

ABSTRACT

Title of Dissertation: **COMBINING PHYSICS-BASED MODELING,
MACHINE LEARNING, AND DATA ASSIMILATION
FOR FORECASTING LARGE, COMPLEX,
SPATIOTEMPORALLY CHAOTIC SYSTEMS**

Alexander Wikner
Doctor of Philosophy, 2023

Dissertation Directed by: **Professor Edward Ott
Department of Physics**

We consider the challenging problem of forecasting high-dimensional, spatiotemporally chaotic systems. We are primarily interested in the problem of forecasting the dynamics of the earth's atmosphere and oceans, where one seeks forecasts that (a) accurately reproduce the true system trajectory in the short-term, as desired in weather forecasting, and that (b) correctly capture the long-term ergodic properties of the true system, as desired in climate modeling. We aim to leverage two types of information in making our forecasts: incomplete scientific knowledge in the form of an imperfect forecast model, and past observations of the true system state that may be sparse and/or noisy. In this thesis, we ask if machine learning (ML) and data assimilation (DA) can be used to combine observational information with a physical knowledge-based forecast model to produce accurate short-term forecasts and consistent long-term climate dynamics.

We first describe and demonstrate a technique called Combined Hybrid-Parallel Prediction (CHyPP) that combines a global knowledge-based model with a parallel ML architecture con-

sisting of many reservoir computers and trained using complete observations of the system’s past evolution. Using the Kuramoto-Sivashinsky equation as our test model, we demonstrate that this technique produces more accurate short-term forecasts than either the knowledge-based or the ML component model acting alone and is scalable to large spatial domains. We further demonstrate using the multi-scale Lorenz Model 3 that CHyPP can incorporate the effect of unresolved short-scale dynamics (subgrid-scale closure).

We next demonstrate how DA, in the form of the Ensemble Transform Kalman Filter (ETKF), can be used to extend the Hybrid ML approach to the case where our system observations are sparse and noisy. Using a novel iterative scheme, we show that DA can be used to obtain training data for successive generations of hybrid ML models, improving the forecast accuracy and the estimate of the full system state over that obtained using the imperfect knowledge-based model.

Finally, we explore the commonly used technique of adding observational noise to the ML model input during training to improve long-term stability and climate replication. We develop a novel training technique, Linearized Multi-Noise Training (LMNT), that approximates the effect of this noise addition. We demonstrate that reservoir computers trained with noise or LMNT regularization are stable and replicate the true system climate, while LMNT allows for greater ease of regularization parameter tuning when using reservoir computers.

COMBINING PHYSICS-BASED MODELING, MACHINE LEARNING,
AND DATA ASSIMILATION FOR FORECASTING LARGE, COMPLEX,
SPATIOTEMPORALLY CHAOTIC SYSTEMS

by

Alexander Wikner

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2023

Advisory Committee:

Professor Edward Ott, Chair/Advisor
Professor Michelle Girvan
Professor Thomas M. Antonsen
Professor Rajarshi Roy
Professor Brian R. Hunt

© Copyright by
Alexander Wikner
2023

Foreword

This dissertation contains work that was co-authored by colleagues and faculty members at the University of Maryland and at other institutions. The author of this thesis made substantial contributions to the relevant aspects of the jointly authored work included in the dissertation.

Dedication

To my mother, for waking me up for school,

To my father, for helping me with my calculus homework, and

To my sister, for staking out my car when my keys were stolen.

Acknowledgments

I would like to express my sincerest gratitude to all those who have made this thesis possible.

First and foremost, I would like to thank my advisor, Prof. Edward Ott. His insight into the challenging problems we have tackled over these past years and his often imaginative solutions to them have been essential to the success of this work. He has never made me want for anything in an advisor; he has been understanding, has always made himself available to me, and has been an excellent resource as I have planned my career beyond the University of Maryland. I can only hope that my future mentors will be as good to me as he has been.

I would next like to thank Prof. Brian Hunt. His mathematical intuition has been invaluable in all of these projects, and he has been an exceptional resource to me as I have learned the in's and out's of data assimilation. As his co-author, I have greatly appreciated his ability to explain complex topics succinctly and unambiguously. I am also immensely grateful for all the time and advice he has given me this past year as I have applied for positions after graduation.

I would likely not be here at the University of Maryland were it not for Prof. Michelle Girvan. As one of my advisors during the TREND Summer REU, she was responsible for introducing me to the exciting field of nonlinear dynamics and machine learning. I will always be grateful for this, in addition to the many other times that she has provided me with advice, recommendations, or ideas.

My thanks are due to the other members of my committee, Prof. Rajarshi Roy and Prof. Thomas Antonsen, for their time spent reviewing my thesis manuscript. I would also like to thank Josiland Chambers and Jessica Crosby for their help navigating my doctoral program and the graduation process. I would additionally like to thank Daniel Serrano for his tireless work as both a COMBINE educator and as TREND coordinator.

I owe so much to my many collaborators and co-authors at the University of Maryland, Texas A & M University, and elsewhere who have helped to make this thesis a reality. Jaideep Pathak played a pivotal role in much of the early technique and code development, for which I will always be grateful. I would also like to acknowledge the contributions from Troy Arcomano, Prof. Istvan Szunyogh, Andrew Pomerance, and Sarthak Chandra. Thanks are additionally due to my other colleagues for their helpful suggestions and comments on this research: Dhruvit Patel, Declan Norton, Ravi Chepuri, Mitchell Tsokatos, and Dylan Elliot.

The friendships that I have made and maintained during my years at the University have helped me in ways that are difficult to describe. To Adam, Dhruvit, Evan, Jake, Jared, Martin, Matt, Nick, Stefano, and Zach: I could hardly have asked for a better graduate cohort. To Alex, Connor, David, Ethan, and Josh: high school might have ended a decade ago, but our friendship has remained an ever-present support in my life. To Em, John, Mike, and Sam: our adventures these past years have kept me sane, and I will continue to cherish them.

The unconditional love and support from my family and from my partner, Em, have allowed me to weather the many hardships one encounters when completing a Ph.D. Their belief in me has been a bulwark against my self-doubt, and they have always given me a home I could come back to whenever I needed it most.

Last but not least, I would like to say that I likely would never have dreamed of becoming

a scientist had it not been for my grandfather, Fred Wikner. His stories about his experiences as a nuclear physicist captivated me as a child, and his later support of my choosing physics as a discipline helped to make that choice possible and motivated me to strive for excellence. I only regret that he passed away before he had the chance to see my thesis finished.

Grandpa, I'm finally going to get my ticket punched. Now, time to press on!

Table of Contents

Foreword	ii
Dedication	iii
Acknowledgements	iv
Table of Contents	vii
List of Tables	x
List of Figures	xi
List of Abbreviations	xix
Chapter 1: Introduction & Background	1
1.1 Chaos and the Limits of Predictability	1
1.2 Numerical Weather Prediction and Climate Modeling	2
1.3 The Rise of Machine Learning	4
1.4 Data Assimilation and Machine Learning	7
1.5 Chapter Outline	8
1.5.1 Combined Hybrid-Parallel Prediction	9
1.5.2 An Iterative Technique for Hybrid Model Training from Sparse and Noisy Observations	9
1.5.3 Stabilizing Machine Learning Prediction of Dynamics using Noise and Noise-Inspired Regularization	10
1.6 Subsequent Projects	10
Chapter 2: Combined Hybrid-Parallel Prediction	12
2.1 Introduction	12
2.2 Reservoir Computing Architecture for CHyPP	15
2.2.1 A Simple Machine Learning Predictor	15
2.2.2 Basic Reservoir Computing	17
2.2.3 Hybrid Reservoir Computing	23
2.2.4 Parallelization	26
2.2.5 Combined Hybrid/Parallel Prediction (CHyPP)	30
2.3 Test Results on the Kuramoto-Sivashinsky Equation	33
2.3.1 Prediction Scalability	35

2.3.2	CHyPP Promotes the Possibility of Good Performance Using a Relatively Small Duration of Training Data	37
2.3.3	Prediction Quality dependence on local overlap length	39
2.4	Test Results on a Multiscale System Prediction and the Problem of Subgrid-Scale Closure	41
2.5	Conclusion and Discussion	48
Chapter 3: An Iterative Technique for Hybrid Model Training with Sparse and Noisy Observations		53
3.1	Introduction	53
3.2	Methods	55
3.2.1	Data Assimilation	56
3.2.2	ML-DA Algorithm	59
3.2.3	Iterated ML-DA Algorithm	64
3.2.4	Experiment Design	66
3.3	Results	70
3.3.1	Lorenz ‘63	70
3.3.2	Kuramoto-Sivashinsky (KS) system	77
3.4	Conclusions and Discussion	81
3.5	Supplemental: The Ensemble Transform Kalman Filter	84
Chapter 4: Stabilizing Machine Learning Prediction of Dynamics using Noise and Noise-Inspired Regularization		86
4.1	Introduction	86
4.1.1	Machine learning prediction of dynamics and the issue of climate stability	87
4.1.2	Stabilization	89
4.1.3	Outline and main results	90
4.1.4	Additional Background: ML Model Robustness to Perturbations	91
4.2	Methods	92
4.2.1	Reservoir computing	93
4.2.2	Training with Regularization	96
4.2.3	Regularization	99
4.2.4	Linearized Multi-Noise Training (LMNT)	104
4.2.5	Prediction and Metrics	109
4.3	Results	113
4.3.1	Kuramoto-Sivashinsky Equation	113
4.3.2	Prediction Test Results	115
4.4	Discussion and Conclusion	126
4.5	Supplemental: LMNT Performance vs. Number of Noise Steps, K	128
4.6	Supplemental: Comparing Performance with Reduced Training Data	131
4.6.1	Reducing the Amount of LMNT Regularization Training Data Only	131
4.6.2	Reducing the Total Amount of Training Data	133
Chapter 5: Concluding Remarks		136

List of Tables

2.1	Hyperparameters used in the results in Sec. 2.3.	35
2.2	Hyperparameters used in the results in Sec. 2.4.	45
3.1	Reservoir and training parameters.	68
4.1	Reservoir computer hyperparameter values used for all prediction tests.	116
4.2	Parameters used for all reservoir training and tests.	116
4.3	The regularization parameters that are searched over for each reservoir prediction test.	117
4.4	Reservoir computer prediction results using different types of regularization. Bold text marks the best performance for the corresponding metric. In every case, regularization parameter values are chosen to maximize the fraction of stable predictions produced by an ensemble of reservoirs over an ensemble of training and testing data sets. If multiple regularization parameter values are found to produce the same fraction of stable predictions, then we choose the regularization parameter(s) from that subset that maximize the median prediction valid time. The $\pm$ error bounds indicate the maximum of the upper and lower 95% confidence intervals for the median [1]. In the case of the valid time, we enlarge this interval slightly to account for the Δt discretization of our predictions.	117
4.5	Reservoir computer prediction results using Tikhonov regularization and LMNT regularization, where we vary the number of noise steps used to compute the LMNT regularization. For a detailed description of how we select regularization parameter values and compute uncertainty bounds, see the Table 4.4 caption. . . .	129
4.6	Reservoir computer prediction results using LMNT regularization computed with different numbers of training samples. For a detailed description of how we select regularization parameter values and compute uncertainty bounds, see the Table 4.4 caption.	132
4.7	The regularization parameters that are searched over for each reservoir prediction test using the reduced training data.	133

List of Figures

2.1	Machine learning prediction device (a) open-loop training phase and (b) closed-loop prediction phase.	16
2.2	Diagram of the reservoir computer setup. In the “open-loop” training phase (analogous to Fig. 2.1(a)), the dashed line representing coupling from the output back to the input is absent. In the “closed-loop” prediction phase (analogous to Fig. 2.1(b)), the coupling from the output back to the input (dashed line) is activated.	18
2.3	Diagram of the hybrid reservoir computer setup. In the “open-loop” training phase (analogous to Fig. 2.1(a)), the dashed line representing coupling from the output back to the input is absent. In the “closed-loop” prediction phase (analogous to Fig. 2.1(b)), the coupling from the output back to the input (dashed line) is activated.	23
2.4	Diagram of parallel reservoir computer setup.	26
2.5	Diagram of the Combined Hybrid/Parallel Prediction (CHyPP) architecture using reservoir computing.	30
2.6	Average normalized root-mean-square error (NRMSE) in the CHyPP prediction where we have fixed the number of system variables predicted in each region to $Q = 8$ and the reservoir spatial density at $P/L = 16/100$. The total number of spatial grid points K in each prediction are $K = 128$ for $L = 100$, $K = 512$ for $L = 400$, and $K = 1024$ for $L = 800$. Predictions are made using an imperfect model with second derivative error $\epsilon = 0.1$. The results shown are the average of 100 predictions where each prediction is made using the same set of reservoirs and training data set but where the CHyPP method is synchronized to different initial conditions for $\mathbf{u}(t)$. we see that the NRMSE curves are relatively invariant as L increases, indicating that our method can be scaled to large systems.	36

2.7	(A) The true dynamics of the Kuramoto-Sivashinsky equation obtained by numerical evolution of Eq.(2.26) normalized to have mean 0 and variance 1. We plot the spatial grid point along the vertical axis and the Lyapunov time on the horizontal axis. The Lyapunov time is defined by $\Lambda_{max}t$ where Λ_{max} is the largest Lyapunov exponent computed from Eq.(2.26). Predictions of this evolution by four different methods are shown in panels (B), (C), (D), and (E). The prediction in (B) is made by a single reservoir computer with 16,000 nodes, the prediction in (C) is made by 16 reservoir computers in parallel each with 4,000 nodes, the prediction in (D) is made by a hybrid with a single reservoir computer with 16,000 nodes(D), and the prediction in (E) is made by CHyPP with 16 reservoir computers in parallel each with 4,000 nodes and an imperfect model with second derivative $\epsilon = 0.1$. The plots on the right (b), (c), (d), and (e) display the difference between the true dynamics (A) and the prediction using each of the corresponding techniques. The vertical black line marks the valid prediction time. Each technique was trained using a 3,375 Lyapunov time sequence of training data.	38
2.8	We plot the average valid time as a function of the duration of the training data, each in Lyapunov time, for the parallel reservoir(s)-only method using (a) 1 reservoir, (b) 4 reservoirs, (c) 8 reservoirs, and (d) 16 reservoirs. We plot the same quantity obtained from the parallel hybrid method using (e) 1 reservoir, (f) 4 reservoirs, (g) 8 reservoirs, and (h) 16 reservoirs. In each case, the imperfect knowledge-based predictor in the hybrid prediction had an error in the second derivative term of $\epsilon = 0.1$, resulting in an average valid time for the model-only prediction of 0.48 of a Lyapunov time. A knowledge-based predictor with no error begun using the most accurate initial condition from the parallel hybrid was able to predict for an average of 11 Lyapunov times. The legend in each plot indicates the number of nodes used in each reservoir.	40
2.9	We plot the valid time for the parallel and non-parallel hybrid and reservoir(s)-only prediction as a function of the local overlap length ℓ	42
2.10	Plot of the true dynamics from Lorenz Model III using the parameters $N = 960$, $K = 32$, $I = 12$, $F = 15$, $b = 10$, and $c = 2.5$. The dynamics have been normalized to set the mean value to 0 and variance to 1. We plot the value of Z color-coded (color bar on the right), the spatial grid point along the vertical axis, and the model time on the horizontal axis.	45
2.11	We plot the predictions of the normalized true dynamics from Lorenz Model III (displayed in Fig. 2.10) where we have measured every spatial grid point of Model III during training. We plot the spatial grid point along the vertical axis and the model time on the horizontal axis. The plots on the left show the predictions made using each of the specified methods, while the plots on the right show the difference between the true dynamics and the prediction (i.e., the prediction error). The parallel reservoir-only prediction used its optimized value of $s = 0.1$, while CHyPP used its optimized value of $s = 0.085$	46

2.12	We plot the predictions of the normalized true dynamics from Lorenz Model III (displayed in Fig. 2.10), where, corresponding to the Model II grid, we have measured <i>only every second</i> spatial grid point of Model III during training. We plot the spatial grid point along the vertical axis and the model time on the horizontal axis. The plots on the left show the predictions made using each of the specified methods, while the plots on the right show the difference between the true dynamics and the prediction (i.e., the prediction error). The parallel reservoir-only prediction used its optimized value of $s = 0.25$, while CHyPP used its optimized value of $s = 0.08$	47
2.13	We plot the predictions of the normalized true dynamics from Lorenz Model III (displayed in Fig. 2.10), where, corresponding to the Model II grid, we have measured <i>only every fourth</i> spatial grid point of Model III during training. We plot the spatial grid point along the vertical axis and the model time on the horizontal axis. The plots on the left show the predictions made using each of the specified methods, while the plots on the right show the difference between the true dynamics and the prediction (i.e., the prediction error). The parallel reservoir-only prediction used its optimized value of $s = 0.25$, while CHyPP used its optimized value of $s = 0.017$	48
2.14	We plot the predictions of the normalized true dynamics from Lorenz Model III (displayed in Fig. 2.10), where, corresponding to the Model II grid, we have measured <i>only every eighth</i> spatial grid point of Model III during training. We plot the spatial grid point along the vertical axis and the model time on the horizontal axis. The plots on the left show the predictions made using each of the specified methods, while the plots on the right show the difference between the true dynamics and the prediction (i.e., the prediction error). The parallel reservoir-only prediction used its optimized value of $s = 0.25$, while CHyPP used its optimized value of $s = 0.0155$	49
2.15	We plot the average valid time for each prediction as a function of the length of training data used. Dashed lines display the results of parallel reservoir-only predictions, whereas solid lines display results from CHyPP where the knowledge-based model is Lorenz Model II with $N = \text{Number of grid points}$, $K = N/30$, and $F = 15$. Each prediction method uses 24 reservoirs.	52
3.1	(a) Training the ML-DA prediction scheme and (b) hybrid forecasting as described in Sec. 3.2.2. The output matrix in (a), $\mathbf{W}_{out}$, is unknown at the time of training. The goal of training is to determine $\mathbf{W}_{out}$ such that $\mathbf{x}_j^H$ best approximates $\mathbf{x}_j^a$ in the regularized least-squares sense.	61
3.2	Training subsequent iterations ($i \geq 2$) of the ML-DA algorithm using the synchronized hybrid model during DA as described in Sec. 3.2.3. The index i denotes the current iteration of the algorithm, and we consider the training procedure shown in Fig. 3.1(a) to occur at $i = 1$. At iteration i , $\mathbf{W}_{out,i-1}$ is known from the previous training iteration, while we endeavor to determine $\mathbf{W}_{out,i}$ such that $\mathbf{x}_{j,i}^H$ best approximates $\mathbf{x}_{j,i}^a$ in the regularized least-squares sense.	65

3.3	Dependence of the forecast Valid Time (t_{VT}) for the Lorenz ‘63 model on the covariance inflation parameter (ρ) when $\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \end{bmatrix}$, measurement noise has magnitude $\sigma_{noise} = 0.1$, and the model error is $\epsilon = 0.1$. Each box plot represents the result of 100 independent trials comparing the baseline and the ML-DA forecasts, each using a different Lorenz ‘63 time series starting from a random initial condition in the attractor. The box plots depict the median, the 25 th and 75 th percentiles, and the 5 th and 95 th percentiles (depicted by dashed lines) computed over these trials. The “*” above a pair of boxes indicates where Mood’s median test determined that the ML-DA median forecast Valid Time was significantly greater than that of the baseline for that value of ρ , with a p-value less than 10^{-3}	73
3.4	Forecast time series plots for the Lorenz ‘63 system using the ML-DA and baseline scheme for $\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$ and model error $\epsilon = 0.1$	75
3.5	Dependence of t_{VT} on the model error (ϵ) for the Lorenz ‘63 system; (a) and (b) differ in the measurement operator used, which is indicated in the figures. Each box plot represents the result of 100 independent trials comparing the baseline and the ML-DA forecasts, each using a different Lorenz ‘63 time series starting from a random initial condition in the attractor. The box plots depict the median, the 25 th and 75 th percentiles, and the 5 th and 95 th percentiles (depicted by dashed lines) computed over these trials. The covariance inflation parameter is independently optimized for both sets of forecasts at each value of the model error (ϵ), and the optimal value is displayed next to each box. The “*” above a pair of boxes indicates where Mood’s median test determined that the ML-DA median forecast Valid Time was significantly greater than that of the baseline for that value of ϵ and optimal ρ , with a p-value less than 10^{-3}	76
3.6	Effect of varying amounts of measurement noise on the baseline (magenta box plots) and ML-DA (light blue box plots) forecast Valid Time for the Lorenz ‘63 system, with model error $\epsilon = 0.2$. Each box plot represents the result of 100 independent trials comparing the baseline and the ML-DA forecasts, each using a different Lorenz ‘63 time series starting from a random initial condition in the attractor. The box plots depict the median, the 25 th and 75 th percentiles, and the 5 th and 95 th percentiles (depicted by dashed lines) computed over these trials. As in Fig. 3.5, the covariance inflation parameter is independently optimized for both sets of forecasts at each value of the measurement noise (σ_{noise}), and the optimal value is displayed next to each box. The “*” above a pair of boxes indicates where Mood’s median test determined that the ML-DA median forecast Valid Time was significantly greater than that of the baseline for that value of σ_{noise} and optimal ρ , with a p-value less than 10^{-3} . The measurement operator is $\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$, and simulated uncorrelated Gaussian noise with standard deviation σ_{noise} is added to the measurements. For comparison, the standard deviation of the X_1 variable is $\sigma_{X_1} \simeq 7.9$	77

3.7	The top 3 panels in this figure are spatiotemporal plots with the 64 spatial grid point locations plotted vertically and the Lyapunov time plotted horizontally. From top to bottom in these panels, the color axis shows the true value of the i^{th} element of $\mathbf{x}_j^t$ for $i = 1, 2, \dots, Q$ from the KS system, the difference between the true i^{th} element of $\mathbf{x}_j^t$ for $i = 1, 2, \dots, Q$ from the KS system and the prediction from the baseline scheme, and the difference between the true i^{th} element of $\mathbf{x}_j^t$ for $i = 1, 2, \dots, Q$ from the KS system and the prediction from the ML-DA scheme. The bottom plot shows the normalized RMS error as function of time for the baseline (magenta) and ML-DA (light blue) schemes, while the dotted vertical lines mark the forecast Valid Time using each scheme. For both schemes, the model error was $\epsilon = 0.1$, and the state was measured at $\Theta = 16$ equally spaced locations of the full $Q = 64$ grid points with measurement noise magnitude $\sigma_{\text{noise}} = 0.1$	80
3.8	Forecast Valid Time of the ML-DA (light blue box plots) and baseline (magenta box plots) schemes for the KS system at different values of the model error (ϵ). Each box plot represents the result of 50 independent trials comparing the baseline and the ML-DA forecasts, each using a different KS time series starting from a random initial condition in the attractor. The box plots depict the median, the 25 th and 75 th percentiles, and the 5 th and 95 th percentiles (depicted by dashed lines) computed over these trials. As in Fig. 3.5, the covariance inflation parameter is independently optimized for both sets of forecasts at each value of the model error (ϵ), and the optimal value is displayed next to each box. The “*” above a pair of boxes indicates where Mood’s median test determined that the ML-DA median forecast Valid Time was significantly greater than that of the baseline for that value of ϵ and optimal ρ , with a p-value less than 10^{-3}	81
3.9	Normalized RMS error during DA as we iteratively use the ML-DA to produce analysis time series for the KS system and use those time series to train the hybrid model. A covariance inflation parameter of $\rho = 2$, which we found to produce the lowest analysis error, was used to produce analysis time series at each step of the iteration. In addition, all knowledge-based models used in this plot had an error of $\epsilon = 0.2$. The plotted values are the median normalized RMS error values computed over 50 different sets of measurement data. The variance in the error over these different data sets is very small; therefore, we have chosen not to plot the standard deviation. At iteration 0, the error values are those from using only the knowledge-based model as our forecast model during DA. We also plot the error values computed over only the measured or the unmeasured variables.	82
4.1	Machine Learning Model with Memory Used for Predicting Dynamics. The above diagrams show an ML model with memory for prediction in the (a) “open-loop” configuration (for training) and (b) “closed-loop” configuration (for prediction). In panels (a) and (b), the dashed line indicates that the output is fed back into the model as the next input.	87

4.2	Diagram of our Reservoir Computer Model. The reservoir is trained in the “open-loop” training phase (analogous to Fig. 4.1(a)). During this phase, the dashed line representing the feedback loop is inactive. Once training is complete, the “closed-loop” prediction phase (analogous to Fig. 4.1(b)) may begin by activating the output-to-input feedback connection (shown by the dashed line). On the left, we indicate the input during training for the standard training (discussed in Sec. 4.2.2) and for noise training (discussed in Sec. 4.2.3.3), as well as the feedback input during prediction (discussed in Sec. 4.2.5).	93
4.3	Comparing Predictions with Increasing $\bar{\epsilon}_{map}$ Values. In this figure, panel (A) shows a histogram of the mean map error values of predictions generated from reservoir computers trained with Jacobian and Tikhonov regularization using a logarithmic grid of regularization parameter values (grid given in Table 4.3). In this panel, the solid blue line marks the $\bar{\epsilon}_{map}$ climate stability threshold, while the dashed red, green, and magenta lines denote the mean map error value for three example predictions: one is a stable prediction with a low $\bar{\epsilon}_{map}$, one is stable with a high $\bar{\epsilon}_{map}$, and one is unstable, respectively. These predictions are obtained using a long $T_{pred} = 1,000,000$ ($12,000 t_{Lyap}$). The dynamics at the end of these predictions are displayed in panel (B), with the stable example prediction with a low $\bar{\epsilon}_{map}$ on the top, the stable example prediction with a high $\bar{\epsilon}_{map}$ in the middle, and the unstable example prediction on the bottom. Panel (C) displays the computed power spectral density of $u_{out,1}(t)$ and $u_{true,1}(t)$ for each of these predictions and the true test data. When computing the power spectral density using Welch’s method, we used a window with 2^{13} samples ($98.304 t_{Lyap}$) and no overlap between windows.	118
4.4	Unstable Predictions of the KS Equation. The top panel shows a testing time series $\mathbf{u}_{true}(t)$ generated from the true KS equations dynamics, while the bottom panels show results from a reservoir computer trained with the regularization type displayed between the left and right panels. Panels on the left show the reservoir prediction, $\mathbf{u}_{out}(t)$, while those on the right show the difference between the reservoir prediction and the true evolution $\mathbf{u}_{true}(t)$. In all panels, the horizontal axis denotes the time measured in Lyapunov times based on the largest positive Lyapunov exponent of the true system, while the vertical axis denotes the spatial coordinate. In the top pa and in the panels on the left, the color denotes the values of $\mathbf{u}_{true}(t)$ and $\mathbf{u}_{out}(t)$, respectively, while in panels on the right, the color denotes the difference, $\mathbf{u}_{out}(t) - \mathbf{u}_{true}(t)$, between the prediction and true dynamics. An entire prediction at a particular time being colored black indicates that this prediction has become unstable and left the region of the true attractor. In the right panels, the cyan line denotes the corresponding prediction valid time.	119

4.5	Stable Predictions of the KS Equation. The top panel shows a testing time series $\mathbf{u}_{true}(t)$ generated from the true KS equations dynamics, while the bottom panels show results from a reservoir computer trained with the regularization type displayed between the left and right panels. Panels on the left show the reservoir prediction, $\mathbf{u}_{out}(t)$, while those on the right show the difference between the reservoir prediction and the true evolution $\mathbf{u}_{true}(t)$. In all panels, the horizontal axis denotes the time measured in Lyapunov times based on the largest positive Lyapunov exponent of the true system, while the vertical axis denotes the spatial coordinate. In the top panel and in the panels on the left, the color denotes the values of $\mathbf{u}_{true}(t)$ and $\mathbf{u}_{out}(t)$, respectively, while in panels on the right, the color denotes the difference, $\mathbf{u}_{out}(t) - \mathbf{u}_{true}(t)$, between the prediction and true dynamics. Black coloring marks where the prediction (left panels) or prediction difference (right panels) is outside the range given by the color bar.	120
4.6	Map Error over Time During Reservoir Predictions. This figure shows the map error ϵ_{map} during each of the unstable predictions shown in Figs. 4.4 and (dotted lines), stable predictions shown in Fig. 4.5 from reservoirs trained with a single regularization type (dashed lines), and stable predictions shown in Fig. 4.5 from reservoirs trained with multiple regularization types (solid lines).	121
4.7	Average Predicted PSD Compared to True PSD. The true KS equation PSD is computed over a single trajectory of duration $T_{pred} = 10,000,000(120,000 t_{Lyap})$, while the prediction PSD is computed as the average over predictions of duration $T_{pred} = 16,000 (192 t_{Lyap})$ produced by the ensemble of reservoirs, training data sets, and testing initial conditions used to test the prediction climate. When computing the PSD with Welch's method, we used a window with 2^{13} samples ($98.304 t_{Lyap}$) and no overlapping windows. The grey filled region surrounding the PSD curve for the true system shows the 95% confidence interval in the mean spectra computed over these windows. The color-filled region surrounding the prediction PSD curves shows the 95% confidence interval in the mean spectra computed over these windows and over the different predictions. We note that for some frequency values, the line width of the plotted PSD curve is wider than the corresponding shaded region. The cutout in the bottom-center panel shows a magnified view of the PSD and 95% confidence interval between frequencies of 0.9 and $0.98 1/t_{Lyap}$	124
4.8	Fraction of Stable Predictions and Median Valid Time using Varying Regularization Parameter Values. Color plots displaying the fraction of stable predictions and median prediction valid time over a grid of different regularization parameters values are shown on the top and bottom, respectively. The regularization parameters used for each regularization type are plotted along the vertical and horizontal axes, respectively. The red dots in the bottom panels mark the regularization parameter values chosen to obtain the largest fraction of stable prediction and to produce the largest median prediction valid time, which is noted in the white box in the top right of the respective panel.	125

4.9	Fraction of Stable Predictions and Median Valid Time using Varying Regularization Parameter Values for Reduced Training Data . Color plots displaying the fraction of stable predictions and median prediction valid time over a grid of different regularization parameters values are shown on the top and bottom, respectively. In the median prediction valid time plots, the red dot marks the regularization parameters values that result in the largest fraction of stable predictions; if there are multiple values with the same fraction, then we mark those that result in the longest median prediction valid time. The blue crosses mark the regularization parameters values which result in the longest median prediction valid time; if there are multiple values with the same valid time, then we mark those that result in the largest fraction of stable predictions. Finally, the green x's mark the parameters values where we have performed the same optimization as for the red dot, but have excluded Tikhonov regularization parameters values above 10^{-4} . If multiple values remain after any of these optimizations, we mark the values with the lowest regularization, giving lowest priority to Tikhonov regularization.	134
-----	--	-----

List of Abbreviations

ANN	Artificial Neural Network
CHyPP	Combined Hybrid-Parallel Prediction
CNN	Convolutional Neural Network
DA	Data Assimilation
DL	Deep Learning
ETKF	Ensemble Transform Kalman Filter
FPGA	Field Programmable Gate Array
GCM	Global Climate Model
GPU	Graphics Processing Unit
LLM	Large Language Model
LMNT	Linearized Multi-Noise Training
ML	Machine Learning
NWP	Numerical Weather Prediction
RC	Reservoir Computer

Chapter 1: Introduction & Background

1.1 Chaos and the Limits of Predictability

This thesis deals with the important problem of accurately modeling high-dimensional chaotic dynamical systems, with a particular interest in addressing terrestrial weather prediction and climate modeling. The problem of weather predictability can trace its way back to the seminal work that jump-started the study of chaotic dynamics: Edward Lorenz’s 1963 paper, “Deterministic Nonperiodic Flow” [2]. Lorenz, with the help of Margaret Hamilton, had developed a primitive model of a weather system from a small set of differential equations. These equations were simulated by Lorenz and Ellen Fetter on a Royal McBee LGP-30 electronic computer that was capable of producing one day of crude weather forecasts every minute. As part of his study of the model, Lorenz wished to run a long-duration prediction beginning from an initial condition taken from the middle of a previous forecast. Lorenz input the initial conditions into the computer from a printout that rounded the output to the third decimal place, assuming that the additional three decimal places stored in the computer memory but not printed would be inconsequential to the results. Returning an hour later, however, he found that this small discrepancy in the initial conditions had caused this prediction to diverge from the earlier output [3]. Lorenz had discovered that physically-derived, deterministic models could become unpredictable at long time scales.

Scientists and mathematicians since 1963 have continued to develop the study of deterministic but unpredictable nonlinear dynamics. These include: James Yorke, who first coined the term “chaos” to describe this type of dynamics [4], Stephen Smale, who expanded understanding of the underlying mechanisms of chaotic systems and the prevalence of unpredictability [5], as well as my thesis advisor, Edward Ott, who, among other contributions, studied the control of chaotic dynamical systems [6].

1.2 Numerical Weather Prediction and Climate Modeling

The expansion of the study of the fundamental limits of predictability in nonlinear dynamical systems coincided with the beginning of useful weather forecasting using numerical models run on computers. The possibility of numerical weather prediction (NWP) and the partial differential equations needed to be solved to make it a reality were first proposed by American meteorologist Cleveland Abbe [7] and Norwegian scientist Vilhelm Bjerknes [8]. The Meteorological Research Project, begun by John von Neumann in 1946 at the Institute for Advanced Study (IAS) at Princeton, saw the first attempts to use the new technology of electronic computers to obtain numerical solutions to simplified versions of the proposed NWP equations in 1950 [9]. By the 1970’s, supercomputers had advanced to the point that it became feasible to integrate a full set of physics-based equations with realistic modeling of unresolved or unaccounted-for physics (in meteorology, this is referred to as “parametrization” [10].) Since then, further advances in computational power and the parametrizations of subgrid-scale processes, as well as improvements in forecast uncertainty estimation methods and forecast initialization, have dramatically increased the forecast skill of NWP models; a 5-day forecast made in 2014 had a higher average

forecast skill than a 3-day forecast made in 1981 [11]. This period has also seen the development of global climate models (GCMs) that couple atmospheric, oceanic, and land-based components to forecast the terrestrial climate variability over decades-long time scales and under different climate change scenarios.

The utility of accurate weather and climate forecasts to society is difficult to understate. Accurate weather forecasts help to mitigate the impacts of damaging weather events and save lives. The economic value of weather forecasts to U.S. households alone was estimated in 2009 to be approximately \$31.5 billion dollars per year, at a total cost to the public and private sector of \$5.1 billion dollars per year [12]. Long-term GCM predictions, on the other hand, provide information essential to public policy, water resource management, urban planning, agriculture, and transportation.

Despite the astounding improvement in NWP models and GCMs in recent decades, current state-of-the-art models still contain substantial imperfections that, in conjunction with the fundamental unpredictability of chaotic systems, lead to erroneous forecasts. These models cannot capture dynamics at the sub-grid scale (e.g., clouds, turbulent fluxes, and interactions with small-scale geographic features). In addition, the parametrizations used to model the effect of unresolved or unmodeled physics will always be imperfect and will lead to model error. These errors are more substantial in GCM's that use a comparatively coarser model grid. The errors in existing models and the significance of chaotic systems to society mean that it is critical to develop techniques to improve prediction of high-dimensional chaotic dynamical systems in a manner that leads to both more accurate short-term forecasts while also capturing the long-term climate variability.

1.3 The Rise of Machine Learning

Recent decades have borne witness to profound advances in the field of machine learning (ML). The current revolution has its roots in latter half of the 20th century, when computer scientists developed the first artificial neural network (ANN)-based ML models that aimed to mimic the activations and connections of neurons in the brain. One of the first such models was the perceptron, developed in 1958 [13]; another important class of ANNs, the convolutional neural network (CNN), was first developed and used to classify handwritten numbers in 1989 [14]. The current paradigm of deep learning (DL) models, consisting of many different ANN layers arranged in series, began around 2009 when it became practical to train and infer with large DL models using graphics processing units (GPUs) [15]. Since then, machine learning algorithms have achieved otherwise inaccessible performance in facial recognition and image classification tasks, and have been used to defeat top-ranking players in games such as Go [16]. More recently, generative machine learning models based on stochastic diffusion have been used to create images from text descriptions [17], while transformer-based large language models (LLMs) like chatGPT have been used to mimic human responses to prompts.

Machine learning models have proved to be effective at analysing and predicting the dynamics of chaotic systems where the equations of motion of said system are unknown. One of the first papers to demonstrate this was that of Jaeger and Haass [18], who used a particular machine learning technique known as “reservoir computing” to predict trajectories of the low-dimensional Mackey-Glass system using only state observations of that system. The reservoir computing technique has since been used to replicate the attractor of other unknown chaotic systems, such as the Lorenz ‘63 equations and the spatiotemporal chaotic Kuramoto-Sivashinsky equation [19, 20].

For large spatiotemporal chaotic systems, Pathak et al. demonstrated that a technique using many reservoir computers in parallel, each trained to predict the dynamics of a small local region of the system, could be used to make accurate forecasts of high-dimensional chaotic systems [21].

While these machine learning models have made accurate predictions by learning dynamics from state observations, they do not make use of all the physical information that is typically available. Namely, there typically already exists an imperfect knowledge-based forecast model component. The possibility of explicitly combining a knowledge-based forecast model and a machine learning component to predict low-dimensional chaotic dynamics was explored by this author and colleagues in Pathak et al. 2018 [22]. In this study, we applied our “hybrid” machine learning approach, which used a reservoir computer as the machine learning component, to predict the dynamics of the Lorenz ‘63 equations and the Kuramoto-Sivashinsky equation using the same equation(s) with initial internal parameters perturbed from their true values as our knowledge-based forecast models. We found that (a) our hybrid approach substantially improved the accuracy of predictions compared to either model acting alone and that (b) our hybrid approach produced accurate forecasts even when the predictions from each component model were so poor that they were essentially useless. Moreover, we found that our hybrid approach was able to replicate the true system climate in cases where the component models did not and that it generally required less training data than the machine learning model alone. While this approach provided a general method for forecasting low-dimensional chaotic systems when both state observations and an existing forecast model are available, it does not scale well to high-dimensional chaotic systems. In addition, this technique requires a time series of complete observations of the true system state that may not be available.

Significant advances in computing power and memory, in addition to the availability of

large time series data sets from GCM predictions (e.g., CMIP5 [23]) and high-resolution estimates of current and past atmospheric states obtained using state-of-the-art NWP models and observation networks (e.g., ERA5 [24]), have enabled the development of fully ML-based NWP models learned from data. Efforts to develop an “ML surrogate” atmospheric model were first published in 2019 [25]. From 2019-2021, a number of groups developed ML models to forecast the weather at a reduced resolution and/or for a subset of forecast variables (e.g., [26–30]); this includes Arcomano et al. 2020 [31], who developed a reduced-resolution ML surrogate model that implements the parallel reservoir technique of Pathak et al. 2018. In 2022, the first ML surrogate model that forecast on the 0.25° -grid of high-resolution reanalysis was developed using graph neural networks [32]. This was followed soon after by a number of other models, including Nvidia’s FourCastNet [33], Google DeepMind’s Graphcast [34], and Huawei’s PanguWeather [35]. The most accurate of these, Graphcast and PanguWeather, demonstrate forecast skills on par with state-of-the-art NWP models, despite operating at a lower resolution. In addition, these models can produce forecasts in a small fraction of the time and, once trained, with considerably less computer resources than comparable NWP models. This speed up has the potential to revolutionize weather forecasting by substantially increasing the size and number of ensemble forecasts, thereby improving analysis and uncertainty estimation (By ensemble forecasts, we refer to a collection of forecasts obtained from initial conditions spanning the uncertainty range of the estimated forecasts of the atmospheric state.) So long as these models remain as accessible as they currently are, this speed-up also has the potential to democratize weather forecasting so that groups with a relatively small amount of computing resources can obtain skillful weather forecasts.

The current state-of-the-art ML surrogate models are not without their flaws, however,

Their forecasts often underestimate atmospheric variability, making it uncertain if they will provide accurate extreme event statistics when used for ensemble forecasting. In addition, these models must be used autoregressively (i.e., with the output fed back into the model as input) to make medium-to-long range predictions. This process often leads to unbounded error growth or climate drift, making these models unsuitable for long-term climate prediction. Finally, while these models implicitly make use of scientific knowledge by training on data from climate model simulations or reanalysis, they contain no explicit scientific knowledge, even in a simplified form. The potential for improving both short and long-term forecasts of high-dimensional chaotic systems, such as the weather, makes developing scalable hybrid techniques that combine ML with a knowledge-based model of paramount importance.

1.4 Data Assimilation and Machine Learning

So far, our discussion of predicting high-dimensional chaotic systems has focused on what Bjerknes termed the “prognostic” step in forecasting, where we seek to accurately evolve the system state from an initial condition to the state at a future time [8]. Before this step, however, Bjerknes envisioned forecasters completing a “diagnostic” step, in which the initial condition for the forecast model is determined from observations. In operational weather forecasting, the diagnostic and prognostic steps are completed cyclically; observations of the atmosphere are constantly taken and combined with the most recent NWP model forecasts to estimate the atmospheric state at regular intervals. These observations are taken at different times and places, are sparse and patchy, and may be of variables that are not a part of our physics-based NWP model. Combining recent forecasts with the older and more recent observations is the task of data as-

simulation (DA). The development and advancement of DA techniques has been one of the main sources of the substantial improvement in weather forecast skill since the beginnings of NWP.

Whereas the assimilated observations may be irregular in space and time and noisy, the estimated state obtained via data assimilation, called the “analysis”, is a full system state on the forecast model grid. Full system states are typically what is required to train an ML model to forecast dynamics; indeed, many of the state-of-the-art ML surrogate models are trained on “re-analysis” states, which are estimated by using current operational NWP models to assimilate past observations. Thus, performing data assimilation prior to training can be necessary if one desires to train their ML model with observational information. However, as discussed in Sec. 1.2, the knowledge-based forecast models which are used during data assimilation often have substantial errors. By training on the analysis generated by these models, the ML model will “inherit” some of this error; this is the case even for state-of-the-art ML surrogate models trained on high-resolution reanalysis (e.g., precipitation forecasts from Graphcast [34]). Given these challenges, it remains an open question as to how to obtain the best performing forecast model using machine learning when one has access to an existing scientific forecast model and sparse, noisy observations.

1.5 Chapter Outline

In this thesis, we consider how machine learning can be combined with physical knowledge-based forecasting and data assimilation to produce substantially more accurate short and long-term predictions of high-dimensional, complex spatiotemporally chaotic systems, with an ultimate view of applying the discussed techniques to weather forecasting and climate modeling.

Chapters 2-4 of this thesis each consider a different aspect of this difficult prediction problem as well as techniques to address them. In Chapter 5, we give some brief concluding remarks.

1.5.1 Combined Hybrid-Parallel Prediction

In Chapter 2, we introduce Combined Hybrid-Parallel Prediction (CHyPP), which merges a global knowledge-based model with a scalable, parallel machine learning framework that employs multiple reservoir computers. We test the short-term forecast skill of this technique against the knowledge-based and ML component methods, as well as against the hybrid approach of Pathak et al. 2018, by using each to predict the dynamics of two paradigmatic spatiotemporally chaotic systems: the Kuramoto-Sivashinsky equation and the multiscale Lorenz Model 3. This chapter was published in the journal *Chaos* [36].

1.5.2 An Iterative Technique for Hybrid Model Training from Sparse and Noisy Observations

In Chapter 3, we illustrate how DA can be harnessed to broaden the application of the Hybrid ML methodology for forecasting complex chaotic systems to scenarios with sparse and noisy system observations. We describe a novel iterative technique for hybrid model training that alternates between using DA to assimilate the observations, then training a hybrid model on the resulting analysis. We test this technique using the Ensemble Transform Kalman Filter to assimilate sparse and noisy observations of the simple 3-variable Lorenz '63 equations and of the Kuramoto-Sivashinsky equation dynamics. This chapter was published in the journal *Chaos* [37]

1.5.3 Stabilizing Machine Learning Prediction of Dynamics using Noise and Noise-Inspired Regularization

In Chapter 4, we consider the problem of climate instability in long-term ML model forecasts of complex chaotic systems. Motivated by the commonly-used instability mitigating technique of adding noise to the ML model input during training, we formulate a novel training technique, Linearized Multi-Noise Training, that approximates the effect of this noise. We then systematically compare the short-term forecast accuracy and long-term climate replication of reservoir computers trained using noise training, LMNT, and other well-known techniques. This chapter was published in the journal *Neural Networks* [38].

1.6 Subsequent Projects

Since completing and testing Combined Hybrid-Parallel prediction (described in Chapter 2), we have worked to implement this parallel hybrid approach to combine a low-resolution knowledge-based atmospheric model with an ML component [39] and a fully ML-based ocean model [40]; this model produces more accurate weather forecasts than either component model, has very low biases over a 70-year stable free run, and is able to capture processes not captured by the component atmospheric model.

We have also followed up our work using data assimilation to train hybrid models using sparse and noisy observations. Our recent work has been into developing an online technique to perform the state estimation and model training simultaneously, while also correcting the potentially erroneous mapping between observation and model space [41]. I have also worked on data

assimilation-related projects outside of the University of Maryland. While interning at Argonne National Lab from May to August 2023, I created a variational DA system to assimilate global radiosonde observations using a state-of-the-art ML weather forecasting model.

Outside of weather and climate, I was a fellow in the Computation and Mathematics for Biological Networks (COMBINE) NSF NRT program, where I participated in coursework, seminars, and research with a focus on network science and its application areas. As part of this program, I created an ML-based technique for context-dependent pooling of predictions from different epidemiological forecast models and applied it to predict deaths due to the COVID-19 pandemic [\[42\]](#).

Chapter 2: Combined Hybrid-Parallel Prediction

This chapter is based on the paper, Combining machine learning with knowledge-based modeling for scalable forecasting and subgrid-scale closure of large, complex, spatiotemporal systems, by Alexander Wikner, Jaideep Pathak, Brian Hunt, Michelle Girvan, Troy Arcomano, Istvan Szunyogh, Andrew Pomerance, and Edward Ott, Chaos 30, 053111 (2020).

2.1 Introduction

In recent years, machine learning techniques have been used to solve a number of complex modeling problems ranging from effective translation between hundreds of different human languages [43] to predicting the bioactivity of small molecules for drug discovery [44]. Typically, the most impressive results have been obtained using artificial neural networks with many hidden neural states [45]. These hidden layers form a “black box” model where internal parameters are trained given a set of measured training data but, after which, only the final model output is observed. This formulation using measured training data contrasts with how models used for forecasting physical spatiotemporally chaotic processes are formulated, which is typically done using the available scientific knowledge of the underlying mechanisms that govern the system’s evolution. For example, in the case of forecasting weather, this knowledge includes the Navier-Stokes equations, the first law of thermodynamics, the ideal gas law, and (see Sec. 2.4) simplified

representations of physics at the unresolved spatial scales [11].

In this chapter, focusing on the key issues of scalability and unresolved subgrid physics, we consider the general problem of forecasting a very large and complex spatiotemporally chaotic system where we have access to both past time series of measurements of the system state evolution and to an imperfect knowledge-based prediction model. We present a method for combining machine learning prediction with imperfect knowledge-based forecasting that is scalable to large systems with the aim that the resulting combined prediction system can be significantly more accurate and efficient than either a pure knowledge-based prediction or a pure machine-learning-based prediction. A main source of difficulty for scalability of the machine learning is that the dimension of the state of the systems we are interested in can be extremely large. For example, in state-of-the-art global numerical weather models the state dimension (number of variables at all grid points) can be on the order of 10^9 . Thus, both the machine learning input (the current atmospheric state) and output (the predicted atmospheric state) have this dimensionality. (In contrast to the description of some machine learning techniques as “deep”, one might refer to the situations we address as “wide”.) The prediction method that we propose for such large complex systems builds on the previous work on parallelizable machine learning prediction [21] and hybridization of knowledge-based modeling with machine learning [22]. We call our technique Combined Hybrid/Parallel Prediction (CHyPP, pronounced “chip”). Although the general method we propose is applicable to different kinds of machine learning, the numerical examples presented in this chapter use a machine learning method known as reservoir computing [46, 47]. Jaeger and Haas [18] described the effectiveness of reservoir computing for predicting low-dimensional chaotic systems. Research surrounding this technique has since expanded [48, 49], and it has recently been shown that reservoir computing using recurrent neural networks can produce similar

quality predictions for chaotic systems to those of other recurrent architectures, such as LSTM's [50] and GRU's [51], while often requiring much less computational time to train [52]. Reservoir computing techniques can additionally be extended to physical implementations using, e.g., photonics [53] and Field Programmable Gate Arrays (FPGA's) [54, 55].

The rest of the chapter is organized as follows. In Sec. 2.2, we first review a simple version of reservoir computing [46, 47] and discuss its shortcomings for forecasting high-dimensional spatiotemporal chaos. We next describe the hybrid reservoir prediction technique (Refs. [22] and [56]), as well as previous work on how machine learning can be parallelized for prediction of spatiotemporal systems [21]. We then present our proposed CHyPP architecture combining the two. In Sec. 2.3, we demonstrate how the CHyPP methodology improves on each of the component prediction methods. For these demonstrations we use the paradigmatic example of the Kuramoto-Sivashinky model as our test model of the spatiotemporally chaotic system that we aim to predict. We highlight the scalability of the proposed method to very large systems as well as its efficient use of training data, which we view as the crucial issues for the general class of applications in which we are interested. In Sec. 2.4, we consider a situation with multiple time and space scales and show by numerical simulation tests, that CHyPP can, through its use of data, effectively account for unknown subgrid scale processes. The main conclusion of this chapter is that our CHyPP methodology provides an extremely promising framework, potentially facilitating significant advances in the forecasting of large, complex, spatiotemporally chaotic systems. We believe that, in addition to weather, the method that we propose may potentially be applicable to a host of important areas, enabling currently unattainable capabilities. Some speculative examples of potential applications are forecasting of ocean conditions, forecasting conditions in the solar wind, magnetosphere and ionosphere (also known as 'space weather', important for its

impact on Earth-orbiting satellites, GPS accuracy, and high frequency communications), forecasting the evolution of forest fires and their response to mitigating interventions, forecasting the responses of ecological systems to climate change, analysis of neuronal activity, etc.

2.2 Reservoir Computing Architecture for CHyPP

2.2.1 A Simple Machine Learning Predictor

To begin, we initially consider the goal of a generic machine learning system for time-series prediction of an unknown dynamical system evolving on an attractor of that system. Later, we will consider that the machine learning system is a reservoir computer and that the unknown chaotic system is not completely unknown, and we will try to make use of that knowledge. Given a finite duration time series of the unknown system state's evolution up to a certain time t_0 , where the state at each time is represented by the K -dimensional vector $\mathbf{u}(t) = [u_1(t), u_2(t), \dots, u_K(t)]^T$, our goal is to predict the subsequent evolution of the state. As illustrated in Fig. 2.1(a), in the initial training phase, at each time $t = n\Delta t \leq t_0$, $\mathbf{u}(t)$ is input to the machine learning system ($\mathbf{u}_{in}(t) = \mathbf{u}(t)$), which is trained to output a time Δt prediction of the dynamical system state $\mathbf{u}(t + \Delta t)$ ($\mathbf{u}_{out}(t + \Delta t) \simeq \mathbf{u}(t + \Delta t)$). We refer to the just-described input-output configuration as the “open-loop” system (Fig. 2.1(a)). To ensure an accurate representation of the true dynamics with a reservoir of limited size, Δt is typically short compared to natural time scales (such as the correlation time or the “Lyapunov time”) present in the unknown dynamical system whose state is to be predicted. Once trained, the machine learning system can be run in a “closed-loop” feedback configuration (Fig. 2.1(b)) to autonomously generate predictions over a finite duration of time. That is, with t_0 representing the time at the

end of the training data, we replace the former input from the training data by the output by inserting an output-to-input feedback loop, shown by the dashed line in Fig. 2.1(b). Then, when $\mathbf{u}_{in}(t_0) = \mathbf{u}(t_0)$ is the input, the reservoir computer produces an output prediction for $\mathbf{u}(t_0 + \Delta t)$, which we refer to as $\tilde{\mathbf{u}}(t_0 + \Delta t) = \mathbf{u}_{out}(t_0 + \Delta t)$. When this predicted state is then used as the input ($\mathbf{u}_{in}(t_0 + \Delta t) = \tilde{\mathbf{u}}(t_0 + \Delta t)$), the reservoir computer produces an output prediction for $\mathbf{u}(t_0 + 2\Delta t)$, denoted $\tilde{\mathbf{u}}(t_0 + 2\Delta t)$ ($\mathbf{u}_{out}(t_0 + 2\Delta t) = \tilde{\mathbf{u}}(t_0 + 2\Delta t)$). This process is then iterated to produce predictions of the system state at $t = t_0 + m\Delta t$ for $m = 1, 2, 3, \dots$ (Fig. 2.1(b)).

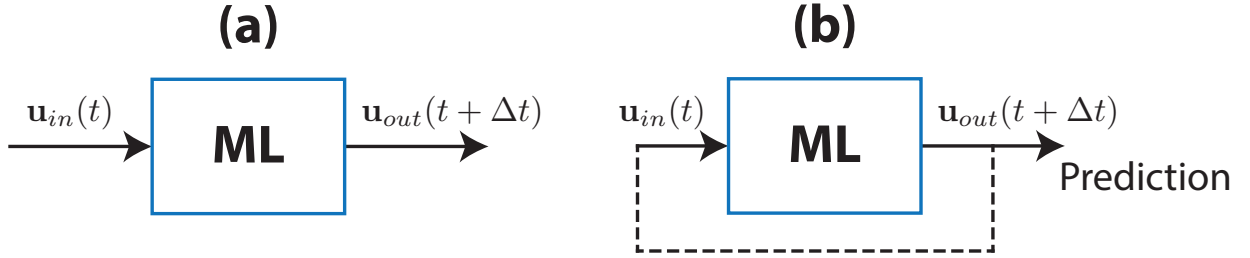


Figure 2.1: Machine learning prediction device (a) open-loop training phase and (b) closed-loop prediction phase.

In the rest of this section, we first present background from previous work (Secs. 2.2.2-2.2.4), then introduce our CHyPP method for combining a knowledge-based model with reservoir-based machine learning to form a scalable system concept suitable for state prediction of very large, spatiotemporally chaotic systems. Specifically in Sec. 2.2.2, we review a basic reservoir computing setup based on the methods of [47, 57] along with the proposal for its use as a predictor carried out in Ref. [18]. In Sec. 2.2.3, we build upon the simple setup of Sec 2.2.2 and describe the methodology from Ref. [21] for hybrid forecasting of the dynamical system using a single reservoir computing network and an imperfect model. In Sec. 2.2.4, the reservoir computing forecasting technique of Sec. 2.2.2 is extended via parallelization of the machine learning with multiple parallel reservoir computers, in order to predict high-dimensional spatiotemporally

chaotic systems, as was first described in [21] (but without the incorporation of a knowledge-based model). Finally, in Sec. 2.2.5, we present our proposed CHyPP architecture and technique for combining the parallel reservoir method of Sec. 2.2.4 with the hybridization of a knowledge-based predictor and a parallel reservoir-based machine learning prediction of Sec. 2.2.3. It is our belief that it is only by means of such a combination that the most effective application of machine-learning-enabled prediction can be realized for large, complex, spatiotemporally chaotic dynamical systems.

2.2.2 Basic Reservoir Computing

We now consider that the ML device shown in Fig. 2.1 is a reservoir computer which, as shown in Fig. 2.2 (and further discussed subsequently), consists of a linear input coupler ($\mathbf{W}_{in}$) that couples the state $\mathbf{u}_{in}(t)$ into the reservoir (the circle in Fig. 2.2). The state of the reservoir is given by a high-dimensional vector $\mathbf{r}(t)$, which is then linearly coupled by $\mathbf{W}_{out}$ to produce an output vector $\mathbf{u}_{out}(t + \Delta t)$ which, through the training, is a very good approximation to $\mathbf{u}(t + \Delta t)$. In this chapter, our implementation of the reservoir is an artificial recurrent neural network with a large number of nodes. The artificial neural network that forms our basis of the reservoir computing implementation is illustrated in Fig. 2.2. The reservoir network adjacency matrix is chosen to be a randomly generated, sparse matrix $\mathbf{A}$ that represents a directed graph with weighted edges. The adjacency matrix $\mathbf{A}$ has dimensions $D_r \times D_r$, where D_r is the number of nodes in the network. Elements of $\mathbf{A}$ are randomly generated such that the average number of incident connections per node (average number of nonzero elements of the matrix in each row) is set to a chosen value $\langle d \rangle$, the “average in-degree”, while the nonzero elements of $\mathbf{A}$ are

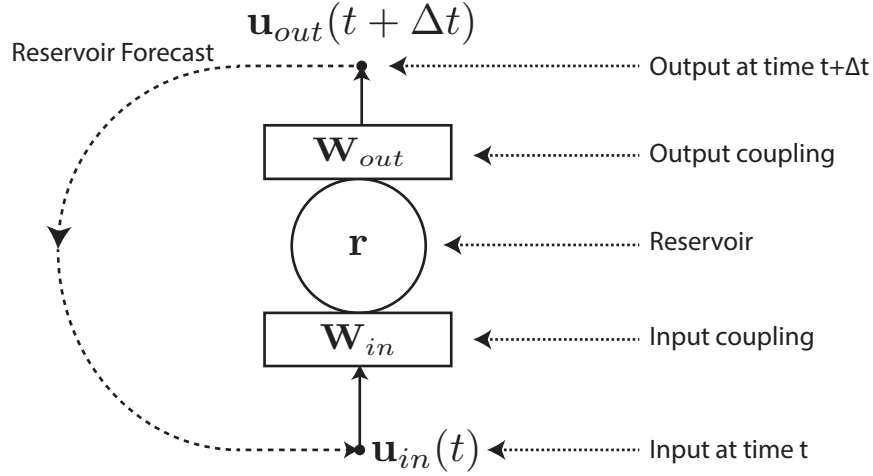


Figure 2.2: Diagram of the reservoir computer setup. In the “open-loop” training phase (analogous to Fig. 2.1(a)), the dashed line representing coupling from the output back to the input is absent. In the “closed-loop” prediction phase (analogous to Fig. 2.1(b)), the coupling from the output back to the input (dashed line) is activated.

chosen from a uniform distribution over the interval $[-1, 1]$. Once generated, $\mathbf{A}$ is re-scaled (i.e., multiplied by a scalar) so that its largest absolute eigenvalue is a prescribed value ρ , called the spectral radius. Each node i in the network has an associated scalar state $r_i(t)$. The state of the network is represented by the D_r dimensional vector $\mathbf{r}(t)$, whose elements are $r_i(t)$ where $i = 1, 2, 3, \dots, D_r$.

The reservoir network state $\mathbf{r}(t)$ evolves dynamically while receiving input through a $K \times D_r$ input coupling matrix, $\mathbf{W}_{in}$. We choose the matrix $\mathbf{W}_{in}$ to contain an equal number of nonzero elements in each column, which corresponds to coupling each element of the reservoir input to an equal number of reservoir node states. Nonzero elements of this matrix are selected randomly and uniformly from the interval $[-\sigma, \sigma]$, where σ is referred to as the input weight. Given the current state of the reservoir $\mathbf{r}(t)$, the reservoir state is advanced at each time using a hyperbolic

tangent activation function,

$$\mathbf{r}(t + \Delta t) = \tanh[\mathbf{A}\mathbf{r}(t) + \mathbf{W}_{in}\mathbf{u}_{in}(t)]. \quad (2.1)$$

Before prediction begins, the reservoir computer is trained in the “open-loop” configuration. During this training phase, $\mathbf{u}_{in}(t) = \mathbf{u}(t) + s\boldsymbol{\eta}(t)$. Here, $\mathbf{u}(t)$ is the measurement of dynamical system state at time t in the form of a K -dimensional vector. As in Ref. [57], we add a small, normally distributed K -dimensional vector $s\boldsymbol{\eta}(t)$ of mean 0 and standard deviation s to the input dynamical system state during training. The elements of the vector $\boldsymbol{\eta}(t)$ are chosen randomly and independently at each time t . The function of this added “stabilization noise” is to allow the reservoir computer to learn how to return to the true trajectory when the input trajectory has been perturbed away from it. We find that, in many cases, this additional small noise input beneficially promotes stability of the closed-loop prediction configuration once training has been completed.

The adjustable constants characterizing the overall prediction system (e.g., D_r , $\langle d \rangle$, σ , s , and Δt) are referred to as “hyperparameters”, and it is important that they be chosen carefully in order for the reservoir computer to predict accurately. For example, as explained in previous literature, the hyperparameters can be chosen so that the reservoir system has the so-called “echo-state property” (see, e.g., Ref. [57]) whereby, when in the open-loop training phase, the reservoir state $\mathbf{r}(t)$, aside from an initial transient and with the random sequence $\boldsymbol{\eta}(t)$ fixed, the reservoir state $\mathbf{r}(t)$ becomes uniquely determined by the reservoir input sequence $\mathbf{u}(t)$ (and hence independent of the initial values of $\mathbf{r}$). Accordingly, prior to initiation of the training, we ignore and discard the reservoir and input states for the first few time steps. The state of the reservoir

at the end of this transient nullification period is labelled $\mathbf{r}(0)$. Starting with $\mathbf{r}(0)$, the training system states $\mathbf{u}(j\Delta t)$ (j an integer, $j\Delta t \leq t_0$) and the resulting reservoir states, $\mathbf{r}((j+1)\Delta t)$, are recorded and saved. We then desire to use these saved states to produce an output, $\tilde{\mathbf{u}}(t + \Delta t)$, when $\mathbf{u}(t)$ is the input, which we desire to be very close to $\mathbf{u}(t + \Delta t)$. To do this, we find it useful to perform an ad-hoc operation on the reservoir state vectors that squares the value of half of the node states. Specifically, we define $\tilde{\mathbf{r}}(j\Delta t)$ such that,

$$\tilde{r}_i = r_i \text{ for } i \text{ odd}, \quad (2.2)$$

$$\tilde{r}_i = r_i^2 \text{ for } i \text{ even}. \quad (2.3)$$

As surmised in footnote [16] of Ref. [21], this operation improves prediction by breaking a particular odd symmetry of the reservoir dynamics that is not generally present in the dynamics to be predicted. We next couple the transformed reservoir state $\tilde{\mathbf{r}}(t + \Delta t)$ via a $K \times D_r$ output coupling matrix $\mathbf{W}_{out}$ to produce an output $\mathbf{u}_{out}(t + \Delta t)$,

$$\mathbf{u}_{out}(t + \Delta t) = \mathbf{W}_{out} \tilde{\mathbf{r}}(t + \Delta t), \quad (2.4)$$

and we endeavor to choose (“train”) the matrix elements of $\mathbf{W}_{out}$ so that $\mathbf{u}_{out}(t + \Delta t)$ is close to $\mathbf{u}(t + \Delta t)$. In general, this will require that $D_r \gg K$. To accomplish this, we try to minimize the L^2 difference between $\mathbf{u}(t + \Delta t)$ and $\mathbf{u}_{out}(t + \Delta t)$. To prevent overfitting, we insert a Tikhonov regularization term to penalize very large values of the matrix elements of $\mathbf{W}_{out}$ [58]; that is, we

find

$$\min_{\mathbf{W}_{out}} \left\{ \sum_{0 \leq t < t_0} \left[\|\mathbf{W}_{out} \tilde{\mathbf{r}}(t) - \mathbf{u}(t)\|^2 \right] + \beta \text{Trace}(\mathbf{W}_{out} \mathbf{W}_{out}^T) \right\}, \quad (2.5)$$

over the KD_r scalar values of the matrix $\mathbf{W}_{out}$. Here, β is a small regularization parameter and $\|\dots\|$ denotes the L^2 norm. This technique is commonly known as ridge regression. In our subsequent numerical experiments (Secs. 2.3 and 2.4), we use the “matrix solution” for the minimization problem to determine the trained $\mathbf{W}_{out}$. In particular, we proceed as follows. We first form a matrix $\tilde{\mathbf{R}}$ where the j^{th} column is the j^{th} transformed reservoir state $\tilde{\mathbf{r}}(j\Delta t)$. We define a target matrix $\mathbf{U}$ consisting of the time series of training data such that the j^{th} column of $\mathbf{U}$ is $\mathbf{u}(j\Delta t)$. We then determine a matrix $\mathbf{W}_{out}$ that satisfies the following linear system,

$$\mathbf{W}_{out}(\tilde{\mathbf{R}}\tilde{\mathbf{R}}^T + \beta I) = \mathbf{U}\tilde{\mathbf{R}}^T. \quad (2.6)$$

This can be done by explicitly calculating the matrix inverse, $\mathbf{W}_{out} = \mathbf{U}\tilde{\mathbf{R}}^T(\tilde{\mathbf{R}}\tilde{\mathbf{R}}^T + \beta I)^{-1}$, or, more efficiently, by using a matrix division algorithm such as `mrdivide` in MATLAB [59] as we do in our numerical experiments. We additionally note that methods of solving Eq.(2.5) for $\mathbf{W}_{out}$ other than direct matrix solution are also available and may sometimes be advantageous (e.g., GMRES [60], stochastic gradient descent [61], etc.). By means of this minimization, it is hoped that $\mathbf{u}_{out}(t) \cong \mathbf{u}(t)$ is achieved. This completes the training process, following which we can switch to the closed-loop configurations (Fig. 2.1(b) and the dashed line in Fig. 2.2) and attempt to predict the subsequent evolution of $\mathbf{u}(t)$. Prediction can then proceed via Eqs.(2.1),

(2.2) and (2.6) where the prediction of the dynamical system state $\tilde{\mathbf{u}}(t) = \mathbf{u}_{out}(t)$ and the reservoir input is received from the feedback loop ($\mathbf{u}_{in}(t) = \mathbf{u}_{out}(t)$).

The closed-loop configuration system can be regarded as a surrogate dynamical system that mimics the original unknown dynamical system. As such, if the original unknown dynamical system is chaotic, the closed-loop predictor system will also be chaotic. Due to the exponential growth of small errors implied by chaos, we cannot expect prediction to be good for more than several Lyapunov times (the Lyapunov time is the typical e-folding time for error growth in a chaotic system). Thus we will regard our predictions to be successful when they are good for a few Lyapunov times.

Now consider that we have made a prediction for $\mathbf{u}(t)$, and, at some later time, we wish to perform another prediction of the same spatiotemporally evolving system with unknown dynamics. It is not necessary to retrain our predictor; we can, instead, re-use the previously obtained $\mathbf{W}_{out}$ [21]. To do so, we re-initialize the reservoir state to zero, switch the reservoir computer into its open-loop configuration, and allow it to evolve given input states of the unknown dynamical system measured at times $t_p - T_S \leq t \leq t_p$ (i.e., $\mathbf{u}_{in}(t) = \mathbf{u}(t)$). T_S is some synchronization time that is sufficiently longer than the characteristic memory of the reservoir computer but, importantly, is much shorter than the necessary training time needed to determine $\mathbf{W}_{out}$. t_p is the time at which we want to begin our prediction. After this synchronization period, the reservoir computer is switched to its standard closed-loop prediction configuration and is used to make predictions at later times.

If the original system is very high dimensional (i.e., the dimension K of the measure vector $\mathbf{u}(t)$ is very large), then $D_r \gg K$ must be exceedingly large. This can make the training to determine $\mathbf{W}_{out}$ infeasible. For example, if we solve Eq.(2.5) by the direct matrix method, Eq.(2.6)

shows that we must solve a $D_r \times D_r$ linear system of equations. For our computational resources, we find that this becomes impossible as D_r approaches 2×10^4 . Due to this and other similar considerations, we deem the method discussed in this section to be untenable for the prediction of large, spatiotemporally chaotic systems of the type we are interested in.

2.2.3 Hybrid Reservoir Computing

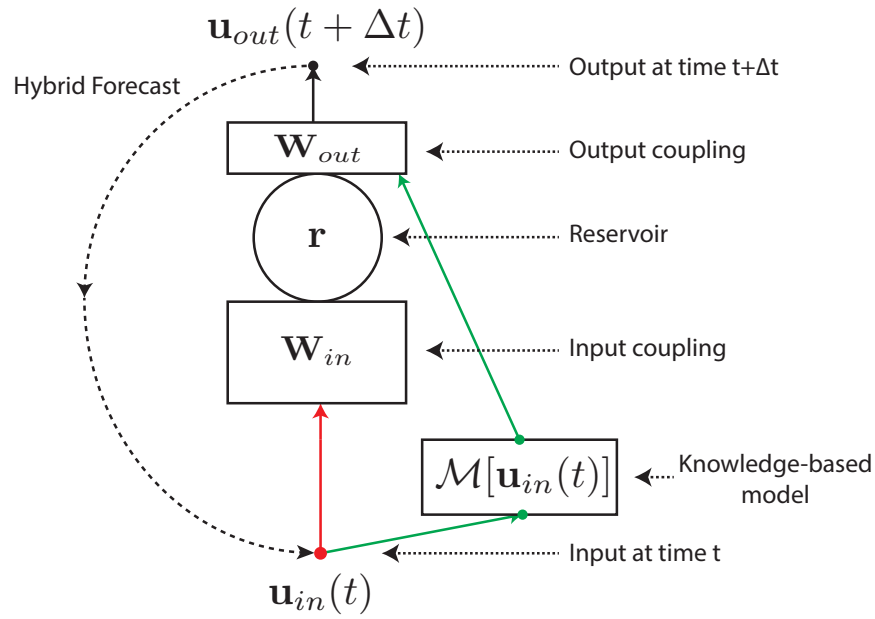


Figure 2.3: Diagram of the hybrid reservoir computer setup. In the “open-loop” training phase (analogous to Fig. 2.1(a)), the dashed line representing coupling from the output back to the input is absent. In the “closed-loop” prediction phase (analogous to Fig. 2.1(b)), the coupling from the output back to the input (dashed line) is activated.

In this section, we briefly review a hybrid scheme proposed in Ref. [22] for combining reservoir computing with an imperfect knowledge-based model of the dynamical system of interest. We again assume access to time series data of measurements of the state of the dynamical system. We further assume that an imperfect knowledge-based model of the system producing the measurements is available and that this imperfect model is capable of forecasting the state

of the dynamical system with some degree of accuracy, which we wish to improve upon. In the hybrid setup of Ref. [22] (Fig. 2.3) described below, it has been shown that the machine learning method and the knowledge-based model augment each other and, in conjunction, can provide a significantly better forecast than either the knowledge-based model or the pure machine learning model acting alone.

As in Sec. 2.2, we assume that the data used for training is given by K measurements of the state of the dynamical system at equally spaced increments in time, Δt , forming a vector time series $\mathbf{u}(t)$. The imperfect knowledge-based model $\mathcal{M}$ is an operator that maps the state $\mathbf{u}(t)$ to a forecast of the state at time $(t + \Delta t)$.

We advance the reservoir state in time using the same activation function as described in Sec. 2.2.2,

$$\mathbf{r}(t + \Delta t) = \tanh[\mathbf{A}\mathbf{r}(t) + \mathbf{W}_{in}\mathbf{u}_{in}(t)]. \quad (2.7)$$

Once again, during the training phase, $\mathbf{u}_{in}(t) = \mathbf{u}(t) + s\boldsymbol{\eta}(t)$. The training process is similar to the one employed in Sec. 2.2 for the basic reservoir computer but with the addition of the knowledge-based prediction (as illustrated in Fig. 2.3). Using ridge regression, we find a linear mapping $\mathbf{W}_{out}$ from $\tilde{\mathbf{r}}(t)$ and $\mathcal{M}[\mathbf{u}(t) + s\boldsymbol{\eta}(t)]$ to produce an approximate prediction of $\mathbf{u}(t + \Delta t)$,

$$\mathbf{W}_{out} \begin{bmatrix} \tilde{\mathbf{r}}(t + \Delta t) \\ \mathcal{M}[\mathbf{u}_{in}(t)] \end{bmatrix} \simeq \mathbf{u}(t + \Delta t). \quad (2.8)$$

Here, $\mathbf{u}_{in}(t)$ is the same as that input to the reservoir, $\mathbf{u}_{in}(t) = \mathbf{u}(t) + s\boldsymbol{\eta}(t)$. We again include the

small $s\boldsymbol{\eta}(t)$ vector in the knowledge-based model input during training to improve the stability of the method. Additionally, recall that $\tilde{\mathbf{r}}$ is related to $\mathbf{r}$ by Eq.(2.2). In the prediction phase, we run the hybrid system in a closed loop feedback configuration (Fig. 2.3 with the dashed line feedback connection present) using Eqs.(2.7), (2.2), and the following equation,

$$\mathbf{u}_{out}(t + \Delta t) = \mathbf{W}_{out} \begin{bmatrix} \tilde{\mathbf{r}}(t + \Delta t) \\ \mathcal{M}[\mathbf{u}_{out}(t)] \end{bmatrix}. \quad (2.9)$$

During the prediction phase, the hybrid forecast $\tilde{\mathbf{u}}_{\mathcal{H}}(t) = \mathbf{u}_{out}(t)$ and the hybrid input is received from the feedback loop ($\mathbf{u}_{in}(t) = \mathbf{u}_{out}(t)$). Note that, in this scheme, the output is a linear combination of the reservoir state and the knowledge-based model output that optimizes the agreement of the combined system output with the training data. Thus, we can regard the result as being an optimum combination of the reservoir and knowledge-based components. Hence, we expect that if one component is superior for some aspect of the prediction, then it will be weighted more highly for that aspect of the prediction. This suggests that predictions by this method may be greatly improved over those available from either the knowledge-based component or the reservoir component acting alone (e.g., see Fig. 7 of Ref. [22]).

In addition to the hybrid configuration shown in Fig. 2.3, we have also tested a modified configuration in which there is an additional input to the reservoir component from the output of the knowledge-based model $\mathcal{M}$. We have empirically found that this modification sometimes yields a small positive improvement in prediction; however, for simplicity, we henceforth only consider the configuration in the figure.

2.2.4 Parallelization

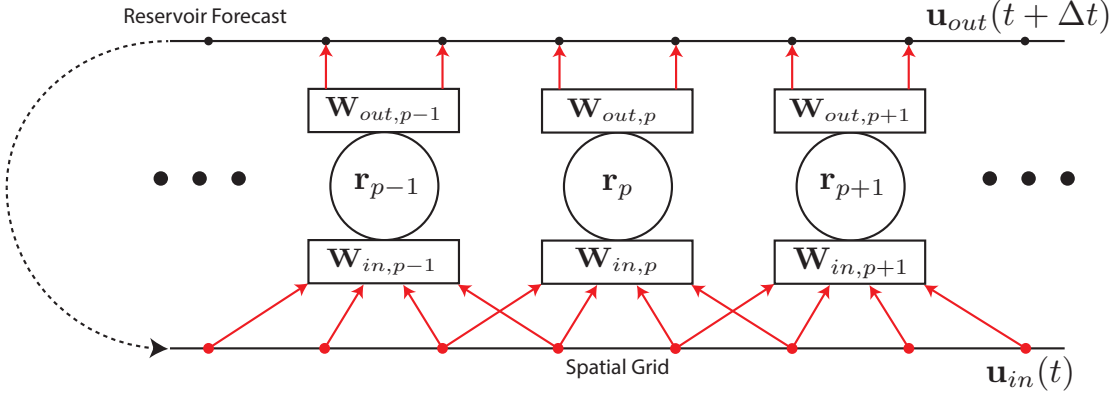


Figure 2.4: Diagram of parallel reservoir computer setup.

To obtain a good prediction of a chaotic dynamical system state using reservoir computing, the reservoir dimensionality must be much greater than that of the dynamical system (i.e., $D_r \gg K$) so that there are enough free parameters available in $\mathbf{W}_{out}$ for fitting the reservoir output state to the measured dynamical system state at time $(t + \Delta t)$. This can cause the computational cost of determining an optimum output matrix to become unfeasibly high for large dynamical systems, e.g. because implementation of this step by the method of Eq.(2.6) involves solving a $D_r \times D_r$ linear system. As a point of reference, we note that the dimension of the state vector of a current typical operational global weather forecasting model is on the order of 10^9 . A method to make consideration of such problems feasible for machine learning approaches was proposed in Ref. [21]. The idea is to exploit the short range of causal interactions over a small time period in many spatiotemporally chaotic dynamical systems. This was shown to allow the use of multiple relatively small reservoir computers that each make predictions of a part of the full dynamical system state as in a local region, illustrated in Fig. 2.4 and explained below. This method has the advantage that, in the training phase, all of the relatively small output matrices $\mathbf{W}_{out,p}$ of each

reservoir computer can be trained independently in parallel, thus greatly reducing the difficulty of training.

For illustrative purposes, consider a spatiotemporal dynamical system in one spatial dimension with periodic boundary conditions. Let the dynamical system state be represented by a K -dimensional vector time series $\mathbf{u}(t) = [u_1(t), u_2(t), \dots, u_K(t)]^T$ where each scalar component $u_j(t)$ represents the time series at a single spatial grid point. We divide the system state into P equally sized, contiguous regions containing Q system variables, where $PQ = K$. We denote the system variables in these regions as $\mathbf{u}_p(t) = [u_{Q(p-1)+1}(t), \dots, u_{Qp}(t)]^T$ where $1 \leq p \leq P$. Each local region in space is predicted by a reservoir R_p , each of which has internal reservoir states $\mathbf{r}_p(t)$ and adjacency matrix $\mathbf{A}_p$ generated via the process described in Sec. 2.2.2. Each reservoir is coupled to its input, $\mathbf{u}_{in,p}(t)$, via a matrix of input weights, $\mathbf{W}_{in,p}$. This input corresponds to a local region of the system that contains the region to be predicted by that reservoir as well as a size ℓ overlap region on either side, $\mathbf{u}_{in,p}(t) = [u_{in,Q(p-1)+1-\ell}(t), u_{in,Q(p-1)+2-\ell}(t), \dots, u_{in,Qp+\ell}(t)]^T$. We denote the dynamical system state in this input region by the size $(2\ell + Q)$ dimensional vector $\mathbf{v}_p(t) = [u_{Q(p-1)+1-\ell}(t), u_{Q(p-1)+2-\ell}(t), \dots, u_{Qp+\ell}(t)]^T$. This overlap accounts for the short range causal interactions across the boundaries of the local regions. The assumption here is that, over the incremental prediction time Δt , state information does not propagate fast enough for nodes outside the input regions of reservoir p to influence the time Δt change in the dynamical system states predicted by reservoir p .

Each reservoir state is advanced using the following equation,

$$\mathbf{r}_p(t + \Delta t) = \tanh[\mathbf{A}_p \mathbf{r}_p(t) + \mathbf{W}_{in,p} \mathbf{u}_{in,p}(t)]. \quad (2.10)$$

During the training phase (Fig. 2.4 with the dashed output-to-input connection absent), $\mathbf{u}_{in,p}(t) = \mathbf{v}_p(t) + s\boldsymbol{\eta}(t)_p$. Here, the $(2\ell + Q)$ dimensional vector $\boldsymbol{\eta}(t)_p$ is the p^{th} local region of a global vector of normally distributed random variables, $\boldsymbol{\eta}(t)$, chosen independently at each time t ,

$$\boldsymbol{\eta}(t)_p = \begin{bmatrix} \eta_{Q(p-1)+1-\ell} \\ \eta_{Q(p-1)+2-\ell} \\ \vdots \\ \eta_{Qp+\ell} \end{bmatrix}. \quad (2.11)$$

After a suitably long transient nullification period, we determine the output matrices $\mathbf{W}_{out,p}$ for each reservoir that solve the least squared optimization problem using ridge regression,

$$\min_{\mathbf{W}_{out,p}} \left\{ \sum_{0 \leq t < t_0} \left[\|\mathbf{W}_{out,p} \tilde{\mathbf{r}}_p(t + \Delta t) - \mathbf{u}_p(t + \Delta t)\|^2 \right] + \beta \text{Trace}(\mathbf{W}_{out,p} \mathbf{W}_{out,p}^T) \right\}. \quad (2.12)$$

Note that, for each p , the matrix $\mathbf{W}_{out,p}$ can be relatively small as the number of outputs is the number of state variables in region p (not the entire global state). Furthermore, the determinations of the relatively small $\mathbf{W}_{out,p}$ matrices are independent for each region p , and thus can be computed in parallel. The “direct matrix method” solution for determining each of the $\mathbf{W}_{out,p}$ matrices proceeds as follows. First, we rewrite Eq.(2.12) as

$$\min_{\mathbf{W}_{out,p}} \|\mathbf{U}_p - \mathbf{W}_{out,p} \tilde{\mathbf{R}}_p\|^2 + \beta \text{Trace}(\mathbf{W}_{out,p} \mathbf{W}_{out,p}^T), \quad (2.13)$$

where, in Eq.(2.13), $\mathbf{U}_p$ and $\tilde{\mathbf{R}}_p$ are analogous to $\mathbf{U}$ and $\tilde{\mathbf{R}}$ in the single reservoir prediction (see Eq.(2.6)). $\mathbf{U}_p$ is the target matrix such that the j^{th} column is $\mathbf{u}_p(j\Delta t)$, while $\tilde{\mathbf{R}}_p$ is obtained from

$\mathbf{R}_p$ analogous to the single reservoir case as described in Eq.(2.2). Each $\mathbf{W}_{out,p}$ is calculated by solving the following equation,

$$\mathbf{W}_{out,p}(\tilde{\mathbf{R}}_p \tilde{\mathbf{R}}_p^T + \beta I) = \mathbf{U}_p \tilde{\mathbf{R}}_p^T. \quad (2.14)$$

Note that, as previously claimed, the minimization problem for each p , Eq.(2.13), is completely independent, and can be solved for the different p in parallel. As in Sec. 2.2.2, in the prediction phase and, after a period of synchronization, we produce a full state prediction $\tilde{\mathbf{u}}(t)$ by running the system in a closed loop feedback configuration (i.e., Fig. 2.4 with the dashed output-to-input feedback connection present). This is done by concatenating the local predictions from each reservoir $\tilde{\mathbf{u}}_p(t) = \mathbf{u}_{out,p}(t)$ (where $\mathbf{u}_{out,p}(t)$ is the output state from each reservoir). Reservoir p then receives inputs from its own outputs in addition to the left and right overlap zone inputs from the ℓ left grid points, and the ℓ right grid points. The entire system thus evolves as follows,

$$\mathbf{u}_{out,p}(t) = \mathbf{W}_{out,p}[\tilde{\mathbf{r}}_p(t)], \quad (2.15)$$

$$\tilde{\mathbf{u}}(t) = \begin{bmatrix} \tilde{\mathbf{u}}_1(t) \\ \tilde{\mathbf{u}}_2(t) \\ \vdots \\ \tilde{\mathbf{u}}_P(t) \end{bmatrix} = \begin{bmatrix} \mathbf{u}_{out,1}(t) \\ \mathbf{u}_{out,2}(t) \\ \vdots \\ \mathbf{u}_{out,P}(t) \end{bmatrix}, \quad \mathbf{u}_{in,p}(t) = \tilde{\mathbf{v}}_p(t) = \begin{bmatrix} \tilde{u}_{Q(p-1)+1-\ell}(t) \\ \tilde{u}_{Q(p-1)+2-\ell}(t) \\ \vdots \\ \tilde{u}_{Qp+\ell}(t) \end{bmatrix}, \quad (2.16)$$

$$\mathbf{r}_p(t + \Delta t) = \tanh[\mathbf{A}_p \mathbf{r}_p(t) + \mathbf{W}_{in,p} \mathbf{u}_{in,p}(t)]. \quad (2.17)$$

$\tilde{\mathbf{r}}_p(t)$ is obtained from $\mathbf{r}_p(t)$ using Eq.(2.2).

Finally, we compare the parallel reservoir method with Convolutional Neural Networks

(CNNs), a commonly used form of artificial neural network architecture that is constructed to learn spatial features [62]. The main differences between the parallel reservoir method and CNN-based methods are in the type of models they produce and the method for training each of them. CNNs capture spatial features via a global model of the local dynamics that parameterizes many different local fields. Our chosen approach, on the other hand, determines many local models for each spatial region which, in the case of a spatially inhomogeneous system such as terrestrial weather, we believe will be more accurate than a global model. In addition, we expect our chosen method is easier to train due to the simplicity of obtaining the least-squares optimal solution for each relatively small local region.

2.2.5 Combined Hybrid/Parallel Prediction (CHyPP)

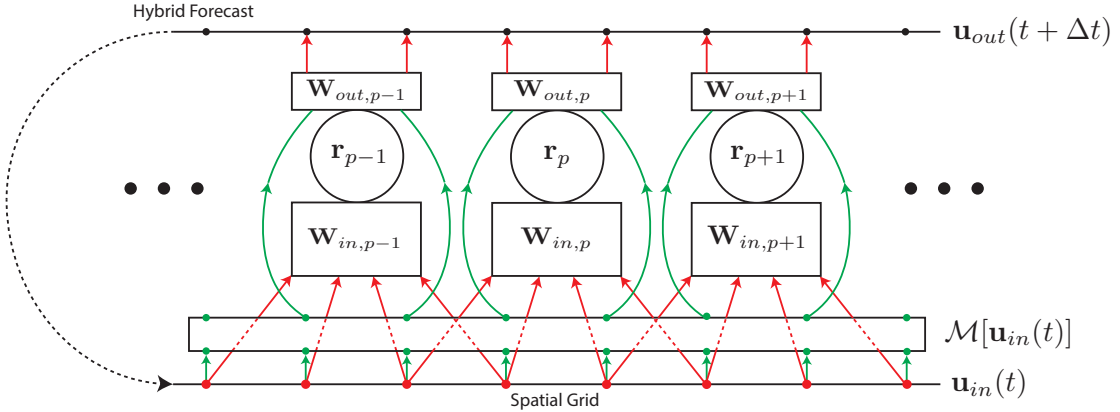


Figure 2.5: Diagram of the Combined Hybrid/Parallel Prediction (CHyPP) architecture using reservoir computing.

In our previous work [22], we constructed a hybrid prediction method using a single reservoir. In this section, we consider a combination of the parallel approach of Sec. 2.2.4 (which enables computationally efficient scaling to high-dimensional spatiotemporal chaos) and the hybrid approach of Sec. 2.2.3 (which allows us to utilize partial prior knowledge of the dynamical

system) where we assume that the knowledge-based system provides *global* predictions over the entire spatial domain. A diagram of this method can be found in Fig. 2.5. While the approach is easily generalized to 2- and 3-dimensional spatial processes, in order to most simply demonstrate our proposed methodology, we again consider a one-dimensional, spatiotemporally chaotic dynamical system with periodic boundary conditions, with a state represented by a K -dimensional vector time series $\mathbf{u}(t) = [u_1(t), u_2(t), \dots, u_K(t)]^T$. Our approximate knowledge-based prediction operator $\mathcal{M}$ gives a global prediction of the full system state for a time Δt : $\mathcal{M}[\mathbf{u}(t)] = \hat{\mathbf{u}}(t + \Delta t)$. As in our parallel reservoir computer prediction described in Sec. 2.2.4, we partition the system state into P equally sized, continuous regions containing Q variables, where $PQ = K$ and each such region is predicted by a reservoir R_p , $p = 1, 2, \dots, P$.

Each reservoir R_p input is coupled to a local region of the system states as in Sec. 2.2.4, and the reservoir state $\mathbf{r}_p(t)$ is advanced using the following equation,

$$\mathbf{r}_p(t + \Delta t) = \tanh[\mathbf{A}_p \mathbf{r}_p(t) + \mathbf{W}_{in,p} \mathbf{u}_{in,p}(t)]. \quad (2.18)$$

During the initial training phase, $\mathbf{u}_{in,p}(t) = \mathbf{v}_p(t) + s\boldsymbol{\eta}(t)_p$. In Eq.(2.18), $\mathbf{W}_{in,p}$ is the input coupling matrix for the local system states, analogous to that described in Sec. 2.2.2. As in Sec. 2.2.4, $\mathbf{v}_p(t)$ is the state measurements at grid points within the local region to be predicted along with ℓ grid points to either side and $\boldsymbol{\eta}(t)_p$ is the p^{th} local region of the global vector of normally distributed random numbers $\boldsymbol{\eta}(t)$. Each reservoir is trained independently in parallel using a set of training data consisting of an equally spaced time series of measured states of the large scale dynamics beginning at $t = 0$ after some initial transient nullification period. Again, we solve the least squares optimization problem with ridge regression to determine an output

mapping for each reservoir (analogous to Eq.(2.13)),

$$\min_{\mathbf{W}_{out,p}} \left\{ \left\| \mathbf{U}_p - \mathbf{W}_{out,p} \begin{bmatrix} \tilde{\mathbf{R}}_p \\ \hat{\mathbf{U}}_p \end{bmatrix} \right\|^2 + \beta \text{Trace}(\mathbf{W}_{out,p} \mathbf{W}_{out,p}^T) \right\}. \quad (2.19)$$

In Eq.(2.19), $\hat{\mathbf{U}}_p$ is a matrix whose j^{th} column is $\hat{\mathbf{u}}_p(j\Delta t)$, where $\hat{\mathbf{u}}_p(j\Delta t)$ is the knowledge-based prediction of the p^{th} local region of the system,

$$\hat{\mathbf{u}}_p(t + \Delta t) = \begin{bmatrix} \mathcal{M}[\mathbf{u}(t) + s\boldsymbol{\eta}(t)]_{Q(p-1)+1} \\ \mathcal{M}[\mathbf{u}(t) + s\boldsymbol{\eta}(t)]_{Q(p-1)+2} \\ \vdots \\ \mathcal{M}[\mathbf{u}(t) + s\boldsymbol{\eta}(t)]_{Qp} \end{bmatrix}. \quad (2.20)$$

The solution to Eq.(2.19) by the direct matrix method is

$$\mathbf{W}_{out,p} \begin{bmatrix} \tilde{\mathbf{R}}_p \\ \hat{\mathbf{U}}_p \end{bmatrix} [\tilde{\mathbf{R}}_p^T, \hat{\mathbf{U}}_p^T] + \beta I = \mathbf{U}_p [\tilde{\mathbf{R}}_p^T, \hat{\mathbf{U}}_p^T]. \quad (2.21)$$

In the prediction phase, we run the system in a closed-loop configuration according to Eqs.(2.22-2.25) to predict each local region of the system given each reservoir state and knowledge-based model prediction, obtaining $\tilde{\mathbf{u}}_p(j\Delta t)$. The local predictions are appropriately concatenated to form a full state prediction, which is then used as input for the next prediction step, as follows,

$$\tilde{\mathbf{u}}_p(t) = \mathbf{u}_{out,p}(t) = \mathbf{W}_{out,p} \begin{bmatrix} \tilde{\mathbf{r}}_p(t) \\ \hat{\mathbf{u}}_p(t) \end{bmatrix}, \quad (2.22)$$

$$\tilde{\mathbf{u}}(t) = \begin{bmatrix} \tilde{\mathbf{u}}_1(t) \\ \tilde{\mathbf{u}}_2(t) \\ \vdots \\ \tilde{\mathbf{u}}_P(t) \end{bmatrix}, \quad \mathbf{u}_{in,p}(t) = \tilde{\mathbf{v}}_p(t) = \begin{bmatrix} \tilde{u}_{Q(p-1)+1-\ell}(t) \\ \tilde{u}_{Q(p-1)+2-\ell}(t) \\ \vdots \\ \tilde{u}_{Qp+\ell}(t) \end{bmatrix} \quad (2.23)$$

$$\hat{\mathbf{u}}_p(t + \Delta t) = \begin{bmatrix} \mathcal{M}[\mathbf{u}_{in}(t)]_{Q(p-1)+1} \\ \mathcal{M}[\mathbf{u}_{in}(t)]_{Q(p-1)+2} \\ \vdots \\ \mathcal{M}[\mathbf{u}_{in}(t)]_{Qp} \end{bmatrix}, \quad (2.24)$$

$$\mathbf{r}_p(t + \Delta t) = \tanh[\mathbf{A}_p \mathbf{r}_p(t) + \mathbf{W}_{in,p} \mathbf{u}_{in,p}(t)]. \quad (2.25)$$

In the above equations, we once again calculate $\tilde{\mathbf{r}}_p(t)$ from $\mathbf{r}_p(t)$ using Eq.(2.2).

2.3 Test Results on the Kuramoto-Sivashinsky Equation

We test the effectiveness of our proposed CHyPP method (Sec. 2.2.5) by forecasting the state evolution of the Kuramoto-Sivashinsky equation with periodic boundary conditions [63,64],

$$\frac{\partial y}{\partial t} = -y \frac{\partial y}{\partial x} - \frac{\partial^2 y}{\partial x^2} - \frac{\partial^4 y}{\partial x^4}, \quad (2.26)$$

where $y = y(x, t)$ and $y(x + L, t) = y(x, t)$. This spatially one-dimensional model generally produces spatiotemporally chaotic dynamics for periodicity length $L \gtrsim 50$. For the purpose of comparing various methods, we regard Eq.(2.26) as generating the state measurements of a putative system that we are interested in, while the imperfect prediction model we use is a modified version of this same equation, where an error term ϵ is introduced in the coefficient of the second derivative term,

$$\frac{\partial y}{\partial t} = -y \frac{\partial y}{\partial x} - (1 + \epsilon) \frac{\partial^2 y}{\partial x^2} - \frac{\partial^4 y}{\partial x^4}, \quad (2.27)$$

We have also investigated the case where the error is introduced by multiplying the $y \partial y / \partial x$ term in Eq.(2.26) by $(1 + \epsilon)$. For the latter case, the results of our method are qualitatively similar to those for Eq.(2.27). We form our simulated measured time series $\mathbf{u}(t)$ by taking the i^{th} element of $\mathbf{u}(t)$ to be $y(i\Delta x, t)$, where $\Delta x = L/K$ is the grid spacing used for our numerical solutions of Eq.(2.26). We numerically solve the Kuramoto-Sivashinky equation on the same discretized spatial grid using fourth-order Runge-Kutta exponential time differencing [65]. As a metric for how long a prediction is valid, we calculate the normalized root-mean-square error (NRMSE) between the true and predicted system states. We define the length of valid prediction, or “valid time”, to be the time at which the NRMSE exceeds 0.2. Since the NRMSE saturates at $\sqrt{2}$, we consider this to be the point when error in the prediction reaches about 15% of its saturation value. In Sec. 2.3.1, we demonstrate that our CHyPP methodology can scale to predict very large systems. In Sec. 2.3.2, we show that the CHyPP method requires significantly less training data than the parallel scheme of Sec. 2.2.4 (using reservoirs without a knowledge-based component). We then discuss, in Sec. 2.3.3, the sensitivity of the CHyPP and parallel machine

learning (Sec. 2.2.4) methods to the local overlap length ℓ .

For all of our numerical experiments in this chapter, in addition to our solution of the imperfect model, we use a digital computer to implement the machine learning. However, if, in the future, CHyPP is applied to very large systems (e.g., weather forecasting) requiring a much larger number of parallel machine learning units, then we envision that it may prove useful to perform the parallel machine learning using a special purpose physically implemented reservoir computing array, e.g., based on FPGA’s or photonic devices [53–55]. Such implementations, called “AI hardware accelerators”, show great promise with respect to low cost, speed, and compactness.

2.3.1 Prediction Scalability

$\langle d \rangle$ (average in-degree)	3		
ρ (spectral radius)	0.6	ℓ (local overlap length)	6
σ (input coupling strength)	0.1	Δt (Prediction time step)	0.25
β (regularization)	10^{-6}	T_S (synchronization time)	25
s (reservoir(s)-only tests)	0.001	s (CHyPP tests)	0

Table 2.1: Hyperparameters used in the results in Sec. 2.3.

We first test the ability of our CHyPP method to scale to large system sizes. We consider the Kuramoto-Sivashinsky equation where we fix the number of system variables (grid points) each reservoir is trained to predict to $Q = 8$, as well as the reservoir spatial density to $P/L = 16/100$ (P is the number of reservoirs), while varying the periodicity length L . For all tests in this section, we use the hyperparameters in Table 2.1 unless otherwise specified. We additionally fix the error in the incorrect model to $\epsilon = 0.1$. Figure 2.6 shows the resulting NRMSE between the true state and our hybrid parallel prediction averaged over 100 prediction periods versus time. For each

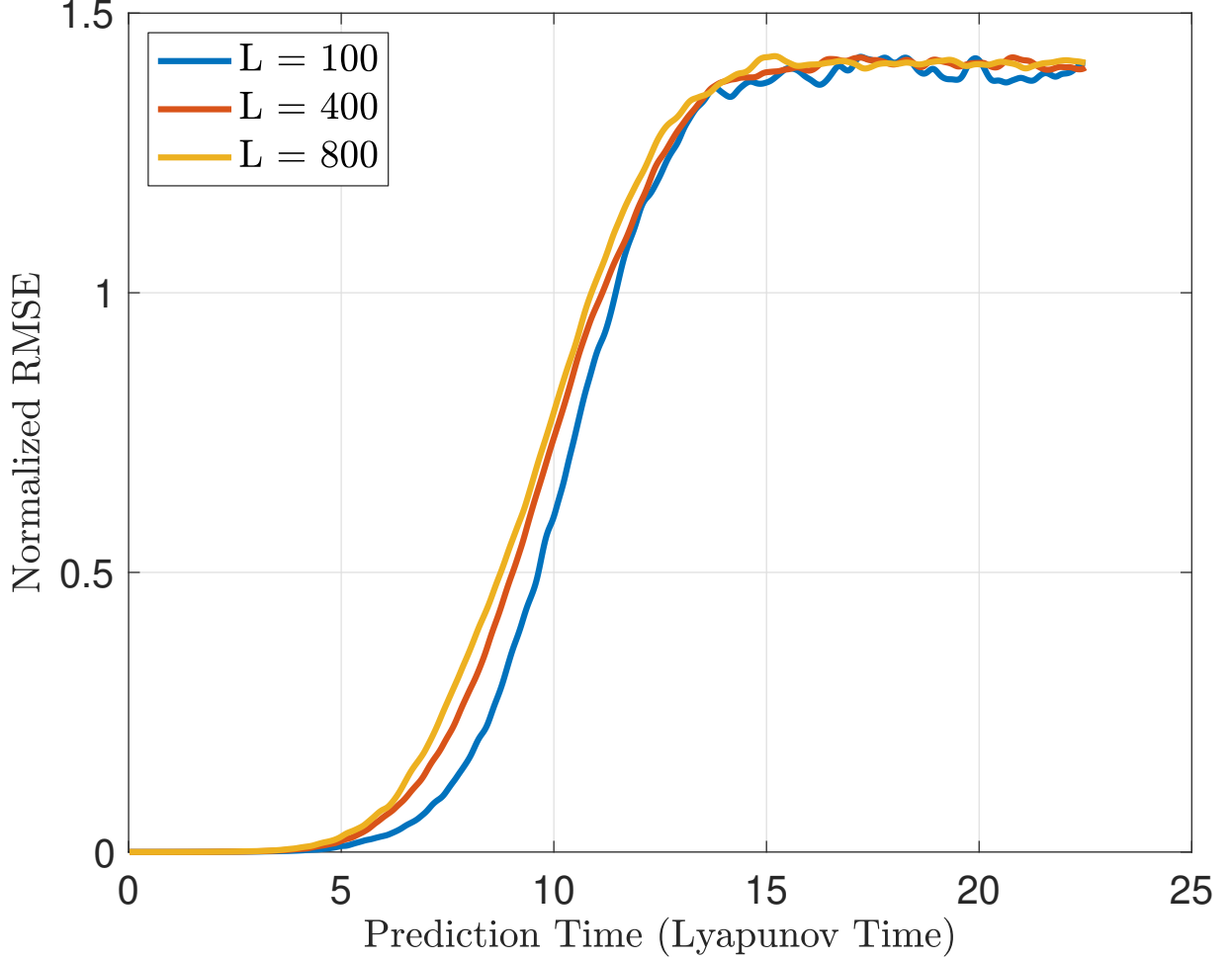


Figure 2.6: Average normalized root-mean-square error (NRMSE) in the CHyPP prediction where we have fixed the number of system variables predicted in each region to $Q = 8$ and the reservoir spatial density at $P/L = 16/100$. The total number of spatial grid points K in each prediction are $K = 128$ for $L = 100$, $K = 512$ for $L = 400$, and $K = 1024$ for $L = 800$. Predictions are made using an imperfect model with second derivative error $\epsilon = 0.1$. The results shown are the average of 100 predictions where each prediction is made using the same set of reservoirs and training data set but where the CHyPP method is synchronized to different initial conditions for $\mathbf{u}(t)$. we see that the NRMSE curves are relatively invariant as L increases, indicating that our method can be scaled to large systems.

value of L plotted in Fig. 2.6, the density of the parallel reservoir computers is kept constant at $P/L = 0.16$. Additionally, the time plotted horizontally is in units of the Lyapunov time (the average chaos-induced e-folding time of small errors in the predicted state orbit). The NRMSE is relatively unchanged as the value of L is increased, indicating that the CHyPP prediction, like

the parallel reservoir-only prediction in [21], can be scaled to very large systems by the addition of more reservoirs.

Finally, we note that our test system, the Kuramoto-Sivashinky equation, is homogeneous in space, while the real systems we are interested in are generally spatially inhomogeneous. For example, weather forecasting accounts for geographic features (continents, mountains, etc.), as well as for latitudinal variation of solar input, among other spatially inhomogeneous factors. In order to ensure that our numerical tests with a homogeneous model are also relevant for a typical inhomogeneous situation, we have not made any use of the homogeneity of Eq.(2.26): the adjacency matrices $\mathbf{A}_p$ corresponding to each reservoir p are independently randomly generated, and each $\mathbf{W}_{out,p}$ is determined separately (rather than taking all $\mathbf{A}_p$ and $\mathbf{W}_{out,p}$ to be the same, as would be possible for a homogeneous system).

2.3.2 CHyPP Promotes the Possibility of Good Performance Using a Relatively Small Duration of Training Data

The prediction results shown in Figs. 2.7 and 2.8 use the parameters contained in Table 2.1. In addition, the number of reservoirs and the length of training data are specified in the figure captions. Our true Kuramoto-Sivashinky equation dynamics and our imperfect model use a periodicity length of $L = 100$ and are realized over $K = 128$ spatial grid points. The imperfect model has an error of $\epsilon = 0.1$ and a small valid prediction time of only 0.48 Lyapunov times. Each plotted valid time is averaged over 100 predictions that are generated using the same set of reservoirs with the same training data sequence but that are synchronized to different initial conditions. Figure 2.7 displays a set of example predictions of one of these test time series. Fig-

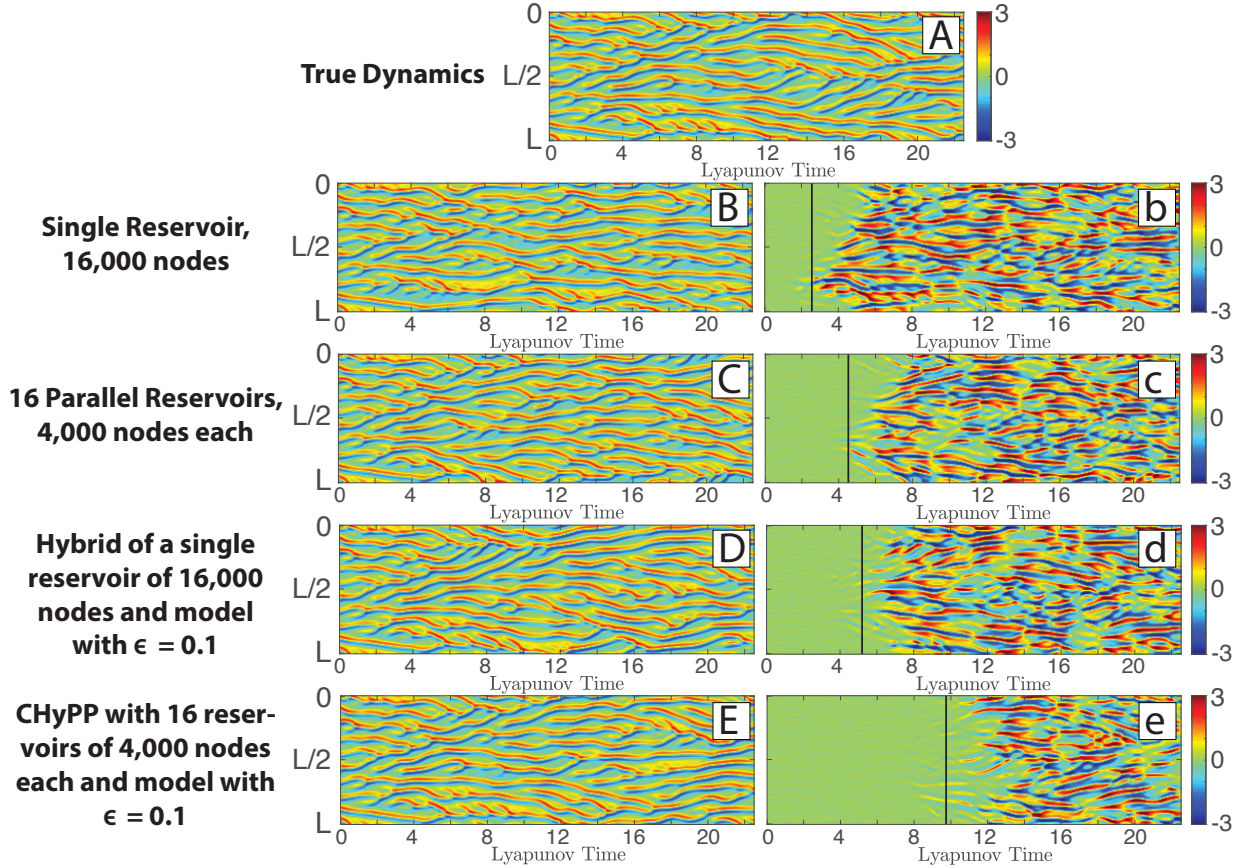


Figure 2.7: (A) The true dynamics of the Kuramoto-Sivashinsky equation obtained by numerical evolution of Eq.(2.26) normalized to have mean 0 and variance 1. We plot the spatial grid point along the vertical axis and the Lyapunov time on the horizontal axis. The Lyapunov time is defined by $\Lambda_{max}t$ where Λ_{max} is the largest Lyapunov exponent computed from Eq.(2.26). Predictions of this evolution by four different methods are shown in panels (B), (C), (D), and (E). The prediction in (B) is made by a single reservoir computer with 16,000 nodes, the prediction in (C) is made by 16 reservoir computers in parallel each with 4,000 nodes, the prediction in (D) is made by a hybrid with a single reservoir computer with 16,000 nodes(D), and the prediction in (E) is made by CHyPP with 16 reservoir computers in parallel each with 4,000 nodes and an imperfect model with second derivative $\epsilon = 0.1$. The plots on the right (b), (c), (d), and (e) display the difference between the true dynamics (A) and the prediction using each of the corresponding techniques. The vertical black line marks the valid prediction time. Each technique was trained using a 3,375 Lyapunov time sequence of training data.

ures 2.7 and 2.8 both demonstrate that CHyPP yields significantly longer valid predictions than the parallel reservoir-only method, which (for our choice of ϵ and reservoir size) produces longer valid predictions than the imperfect model alone. Both the parallel hybrid and parallel reservoir-only predictions outperform the imperfect model-only approach. From Fig. 2.8, we also observe that the CHyPP saturates, or reaches a valid prediction time that increases only negligibly with the addition of more training data, at a much shorter length of training data than the parallel reservoir-only. As a result of this, the length of training data before which we would not benefit from increasing the size of the reservoir is also much shorter in the CHyPP prediction. For example, consider the plots in (b) and (f) in Fig. 2.8, where each prediction uses 4 reservoirs. If we have a 500 Lyapunov time length of training data, we would not benefit from increasing the number of nodes per reservoir above 2000 in the reservoirs-only prediction, but could obtain a significant performance improvement if we did so using CHyPP. CHyPP exhibits its most impressive performance for very short lengths of training data. With only 22.5 Lyapunov times worth of training data, the parallel reservoir-only method is able to predict for only 0.44 Lyapunov times, whereas CHyPP with 16 reservoirs of 2000 nodes each predicts for 3.35 Lyapunov times on average, matching the saturated single reservoir-only prediction that is trained using 150 times more training data.

2.3.3 Prediction Quality dependence on local overlap length

In this section, we investigate the dependence of CHyPP performance on the local overlap length ℓ . Figure 2.9 shows that when the local overlap length ℓ is zero or close to zero, the parallel reservoir-only prediction valid time (solid red line) is very poor, predicting for only around 0.5

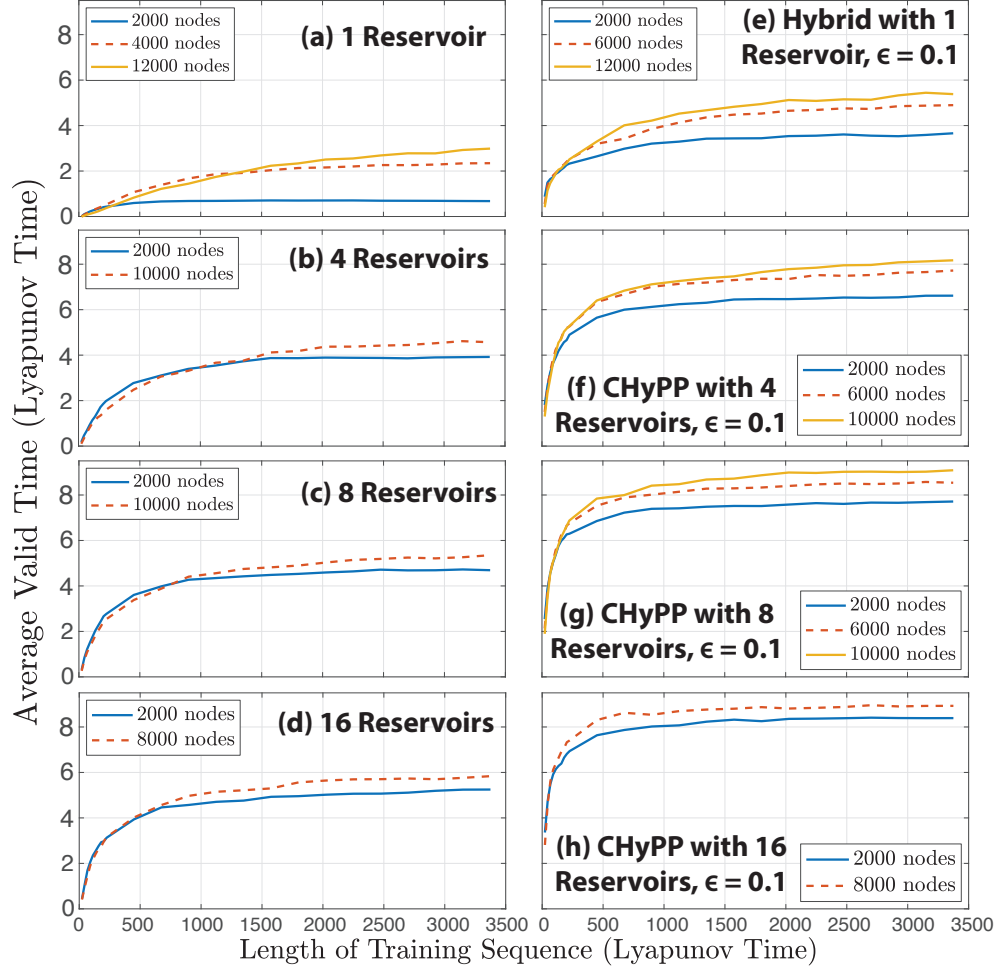


Figure 2.8: We plot the average valid time as a function of the duration of the training data, each in Lyapunov time, for the parallel reservoir(s)-only method using (a) 1 reservoir, (b) 4 reservoirs, (c) 8 reservoirs, and (d) 16 reservoirs. We plot the same quantity obtained from the parallel hybrid method using (e) 1 reservoir, (f) 4 reservoirs, (g) 8 reservoirs, and (h) 16 reservoirs. In each case, the imperfect knowledge-based predictor in the hybrid prediction had an error in the second derivative term of $\epsilon = 0.1$, resulting in an average valid time for the model-only prediction of 0.48 of a Lyapunov time. A knowledge-based predictor with no error begun using the most accurate initial condition from the parallel hybrid was able to predict for an average of 11 Lyapunov times. The legend in each plot indicates the number of nodes used in each reservoir.

Lyapunov times.

CHyPP, however, is still able to obtain good predictions with very little local overlap length.

For CHyPP, this indicates that it is able to utilize the inaccurate model prediction to infer the propagation of dynamical influences between the adjacent prediction regions. We note that neither

CHyPP nor the 16 parallel reservoirs-only prediction are able to improve upon a corresponding single, large reservoir hybrid prediction or corresponding single large reservoir-only prediction when the local overlap length is near 0. Regarding this comparison, however, it is important to recognize that in systems larger than the one we have used to test our method here, use of a single reservoir prediction is not possible because the reservoir size necessary for prediction becomes infeasible, as discussed in Sec. 2.2. This result is nevertheless very important for large-scale implementations of the proposed method. In realistic systems with 3 spatial dimensions, a small increase in the local overlap length ℓ can lead to increasingly large memory requirements and increasingly large amounts of information that must be communicated between reservoirs at each prediction iteration. Being able to obtain a good prediction with less local overlap length when the propagating dynamics is adequately explained by the imperfect model is thus another advantage of CHyPP.

2.4 Test Results on a Multiscale System Prediction and the Problem of Subgrid-Scale Closure

A common difficulty in numerical modeling arises because many physical processes involve dynamics on multiple scales. As a result, fundamentally crucial subgrid scale dynamics is often only crudely captured in an ad-hoc manner. The formulation of subgrid scale models by analytical techniques has been extensively studied (e.g., see Ref. [66]) and is sometimes referred to as “subgrid-scale closure”. In this section, we show that our CHyPP methodology provides a very effective data-based (as opposed to analysis-based) approach to subgrid-scale closure.

In particular, we test our CHyPP prediction method on a dynamical system with multiple

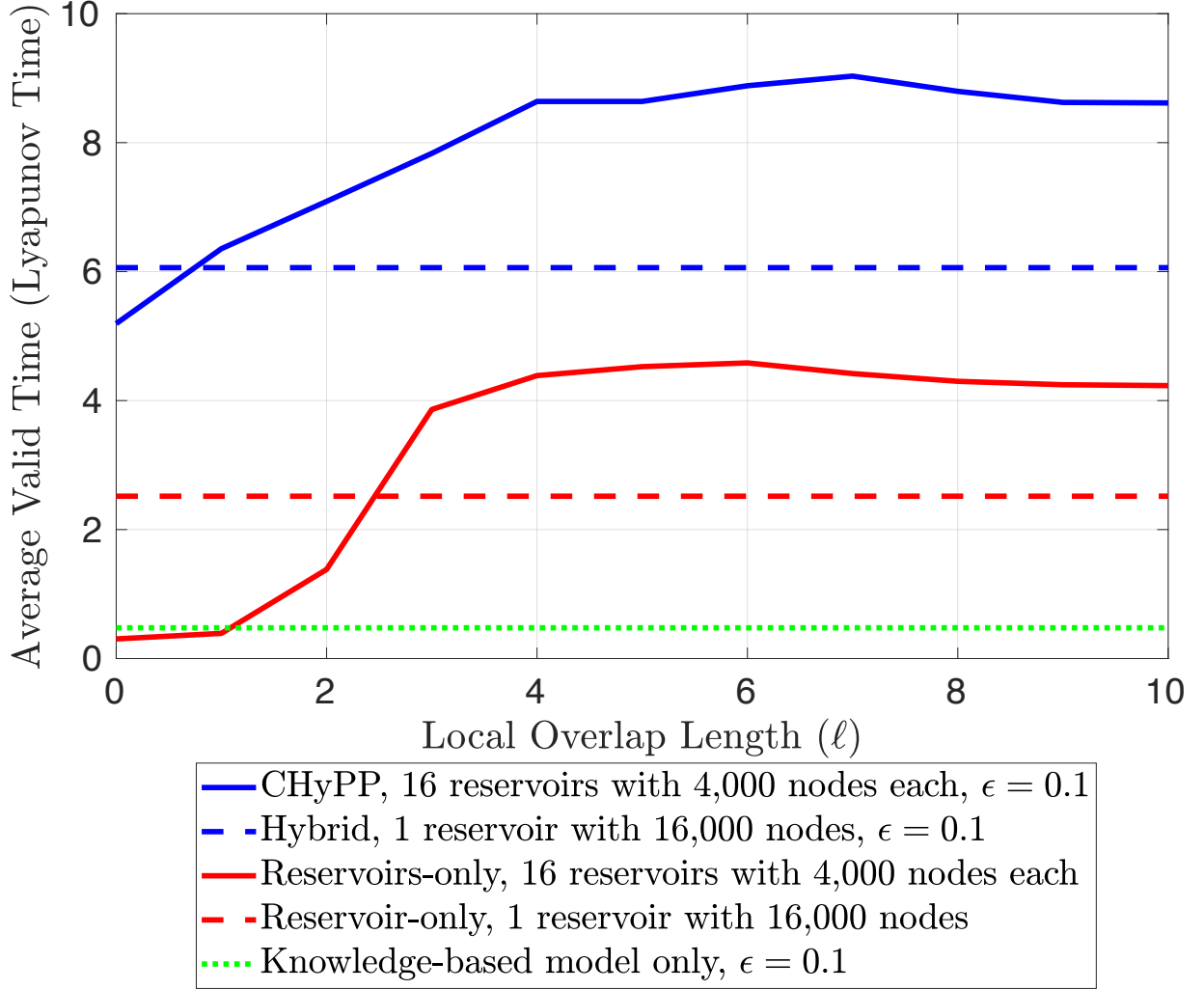


Figure 2.9: We plot the valid time for the parallel and non-parallel hybrid and reservoir(s)-only prediction as a function of the local overlap length ℓ .

spatial and temporal scales. The particular system we choose to predict is the multiscale “toy” atmospheric model formulated by E.N. Lorenz in his 2005 paper [67]. We refer to this model as Lorenz Model III. This model is a smoothed extension of Lorenz’s original “toy” atmospheric model described in his 1996 paper [68] (hereafter referred to as Lorenz Model I), with the addition of small scale dynamical activity. Lorenz Model III describes the evolution of a single atmospheric variable, Z , on a one-dimensional grid with N grid points and periodic boundary conditions, representing a single latitude. The value of Z at each grid point, Z_n , evolves accord-

ing to the following equation,

$$dZ_n/dt = [X, X]_{K,n} + b^2[Y, Y]_{1,n} + c[Y, X]_{1,n} - X_n - bY_n + F. \quad (2.28)$$

In Eq.(2.28), X is a smoothed version of Z , and Y is the difference between Z and X ,

$$X_n = \sum_{i=-I}^I (\alpha - \beta|i|) Z_{n+i}, \quad Y_n = Z_n - X_n, \quad (2.29)$$

$$\alpha = (3I^2 + 3) / (2I^3 + 4I), \quad \beta = (2I^2 + 1) / (I^4 + 2I^2). \quad (2.30)$$

Here, I denotes the smoothing distance. Thus, X describes the large-spatial-scale, long-time-scale wave component of Z , while Y describes the small-spatial-scale, short-time-scale wave component. $[V, W]_{K,n}$ indicates a coupling between the variables V and W [i.e., interaction within scales ($V = W = X$ or Y) or between scales for ($V = X, W = Y$)]. For K even, this coupling takes the form,

$$[V, W]_{K,n} = \sum_{j=-J}^J ' \sum_{i=-J}^J ' (-V_{n-2K-i} W_{n-K-j} + V_{n-K+j-i} W_{n+K+j}) / K^2. \quad (2.31)$$

Here, $J = K/2$ and $\sum'$ denotes a modified summation where the first and last summands are divided by 2. For K odd, $J = (K - 1)/2$ and each $\sum'$ becomes a standard summation $\sum$. In Eq.(2.28), the parameters K , b , c , and F describe the coupling distance of the system's large scale dynamics, the increase in small scale oscillation rapidity and decrease in amplitude (relative to the large scale dynamics), the degree of interaction between the large and small scale dynamics, and the overall forcing in the system, respectively. We note that when $b = c = 0$ and $K = 1$, this

model reduces to the Lorenz Model I.

For the purposes of testing our CHyPP method, we also introduce another model formulated by Lorenz, which we will refer to as Lorenz Model II [67]. This model is equivalent to Lorenz Model III with no distinct small scale wave component or smoothing present in the equation,

$$dZ_n/dt = [Z, Z]_{K,n} - Z_n + F. \quad (2.32)$$

Important for our tests, Lorenz notes that, for constant F , the dominant wavenumber in Lorenz Model II depends only on the ratio N/K .

We test our CHyPP method by using it to predict the dynamics generated from Lorenz Model III using Lorenz Model II as our imperfect knowledge-based predictor. As we discussed in Sec. 2.1, knowledge-based models used to predict physical multiscale systems (i.e., the weather) often use simplified representations of subgrid scale dynamics. To replicate this type of imperfection in the knowledge-based model in CHyPP, we realize the Lorenz Model II dynamics over an equal or fewer number of grid points $N_{\text{Model II}}$ while using the same N/K ratio as used to generate the true Lorenz Model III dynamics (i.e., $N_{\text{Model III}}/K_{\text{Model III}} = N_{\text{Model II}}/K_{\text{Model II}}$). Our imperfect model thus incorrectly represents the effect of small scale dynamics that, when $N_{\text{Model III}} > N_{\text{Model II}}$, are also subgrid. In our tests, each of these models is solved numerically using a fourth-order Runge-Kutta scheme.

For the following results in this section, we use the parameters in Table 2.2. In addition, the value of s used in each of the reservoir computing-based prediction methods has been selected for each method to maximize the valid prediction time. Figure 2.10 shows a solution of Lorenz

$\langle d \rangle$ (average in-degree)	3	ρ (spectral radius)	0.6
ℓ (local overlap length)	$N/12$	σ (input coupling strength)	1
Δt (Prediction time step)	0.005	β (regularization)	10^{-4}
T_S (synchronization time)	0.5		

Table 2.2: Hyperparameters used in the results in Sec. 2.4.

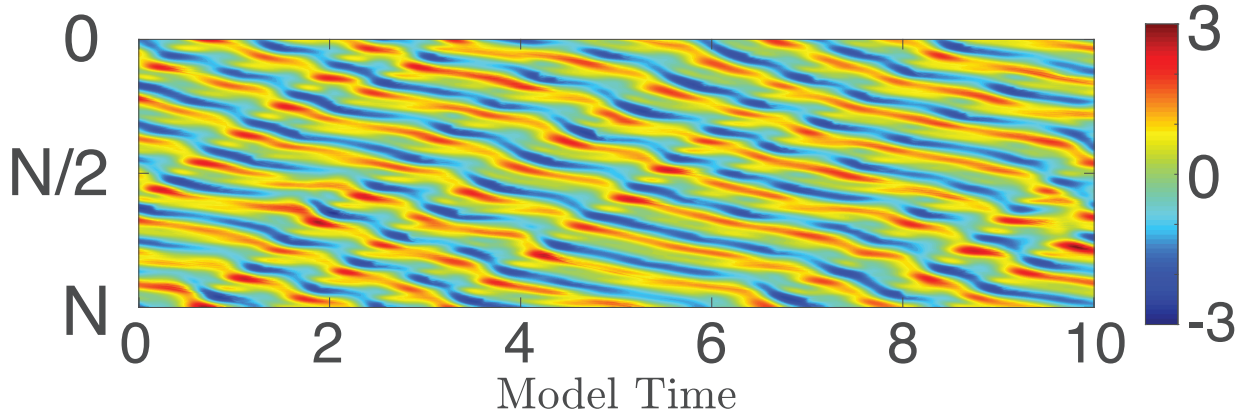


Figure 2.10: Plot of the true dynamics from Lorenz Model III using the parameters $N = 960$, $K = 32$, $I = 12$, $F = 15$, $b = 10$, and $c = 2.5$. The dynamics have been normalized to set the mean value to 0 and variance to 1. We plot the value of Z color-coded (color bar on the right), the spatial grid point along the vertical axis, and the model time on the horizontal axis.

Model III where the value of Z is color-coded. The spatial variable is plotted vertically, time is plotted horizontally, and the model parameters are given in the caption. As seen in Fig. 2.10, there is a wave-like motion with a dominant wavenumber of ≈ 7 (7 oscillations along the vertical periodicity length). This corresponds to Lorenz’s design of the model to mimic atmospheric dynamics, which has a predominant wavenumber for Rossby waves of this order as one goes around a mid-latitude circle. Figures 2.11-2.14 show predictions of the true Lorenz Model III dynamics (in Fig. 2.10) for grid resolutions of varying coarseness. In particular, while the truth (Fig. 2.10) is obtained from Model III with $N = 960$ grid points, the number of Model II grid points is $N = 960, 480, 240$, and 120 for Figs. 2.11, 2.12, 2.13, and 2.14 respectively. Also note that the measurements are taken from the Model III result only at the Model II grid points.

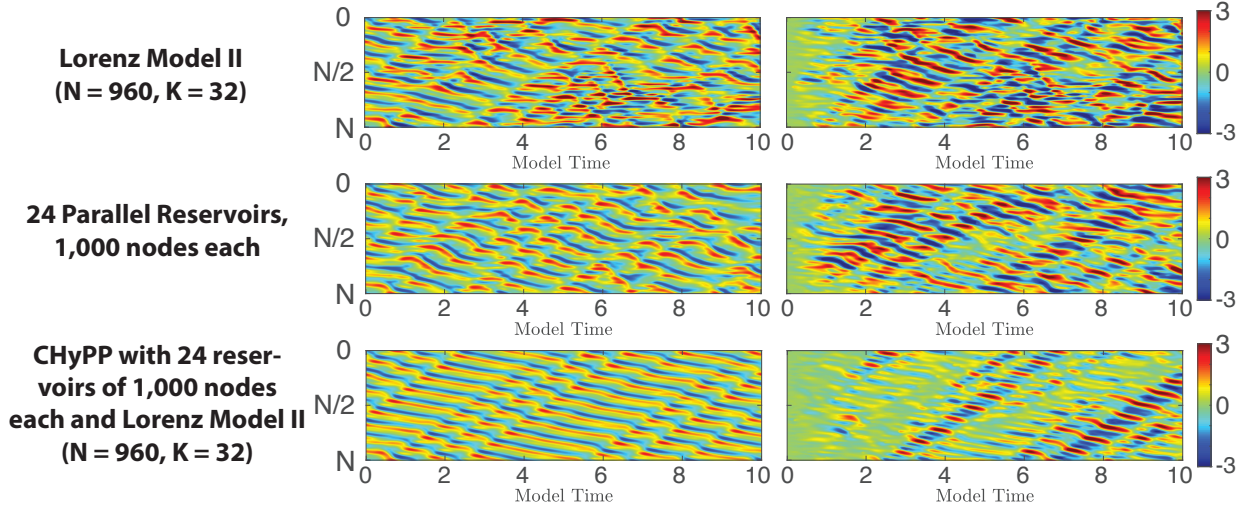


Figure 2.11: We plot the predictions of the normalized true dynamics from Lorenz Model III (displayed in Fig. 2.10) where we have measured every spatial grid point of Model III during training. We plot the spatial grid point along the vertical axis and the model time on the horizontal axis. The plots on the left show the predictions made using each of the specified methods, while the plots on the right show the difference between the true dynamics and the prediction (i.e., the prediction error). The parallel reservoir-only prediction used its optimized value of $s = 0.1$, while CHyPP used its optimized value of $s = 0.085$.

For both the parallel reservoir-only predictions and CHyPP, we fix the ratio of the number of nodes per reservoir to the number of grid points each reservoir predicts to be $D_r/Q = 25$. We choose the Lorenz Model III time units rather than the Lyapunov time as our time scale for these plots since the largest Lyapunov exponent of Model III corresponds to the small-spatial-scale, short-time-scale dynamics and thus will not be relevant to our intended goal of forecasting the large-spatial-scale, long-time-scale dynamics.

We find that CHyPP significantly outperforms each of its component methods, and that this is true for all of the grid resolutions we tested. We note that, unlike in the Model II and parallel reservoir-only predictions, the prediction error in CHyPP seems to appear locally in a small region and manifests as slanted streaks in the error plots in Figs. 2.11-2.14. We have verified that the slant of these streaks corresponds to the group velocity of the dominant wave motion (wavenum-

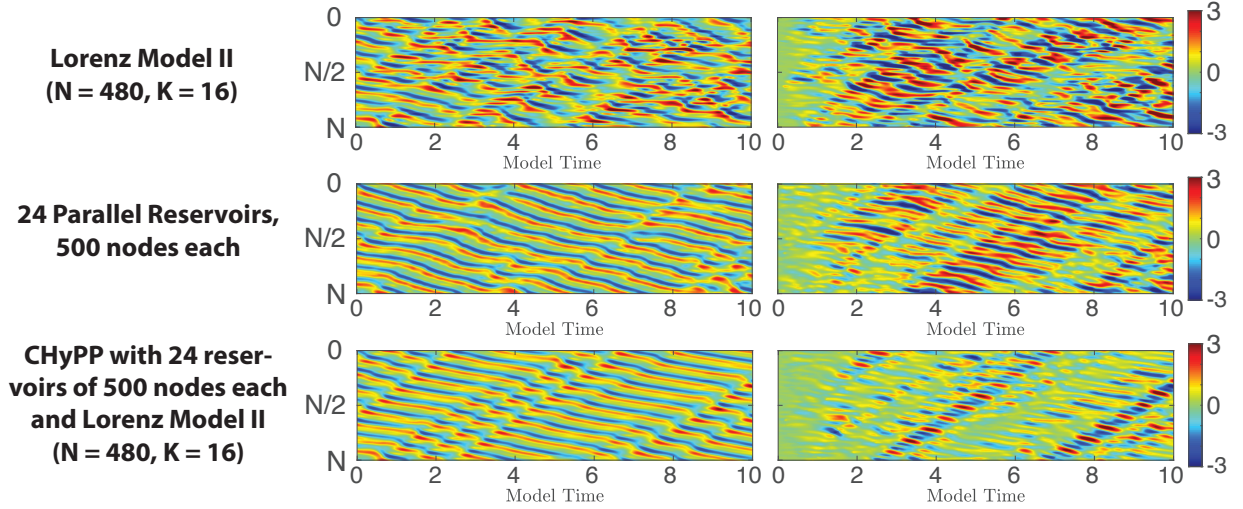


Figure 2.12: We plot the predictions of the normalized true dynamics from Lorenz Model III (displayed in Fig. 2.10), where, corresponding to the Model II grid, we have measured *only every second* spatial grid point of Model III during training. We plot the spatial grid point along the vertical axis and the model time on the horizontal axis. The plots on the left show the predictions made using each of the specified methods, while the plots on the right show the difference between the true dynamics and the prediction (i.e., the prediction error). The parallel reservoir-only prediction used its optimized value of $s = 0.25$, while CHyPP used its optimized value of $s = 0.08$.

ber ≈ 7). For all predictions made, we again calculate a valid time of prediction; however, since in this case early error growth using CHyPP is local in space and affects only a small part of the prediction domain, we choose to use a higher valid time error threshold of 0.85 (approximately 60% of the error saturation value) so that the valid time metric reflects when error is present in the entire prediction domain. Figure 2.15 shows the valid time averaged over 100 predictions from different initial conditions versus training time for different grid resolutions and reservoir sizes. The dashed lines are parallel reservoir-only predictions, while the solid lines are CHyPP predictions. In this figure, the reservoir sizes are scaled in proportion to the number of Model II grid points used. We find that, while the quality of the scaled parallel reservoir-only prediction degrades significantly as the grid resolution decreases, the quality of the scaled CHyPP prediction degrades much more slightly from an average valid time of 3.23 at full resolution to 3.05 at $1/8$

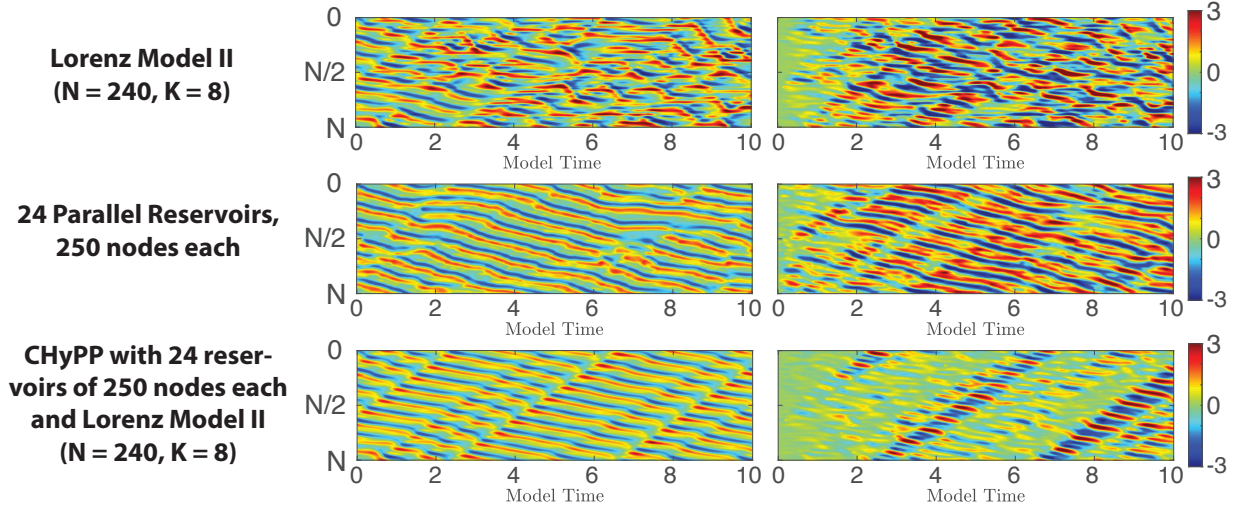


Figure 2.13: We plot the predictions of the normalized true dynamics from Lorenz Model III (displayed in Fig. 2.10), where, corresponding to the Model II grid, we have measured *only every fourth* spatial grid point of Model III during training. We plot the spatial grid point along the vertical axis and the model time on the horizontal axis. The plots on the left show the predictions made using each of the specified methods, while the plots on the right show the difference between the true dynamics and the prediction (i.e., the prediction error). The parallel reservoir-only prediction used its optimized value of $s = 0.25$, while CHyPP used its optimized value of $s = 0.017$.

resolution. The Model II valid time is essentially constant at roughly 0.95, corresponding to the fact that all of the grid spacings tested are well below the characteristic Model II spatial scale. We see from Fig. 2.15 that the parallel reservoir-only prediction and CHyPP prediction appear to reach valid time saturation at about the same training data length for each grid resolution.

2.5 Conclusion and Discussion

In this chapter, we address the general goal of utilizing machine learning to enable expanded capability in the forecasting of a large, complex, spatiotemporally chaotic systems for which an imperfect knowledge-based model exists. Some typical common sources of imperfection in such a knowledge-based model are unresolved subgrid scale processes and lack of first principles knowledge or computational ability for modeling some necessary aspect or aspects of

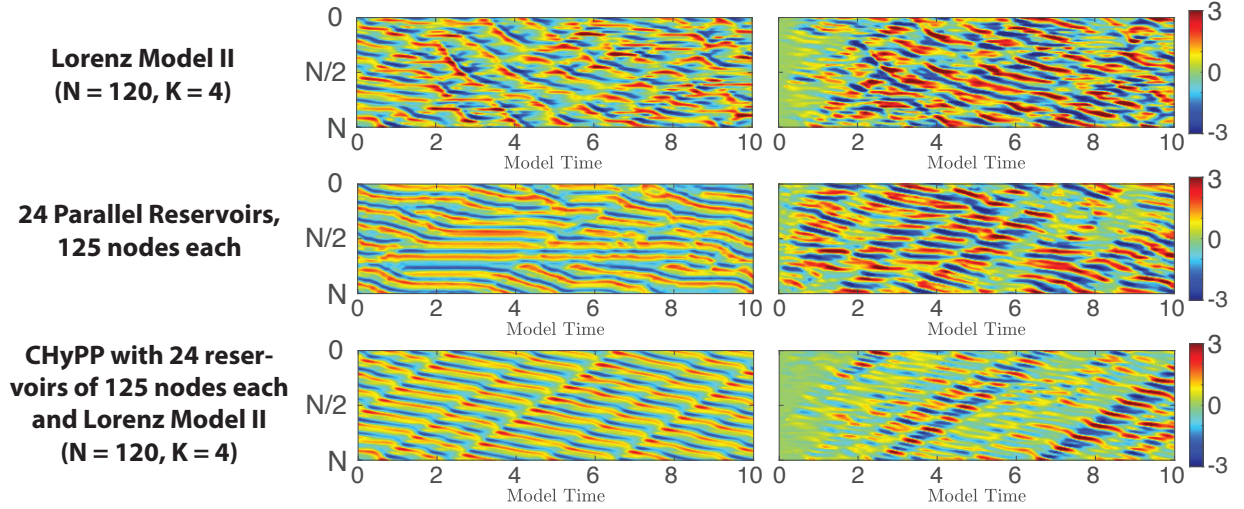


Figure 2.14: We plot the predictions of the normalized true dynamics from Lorenz Model III (displayed in Fig. 2.10), where, corresponding to the Model II grid, we have measured *only every eighth* spatial grid point of Model III during training. We plot the spatial grid point along the vertical axis and the model time on the horizontal axis. The plots on the left show the predictions made using each of the specified methods, while the plots on the right show the difference between the true dynamics and the prediction (i.e., the prediction error). The parallel reservoir-only prediction used its optimized value of $s = 0.25$, while CHyPP used its optimized value of $s = 0.0155$.

the physics. The hope is that these “imperfections” can be compensated for by use of measured time series data and machine learning. The two main foreseeable difficulties in realizing this hope are how to effectively combine the machine learning component with the knowledge-based component in such a way that they mutually enhance each other, and how to promote feasible scaling of the machine learning requirements with respect to its computational cost and necessary amount of training data. Note that addressing the first of these issues necessarily lessens the difficulty of dealing with the second issue, since good use of any valid scientific knowledge of the system being forecast can potentially reduce the amount of learning required from the machine learning component.

To address these two issues, we propose a methodology (CHyPP) that combines two previously proposed techniques: (i) a hybrid utilization of an imperfect knowledge-based model with

a single machine learning component [22, 56], and (ii) a parallel machine learning scheme using many spatially distributed machine learning devices [21]. We note that (ii) applies to large spatial systems with the common attribute of what we have called ‘local short-time causal interactions’. Numerical tests of our proposed combination of (i) and (ii) are presented in Secs. 2.3 and 2.4 and demonstrate good quality prediction that is scalable with size (Figs. 2.6 and 2.7), reduced required length of training data (Fig. 2.8), and ability to compensate for unresolved subgrid-scale processes (Figs. 2.10-2.15).

In this chapter, our proof-of-principle problems have been one-dimensional in space with relatively simple mathematical formulations. We note, however, that the CHyPP methodology is readily applicable to higher dimensions and more complicated situations. For example, we are currently in the process of applying CHyPP to global atmospheric weather forecasting for which the atmospheric state is spatially three-dimensional and the system is strongly inhomogeneous, e.g., due to the presence of complex geographic features (including continents, mountains, and oceans), as well as the latitudinal variation of solar heating.

Finally, we note that in many situations, such as operational weather forecasting, predictions are produced cyclically once every cycle time interval. To do this, an accurate estimation of the initial condition is required at the beginning of each cycle. The process of formulating accurate initial conditions is called “data assimilation” and is vital to account for sparse and uncertain measurements. Once every cycle time interval, a set of forecasts are produced for a set of forecast times, where one of these times is equal to the cycle time interval. Then, at the beginning of the next cycle, the previous forecast for the state at the cycle time interval is combined with state measurements taken during the previous cycle and their corresponding estimated uncertainty to obtain an estimate of the current state of the system to be predicted. This state estimate is then

used as an initial condition for a forecast model making the next set of predictions. The problem of formulating data assimilation for our improved prediction model (CHyPP) awaits further study, which we are currently pursuing.

In conclusion, we have shown that data-assisted forecasting via parallel reservoir computing and an imperfect knowledge-based model can significantly improve prediction of a large spatiotemporally chaotic system over existing methods. This method is scalable to even larger systems, requires significantly less training data than previous methods to obtain high quality predictions, is able to effectively utilize a knowledge-based forecast of information propagation between local regions to improve prediction quality, and effectively provides a means of using data to compensate for unresolved subgrid scale processes (playing a role akin to traditional analysis-based closure schemes).

Acknowledgements

We thank Sarthak Chandra for his helpful discussion and comments. This work was supported by DARPA contract DARPA-PA-18-01(HR111890044).

Data Availability

The data that support the findings of this study are available from the corresponding author upon reasonable request. Our software implementation of CHyPP in MATLAB is publicly available at <https://github.com/awikner/CHyPP>.

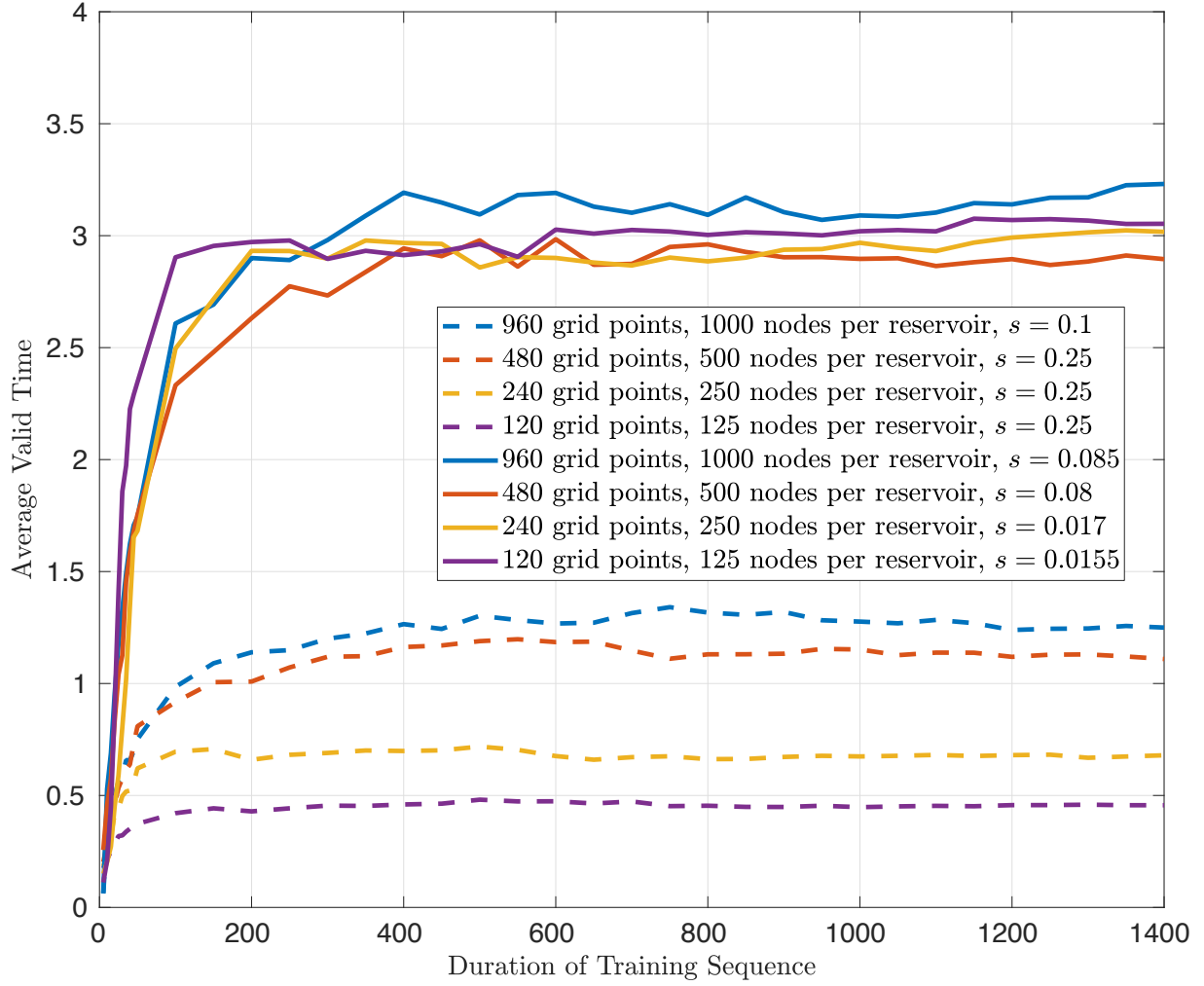


Figure 2.15: We plot the average valid time for each prediction as a function of the length of training data used. Dashed lines display the results of parallel reservoir-only predictions, whereas solid lines display results from CHyPP where the knowledge-based model is Lorenz Model II with $N = \text{Number of grid points}$, $K = N/30$, and $F = 15$. Each prediction method uses 24 reservoirs.

Chapter 3: An Iterative Technique for Hybrid Model Training with Sparse and Noisy Observations

This chapter is based on the paper, Using data assimilation to train a hybrid forecast system that combines machine-learning and knowledge-based components, by Alexander Wikner, Jaideep Pathak, Brian Hunt, Istvan Szunyogh, Michelle Girvan, and Edward Ott, Chaos 31, 053114 (2021).

3.1 Introduction

The application of machine learning (ML) to forecasting chaotic dynamical systems (e.g., [2]) has been a subject of much recent interest [18, 19, 21, 22, 31, 36, 69–74]. Promising results of tests promote the possibility of using ML for large scale applications such as weather forecasting. Particularly, Ref. [21] indicates that machine-learning techniques can be scaled effectively to forecast arbitrarily high-dimensional chaotic attractors. In cases like weather forecasting where a skillful knowledge-based model is available, hybrid approaches that use ML to correct model deficiencies are especially promising [22, 75]. By “knowledge-based”, we mean derived from fundamental principles such as the laws of physics.

So far, most data-driven forecasting approaches assume that training data is available for all the variables to be forecast. This assumption is relaxed in [70, 72–74], which describe techniques

to develop a data-driven forecast model on a specified, but potentially only partially measured, model state space. Using an approach different from those in Refs. [70,72–74], in this chapter, we develop a technique for hybrid knowledge-based/ML systems in the partially measured scenario. Both our hybrid approach and the purely data-driven approaches in [70,72,73,76] are based on the paradigm of data assimilation (DA), which has long been used in weather forecasting and other applications.

When a forecast model is available, DA (e.g., [77–79]) determines initial conditions for an ongoing sequence of forecasts, based on an ongoing sequence of measurements. DA does not require that current measurements are sufficient to determine the current model state; instead, it typically uses the forecast model to propagate information from past measurements to augment current information. This technique helps both to constrain the model state and to filter noise in the measurements.

Ideas from ML and DA have been combined in various ways in the literature. DA has been used with data-driven forecast models based on delay-coordinate embedding [80] and analog forecasting [81]. On the other hand, with appropriate training data, machine-learning techniques can be taught to perform DA, without explicitly formulating a forecast model [48]. DA has also been employed as a paradigm and a tool for the training of machine-learning systems [70,72–74, 82–84].

We give a brief introduction to DA in Sec. 3.2.1. In Sec. 3.2.2, we formulate a potentially effective scheme combining DA (Sec. 3.2.1) with machine-learning-assisted prediction (as described in Ref. [22]) to create a cyclic forecast system. In Sec. 3.2.3, we propose an iterative procedure to improve the DA cycle and potentially produce improved ML-assisted forecast models. In Sec. 3.3, we show the results of numerical experiments.

3.2 Methods

We consider the problem of forecasting an autonomous, chaotic, nonlinear dynamical system represented by the equation:

$$\frac{d\tilde{\mathbf{x}}(t)}{dt} = \mathbf{F}[\tilde{\mathbf{x}}(t)]. \quad (3.1)$$

In Eq. (3.1), $\tilde{\mathbf{x}}(t)$ is the true dynamical system state at time t and $\mathbf{F}$ is the vector field. We allow for the case where $\tilde{\mathbf{x}}$ is infinite-dimensional, as it is in a partial differential equation. We assume that we are only able to obtain partial measurements of the state $\tilde{\mathbf{x}}$ at regular time intervals. We denote the measurements made at time $k\Delta t$ by $\mathbf{y}_k$, so that

$$\mathbf{y}_k = \tilde{\mathbf{H}}\tilde{\mathbf{x}}(k\Delta t) + \eta_k. \quad (3.2)$$

In Eq. (3.2), $\tilde{\mathbf{H}}$ is often called the observation operator; here, we call it the measurement operator. For our exposition, we assume $\tilde{\mathbf{H}}$ to be linear, as will be true in the numerical experiments performed in this chapter. (Many DA methods allow a nonlinear measurement operator, and our ML method is compatible with this case.) The random variable η_k represents measurement noise, which is typically assumed to be Gaussian with mean $\mathbf{0}$ and covariance matrix $\mathbf{R}$.

In applications, we do not have full knowledge of the dynamics of the system, and, as such, Eq. (3.1) is unknown. What often is available is an imperfect model of the dynamics, with model error due to imperfect representations of some of the physical processes that govern the system. We assume that we have access to such an imperfect “knowledge-based” model denoted by $\mathbf{G}$

and that the dynamical equation,

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{G}[\mathbf{x}(t)], \quad (3.3)$$

can, via evolution of $\mathbf{x}$, be used to forecast future $\tilde{\mathbf{x}}$ -dependent properties of interest. In some applications, Eq. (3.1) represents a partial differential equation, and $\mathbf{x}$ approximates $\tilde{\mathbf{x}}$ on a model grid. Thus, we assume there is a projection operator (which could be the identity) from the space of $\tilde{\mathbf{x}}$ to the space of $\mathbf{x}$. We write $\mathbf{x}_k^t$ to represent the projection of the true state $\tilde{\mathbf{x}}(k\Delta t)$ onto the space of $\mathbf{x}$, so that $\mathbf{x}(k\Delta t)$ is meant to approximate $\mathbf{x}_k^t$. We rewrite Eq. (3.2) as

$$\mathbf{y}_k \approx \mathbf{H}\mathbf{x}_k^t + \eta_k, \quad (3.4)$$

where $\mathbf{H}$ approximates $\tilde{\mathbf{H}}$ (for example, via interpolation from a model grid to an observation location).

3.2.1 Data Assimilation

In DA, the initial state $\mathbf{x}(0)$ for the forecast model, Eq. (3.3), is determined from the current measurement $\mathbf{y}_0$ and past measurements $\{\mathbf{y}_k\}_{-T_m \leq k < 0}$, using the same model to propagate information (imperfectly) from the past to the present. Here, T_m represents the number of past times $k\Delta t$ from which measurements are available. Many DA methods can be formulated as the approximate solution of a nonlinear least-squares problem [85–87], similar to the following. Given the measurements $\{\mathbf{y}_k\}_{-T_m \leq k \leq 0}$, find a trajectory $\mathbf{x}(t)$ of Eq. (3.3) that minimizes the cost

function given by

$$J_0(\mathbf{x}(t)) = \sum_{k=-T_m}^0 \rho^k [\mathbf{y}_k - \mathbf{H}\mathbf{x}(k\Delta t)]^\top \mathbf{R}^{-1} [\mathbf{y}_k - \mathbf{H}\mathbf{x}(k\Delta t)]. \quad (3.5)$$

Usually, ρ is taken to equal 1, but here we allow also $\rho > 1$ in order to discount past measurements, in part to compensate for model error in $\mathbf{G}$; we will discuss the role of ρ further after Eq. (3.8). In the situation where the measurement errors (η_k) are Gaussian and the model $\mathbf{G}$ is a perfect representation of the dynamical system, i.e., $\mathbf{G} = \mathbf{F}$, minimizing Eq. (3.5) with $\rho = 1$ will give us the trajectory of the model $\mathbf{x}(t)$ that is the most likely trajectory of the dynamical system in the Bayesian sense (here, we assume a “flat” prior likelihood; the choice of prior distribution becomes increasingly unimportant as T increases.)

In forecast applications, the purpose of DA is to determine an estimate $\mathbf{x}(0)$ of the current true system state $\mathbf{x}_0^t$. This estimate is used as an initial condition for Eq. (3.3) to forecast $\mathbf{x}(t)$ for $t > 0$. As such, the trajectory $\mathbf{x}(t)$ for $t < 0$ that minimizes the cost function J_0 is often not computed explicitly; only its final state $\mathbf{x}(0)$ is needed as the algorithmic output. DA is typically done sequentially, using the state estimate from an earlier time to compute the estimate for the current time. To be more precise, consider the cost function J_j that generalizes Eq. (3.5) as follows:

$$J_j(\mathbf{x}(t)) = \sum_{k=-T_m}^j \rho^{k-j} [\mathbf{y}_k - \mathbf{H}\mathbf{x}(k\Delta t)]^\top \mathbf{R}^{-1} [\mathbf{y}_k - \mathbf{H}\mathbf{x}(k\Delta t)]. \quad (3.6)$$

Then,

$$J_j(\mathbf{x}(t)) = \rho^{-1} J_{j-1}(\mathbf{x}(t)) + [\mathbf{y}_j - \mathbf{H}\mathbf{x}(j\Delta t)]^\top \mathbf{R}^{-1} [\mathbf{y}_j - \mathbf{H}\mathbf{x}(j\Delta t)]. \quad (3.7)$$

Suppose for some $-T_m < j \leq 0$ that $\mathbf{x}_{j-1}^a$ represents an estimate of $\mathbf{x}_{j-1}^t$ based on the cost function J_{j-1} ; i.e., based on measurements $\mathbf{y}_k$ up to and including $k = j - 1$. The next estimate $\mathbf{x}_j^a$ is computed in two steps as follows:

1. **Forecast:** The model, Eq. (3.3), is applied from time $(j - 1)\Delta t$ to $j\Delta t$, using $\mathbf{x}_{j-1}^a$ as the initial condition, to obtain an estimate $\mathbf{x}_j^b$ to $\mathbf{x}_j^t$, called the “background” state at time $j\Delta t$.
2. **Analysis:** The background $\mathbf{x}_j^b$ is used along with the current measurement vector $\mathbf{y}_j$ to compute the “analysis” state $\mathbf{x}_j^a$ that estimates the minimizer of J_j at time $j\Delta t$.

These two steps are repeated to compute $\mathbf{x}_{j+1}^b$, $\mathbf{x}_{j+1}^a$, and so on.

Different DA methods perform the analysis step differently [77–79, 87]. In some methods, including those based on the Kalman filter, the cost functions are approximated by quadratic functions of $\mathbf{x}(j\Delta t)$, so that the right side of Eq. (3.7) becomes

$$\rho^{-1} [\mathbf{x}(j\Delta t) - \mathbf{x}_j^b]^\top (\mathbf{P}_j^b)^{-1} [\mathbf{x}(j\Delta t) - \mathbf{x}_j^b] + [\mathbf{y}_j - \mathbf{H}\mathbf{x}(j\Delta t)]^\top \mathbf{R}^{-1} [\mathbf{y}_j - \mathbf{H}\mathbf{x}(j\Delta t)]. \quad (3.8)$$

Here, $\mathbf{P}_j^b$ is a covariance matrix associated with the background state at time $j\Delta t$. Thus, the quantity to be minimized is a sum of (squared) weighted distances from $\mathbf{x}(j\Delta t)$ to the background state (which is forecast from previous observations) and to the current observations. In an ensemble Kalman filter [88–90], $\mathbf{P}_j^b$ is computed as the sample covariance of an ensemble of

forecasts. The parameter ρ effectively inflates this covariance, compensating for (in addition to model error) sampling errors due to finite ensemble size and the effect of nonlinearity on the error dynamics; this technique is called multiplicative covariance inflation [86, 87, 91]. We refer to ρ as the “covariance inflation parameter”.

In Appendix A, we describe the particular DA method we use, the “Ensemble Transform Kalman Filter” (ETKF) [92, 93], a type of ensemble Kalman filter. We emphasize that the approach we describe in Sec. 3.2.2 can be used with any DA method. The forecast and analysis steps we have described are applied iteratively to produce a time series of analysis states $\mathbf{x}_j^a$. We will use this time series in Sec. 3.2.2 to train the ML component of our method.

3.2.2 ML-DA Algorithm

In what follows, we use a reservoir computer implementation of our ML-DA algorithm based on Refs. [18, 69], which is similar to that of Ref. [22]. The main component of a reservoir computer is the reservoir network. The reservoir network is a sparse, randomly generated graph with weighted, directed edges and dynamical nodes whose states evolve in response to input signals transported along network edges. The network topology can be represented completely by a weighted adjacency matrix, denoted $\mathbf{A}$. We choose $\mathbf{A}$ to be a $D_r \times D_r$ sparse, randomly generated matrix. The network is constructed to have an average in-degree (number of incoming edges per node) denoted by $\langle d \rangle$, and the nonzero elements of $\mathbf{A}$, representing the edge weights in the network, are initially chosen independently from the uniform distribution over the interval $[0, 1]$. All the edge weights in the network are then uniformly scaled via multiplication of the adjacency matrix by a constant factor to set the largest magnitude eigenvalue of the matrix to

a desired value ω , which is called the “spectral radius” of $\mathbf{A}$. The state of the reservoir, given by the vector $\mathbf{r}(t)$, consists of the components r_n for $1 \leq n \leq D_r$ where $r_n(t)$ denotes the scalar state of the n^{th} node in the network. The reservoir is coupled to the M dimensional input through a $D_r \times M$ dimensional matrix $\mathbf{W}_{in}$. Each row of the matrix $\mathbf{W}_{in}$ has exactly one nonzero element which is independently and randomly chosen from the uniform distribution on the interval $[-\zeta, \zeta]$, where ζ is a tunable parameter. We choose the locations of these nonzero elements such that each node receives input from only one input variable and each input variable is input to an equal number of nodes.

As outlined at the beginning of Sec. 3.2, we note that the finite-dimensional representation of the true dynamical system states are given by $\mathbf{x}_j^t$. We have $T + T_s$ measurements given by $\{\mathbf{y}_j\}$, in the interval $-T - T_s \leq j \leq 0$, which are related to the true state of the dynamical system by Eq. (3.2). We refer to T and T_s as the number of training and synchronization samples, where $T \gg T_s$. We further assume that the model $\mathbf{G}$ (Eq. (3.3)) is imperfect. Using the model $\mathbf{G}$ and the DA algorithm outlined in Sec. 3.5, we obtain an analysis state $\mathbf{x}_j^a$ at each time step j , $-T - T_s \leq j \leq 0$. We are interested in forecasting the state of the dynamical system for $j > 0$. In the standard forecast approach (without ML), one would predict the future state of the dynamical system (Eq. (3.1)) using $\mathbf{x}_0^a$ as the initial condition and Eq. (3.3) as the dynamical model. We will call this forecast the baseline forecast, to which we will compare the forecast made by our ML-DA algorithm. The ML-DA algorithm is an extension, to the case where the model state is only partially observed, of the algorithm proposed in Ref. [22] for hybrid forecasting of chaotic dynamical systems using ML in conjunction with a knowledge-based model. The analysis states $\mathbf{x}_j^a$ lie in the model state space; once they are determined by the DA procedure, training proceeds as in the hybrid method of Ref. [22]. Figure 3.1 shows a schematic illustration of the ML-DA

scheme described below.

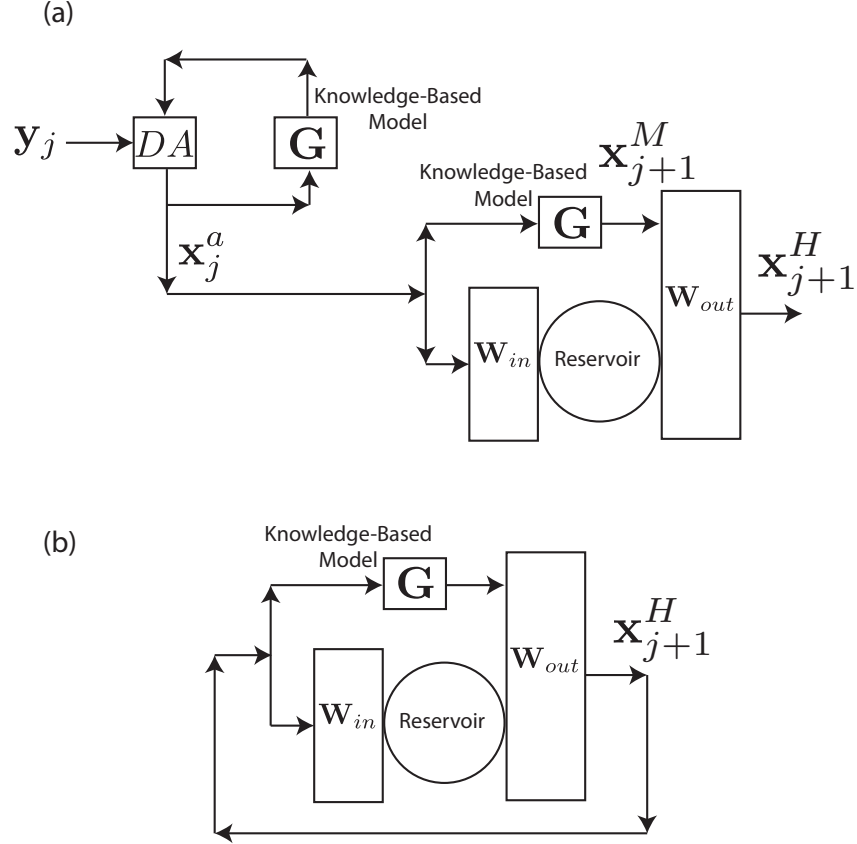


Figure 3.1: (a) Training the ML-DA prediction scheme and (b) hybrid forecasting as described in Sec. 3.2.2. The output matrix in (a), W_{out} , is unknown at the time of training. The goal of training is to determine W_{out} such that x_j^H best approximates x_j^a in the regularized least-squares sense.

3.2.2.1 Training

1. Use the model G to create a set of forecasts from the latest T analysis states x_j^a , $-T \leq j \leq 0$. We denote these time- Δt forecasts by x_j^M , $-T + 1 \leq j \leq 1$.
2. Initialize the reservoir state $\mathbf{r}_{-T-T_s}$ to a random value.

3. Evolve the reservoir computer using the following equation

$$\mathbf{r}_{j+1} = \tanh[\mathbf{A}\mathbf{r}_j + \mathbf{W}_{in}\mathbf{x}_j^a] \quad (3.9)$$

for $-T - T_s \leq j \leq 0$. Here, $\tanh$ is applied element-by-element to its input vector.

4. Find a set of output weights $\mathbf{W}_{out}$ so that

$$\mathbf{x}_j^H = \mathbf{W}_{out} \begin{bmatrix} \mathbf{r}_j \\ \mathbf{x}_j^M \end{bmatrix} \simeq \mathbf{x}_j^a \quad (3.10)$$

for $-T + 1 \leq j \leq 0$. The matrix $\mathbf{W}_{out}$ is computed using regularized least squares regression. Thus, we find the $\mathbf{W}_{out}$ that minimizes the following cost function:

$$\ell(\mathbf{W}_{out}) = \sum_{j=-T+1}^0 \left\| \mathbf{W}_{out} \begin{bmatrix} \mathbf{r}_j \\ \mathbf{x}_j^M \end{bmatrix} - \mathbf{x}_j^a \right\|^2 + \beta \|\mathbf{W}_{out}\|^2. \quad (3.11)$$

Here, β is a regularization parameter. Notice that we have not used the first T_s reservoir states to find the optimal $\mathbf{W}_{out}$. This is to ensure that the reservoir states used in the computation of $\mathbf{W}_{out}$ do not depend strongly on the initial random value of the reservoir state, $\mathbf{r}_{-T-T_s}$, and thus are “synchronized” to the measurements that drive Eq. (3.9) through the analysis states $\mathbf{x}_j^a$.

3.2.2.2 Prediction

We now describe the hybrid forecast model that uses the trained reservoir along with the knowledge-based model $\mathbf{G}$ to forecast the state $\mathbf{x}(j\Delta t)$ for $j > 0$.

The hybrid reservoir runs in a closed-loop feedback mode according to the following steps:

1. Obtain the hybrid ML prediction at time $j\Delta t$ (denoted $\mathbf{x}_j^H$) according to:

$$\mathbf{x}_j^H = \mathbf{W}_{out} \begin{bmatrix} \mathbf{r}_j \\ \mathbf{x}_j^M \end{bmatrix}. \quad (3.12)$$

The reservoir state and knowledge-based forecasts at the first step, $\mathbf{r}_1$ and $\mathbf{x}_1^M$, are computed from the final training analysis $\mathbf{x}_0^a$ according to steps 1 and 3 of the training procedure.

2. Compute the imperfect model forecast at time $(j+1)\Delta t$, $\mathbf{x}_{j+1}^M$, by evolving $\mathbf{x}_j^H$ using the imperfect model $\mathbf{G}$.
3. Evolve the reservoir according to

$$\mathbf{r}_{j+1} = \tanh [\mathbf{A}\mathbf{r}_j + \mathbf{W}_{in}\mathbf{x}_j^H] \quad (3.13)$$

4. Increment j and repeat steps 1 to 3.

If one wants to initialize a forecast at a time $J\Delta t$ for $J > 0$, one should continue to run the DA procedure for J more steps after the training time period, set $\mathbf{x}_J^H = \mathbf{x}_J^a$, then proceed as above starting with step 2 for $j = J$. The J DA cycles after time 0 can be performed using the trained

hybrid model in place of the knowledge-based model, similarly to the approach described in the next section.

We note that the ensemble-based DA method we use could be used to initialize an ensemble forecast. However, for simplicity, our results use only forecasts from the ensemble mean.

3.2.3 Iterated ML-DA Algorithm

Once we have trained a hybrid model using the process described in Sec. 3.2.2.1, we can iterate the training procedure in the following sense. An iterative approach, initialized differently, is described in Ref. [72]. As shown in Fig. 3.2, we replace the knowledge-based model used during DA with the previously trained hybrid model. At iteration $i = 1$, the hybrid is trained using the analysis states produced using only the knowledge-based model. We denote the output matrix to be determined by the training, the state of the reservoir at time $j\Delta t$, the analysis state at time $j\Delta t$, and the hybrid model predictions for time $j\Delta t$ at iteration i as $\mathbf{W}_{out,i}$, $\mathbf{r}_{j,i}$, $\mathbf{x}_{j,i}^a$, and $\mathbf{x}_{j,i}^H$, respectively.

Training proceeds as in Sec. 3.2.2.1 with the following modifications.

3.2.3.1 Training

1. For $-T - T_s \leq j < -T$, we use the knowledge-based model to perform DA, producing analysis states $\mathbf{x}_{j,i}^a$. We do not use the hybrid model during this synchronization time period because the reservoir states $\mathbf{r}_{j,i}$ might not yet be synchronized with the measurements.

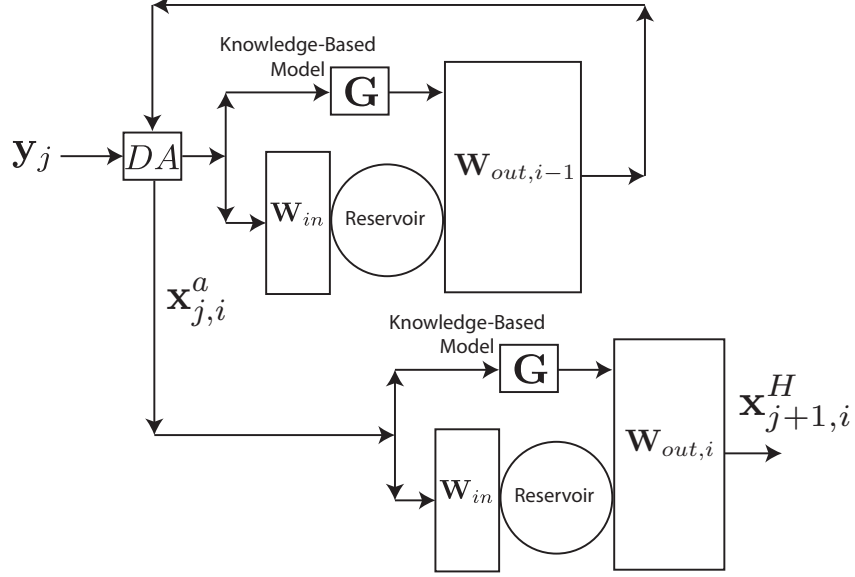


Figure 3.2: Training subsequent iterations ($i \geq 2$) of the ML-DA algorithm using the synchronized hybrid model during DA as described in Sec. 3.2.3. The index i denotes the current iteration of the algorithm, and we consider the training procedure shown in Fig. 3.1(a) to occur at $i = 1$. At iteration i , $\mathbf{W}_{out,i-1}$ is known from the previous training iteration, while we endeavor to determine $\mathbf{W}_{out,i}$ such that $\mathbf{x}_{j,i}^H$ best approximates $\mathbf{x}_{j,i}^a$ in the regularized least-squares sense.

These reservoir states are generated according to step 3 of Sec. 3.2.2.1, i.e.,

$$\mathbf{r}_{j+1,i} = \tanh[\mathbf{A}\mathbf{r}_{j,i} + \mathbf{W}_{in}\mathbf{x}_{j,i}^a]. \quad (3.14)$$

2. For $T \leq j \leq 0$, we use the hybrid model to perform DA in place of the knowledge-based model, producing analysis states $\mathbf{x}_{j,i}^a$, as depicted in Fig. 3.2. We continue to use Eq. (3.14) to evolve the reservoir states.
3. Using these new analysis states, we follow the steps contained in Sec. 3.2.2.1 to compute a new hybrid output matrix, $\mathbf{W}_{out,i+1}$.
4. Once the new hybrid output matrix has been computed, predictions can be produced by following Sec. 3.2.2.2.

In the case of an ensemble-based assimilation method (such as the ETKF described in Appendix A), we augment each ensemble member to include, in addition to the system state $\mathbf{x}_j^{a,k}$, a corresponding reservoir state $\mathbf{r}_j^k$. This reservoir state is synchronized to the corresponding ensemble system state by evolving it as in Eq. (3.14):

$$\mathbf{r}_{j+1,i}^k = \tanh[\mathbf{A}\mathbf{r}_{j,i}^k + \mathbf{W}_{in}\mathbf{x}_{j,i}^{a,k}]. \quad (3.15)$$

Then, $\mathbf{r}_{j+1,i}^k$ is used in the hybrid model to produce the k^{th} background ensemble member for the next analysis cycle. We note that the reservoir state(s) at $j = -T + 1$ needed to begin DA using the hybrid model must only be computed at the first iteration. Afterwards, these initial synchronized states may be reused at the beginning of subsequent DA cycles.

3.2.4 Experiment Design

We evaluate the forecast performance of the ML-DA algorithm in comparison with the performance of baseline for forecasts prepared without using DA. We consider two dynamical systems as our examples: the Lorenz ‘63 system [2] and the Kuramoto-Sivashinsky (KS) system [94, 95]. For the latter system, we additionally evaluate the error in the analysis states from using the iterative method described in Sec. 3.2.3.

For the purpose of evaluating forecast performance, simulated true states $\mathbf{x}_j^t$ are generated by the perfect model dynamics in the time interval $-T - T_S \leq j \leq P$, where T and T_S are described in Sec. 3.2.2. In all cases, transient dynamics are discarded so that the simulated true states lie in the model attractor. The data in the interval $1 \leq j \leq P$ is set aside to verify the baseline and ML-DA forecasts but is not otherwise used, since it is considered to be in the future.

We assume that we only have access to the measurements $\{\mathbf{y}_j\}_{-T-T_S \leq j \leq 0}$ generated from the simulated true states via Eq. (3.2) and do not know the perfect model, Eq. (3.1). We further assume that we have access to an imperfect model $\mathbf{G}$ (Eq. (3.3)). We use this imperfect model to perform DA, to train the ML-DA, and to prepare both the baseline and ML-DA forecasts for $j > 0$.

For DA, we use the ETKF, as described in Appendix A, with 15 ensemble members for the Lorenz '63 system and 30 ensemble members for the KS system. To initialize the forecasts and the ML training, we perform DA using the measurements $\mathbf{y}_j$ ($-T - T_S \leq j \leq 0$) and the model $\mathbf{G}$ to obtain the analysis states $\mathbf{x}_{-T-T_S}^a, \dots, \mathbf{x}_0^a$.

3.2.4.1 Baseline Forecasts:

Using $\mathbf{x}_0^a$ as the initial condition, we compute a forecast using the model $\mathbf{G}$ from time $j = 1$ to $j = P$. We call this forecast $\mathbf{x}_j^{f,base}$, $1 \leq j \leq P$.

3.2.4.2 ML-DA Forecasts:

We follow the ML-DA scheme detailed in Sec. 3.2.2 using a reservoir computer with the reservoir and training parameters listed in Table 3.1 for the Lorenz '63 and the Kuramoto-Sivashinsky (KS) dynamical systems. We use the measurements $\mathbf{y}_j$ ($-T - T_S \leq j \leq 0$) and the imperfect model $\mathbf{G}$ corresponding to the Lorenz and KS systems respectively to train the reservoir computer. The exact form of the measurements and of the imperfect model is described in Secs. 3.3.1 and 3.3.2. We then forecast using the ML-DA scheme from $j = 1$ to $j = P$. We call this forecast $\mathbf{x}_j^{f,ml}$, $1 \leq j \leq P$.

Parameter	Lorenz ‘63	KS
D_r	1000	2000
$\langle d \rangle$	3	3
ω	0.9	0.6
ζ	0.1	1.0
β	10^{-4}	10^{-4}
T	20,000	100,000

Table 3.1: Reservoir and training parameters.

3.2.4.3 Normalized RMS error

The normalized RMS error in the baseline forecast (e_j^{base}) and the RMS error in ML-DA forecast (e_j^{ml}) are calculated as follows:

$$e_j = \frac{\|\mathbf{x}_j^f - \mathbf{x}_j^t\|}{\sqrt{\langle \|\mathbf{x}_j^t\|^2 \rangle}}, \quad 1 \leq j \leq P, \quad (3.16)$$

where $\|\dots\|$ denotes the Euclidean norm, $\langle \dots \rangle$ denotes an average over the prediction interval ($1 \leq j \leq P$), and $e_j, \mathbf{x}_j^f$ denote either $e_j^{base}, \mathbf{x}_j^{f,base}$ or $e_j^{ml}, \mathbf{x}_j^{f,ml}$. The superscript f in $\mathbf{x}_j^f$ stands for forecast.

3.2.4.4 Forecast Valid Time

We define the forecast Valid Time (t_{VT}) as the time $j\Delta t$ at which the RMS error (e_j) first exceeds a threshold κ chosen to be 0.9. In order to report the forecast quality on a meaningful natural time scale, we normalize the forecast Valid Time by the ‘Lyapunov time’ of the dynamical system. The Lyapunov time is defined as the inverse of the Largest Lyapunov Exponent (LLE), denoted Λ_{max} , of the dynamical system being forecast. In a chaotic system, infinitesimal errors grow exponentially on average as $\exp(\Lambda_{max}t)$. Thus, Λ_{max}^{-1} is a natural time scale for evaluating

forecast quality in a chaotic system.

Since forecast Valid Time can vary considerably depending on what part of the attractor forecasts are initialized from, we present Valid Time results as box plots reflecting 100 different runs for the Lorenz System and 50 runs for the KS system. Each run uses a different initial condition for the training data (and therefore for the following forecast) and a different random realization of the reservoir. The box plots depict the median, the 25th and 75th percentiles, and the 5th and 95th percentiles (depicted by dashed lines) computed over these trials.

We evaluate the statistical significance of the difference between each sample of ML-DA and baseline forecast Valid Times using Mood’s median test [96]. This test considers the null hypothesis that the two samples come from distributions with the same median. We compute the median of the union of the two samples, then count the number of forecast Valid Times in each sample that exceeds this median. We then compute the p-value of these counts using Pearson’s chi-squared test; this represents the probability that a discrepancy between the two counts of at least the same size would have occurred if the null hypothesis were true. We choose to reject the null hypothesis, and declare the difference in the samples to be statistically significant, if the p-value is less than 10^{-3} .

3.2.4.5 Analysis RMS Error

For the iterative method in Sec. 3.2.3, we evaluate the error in the analysis state at a particular iteration i by computing the normalized RMS error over the entire analysis period,

$$e_{tot,i}^a = \sqrt{\frac{\langle \|\mathbf{x}_{j,i}^a - \mathbf{x}_j\|^2 \rangle}{\langle \|\mathbf{x}_j\|^2 \rangle}}, \quad (3.17)$$

where $\langle \dots \rangle$ denotes an average over the DA interval ($-T - T_S \leq j\Delta t \leq 0$). In addition, we evaluate the analysis error in the measured or unmeasured system variables by including only the particular set of variables in the norms in Eq. (3.17).

3.3 Results

We evaluate the performance of ML-DA using the procedures described in Sec. 3.2.4.

3.3.1 Lorenz ‘63

The Lorenz ‘63 system is described by the equations,

$$\frac{dX_1}{dt} = -aX_1 + aX_2 \quad (3.18)$$

$$\frac{dX_2}{dt} = bX_1 - X_2 - X_1X_3 \quad (3.19)$$

$$\frac{dX_3}{dt} = -cX_3 + X_1X_2 \quad (3.20)$$

where $a = 10$, $b = 28$, and $c = 8/3$. In this example, the true dynamical system, Eq. (3.1), is given by Eqs. (3.18-3.20). We obtain simulated data by integrating Eqs. (3.18-3.20) using a fourth order Runge-Kutta solver with time step $\tau = 0.01$. We sample the time series at intervals $\Delta t = 0.01$. This data is taken to represent the true system state $\mathbf{x}_j^t$, $-T - T_S \leq j \leq P$, where the subscript j indicates that the data was sampled at time $j\Delta t$. We use the data in the interval

$-T - T_S \leq j \leq 1$ to create simulated measurements

$$\mathbf{y}_j = \mathbf{H}\mathbf{x}_j^t + \boldsymbol{\eta}_j. \quad (3.21)$$

We will study two examples with two different forms of the measurement operator $\mathbf{H}$, one in which only the X_1 variable is measured,

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}, \quad (3.22)$$

and another in which the X_1 and X_3 variables are measured,

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (3.23)$$

Measurement noise $\boldsymbol{\eta}_j$ with a Gaussian distribution that has mean zero and covariance matrix $\mathbf{R}$ is added to the measurements. The covariance matrix is diagonal, with the diagonal elements equal to σ_{noise}^2 , which corresponds to the situation in which the errors in the measurements of the different state vector components are statistically independent. In Sec. 3.3.1.1 and Sec. 3.3.1.2, we let $\sigma_{noise} = 0.1$. For comparison, the standard deviations $\sigma_{X_1}, \sigma_{X_2}, \sigma_{X_3}$ of the variables X_1, X_2, X_3 are 7.9, 8.9 and 8.6, respectively. In Sec. 3.3.1.3, we consider the effect of varying σ_{noise} .

We let the imperfect model equations be given by

$$\frac{dX_1}{dt} = -aX_1 + aX_2, \quad (3.24)$$

$$\frac{dX_2}{dt} = b(1 + \epsilon)X_1 - X_2 - X_1X_3, \quad (3.25)$$

$$\frac{dX_3}{dt} = -cX_3 + X_1X_2. \quad (3.26)$$

Thus, the imperfect model **G** differs from the perfect model **F** in the parameter b by a multiplicative factor $(1 + \epsilon)$. Notice also that the model error is in the evaluation of X_2 , which is unmeasured in our experiments.

3.3.1.1 Optimizing the Covariance Inflation Parameter

The quality of both the baseline and ML-DA forecasts depends on the covariance inflation parameter ρ (see Eqs. (3.8) and (3.32)). This parameter is used in the DA cycle that initializes both forecasts, as well as forming the training data for the ML-DA. It is thus crucial that the covariance inflation parameter is optimized independently for both the forecast schemes. In Fig. 3.3, for $\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, we demonstrate the dependence of t_{VT} on the covariance inflation parameter ρ when the model error is $\epsilon = 0.1$. Each forecast Valid Time is shown for a set of 100 forecasts (see Sec. 3.2.4.4) made with the baseline scheme (magenta box plots) and the ML-DA scheme (light blue box plots). Figure 3.3 shows that the ML-DA scheme dramatically improves t_{VT} when the model has substantial error. In particular, the best median t_{VT} for the ML-DA (which occurs at $\rho = 1.2$) exceeds the best median t_{VT} baseline (which occurs at $\rho = 1.05$) by about a factor of 3.

In the results that follow, for each choice of method, model error, and noise magnitude, we

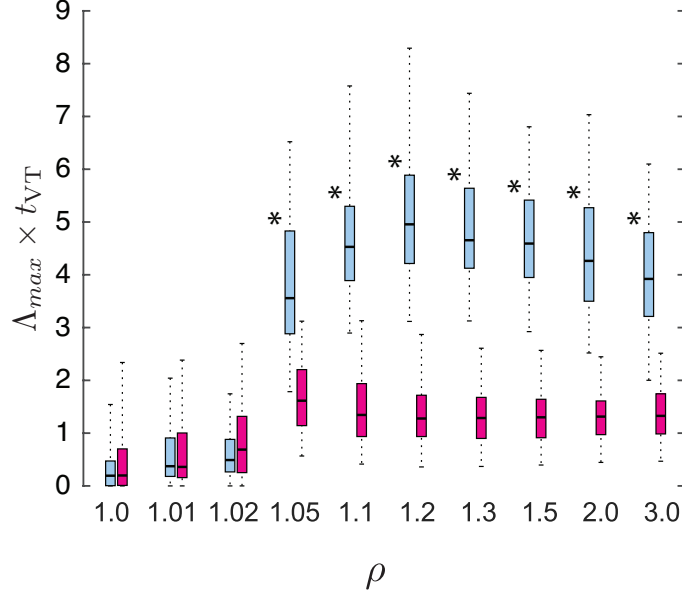


Figure 3.3: Dependence of the forecast Valid Time (t_{VT}) for the Lorenz ‘63 model on the covariance inflation parameter (ρ) when $\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, measurement noise has magnitude $\sigma_{noise} = 0.1$, and the model error is $\epsilon = 0.1$. Each box plot represents the result of 100 independent trials comparing the baseline and the ML-DA forecasts, each using a different Lorenz ‘63 time series starting from a random initial condition in the attractor. The box plots depict the median, the 25th and 75th percentiles, and the 5th and 95th percentiles (depicted by dashed lines) computed over these trials. The “*” above a pair of boxes indicates where Mood’s median test determined that the ML-DA median forecast Valid Time was significantly greater than that of the baseline for that value of ρ , with a p-value less than 10^{-3} .

choose the covariance inflation parameter ρ to be the value that yields the highest median t_{VT} , among the values shown in Fig. 3.3. For our usual values $\epsilon = 0.1$ and $\rho_{noise} = 0.1$, Figure 3.4 shows a typical example comparing the true and predicted time series; notice that the ML-DA scheme improves the forecast for all three variables, including those that are unobserved.

3.3.1.2 Dependence on Model Error

Figure 3.5 demonstrates the effectiveness of the ML-DA algorithm at different values of the model error (ϵ). As expected, we find that the ML-DA optimized over ρ yields the most improvement relative to the baseline optimized over ρ when the model error is large. In the case when the model error is zero, the ML-DA algorithm degrades the forecasts somewhat; since there is no model error to correct, the modification made to the model by the ML is spurious.

3.3.1.3 Effects of measurement Noise

Figure 3.6 shows the effects of varying amounts of measurement noise on the baseline (magenta box plots) and ML-DA (light blue box plots) forecast Valid Time for the Lorenz ‘63 system. The measurement operator is $\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$ and simulated uncorrelated Gaussian noise with standard deviation σ_{noise} is added to the measurements. We assume knowledge of the correct measurement noise, so that the DA procedure uses $\mathbf{R} = \sigma_{noise}^2 \mathbf{I}$. For comparison, the standard deviation of the X_1 variable is $\sigma_{X_1} \simeq 7.9$. We see that the ML-DA outperforms the baseline forecast by a larger amount when the noise is smaller.

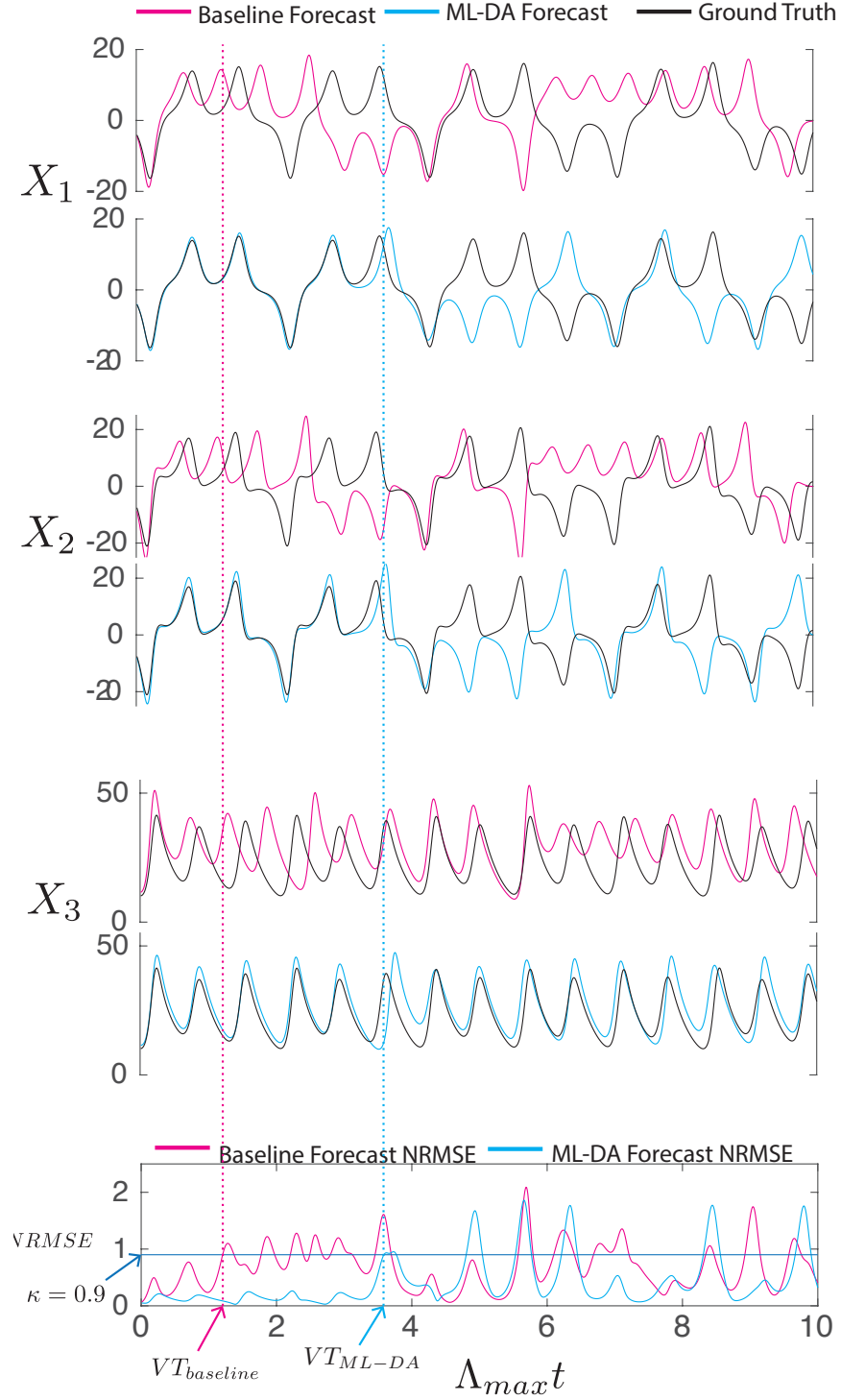


Figure 3.4: Forecast time series plots for the Lorenz '63 system using the ML-DA and baseline scheme for $\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$ and model error $\epsilon = 0.1$.

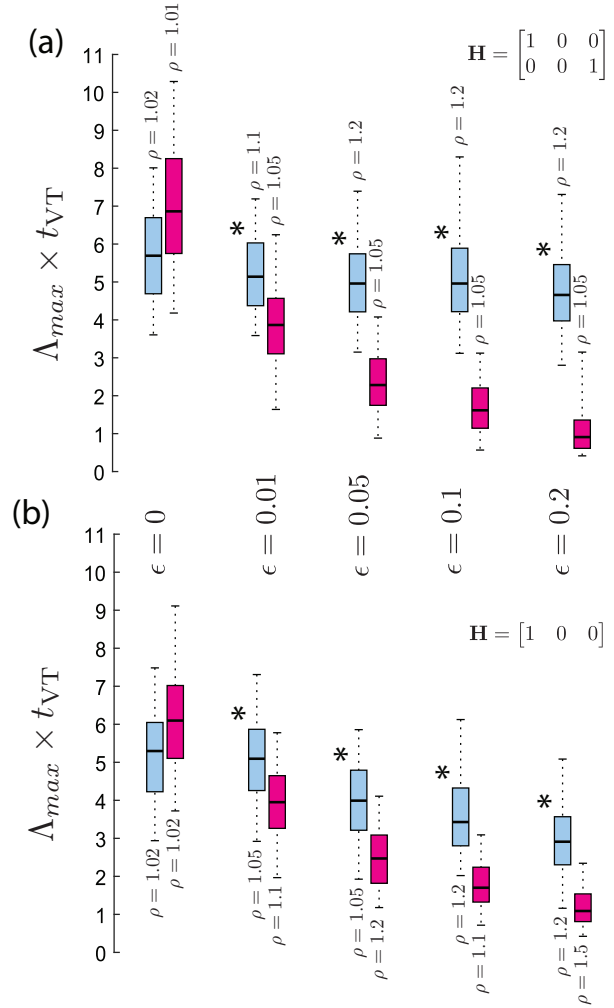


Figure 3.5: Dependence of t_{VT} on the model error (ϵ) for the Lorenz ‘63 system; (a) and (b) differ in the measurement operator used, which is indicated in the figures. Each box plot represents the result of 100 independent trials comparing the baseline and the ML-DA forecasts, each using a different Lorenz ‘63 time series starting from a random initial condition in the attractor. The box plots depict the median, the 25th and 75th percentiles, and the 5th and 95th percentiles (depicted by dashed lines) computed over these trials. The covariance inflation parameter is independently optimized for both sets of forecasts at each value of the model error (ϵ), and the optimal value is displayed next to each box. The “*” above a pair of boxes indicates where Mood’s median test determined that the ML-DA median forecast Valid Time was significantly greater than that of the baseline for that value of ϵ and optimal ρ , with a p-value less than 10^{-3} .

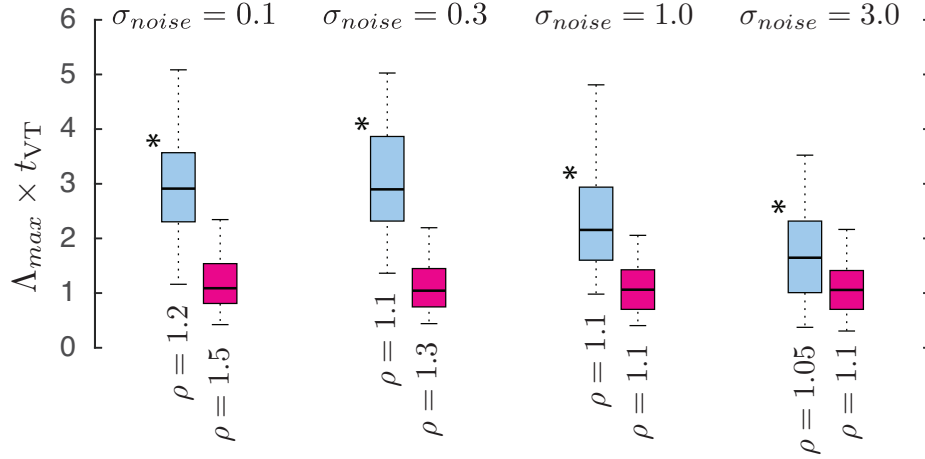


Figure 3.6: Effect of varying amounts of measurement noise on the baseline (magenta box plots) and ML-DA (light blue box plots) forecast Valid Time for the Lorenz ‘63 system, with model error $\epsilon = 0.2$. Each box plot represents the result of 100 independent trials comparing the baseline and the ML-DA forecasts, each using a different Lorenz ‘63 time series starting from a random initial condition in the attractor. The box plots depict the median, the 25th and 75th percentiles, and the 5th and 95th percentiles (depicted by dashed lines) computed over these trials. As in Fig. 3.5, the covariance inflation parameter is independently optimized for both sets of forecasts at each value of the measurement noise (σ_{noise}), and the optimal value is displayed next to each box. The “*” above a pair of boxes indicates where Mood’s median test determined that the ML-DA median forecast Valid Time was significantly greater than that of the baseline for that value of σ_{noise} and optimal ρ , with a p-value less than 10^{-3} . The measurement operator is $\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$, and simulated uncorrelated Gaussian noise with standard deviation σ_{noise} is added to the measurements. For comparison, the standard deviation of the X_1 variable is $\sigma_{X_1} \simeq 7.9$.

3.3.2 Kuramoto-Sivashinsky (KS) system

In this section we consider an example of a spatiotemporal chaotic dynamical system called the the Kuramoto-Sivashinsky (KS) system defined by the Partial Differential Equation (PDE),

$$\frac{\partial u(x, t)}{\partial t} + u \frac{\partial u(x, t)}{\partial x} + \frac{\partial^2 u(x, t)}{\partial x^2} + \frac{\partial^4 u(x, t)}{\partial x^4} = 0. \quad (3.27)$$

Equation (3.27) defines the evolution of a one-dimensional spatiotemporal scalar field

$u(x, t)$ on the spatial domain $x \in [0, L]$. We assume periodic boundary conditions so that $u(x + L, t) = u(x, t)$. In this example, we let Eq. (3.27) represent the true, unknown space and time continuous dynamics. We obtain a time series of finite-dimensional representations of the state that we assume to satisfy Eq. (3.1) by discretizing the domain $[0, L]$ into Q grid-points and integrating Eq. (3.27) using a pseudo-spectral PDE solver [97] with time step $\tau = 0.25$. We sample the time series at intervals of $\Delta t = 0.25$. Thus, our simulated true system states takes the form of a Q -dimensional vector $\mathbf{x}_j^t$, $-T - T_S \leq j \leq P$, where the subscript j indicates that the data was sampled at time $j\Delta t$. As in the previous example of the Lorenz ‘63 system (Sec. 3.3.1), we generate simulated measurements by adding random Gaussian noise to the simulated true states. We use the simulated measurements in the interval $-T - T_S \leq j \leq 0$ as the training data for the ML-DA scheme and set aside the data in the interval $1 \leq j \leq P$ for forecast verification.

We assume that we have access to an imperfect model, $\mathbf{G}$, of the KS dynamical system given by the equations

$$\frac{\partial u(x, t)}{\partial t} + u \frac{\partial u(x, t)}{\partial x} + (1 + \epsilon) \frac{\partial^2 u(x, t)}{\partial x^2} + \frac{\partial^4 u(x, t)}{\partial x^4} = 0. \quad (3.28)$$

Thus, if $\epsilon = 0$, then the model represents the true dynamics perfectly, while a nonzero value of ϵ indicates an imperfect model. We also assume that our measurements are of the form given by Eq. (3.2). The measurement operator $\mathbf{H}$ is a projection operator that outputs Θ uniformly spaced measurements (of the Q total variables) in $\mathbf{x}_j^t$. The measurement noise is Gaussian with zero mean and a diagonal covariance matrix with diagonal elements equal to σ_{noise}^2 , where $\sigma_{noise} = 0.1$. For comparison, the standard deviation of the KS time series variable is $\sigma_{KS} = 1.3$.

3.3.2.1 Forecast Quality and Dependence on Model Error

To demonstrate the effectiveness of the ML-DA technique, we consider a KS system with $L = 35$, $Q = 64$ and a measurement operator $\mathbf{H}$ that corresponds to $\Theta = 16$ uniformly spaced measurements. The upper three panels of Fig. 3.7 shows (from top to bottom) the time dependence of the true discretized Kuramoto-Sivashinsky equation dynamics, the error in the baseline prediction, and the error in the ML-DA prediction when the model error is $\epsilon = 0.1$. The bottom panel in Fig. 3.7 shows the normalized RMS error of both predictions. Notice that the ML-DA again significantly outperforms the baseline scheme.

Figure 3.8 shows t_{VT} for 50 different forecasts per box plot made with the ML-DA and baseline scheme at four different values of the model error ϵ . As for the Lorenz '63 system, as the size of the error in the imperfect model increases, the ML-DA scheme outperforms the baseline scheme by an increasing ratio, but as expected, it degrades performance somewhat when the model error is zero (and, thus, no ML correction is needed). In general, this degradation must persist for sufficiently small model error; for the KS system, model error $\epsilon = 0.01$ is not large enough for us to see a clear advantage to either scheme.

3.3.2.2 Analysis Error using Iterative ML-ETKF

To demonstrate that the ML-DA hybrid model is also capable of improving the analysis states in a DA cycle, we re-do the DA run that produced the training data using the hybrid model in place of the knowledge-based model. We then re-train the hybrid model, and iterate as described in Sec. 3.2.3. Figure 3.9 shows the error in analysis states computed using iteratively produced hybrid models. Here, iteration 0 denotes the original set of analysis states computed using only

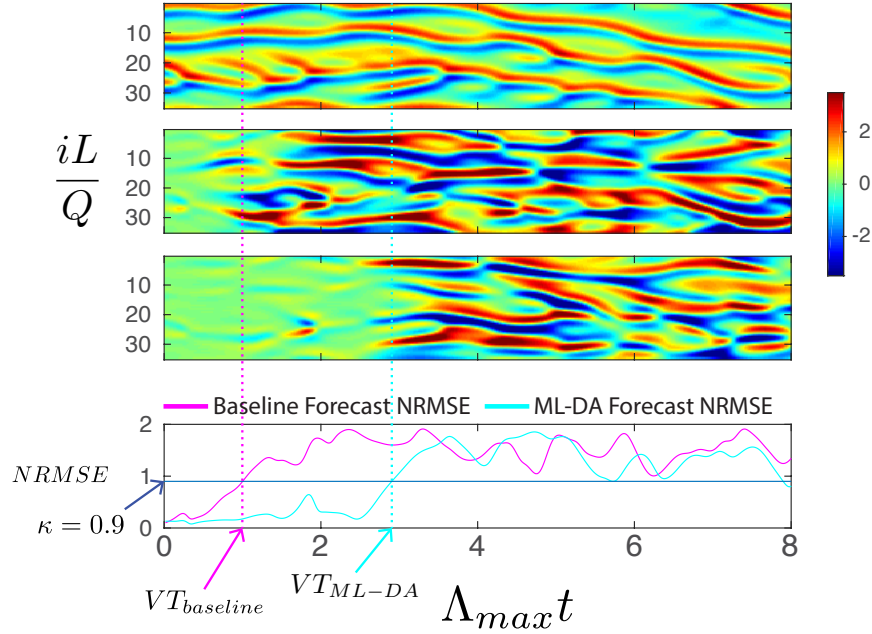


Figure 3.7: The top 3 panels in this figure are spatiotemporal plots with the 64 spatial grid point locations plotted vertically and the Lyapunov time plotted horizontally. From top to bottom in these panels, the color axis shows the true value of the i^{th} element of $\mathbf{x}_j^t$ for $i = 1, 2, \dots, Q$ from the KS system, the difference between the true i^{th} element of $\mathbf{x}_j^t$ for $i = 1, 2, \dots, Q$ from the KS system and the prediction from the baseline scheme, and the difference between the true i^{th} element of $\mathbf{x}_j^t$ for $i = 1, 2, \dots, Q$ from the KS system and the prediction from the ML-DA scheme. The bottom plot shows the normalized RMS error as function of time for the baseline (magenta) and ML-DA (light blue) schemes, while the dotted vertical lines mark the forecast Valid Time using each scheme. For both schemes, the model error was $\epsilon = 0.1$, and the state was measured at $\Theta = 16$ equally spaced locations of the full $Q = 64$ grid points with measurement noise magnitude $\sigma_{noise} = 0.1$.

the knowledge-based model. We see that using the first 2 generations of the trained ML-DA to perform DA results in a significant reduction in error in the analyses of both the measured and unmeasured variables. Later iterations give smaller reductions in error. We also measured the forecast Valid Time for each hybrid model trained on the iteratively-produced analysis time series. We found that there was only a small increase in the median Valid Time after one iteration ($\sim 10\%$), and that successive iterations did not improve the Valid Time.

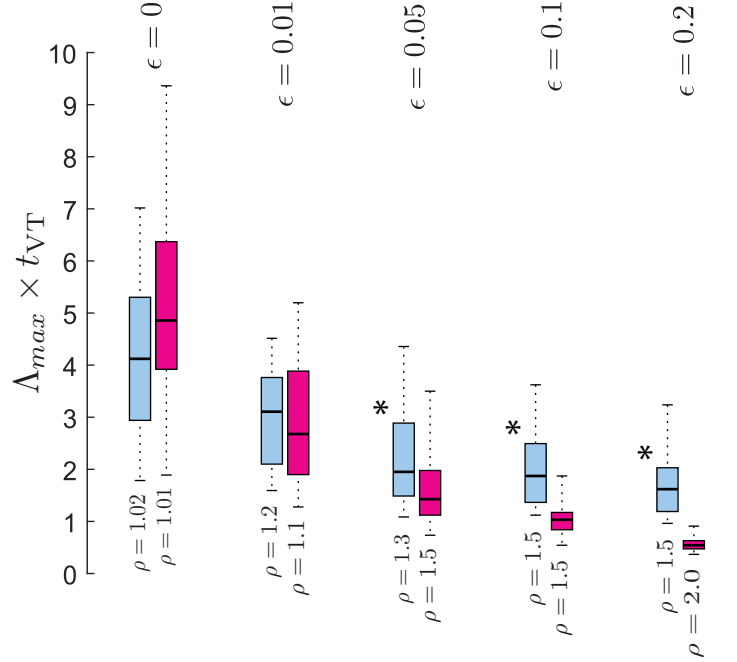


Figure 3.8: Forecast Valid Time of the ML-DA (light blue box plots) and baseline (magenta box plots) schemes for the KS system at different values of the model error (ϵ). Each box plot represents the result of 50 independent trials comparing the baseline and the ML-DA forecasts, each using a different KS time series starting from a random initial condition in the attractor. The box plots depict the median, the 25th and 75th percentiles, and the 5th and 95th percentiles (depicted by dashed lines) computed over these trials. As in Fig. 3.5, the covariance inflation parameter is independently optimized for both sets of forecasts at each value of the model error (ϵ), and the optimal value is displayed next to each box. The “*” above a pair of boxes indicates where Mood’s median test determined that the ML-DA median forecast Valid Time was significantly greater than that of the baseline for that value of ϵ and optimal ρ , with a p-value less than 10^{-3} .

3.4 Conclusions and Discussion

We have demonstrated that DA can successfully extend the hybrid approach introduced in [22], which trains a machine-learning component based on reservoir computing to improve a knowledge-based forecast model, to the case where a multivariate time series of complete, directly-measured model states is not available for training. Though we use a specific ensemble DA scheme, ETKF, in our numerical experiments, any DA scheme could be used to process the available measurements into estimated model states for use as training data. Using a knowledge-

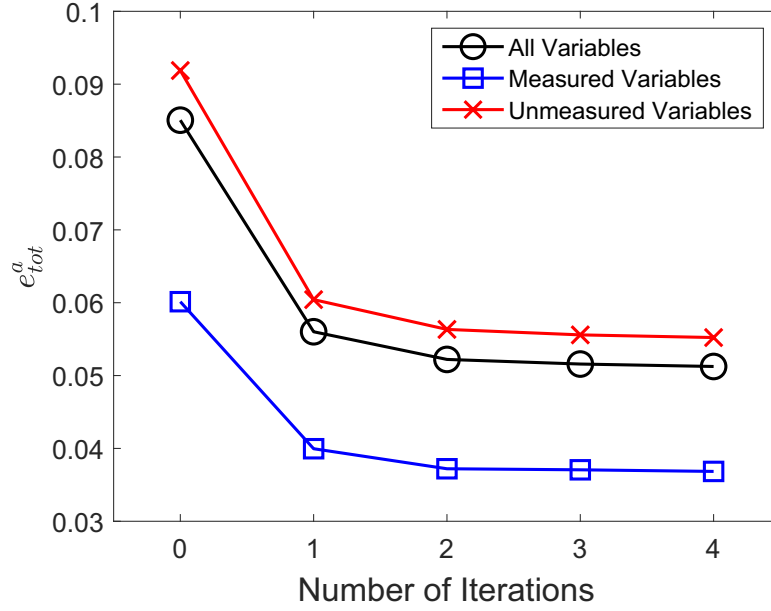


Figure 3.9: Normalized RMS error during DA as we iteratively use the ML-DA to produce analysis time series for the KS system and use those time series to train the hybrid model. A covariance inflation parameter of $\rho = 2$, which we found to produce the lowest analysis error, was used to produce analysis time series at each step of the iteration. In addition, all knowledge-based models used in this plot had an error of $\epsilon = 0.2$. The plotted values are the median normalized RMS error values computed over 50 different sets of measurement data. The variance in the error over these different data sets is very small; therefore, we have chosen not to plot the standard deviation. At iteration 0, the error values are those from using only the knowledge-based model as our forecast model during DA. We also plot the error values computed over only the measured or the unmeasured variables.

based model rather than a purely data-driven approach allows us to perform DA before training the machine-learning component. A potential drawback to this approach is that biases present in the knowledge-based model will affect the training data, reducing the effectiveness of the training in correcting for these biases. Using the iterative approach described in Sec. 3.2.3, however, we are able to reduce the effect of these biases on the analysis errors by using the previously trained hybrid model as the forecast model on the DA. In applications like weather forecasting, where multiple knowledge-based models are available, another approach would be to use a different model in the DA procedure than in the hybrid training and prediction procedure.

Another natural extension of the method described in this chapter would be to integrate the machine-learning training with the data-assimilation cycle in such a way that the training is updated each time new data becomes available. The use of reservoir computing as the machine-learning component of our hybrid model allows a large number of parameters (the elements of the output weight matrix) to be trained efficiently using linear regression, which could be done sequentially (like DA) through techniques such as recursive least squares. It could be fruitful to more closely integrate the state estimation and the estimation of the parameters of the hybrid model. The unified Bayesian framework described in [73] and [74] could be applied to a hybrid model, though a successful implementation may require limiting the number of machine-learning parameters.

Acknowledgement

This work was supported by DARPA grant HR00111890044. The work of Alexander Wikner was supported in part by NSF award DGE-1632976. The work of Istvan Szunyogh was also supported by ONR Award N00014-18-2509. We thank the reviewers for their helpful comments.

Data Availability

The data that support the findings of this study are available from the corresponding author upon reasonable request. A software implementation in MATLAB of the hybrid reservoir computer described here (as well as in Refs. [22] and [36]) may be found at <https://github.com/awikner/CHyPP>.

3.5 Supplemental: The Ensemble Transform Kalman Filter

The particular data assimilation algorithm used in this chapter is the Ensemble Transform Kalman Filter (ETKF) [92, 93], a variant of the Ensemble Kalman Filter (EnKF) [88–90]. Our formulation is based on the local analysis used in the LETKF [86, 98, 99] for spatially extended systems larger than the KS system used in Sec. 3.3.2.

Recall from Sec. 3.2.1 that the analysis state at time $j\Delta t$ is computed from the analysis state at time $(j-1)\Delta t$ by a forecast step followed by an analysis step. An EnKF uses an ensemble $\{\mathbf{x}_j^{a,k}\}_{1 \leq k \leq E}$ of analysis states whose mean $\bar{\mathbf{x}}_j^a$ represents the estimated state and whose covariance $\mathbf{P}_j^a$ represents the uncertainty in the estimate. We now describe the forecast and analysis steps for the ensemble.

In an EnKF, the background ensemble members $\mathbf{x}_j^{b,k}$ are forecast separately using the model, Eq. (3.3), so that

$$\mathbf{x}_j^{b,k} = \mathbf{x}^k(j\Delta t) \quad \text{where} \quad \frac{d\mathbf{x}^k}{dt} = \mathbf{G}[\mathbf{x}^k(t)], \quad \mathbf{x}^k((j-1)\Delta t) = \mathbf{x}_{j-1}^{a,k}. \quad (3.29)$$

The sample covariance $\mathbf{P}_j^b$ of the background ensemble is

$$\mathbf{P}_j^b = (E-1)^{-1} \sum_{i=1}^E (\mathbf{x}_j^{b,i} - \bar{\mathbf{x}}_j^b)(\mathbf{x}_j^{b,i} - \bar{\mathbf{x}}_j^b)^T, \quad (3.30)$$

where $\bar{\mathbf{x}}_j^b$ is the mean of the background ensemble members. The goal of the analysis step, in which the types of EnKF differ, is to compute an analysis ensemble whose mean $\bar{\mathbf{x}}_j^a$ and sample covariance $\mathbf{P}_j^a$ are consistent with the Kalman filter [100].

We now present the computations that form the analysis step of the ETKF, as formulated in Ref. [86] (in which the consistency with the Kalman filter is explained). Recall that $\mathbf{y}_j$ is a vector of measurements made at time $j\Delta t$.

1. Create the matrix

$$\mathbf{X}_j^b = \begin{bmatrix} \mathbf{x}_j^{b,1} - \bar{\mathbf{x}}_j^b & | & \mathbf{x}_j^{b,2} - \bar{\mathbf{x}}_j^b & | & \dots & | & \mathbf{x}_j^{b,E} - \bar{\mathbf{x}}_j^b \end{bmatrix}, \quad (3.31)$$

whose columns represent deviations of the ensemble members from the ensemble mean.

2. Compute the matrix $\mathbf{Y}_j^b = \mathbf{H}\mathbf{X}_j^b$ and the vector $\bar{\mathbf{y}}_j^b$.
3. Create a matrix

$$\tilde{\mathbf{P}}_j^a = [(E-1)I/\rho + \mathbf{C}\mathbf{Y}_j^b] \quad (3.32)$$

where $\mathbf{C} = (\mathbf{Y}_j^b)^T \mathbf{R}^{-1}$ and ρ is the covariance inflation parameter described in Sec. 3.2.1.

4. Compute $\mathbf{W}_j^a = [(E-1)\tilde{\mathbf{P}}_j^a]^{1/2}$, where the $[\cdot]^{1/2}$ denotes the symmetric square root.
5. Compute $\bar{\mathbf{w}}_j^a = \tilde{\mathbf{P}}^a \mathbf{C}(\mathbf{y}_j - \bar{\mathbf{y}}_j^b)$ and add it to the k^{th} column of $\mathbf{W}_j^a$ to obtain $\mathbf{w}_j^{a,k}$.
6. Compute $\mathbf{x}_j^{a,k} = \mathbf{X}_j^b \mathbf{w}_j^{a,k} + \bar{\mathbf{x}}_j^b$. The analysis ensemble is given by $\{\mathbf{x}_j^{a,k}\}_{1 \leq k \leq E}$.

For our results, the analysis ensemble is initialized arbitrarily in a neighborhood of the model attractor. Then, the forecast and analysis steps described above are applied iteratively to generate the analysis time series used to train the ML system.

Chapter 4: Stabilizing Machine Learning Prediction of Dynamics using Noise and Noise-Inspired Regularization

This chapter is based on the paper, Using data assimilation to train a hybrid forecast system that combines machine-learning and knowledge-based components, by Alexander Wikner, Joseph Harvey, Michelle Girvan, Brian Hunt, Andrew Pomerance, Thomas Antonsen, and Edward Ott, Neural Networks 0893-6080 (2023).

4.1 Introduction

Learning dynamics solely from state time-series measurements of otherwise unknown complex dynamical systems is a challenging problem for which, in recent years, machine learning (ML) has been shown to be a promising solution. For example, ML models trained on time series measurements have been applied to obtain accurate predictions of terrestrial weather [28, 29, 31, 33–35]. Chaotic dynamics is of particular interest due to its common occurrence in complex systems. Time series measurements from such systems have been accurately predicted using ML (e.g., in Refs. [21, 101–103]), although, due to the exponentially sensitive dependence of chaotic orbits on perturbations, the time duration for which specific measurements can be predicted is necessarily limited (e.g., as in weather forecasting). Nonetheless, ML models can also produce arbitrarily long-term predictions that approximate the correct “climate” [104, 105], by which we

mean a statistical description of the long-term system behavior. However, as with numerical methods for solving differential equations, ML models sometimes generate artificial instabilities that lead to an inaccurate climate, as was seen in Refs. [25,34,35,102,106]. We call this situation a “climate instability”; such an instability might or might not degrade the accuracy of a short term forecast.

4.1.1 Machine learning prediction of dynamics and the issue of climate stability

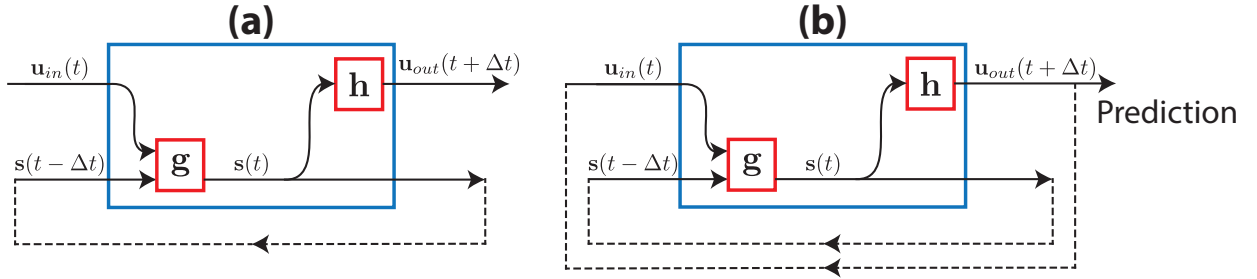


Figure 4.1: Machine Learning Model with Memory Used for Predicting Dynamics. The above diagrams show an ML model with memory for prediction in the (a) “open-loop” configuration (for training) and (b) “closed-loop” configuration (for prediction). In panels (a) and (b), the dashed line indicates that the output is fed back into the model as the next input.

In this chapter, we consider the commonly employed ML scheme used for predicting dynamical system states from time-series measurements $\{\mathbf{u}(t + \Delta t)\}$. We are concerned here with ML models that contain some form of a “memory” of previous inputs, so that $\mathbf{u}_{out}(t + \Delta t)$ depends not only on $\mathbf{u}_{in}(t)$, but also on a memory vector $\mathbf{s}(t - \Delta t)$ that contains information on $\mathbf{u}_{in}(t - \Delta t)$, $\mathbf{u}_{in}(t - 2\Delta t)$, etc. The general structure of such models is shown in Fig. 4.1. The model output is given by the memory evolution function, $\mathbf{g}$, and the output function, $\mathbf{h}$, as follows:

$$\begin{aligned} \mathbf{s}(t) &= \mathbf{g}(\mathbf{u}_{in}(t), \mathbf{s}(t - \Delta t)), \\ \mathbf{u}_{out}(t + \Delta t) &= \mathbf{h}(\mathbf{s}(t)). \end{aligned} \tag{4.1}$$

Notice that our formulation allows $\mathbf{s}(t)$ to include auxiliary variables that are used to compute $\mathbf{h}(\mathbf{s}(t))$ but are ignored as inputs by $\mathbf{g}$, so that they are not, strictly speaking, part of the model’s memory (The case of ML without memory can be represented by making $\mathbf{g}$ depend only on $\mathbf{u}_{in}(t)$). In general, both $\mathbf{g}$ and $\mathbf{h}$ can contain parameters that will be learned during the ML model training. The ML model memory is a necessary component when one does not have access to full system measurements, and it often improves model performance even when full system state measurements are available.

The goal during ML model training in this scheme (Fig. 4.1(a)) is to adjust the model weights such that when the model input has been $\mathbf{u}_{in} = \mathbf{u}$ up to and including time t , the output $\mathbf{u}_{out}(t + \Delta t)$ closely matches the measurements $\mathbf{u}(t + \Delta t)$. We refer to Fig. 4.1(a) as the “open-loop” configuration. After training, the prediction is initialized after time $t = T_{init}\Delta t$ by switching the model to the configuration shown in Fig. 4.1(b), which we refer to as the “closed-loop” configuration. In this configuration, the output $\mathbf{u}_{out}((T_{init} + n)\Delta t)$ is used as the next input $\mathbf{u}_{in}((T_{init} + n)\Delta t)$ for $n = 1, 2, 3, \dots$, and the outputs $\{\mathbf{u}_{out}((T_{init} + n)\Delta t)\}$ also form the predicted time series of measurements. This closed-loop configuration is used for prediction in Refs. [21, 28, 29, 31, 33, 101, 102, 107] for ML models both with and without memory.

Once a trained ML model has been placed into the closed-loop configuration, it acts as an autonomous dynamical system. Assuming that the unknown dynamical system generating the training data $\{\mathbf{u}(t)\}$ is evolving on an “attractor” (an invariant set of the state space dynamics that “attracts” nearby orbits [108]), the closed-loop ML model plausibly has an invariant set that approximates this attractor [20]. From examples, it appears that this approximating invariant set is often indeed an attractor for the ML model, with ergodic properties nearly identical to those of the true dynamical system being measured [19]. However, even in cases where the existence of

an approximating invariant set can be guaranteed, it might be that small perturbations transverse to the invariant set grow with time, so that, eventually, the ML model climate grossly differs from that of the dynamical system producing the training data $\{\mathbf{u}(t)\}$ [20]. Climate stability requires suppressing the growth of such perturbations. In chaotic systems where accurate long-term prediction is impossible, such as the earth’s atmosphere, obtaining the correct climate is often the goal of long-term predictions. In addition, climate instability can limit the duration of accurate short-term state forecasts (as shown, e.g., in Figs. 4 and 5 of Ref. [19]). This can occur if the growth time of the climate instability is fast enough that it causes substantial deviation from the invariant set of the closed-loop system before the predictions break down due to the natural chaos of the orbits on the original attractor of the unknown system being predicted.

4.1.2 Stabilization

One of our primary goals in this chapter is to study the effect of adding noise to the model input. Adding noise as a form of regularization is an established technique for improving the robustness and generalization of artificial neural networks [109–114]. The addition of noise as regularization is also an essential component of NARMAX models for nonlinear system identification [115]. Adding input noise is a commonly used technique in reservoir computing (see, e.g., Refs. [46, 102, 116]). Here, we systematically compare the utility of noise regularization with other regularization techniques for promoting climate stability and prediction accuracy.

In the context of learning to forecast chaotic dynamics, one can view the added noise as perturbing the input training orbit off the target invariant set of the dynamics so that, during the training, the ML model learns to respond to input from a neighborhood of the invariant set. Thus,

perturbations from the invariant set are trained to be pulled back toward the invariant set, tending to make it stable (i.e., make it an attractor). We emphasize, however, that this reasoning is only heuristic, because it is based on the open-loop system (Fig. 4.1(a)), while the prediction uses the closed-loop system (Fig. 4.1(b)).

In addition to systematically considering the utility of input noise, we also formulate and test a new penalty term in the ML loss function based on the idea of adding noise to the input training data. This term is computed by linearly approximating the mean effect of added noise from a prescribed number of most-recent inputs to the ML model with memory. This penalty term is deterministic, approximating the effect of an arbitrarily large number of noise realizations without sampling error. We refer to the technique of training with this penalty as Linearized Multi-Noise Training (LMNT) and to the resulting regularization as LMNT regularization. In the context of our chosen machine learning method, reservoir computing, we find that LMNT regularization greatly simplifies the tuning of the hyperparameter associated with the regularization strength when compared to the approach of adding noise to the model input, though this particular benefit might be specific to the training protocol for reservoir computing.

4.1.3 Outline and main results

This chapter is structured as follows: In Sec. 4.2, we describe the process of training an ML predictor with memory to produce short-term forecasts and then using it for long-term climate prediction. We discuss in Sec. 4.2.1 our implementation of reservoir computing using a recurrent neural network (RNN), describe the process of training in Sec. 4.2.2, and discuss different regularization techniques, whose performance we will test, in Sec. 4.2.3. We then describe our

LMNT technique and the resulting regularization in Sec. 4.2.4. In Sec. 4.2.5, we discuss how we will produce and evaluate short and long-term predictions using each of the regularization techniques described. Our results using reservoir computing with training data from a chaotic test system, the Kuramoto-Sivashinsky equation, are discussed in Sec. 4.3. We discuss alternative implementations of LMNT and give concluding remarks in Sec. 4.4.

4.1.4 Additional Background: ML Model Robustness to Perturbations

A number of techniques have been used to improve ML model “robustness” – insensitivity to small spurious changes – for input/output situations analogous to Fig. 4.1(a). Since this relates to the subject of this chapter, we discuss some of this past background. These techniques include regularization, such as Tikhonov regularization [58] (also known as ridge regression), LASSO regression [117], and Jacobian regularization [118]. Training using a dropout scheme [119] is another method which introduces random noise to improve model robustness. In RNN models, teacher forcing [120], the more sophisticated professor forcing [121], or the online weight adjustment technique known as FORCE learning [122] are used to encourage the dynamics during the closed-loop prediction to more closely resemble the dynamics during training. Noise may also be added to an RNN model’s internal state to encourage stability [123]. In the case where an ML prediction model runs in a closed-loop configuration (as in our Fig. 4.1(b)), using a loss function that incorporates multiple feedback loops, which can be done simultaneously, as in Refs. [28,34], or iteratively, as in Ref. [33] (referred to as “fine-tuning”), may also improve model stability. In addition, hybrid methods which combine machine learning with a science-based model can also improve model stability and climate replication [36,39]. Many of these regularization techniques

introduce additional model “hyperparameters” – algorithmic parameters that are not optimized within the training procedure – to obtain optimal performance. Tuning hyperparameters can often be computationally expensive, even when *full* optimization is not attempted.

4.2 Methods

We consider the case where we have a finite-duration time series of training data consisting of M simultaneous measurements of state variables from our unknown dynamical system, $\mathbf{u}(t) = [u_1(t), u_2(t), \dots, u_M(t)]^\top$, obtained from time t_0 to time $t_0 + t_{\text{train}}$ with a sampling time-step of Δt . We standardize $\mathbf{u}$ by applying, for each k , a linear transformation to the k^{th} measurement time series so that $\mathbf{u}_k(t)$ has mean 0 and standard deviation 1. We assume that the sampling time-step is short compared to the time scale of the unknown dynamical system (e.g., for a chaotic system, the average e -fold error growth time, known as the “Lyapunov time”).

We perform our model training in the open-loop configuration shown in Fig. 4.1(a). Beginning with an arbitrary initial memory vector, e.g., $\mathbf{s}(t) = 0$, we input samples from our training time series to our ML model ($\mathbf{u}_{in}(t) = \mathbf{u}(t)$), obtaining an M -dimensional output $\mathbf{u}_{out}(t + \Delta t)$ and the evolved memory vector. The goal of the training is to adjust the parameters of our ML model, contained in the functions $\mathbf{g}$ and $\mathbf{h}$, so that $\mathbf{u}_{out}(t + \Delta t) \simeq \mathbf{u}(t + \Delta t)$ for $t_0 \leq t \leq t_0 + t_{\text{train}} - \Delta t$. Once trained, our model takes an input $\mathbf{u}(t)$ and outputs a prediction of the dynamical system state at time $t + \Delta t$. Following training, we use the trained model to produce predictions by switching our ML model into the closed-loop configuration shown in Fig. 4.1(b). After receiving input up to time $t_{init} \geq t_0 + t_{\text{train}}$ and evolving the memory vector accordingly, the model in this configuration functions autonomously by feeding the model output back into the model as

input ($\mathbf{u}_{in}(t + \Delta t) = \mathbf{u}_{out}(t + \Delta t)$), shown by the bottom dashed line in Fig. 4.1(b). We obtain a prediction for the unknown dynamical system state at time $t_{init} + T\Delta t$, where T is a positive integer, by cycling the model as depicted T times and recording the final model output as the prediction.

4.2.1 Reservoir computing

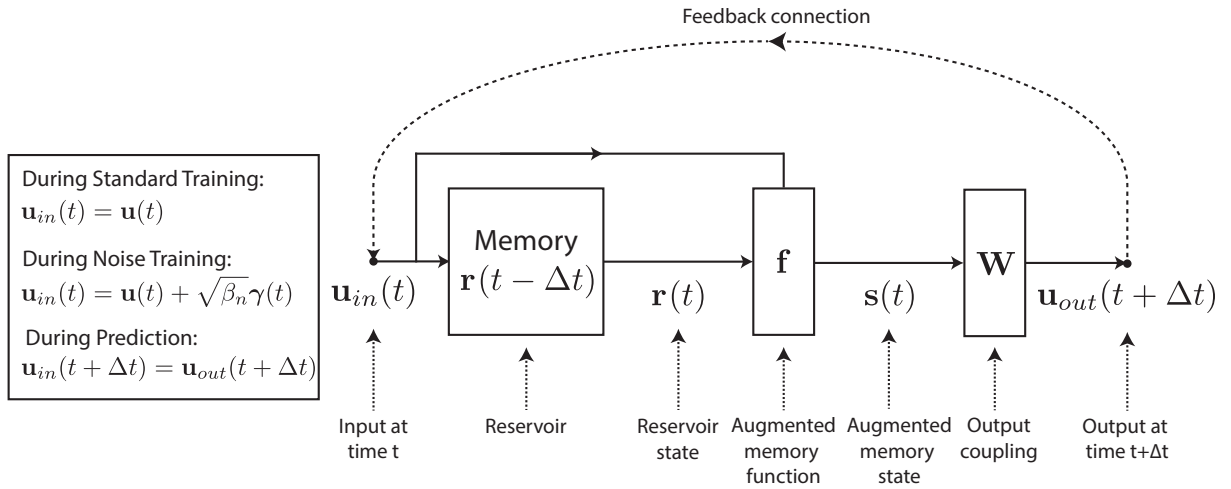


Figure 4.2: Diagram of our Reservoir Computer Model. The reservoir is trained in the “open-loop” training phase (analogous to Fig. 4.1(a)). During this phase, the dashed line representing the feedback loop is inactive. Once training is complete, the “closed-loop” prediction phase (analogous to Fig. 4.1(b)) may begin by activating the output-to-input feedback connection (shown by the dashed line). On the left, we indicate the input during training for the standard training (discussed in Sec. 4.2.2) and for noise training (discussed in Sec. 4.2.3.3), as well as the feedback input during prediction (discussed in Sec. 4.2.5).

We next describe the particular ML technique with memory used in this chapter, known as reservoir computing. In reservoir computing, the M -dimensional input vector $\mathbf{u}_{in}(t)$ is coupled by an $N \times M$ input coupling matrix $\mathbf{B}$ to a high-dimensional reservoir with an N -dimensional internal state vector $\mathbf{r}(t)$. The internal reservoir state at time t depends on both this input and the internal state at time $t - \Delta t$. This internal state is an essential component of the memory vector and allows the reservoir to produce useful predictions even when the M -dimensional vector $\mathbf{u}_{in}(t)$

does not have enough components to represent the full dynamical state of the unknown system to be predicted. The evolved memory vector is finally coupled by an output coupling matrix $\mathbf{W}$ to produce an output $\mathbf{u}_{out}(t + \Delta t)$. In reservoir computing, only the matrix elements of $\mathbf{W}$ (i.e., the parameters of the output function $\mathbf{h}(\mathbf{s}(t)) = \mathbf{W}\mathbf{s}(t)$) are adjusted during training, while the input coupling matrix elements and the internal reservoir coupling parameters (i.e., the parameters of the memory evolution function $\mathbf{g}$) are fixed after initialization.

For the reservoir computer implementation used in this chapter, we use an artificial recurrent neural network with a large number of computational nodes. This network's adjacency matrix, $\mathbf{A}$, is an $N \times N$ sparse matrix with randomly generated non-zero elements. These non-zero elements represent edges in a directed, weighted graph of N nodes, and the i^{th} element of the reservoir state vector $\mathbf{r}(t)$ represents the scalar state of the i^{th} network node. Values of the non-zero elements of $\mathbf{A}$ are sampled from a uniform random distribution over the interval $[-1, 1]$, and are assigned such that the average number of non-zero elements per row is equal to $\langle d \rangle$, called the “average in-degree”. For $\mathbf{A}$ to be sparse, we choose $\langle d \rangle \ll N$. Then, $\mathbf{A}$ is re-scaled ($\mathbf{A} \rightarrow [\text{constant}] \times \mathbf{A}$) such that its “spectral radius” ρ , the magnitude of the maximum magnitude eigenvalue of $\mathbf{A}$, is equal to some chosen value. The spectral radius is chosen to be small enough (typically less than 1 suffices) that the reservoir will satisfy the “echo-state” property [46] and large enough that the reservoir has substantial memory. We choose the input coupling $\mathbf{B}$ as a $N \times M$ matrix with exactly one non-zero value per row so that each reservoir node is coupled to exactly one coordinate of $\mathbf{u}_{in}(t)$. We choose the location of these non-zero elements such that each input is coupled to an approximately equal number of reservoir nodes. These non-zero elements are sampled from a uniform random distribution over the interval $[-\sigma, \sigma]$; we refer to σ as the “input scaling”.

Given an input $\mathbf{u}_{in}(t)$ and the previous reservoir internal state $\mathbf{r}(t - \Delta t)$, the reservoir internal state evolves according to the following equation:

$$\mathbf{r}(t) = (1 - \alpha)\mathbf{r}(t - \Delta t) + \alpha \tanh(\mathbf{A}\mathbf{r}(t - \Delta t) + \mathbf{B}\mathbf{u}_{in}(t) + \mathbf{C}), \quad (4.2)$$

where $\tanh$ is applied element-wise. Here, $\mathbf{C}$ is an N -dimensional bias vector with elements sampled from a uniform random distribution over the interval $[-\theta, \theta]$; we refer to θ as the “input bias”. The leaking rate, α , controls the time scale of the reservoir evolution, and is chosen from the interval $(0, 1]$.

Some implementations of reservoir computing use $\mathbf{s}(t) = \mathbf{r}(t)$ as the input to the output coupling $\mathbf{W}$. Here, we use an augmented memory vector $\mathbf{s}(t)$ defined by:

$$\mathbf{s}(t) = \mathbf{f}(\mathbf{r}(t), \mathbf{u}_{in}(t)) = \begin{bmatrix} 1 \\ \mathbf{u}_{in}(t) \\ \mathbf{r}(t) \\ \mathbf{r}(t)^2 \end{bmatrix}. \quad (4.3)$$

Here, the $\mathbf{r}(t)^2$ represents an element-wise power of 2, while $[\cdot]$ represents vertical concatenation of the contained elements. We then compute the reservoir output as:

$$\mathbf{u}_{out}(t + \Delta t) = \mathbf{h}(\mathbf{s}(t)) = \mathbf{W}\mathbf{s}(t). \quad (4.4)$$

The most essential component of $\mathbf{s}(t)$ is the reservoir state vector $\mathbf{r}(t)$, which is a high dimensional vector ($N \gg M$) mapped from the much lower dimensional (M) input state space. In

addition to $\mathbf{r}(t)$, the memory vector also contains a constant (the 1), the reservoir input ($\mathbf{u}_{in}(t)$), and the squared reservoir state ($\mathbf{r}(t)^2$). We include the reservoir input so that our model need only learn the change from $\mathbf{u}(t)$ to $\mathbf{u}(t + \Delta t)$. Finally, we find empirically, as in Ref. [21], that including the squared reservoir state improves the accuracy of our forecasts.

The parameters used to generate the reservoir computer (N , $\langle d \rangle$, ρ , σ , θ , and α) and the regularization parameters used during training (which we will describe in Secs. 4.2.2-4.2.4) are “hyperparameters” (see Sec. 4.1.4). To obtain an accurate short-term prediction and a long-term prediction with a climate similar to that of the true system, it is necessary to carefully choose the values of these hyperparameters. In reservoir computing, modifying the hyperparameters used to generate a trained reservoir computer will require that one re-perform the training before the reservoir can again be used for prediction. Regularization parameters for some regularization types, however, can be changed without having to re-perform all of the steps in the reservoir training. We will discuss this more in Sec. 4.2.3.

We next describe the reservoir training process and loss function for the different types of regularization we will discuss in this chapter.

4.2.2 Training with Regularization

The goal of the training is to determine the $M \times (1 + M + 2N)$ output coupling matrix $\mathbf{W}$ such that $\mathbf{W}\mathbf{s}(t) \approx \mathbf{u}(t + \Delta t)$ when $\mathbf{u}(t)$ is generated by the unknown dynamical system. Training is accomplished with the reservoir in the “open-loop” configuration with no output feedback present. Our training data consists of measurements obtained at $T_{sync} + T_{train} + 1$ equally-spaced

values:

$$\begin{aligned} \{\mathbf{u}(t)\} = \{\mathbf{u}(0), \mathbf{u}(\Delta t), \dots, \\ \mathbf{u}((T_{sync} + T_{train} - 1)\Delta t), \mathbf{u}((T_{sync} + T_{train})\Delta t)\}. \end{aligned} \quad (4.5)$$

Each of the M components of $\mathbf{u}(t)$ has been standardized such that its mean value and standard deviation over the $T_{train} + T_{sync} + 1$ time steps are 0 and 1, respectively. Here, T_{train} represents the number of training samples, while, as explained subsequently, T_{sync} represents the number of synchronization samples. We begin training by initializing the reservoir so that $\mathbf{r}(-\Delta t) = \mathbf{0}$. We then input $\mathbf{u}_{in}(0) = \mathbf{u}(0)$, computing the evolved reservoir state $\mathbf{r}(0)$ and recording the memory vectors(0). Iterating from $t = \Delta t$ to $t = (T_{sync} + T_{train} - 1)\Delta t$, we perform this process, obtaining a set of memory vectors from $t = 0$ to $t = (T_{sync} + T_{train} - 1)\Delta t$. We can express the evolution of the memory vector during training using the open-loop memory evolution function, $\mathbf{g}_o$, as follows:

$$\mathbf{s}(t) = \mathbf{g}_o(\mathbf{u}(t), \mathbf{s}(t - \Delta t)) = \begin{bmatrix} 1 \\ \mathbf{u}(t) \\ \mathbf{g}_r(\mathbf{u}(t), \mathbf{s}(t - \Delta t)) \\ (\mathbf{g}_r(\mathbf{u}(t), \mathbf{s}(t - \Delta t)))^2 \end{bmatrix}, \quad (4.6)$$

where

$$\begin{aligned} \mathbf{g}_r(\mathbf{u}(t), \mathbf{s}(t - \Delta t)) = (1 - \alpha) \begin{bmatrix} \mathbf{0}_{N \times (1+M)}, \mathbf{I}_{N \times N}, \mathbf{0}_{N \times N} \end{bmatrix} \mathbf{s}(t - \Delta t) \\ + \alpha \tanh \left(\begin{bmatrix} \mathbf{0}_{N \times (1+M)}, \mathbf{A}, \mathbf{0}_{N \times N} \end{bmatrix} \mathbf{s}(t - \Delta t) + \mathbf{B}\mathbf{u}(t) + \mathbf{C} \right). \end{aligned} \quad (4.7)$$

In Eq. 4.7, $[\dots]$ denotes horizontal concatenation.

To ensure that the memory vectors have minimal dependence on the initial reservoir state, we do not use the first T_{sync} memory vectors to train the reservoir. The first T_{sync} inputs are only used to “synchronize” the reservoir state $\mathbf{r}(t)$ to the unknown dynamical system trajectory [124]. For a reservoir computer with typical hyperparameter values, T_{sync} may be set such that $T_{sync} \ll T_{train}$. We thus obtain T_{train} dynamical system state inputs to the synchronized reservoir, reservoir internal states, and memory vectors to be used for the training, which we will denote as $\{\mathbf{u}_j\}$, $\{\mathbf{r}_j\}$, and $\{\mathbf{s}_j\}$. Here, $0 \leq j \leq T_{train} - 1$ and sample j is obtained at time $t = (T_{sync} + j)\Delta t$. We additionally introduce the target time series data, $\{\mathbf{v}_j\}$, which contains the unknown dynamical system states $\mathbf{u}(T_{sync} + j + 1)$ that we desire to approximate at each time index j .

We train $\mathbf{W}$ such that $\mathbf{W}\mathbf{s}_j \approx \mathbf{v}_j$ by minimizing the following regularized least-squares loss function, which, for our purposes, has the form:

$$\ell(\mathbf{W}) = \frac{1}{T_{train}} \sum_{j=0}^{T_{train}-1} \|\mathbf{W}\mathbf{s}_j - \mathbf{v}_j\|_2^2 + \sum_i \beta_i \text{Tr}(\mathbf{W}\mathbf{R}_i\mathbf{W}^\top). \quad (4.8)$$

In Eq. 4.8 and in future equations, $\text{Tr}(\dots)$ denotes the trace of a matrix, and $\|\dots\|_2$ is the Euclidean norm. Here, $\mathbf{R}_i$ is a regularization matrix, which we denote with an index i , with an associated tunable regularization parameter β_i . Depending on the type of regularization used, $\mathbf{R}_i$ might or might not depend on $\{\mathbf{u}_j\}$ and $\{\mathbf{s}_j\}$. We will discuss the different types of regularization used in this chapter and their associated penalties in the loss function in Secs. 4.2.3–4.2.4. For all regularization types used, we will minimize this loss function using the “matrix solution”, as follows. We first form the matrices $\mathbf{S}$ and $\mathbf{V}$, where the j^{th} columns of $\mathbf{S}$ and $\mathbf{V}$ are $\mathbf{s}_j$ and $\mathbf{v}_j$,

respectively. We then determine a matrix $\mathbf{W}$ which solves the following linear system obtained by setting the derivative of Eq. 4.8 with respect to $\mathbf{W}$ to zero:

$$\mathbf{W} \left(\frac{1}{T_{train}} \mathbf{S} \mathbf{S}^\top + \sum_i \beta_i \mathbf{R}_i \right) = \frac{1}{T_{train}} \mathbf{V} \mathbf{S}^\top. \quad (4.9)$$

While all of the regularization methods which we test in this chapter may be incorporated into the matrix solution, we note that this is not the case for all regularization types (e.g., LASSO regularization [117]). In such cases, it would be necessary to use a method other than the matrix solution to minimize Eq. 4.8 (e.g., proximal gradient methods for LASSO [125]). We solve for $\mathbf{W}$ in Eq. 4.9 using the `gesv` function in LAPACK [126], implemented via `numpy.linalg.solve` [127] in Python.

4.2.3 Regularization

We now discuss the regularization techniques to improve an ML model’s climate stability that we will test in this chapter. Each of these techniques adds a penalty term to the loss function that, for the reservoir computer training, can be reduced to the form of Eq. 4.8. Noise training, which replaces the vectors $\mathbf{s}_j$ in Eq. 4.8 with perturbed vectors $\tilde{\mathbf{s}}_j$, is the one exception to this general form discussed in this chapter.

4.2.3.1 Tikhonov Regularization

Tikhonov regularization [58], also known as “ridge regression”, places a penalty on the Frobenius (element-wise L^2) norm of the trainable model parameters to prevent model overfitting. In this chapter, we consider Tikhonov regularization using a single scalar regularization

parameter, β_T ; however, one may generally use a matrix of regularization parameters to penalize different trainable model parameters to different degrees. The Tikhonov regularization function for the reservoir computer is

$$\beta_T \|\mathbf{W}\|_F^2 = \beta_T \text{Tr}(\mathbf{W}\mathbf{W}^\top) \quad (4.10)$$

where $\|\dots\|_F$ is the Frobenius matrix norm, i.e., the square root of the sum of the squares of all the matrix elements. For use in the last term on the right-hand side of Eq. 4.8, we define a Tikhonov regularization matrix as

$$\mathbf{R}_T = \mathbf{I}_{(1+M+2N) \times (1+M+2N)}, \quad (4.11)$$

and express $\text{Tr}(\mathbf{W}\mathbf{W}^\top)$ as $\text{Tr}(\mathbf{W}\mathbf{R}_T\mathbf{W}^\top)$.

4.2.3.2 Jacobian Regularization

Jacobian regularization penalizes the Frobenius norm of an ML model's input-output Jacobian matrix to promote model robustness with respect to input perturbations [118]. For ML models with memory, we use the Jacobian matrix of $\mathbf{u}_{out}(t + \Delta t) = \mathbf{h}(\mathbf{g}(\mathbf{u}_{in}, \mathbf{s}(t - \Delta t)))$ with respect to $\mathbf{u}_{in}(t)$. Notably, Jacobian regularization only penalizes the Jacobian with respect to the most recent model input and does not account for the dependence of $\mathbf{s}(t - \Delta t)$ on earlier inputs. Using the notation of Fig. 4.1 and Eq. 4.1, the Jacobian regularization penalty term for a single

training sample can be expressed as

$$\beta_J \|\nabla_{\mathbf{s}} \mathbf{h}(\mathbf{s}_j) \nabla_{\mathbf{u}} \mathbf{g}(\mathbf{u}_j, \mathbf{s}_{j-1})\|_F^2. \quad (4.12)$$

Here, β_J is a scalar regularization term and $\nabla_{\mathbf{x}} \mathbf{f}(\mathbf{x}_j, \mathbf{y}_j)$ denotes the Jacobian of $\mathbf{f}(\mathbf{x}, \mathbf{y})$ with respect to $\mathbf{x}$ evaluated at $\mathbf{y} = \mathbf{y}_j$ and $\mathbf{x} = \mathbf{x}_j$.

For the reservoir computer, we use the reservoir open-loop memory evolution function $\mathbf{g}_o(\mathbf{u}(t), \mathbf{s}(t - \Delta t))$ for an input $\mathbf{u}_j$ and that $\nabla_{\mathbf{s}} \mathbf{h}(\mathbf{s}_j) = \mathbf{W}$. From this, we can express the Jacobian regularization function computed for all training samples as

$$\frac{\beta_J}{T_{train} - 1} \sum_{j=1}^{T_{train}-1} \|\mathbf{W} \nabla_{\mathbf{u}} \mathbf{g}_o(\mathbf{u}_j, \mathbf{s}_{j-1})\|_F^2 = \beta_J \text{Tr}(\mathbf{W} \mathbf{R}_J \mathbf{W}^\top), \quad (4.13)$$

where the Jacobian regularization matrix is

$$\mathbf{R}_J = \frac{1}{T_{train} - 1} \sum_{j=1}^{T_{train}-1} \nabla_{\mathbf{u}} \mathbf{g}_o(\mathbf{u}_j, \mathbf{s}_{j-1}) \nabla_{\mathbf{u}} \mathbf{g}_o(\mathbf{u}_j, \mathbf{s}_{j-1})^\top. \quad (4.14)$$

We have introduced a normalizing factor of $1/(T_{train} - 1)$ so that the scaling of $\mathbf{R}_J$ is approxi-

mately independent of T_{train} . We compute $\nabla_{\mathbf{u}} \mathbf{g}_o(\mathbf{u}_j, \mathbf{s}_{j-1})$ from Eqs. 4.6 and 4.7 as follows.

$$\nabla_{\mathbf{u}} \mathbf{g}_o(\mathbf{u}_j, \mathbf{s}_{j-1}) = \begin{bmatrix} 0 \\ \mathbf{I}_{M \times M} \\ \alpha \text{diag}(\mathbf{d}(\mathbf{r}_{j-1}, \mathbf{u}_j)) \mathbf{B} \\ \alpha \text{diag}(2\mathbf{r}_j) \text{diag}(\mathbf{d}(\mathbf{r}_{j-1}, \mathbf{u}_j)) \mathbf{B} \end{bmatrix}, \quad \text{where} \quad (4.15)$$

$$\text{diag}(\mathbf{d}) = \begin{bmatrix} d_1 & 0 \\ & \ddots \\ 0 & d_N \end{bmatrix}, \quad \text{and} \quad (4.16)$$

$$\mathbf{d}(\mathbf{r}_{j-1}, \mathbf{u}_j) = \text{sech}^2(\mathbf{A}\mathbf{r}_{j-1} + \mathbf{B}\mathbf{u}_j + \mathbf{C}), \quad (4.17)$$

where the sech^2 in Eq. 4.17 is the derivative of $\tanh$. We note that Eq. 4.14 depends on the reservoir model parameters ($\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and α) and the internal reservoir state, $\mathbf{r}_{j-1}$, when the reservoir receives the input $\mathbf{u}_j$.

4.2.3.3 Noise Training

Similar to our motivation for using Jacobian regularization, we can use noise added to the reservoir input to encourage the reservoir output to be insensitive to perturbations of the input. In noise training, we add a scaled noise vector $\sqrt{\beta_N} \boldsymbol{\gamma}(t)$ to each input to the reservoir during training:

$$\mathbf{u}_{in}(t) = \mathbf{u}(t) + \sqrt{\beta_N} \boldsymbol{\gamma}(t), \quad (4.18)$$

where $\gamma(t)$ is the noise vector and β_N is the noise variance. We generate the set of noise vectors, $\{\gamma_j\}$, to be added during training for each of the M components of $\gamma(t)$ at sample time t , sampling independently from a normal distribution with mean 0 and standard deviation 1. Following the process described earlier in this section, we obtain the noisy memory vectors, denoted by $\{\tilde{\mathbf{s}}_j\}$. We then train $\mathbf{W}$ such that $\mathbf{W}\tilde{\mathbf{s}}_j \approx \mathbf{v}_j$ by minimizing the noisy loss function:

$$\ell(\mathbf{W}) = \frac{1}{T_{train}} \sum_{j=1}^{T_{train}} \|\mathbf{W}\tilde{\mathbf{s}}_j - \mathbf{v}_j\|_2^2 + \sum_i \beta_i \text{Tr}(\mathbf{W}\mathbf{R}_i\mathbf{W}^\top), \quad (4.19)$$

where we include potential additional regularization terms as well. In practice, Tikhonov regularization is often used in conjunction with noise training. We minimize Eq. 4.19 as we did Eq. 4.8: by constructing a matrix $\tilde{\mathbf{S}}$, where the j^{th} column of $\tilde{\mathbf{S}}$ is $\tilde{\mathbf{s}}_j$, and solving the following linear system:

$$\mathbf{W} \left(\frac{1}{T_{train}} \tilde{\mathbf{S}}\tilde{\mathbf{S}}^\top + \sum_i \beta_i \mathbf{R}_i \right) = \frac{1}{T_{train}} \mathbf{V}\tilde{\mathbf{S}}^\top. \quad (4.20)$$

We note that while we can solve Eq. 4.20 using the same method used for Eq. 4.9, changing the noise scaling β_N requires us to re-compute the matrix $\tilde{\mathbf{S}}$ before we can solve Eq. 4.20 again. By contrast, changing the Tikhonov or Jacobian regularization parameter only requires that we re-scale the already-determined regularization matrices before we can solve Eq. 4.9 again. This re-computation generally makes tuning the noise scaling more computationally costly than the other regularization parameters.

4.2.4 Linearized Multi-Noise Training (LMNT)

We now introduce Linearized Multi-Noise Training (LMNT), pronounced as an initialism. LMNT introduces a penalty term to the loss function that penalizes the sum of the squares of the derivatives of the ML model output with respect to the K most-recent model inputs, averaged over roughly the entirety of the training data and scaled by an overall constant (which represents the approximated noise variance). This penalty term aims to approximate the effect of many independent realizations of small-amplitude input noise added to the unregularized least-squares loss function about a minimum of this loss function. The penalty term for LMNT can be written using the notation of Eq. 4.1 as

$$\ell_{LMNT} = \beta_L \frac{1}{T_{train} - K} \sum_{j=K}^{T_{train}-1} \sum_{k=j-K+1}^j \|\nabla_{\mathbf{s}} \mathbf{h}(\mathbf{s}_j) \nabla_{\mathbf{u}}(j, k)\|_F^2, \quad (4.21)$$

where β_L is the scaling constant representing input noise variance and $\nabla_{\mathbf{u}}(j, k)$ is the Jacobian matrix of $\mathbf{s}_j$ with respect to $\mathbf{u}_k$, where $k \leq j$:

$$\begin{aligned} \nabla_{\mathbf{u}}(j, k) &= \nabla_{\mathbf{s}} \mathbf{g}(\mathbf{u}_j, \mathbf{s}_{j-1}) \nabla_{\mathbf{s}} \mathbf{g}(\mathbf{u}_{j-1}, \mathbf{s}_{j-2}) \dots \\ &\quad \nabla_{\mathbf{s}} \mathbf{g}(\mathbf{u}_{k+1}, \mathbf{s}_k) \nabla_{\mathbf{u}} \mathbf{g}(\mathbf{u}_k, \mathbf{s}_{k-1}). \end{aligned} \quad (4.22)$$

Here $\{\mathbf{u}_j\}$ and $\{\mathbf{s}_j\}$ are the training inputs and the resulting memory vectors, as in Sec. 4.2.2. Though this training technique is motivated by noise training, it is deterministic. For the reservoir computer training, this results in a regularization matrix that may be easily re-scaled for efficient regularization parameter tuning (thus avoiding the drawback mentioned at the end of Sec. 4.2.3).

Next, we illustrate the connection between the LMNT penalty term and the effect of input

noise on a mean-squared-error loss function in the case of reservoir computing. Suppose that we compute the noisy memory vectors as described in Sec. 4.2.3.3 for P different noise realizations added to the input during training, where $P \gg 1$. That is, we compute the memory vectors for a particular set of noise vectors $\{\gamma_{i,1}\}$; then, beginning the computation again, we compute the memory vectors for another set of noise vectors $\{\gamma_{i,2}\}$; and so on, up to iteration P . We denote these memory vectors as $\{\tilde{\mathbf{s}}_{j,p}\}$, where the index $p = 1, 2, \dots, P$ denotes the noise realization used. The least-squares loss function without additional regularization is

$$\ell(\mathbf{W}) = \frac{1}{P} \sum_{p=1}^P \frac{1}{T_{train}} \sum_{j=0}^{T_{train}-1} \|\mathbf{W}\tilde{\mathbf{s}}_{j,p} - \mathbf{v}_j\|_2^2 \quad (4.23)$$

where we have added a factor of $1/P$ to normalize the loss function to have values similar to Eq. 4.8. We perform a bias-variance decomposition [128] on the loss function, obtaining:

$$\begin{aligned} & \frac{1}{P} \sum_{p=1}^P \frac{1}{T_{train}} \sum_{j=0}^{T_{train}-1} \|\mathbf{W}\tilde{\mathbf{s}}_{j,p} - \mathbf{v}_j\|_2^2 = \\ & \underbrace{\frac{1}{T_{train}} \sum_{j=0}^{T_{train}-1} \|\mathbf{W}\bar{\mathbf{s}}_j - \mathbf{v}_j\|_2^2}_{\text{bias}} + \underbrace{\frac{1}{P} \sum_{p=1}^P \frac{1}{T_{train}} \sum_{j=0}^{T_{train}-1} \|\mathbf{W}\mathbf{q}_{j,p}\|_2^2}_{\text{variance}} \end{aligned} \quad (4.24)$$

In Eq. 4.24, $\bar{\mathbf{s}}_j$ is the mean of the memory vector computed over P noise realizations for time index j , while $\mathbf{q}_{j,p} = \tilde{\mathbf{s}}_{j,p} - \bar{\mathbf{s}}_j$ is the deviation of $\tilde{\mathbf{s}}_{j,p}$ from $\bar{\mathbf{s}}_j$.

We next approximate Eq. 4.24 by assuming that: (a) $P \rightarrow \infty$; (b) β_N is small, allowing us to approximate to linear order in $\sqrt{\beta_N}$; and (c) due the reservoir computer's decaying memory, we can consider only the K most-recent additions of noise prior to time index j , where $K \leq T_{sync}$.

Considering assumption (a), we can write the variance term in Eq. 4.24 as:

$$\begin{aligned}
& \lim_{P \rightarrow \infty} \frac{1}{P} \sum_{p=1}^P \frac{1}{T_{train}} \sum_{j=0}^{T_{train}-1} \|\mathbf{W} \mathbf{q}_{j,p}\|_2^2 \\
&= \lim_{P \rightarrow \infty} \frac{1}{T_{train}} \sum_{j=0}^{T_{train}-1} \text{Tr} \left[\mathbf{W} \left(\frac{1}{P} \sum_{p=1}^P \mathbf{q}_{j,p} \mathbf{q}_{j,p}^\top \right) \mathbf{W}^\top \right] \\
&= \frac{1}{T_{train}} \sum_{j=0}^{T_{train}-1} \text{Tr}[\mathbf{W} \Sigma_j \mathbf{W}^\top],
\end{aligned} \tag{4.25}$$

where Σ_j is the covariance of $\mathbf{q}_{j,p}$, $\Sigma_j = \lim_{P \rightarrow \infty} P^{-1} \sum_{p=1}^P \mathbf{q}_{j,p} \mathbf{q}_{j,p}^\top$. Next considering assumptions (b) and (c), we write $\tilde{\mathbf{s}}_{j,p}$ as:

$$\tilde{\mathbf{s}}_{j,p} \approx \mathbf{s}_j + \sqrt{\beta_N} \sum_{k=j-K+1}^j \nabla_{\mathbf{u}}(j, k) \gamma_{k,p} + \mathcal{O}(\beta_N), \tag{4.26}$$

where $\mathbf{s}_j$ is the feature vector for noiseless training and $\nabla_{\mathbf{u}}(j, k)$ is computed from Eq. 4.21 with $\mathbf{g} = \mathbf{g}_o$ from Eq. 4.6. Then, since $\gamma_{k,p}$ is chosen from a distribution with mean zero,

$$\lim_{P \rightarrow \infty} \bar{\mathbf{s}}_j = \mathbf{s}_j + \mathcal{O}(\beta_N), \tag{4.27}$$

and for large P,

$$\mathbf{q}_{j,p} \approx \sqrt{\beta_N} \sum_{k=j-K+1}^j \nabla_{\mathbf{u}}(j, k) \gamma_{k,p} + \mathcal{O}(\beta_N). \tag{4.28}$$

We use Eq. 4.27 to approximate the bias term in Eq. 4.24, and Eq. 4.28 to approximate the variance term. We assume that the $\mathcal{O}(\beta_N)$ term in Eq. 4.27, which depends on j , has fluctuations that are independent of the fluctuations in $\mathbf{W} \mathbf{s}_j - \mathbf{v}_j$, and that the mean over j of $\mathbf{W} \mathbf{s}_j - \mathbf{v}_j$ is

approximately zero, as it should be for least-squares fitting. Thus we assume that changing $\bar{\mathbf{s}}_j$ to $\mathbf{s}_j$ in Eq. 4.24 makes a change that is small compared to β_N , due to the cancellation of the $\mathcal{O}(\beta_N)$ terms in the expression $\|\mathbf{W}\bar{\mathbf{s}}_j - \mathbf{v}_j\|_2^2 - \|\mathbf{W}\mathbf{s}_j - \mathbf{v}_j\|_2^2 \approx 2\mathbf{W}\mathcal{O}(\beta_N)(\mathbf{W}\mathbf{s}_j - \mathbf{v}_j)^\top$ after summing over j .

Next, since the noise vectors $\gamma_{j,p}$ are independent of each other and each has covariance $\mathbf{I}$, we approximate Σ_j in Eq. 4.25 as

$$\Sigma_j = \sum_{k=j-K+1}^j \nabla_{\mathbf{u}}(j, k) (\beta_N \mathbf{I}) \nabla_{\mathbf{u}}(j, k)^\top = \beta_N \sum_{k=j-K+1}^j \nabla_{\mathbf{u}}(j, k) \nabla_{\mathbf{u}}(j, k)^\top. \quad (4.29)$$

Combining Eqs. 4.25 and 4.22, we obtain the LMNT regularization function for the reservoir computer:

$$\beta_L \frac{1}{T_{train} - K} \sum_{j=K}^{T_{train}-1} \sum_{k=j-K+1}^j \|\mathbf{W} \nabla_{\mathbf{u}}(j, k)\|_F^2, \quad (4.30)$$

where we have replaced β_N by β_L . Notice that Eq. 4.30 agrees with Eq. 4.21 in the case that $\mathbf{h}(\mathbf{s}) = \mathbf{W}\mathbf{s}$. We restrict to $j \geq K$ so that $k \geq 1$ and, thus, $k-1 \geq 0$ in Eq. 4.22. In Eq. 4.22, $\mathbf{u}_k$ and $\mathbf{s}_k$ are computed without any addition of noise, making this regularization deterministic. We use Eq. 4.15 for $\nabla_{\mathbf{u}} \mathbf{g}_o$, and we compute $\nabla_{\mathbf{s}} \mathbf{g}_o$ using the open-loop memory vector evolution

described by Eq. 4.6 and shown in Fig. 4.2:

$$\nabla_{\mathbf{s}\mathbf{g}_o}(\mathbf{u}_j, \mathbf{s}_{j-1}) = \begin{bmatrix} \mathbf{0}_{(1+M) \times (1+M+2N)} \\ \nabla_{\mathbf{s}\mathbf{g}_r}(\mathbf{s}_{j-1}, \mathbf{u}_j) \\ \text{diag}(2\mathbf{r}_j) \nabla_{\mathbf{s}\mathbf{g}_r}(\mathbf{s}_{j-1}, \mathbf{u}_j) \end{bmatrix}, \quad \text{where} \quad (4.31)$$

$$\begin{aligned} \nabla_{\mathbf{s}\mathbf{g}_r}(\mathbf{s}_{j-1}, \mathbf{u}_j) = & \alpha \text{diag}(\mathbf{d}(\mathbf{r}_{j-1}, \mathbf{u}_j)) \begin{bmatrix} \mathbf{0}_{N \times (1+M)}, & \mathbf{A}, & \mathbf{0}_{N \times N} \end{bmatrix} \\ & + \begin{bmatrix} \mathbf{0}_{N \times (1+M)}, & (1 - \alpha)\mathbf{I}_{N \times N}, & \mathbf{0}_{N \times N} \end{bmatrix} \end{aligned} \quad (4.32)$$

and where $\text{diag}(\dots)$ and $\mathbf{d}(\mathbf{r}_{j-1}, \mathbf{u}_j)$ are the same as in Eqs. 4.16 and 4.17, respectively. When computed using the inputs and internal reservoir states during training, the resulting regularization matrix is:

$$\mathbf{R}_L = \frac{1}{T_{train} - K} \sum_{j=K}^{T_{train}-1} \left[\sum_{k=j-K+1}^j \nabla_{\mathbf{u}}(j, k) \nabla_{\mathbf{u}}(j, k)^\top \right]. \quad (4.33)$$

In the case where $K = 1$, this regularization matrix is identical to the Jacobian regularization matrix, $\mathbf{R}_J$. For all results in this chapter, we will compute the LMNT regularization using $K = 4$; we justify this choice of K , show results for other K values, and discuss general considerations for selecting K in Sec. 4.5. We also note that the LMNT regularization is computed here using approximately the same number of training samples as is used to train the reservoir. In Sec. 4.6.1, we discuss computing the LMNT regularization with a greatly reduced number of training samples, while in Sec. 4.6.2, we compare prediction stability and accuracy when the total amount of training samples is greatly reduced.

4.2.5 Prediction and Metrics

Once we have determined $\mathbf{W}$, we are ready to begin prediction. Prior to switching on the feedback loop shown in Fig. 4.2, we reset the internal reservoir state to $\mathbf{0}$, and then input a short sequence of measurements, $\{\mathbf{u}_{true}(t)\}$, from the system we intend to predict. More precisely, assume that $\mathbf{u}_{true}(t)$ is known up to time $T_{init}\Delta t$. To re-synchronize the reservoir to the true system trajectory, we begin at time $(T_{init} - T_{sync})\Delta t$, inputting $\mathbf{u}_{in}((T_{init} - T_{sync})\Delta t) = \mathbf{u}_{true}((T_{init} - T_{sync})\Delta t)$. We iterate this process from $t = (T_{init} - T_{sync})\Delta t$ to $t = T_{init}\Delta t$, recording the resulting reservoir states until we obtain $\mathbf{r}(T_{init}\Delta t)$. We then compute our prediction for the system state at time $(T_{init} + 1)\Delta t$ as $\mathbf{u}_{out}((T_{init} + 1)\Delta t) = \mathbf{W}\mathbf{s}(T_{init}\Delta t)$, where $\mathbf{s}(T_{init}\Delta t) = f(\mathbf{r}(T_{init}\Delta t), \mathbf{u}_{true}(T_{init}\Delta t))$ (see Eq. 4.3). We then activate the feedback loop shown in Fig. 4.2 so that $\mathbf{u}_{in}(t) = \mathbf{u}_{out}(t)$ for $t \geq (T_{init} + 1)\Delta t$. We compute $\mathbf{u}_{out}((T_{init} + 2)\Delta t) = \mathbf{W}\mathbf{s}((T_{init} + 1)\Delta t)$, feed back this prediction as input, and so on, until we have reached our desired prediction time, $t_{pred} = (T_{init} + T_{pred})\Delta t$, where T_{pred} is a positive integer. The closed-loop reservoir dynamics can be expressed as a single evolution function $\mathbf{s}(t) = \mathbf{g}_c(\mathbf{s}(t - \Delta t))$, where

$$\mathbf{g}_c(\mathbf{s}(t - \Delta t)) = \begin{bmatrix} 1 \\ \mathbf{W}\mathbf{s}(t - \Delta t) \\ \mathbf{g}_s(\mathbf{s}(t - \Delta t)) \\ (\mathbf{g}_s(\mathbf{s}(t - \Delta t)))^2 \end{bmatrix} \quad (4.34)$$

and

$$\begin{aligned} \mathbf{g}_s(\mathbf{s}(t - \Delta t)) = (1 - \alpha) & \begin{bmatrix} \mathbf{0}_{N \times (1+M)}, & \mathbf{I}_{N \times N}, & \mathbf{0}_{N \times N} \end{bmatrix} \mathbf{s}(t - \Delta t) \\ & + \alpha \tanh \left(\begin{bmatrix} \mathbf{0}_{N \times 1}, & \mathbf{B}\mathbf{W}, & \mathbf{A}, & \mathbf{0}_{N \times N} \end{bmatrix} \mathbf{s}(t - \Delta t) + \mathbf{C} \right). \end{aligned} \quad (4.35)$$

The right side of Eq. 4.35 is obtained by substituting $\mathbf{u}(t) = \mathbf{W}\mathbf{s}(t + \Delta t)$ into the right side of Eq. 4.7.

When evaluating the performance of a prediction from our machine learning model, we are interested in:

1. For what duration of time is the near-term prediction approximately valid? (In other words, how long does the near-term prediction error remain below some chosen threshold?)
2. Is the long-term climate “stable”? (In other words, does the ML model prediction remain within approximately the same region of space as the training data, or does it escape to some other region?)
 - Due to the chaotic nature of the systems we are interested in, we expect predictions to be “unstable” in the sense that the predicted trajectory will exponentially diverge from the true trajectory due to any error in the initial condition, no matter how small the error is in the model. We distinguish this type of instability from climate instability.
3. If the prediction is stable, are its statistical properties, or “climate”, similar to that of the unknown dynamical system?

To evaluate each of these criteria, we use the following metrics:

1. **Prediction Valid Time:** The prediction valid time is computed as

$$VT = \min_{T_{init}\Delta t \leq t \leq (T_{init}+T_{pred})\Delta t} \left\{ t \mid \frac{\|\mathbf{u}_{out}(t) - \mathbf{u}_{true}(t)\|_2}{\bar{E}} > \epsilon_{VT} \right\} - \Delta t. \quad (4.36)$$

Here, ϵ_{VT} is the valid time error threshold, and $\bar{E}$ is the average error between true system states computed from the training data as the mean of $\|\mathbf{u}_j - \mathbf{u}_k\|_2$ over $0 \leq j < k \leq T_{train}$.

In all of our tests, we choose $\epsilon_{VT} = 0.2$, representing a 20% error in the prediction.

2. **Climate Stability:** It is often the case that ML model predictions will diverge exponentially from the region of the true attractor, eventually resulting in numerical overflow in the floating-point computation. These predictions can be easily identified as unstable. However, grossly inaccurate predictions of the true system climate may also result when the system settles into some other region of state space significantly removed from the true attractor. For the purposes of this work, we also classify the latter situation as instability. We desire to formulate a single metric that can be used to signal the presence of instability of either of the two types described above. We determine if such a prediction is stable by computing the mean of the normalized “map error” over the entire prediction. This metric presumes a diagnostic scenario where we know the true evolution equations for $\mathbf{u}(t)$ and that $\mathbf{u}(t)$ is the entire state of the system to be predicted. The map error at time t is the norm of the difference between $\mathbf{u}_{out}(t)$ and the result of evolving the true evolution equations for time Δt from initial condition $\mathbf{u}_{out}(t - \Delta t)$. The normalized map error is

$$\epsilon_{map}(t) = \frac{\|\mathbf{u}_{out}(t) - \mathbf{F}(\mathbf{u}_{out}(t - \Delta t), t - \Delta t, t)\|_2}{\bar{E}_{map}}. \quad (4.37)$$

Here, $\mathbf{F}(\mathbf{u}(t_0), t_0, t_f)$ is a function that integrates the true evolution equations with an initial condition $\mathbf{u}(t_0)$ from t_0 to t_f . We have normalized the map error using the mean error of the persistence forecast computed from the training data,

$$\bar{E}_{map} = \overline{\|\mathbf{u}_{j+1} - \mathbf{u}_j\|_2}, \quad (4.38)$$

where the horizontal bar $\overline{(\dots)}$ denotes a mean computed over the training time indices from $j = 0$ to $j = T_{train} - 1$.

We denote by $\bar{\epsilon}_{map}$ the mean of $\epsilon_{map}(t)$ for $T_{init}\Delta t \leq t \leq (T_{init} + T_{pred})\Delta t$. We choose a threshold for the mean normalized map error that, if exceeded, characterizes the prediction as unstable. We choose this threshold by first producing predictions using an ensemble of reservoir realizations, training data sets, and testing data sets that are each trained using regularization parameter values from a logarithmic grid of parameter values. We then compute a histogram of the mean map error values from predictions that have not resulted in numerical overflow. In all cases we test, we are able to clearly see a multimodal distribution, with those points with low mean map error corresponding to stable predictions and those with high mean map error corresponding to unstable predictions. We show an example of a histogram of mean map error values in Sec. 4.3.2. From this histogram, we have chosen $\bar{\epsilon}_{map} = 1.0$ as our stability cutoff. To measure the maximum deviation from the true evolution equations during prediction, we also compute the maximum map error

for each prediction,

$$\epsilon_{map}^{max} = \max_{T_{init}\Delta t \leq t \leq (T_{init}+T_{pred})\Delta t} \epsilon_{map}(t). \quad (4.39)$$

3. **Climate Similarity:** To compare the climate of the true system and the stable predictions generated from our reservoir computer, we use the power spectral density (PSD), $S_{uu}(f)$. We estimate the PSD of a particular element of $\mathbf{u}_{true}(t)$ or $\mathbf{u}_{out}(t)$ using a smoothed periodogram. For a 1-dimensional time series data set $\{u_j\} = \{u_0, u_1, \dots, u_{J-2}, u_{J-1}\}$, where $J = T_{pred}/\Delta t$ and $u_j = u((T_{init} + j + 1)\Delta t)$, this estimate is computed using Welch's method [129].

4.3 Results

4.3.1 Kuramoto-Sivashinsky Equation

The test system that we will use to examine near-term accuracy and long-term climate stability is the Kuramoto-Sivashinsky (KS) equation [94, 95] with periodic boundary conditions:

$$\frac{\partial y(x, t)}{\partial t} + y(x, t) \frac{\partial y(x, t)}{\partial x} + \frac{\partial^2 y(x, t)}{\partial x^2} + \frac{\partial^4 y(x, t)}{\partial x^4} = 0, \quad (4.40)$$

where $y(x, t) = y(x + L, t)$ for a chosen spatial periodicity length L . For particular choices of L , the Kuramoto-Sivashinsky equation exhibits chaotic dynamics. To obtain the dynamical system states that we will use as our training and testing data, and to evaluate the map error as discussed in Sec. 4.2.5, we simulate Eq. 4.40 on a spatial grid consisting of 64 grid points equally-spaced

at intervals of $\Delta x = L/64$ using the ETRK4 method [97, 130] to integrate the system at a time step of $\Delta t = 0.25$. The resulting discretized system state is:

$$\mathbf{y}(t) = [y(0, t), y(\Delta x, t), \dots, y((M - 1)\Delta x, t)]^\top, \quad (4.41)$$

where $M = 64$. For each time series used for training and testing, we choose a different random initial condition, where each coordinate of $\mathbf{u}(0)$ is sampled from a uniform distribution on the interval $[-0.6, 0.6]$, with the spatial mean value of $\mathbf{u}(0)$ adjusted to be 0. We then integrate the equation and discard the states obtained before $t = 500$ to avoid the effect of any transient dynamics that are not on the true attractor. Our numerical integrations of Eq. 4.40 yield chaotic trajectories and (as they should) preserve the zero spatial average of $\mathbf{y}(t)$ for $t > 0$. We standardize $\mathbf{y}(t)$ as follows to form $\mathbf{u}(t)$ that is used for training and testing the reservoir. To each coordinate of $\mathbf{y}(t)$, we apply a linear transformation so that the resulting coordinate of $\mathbf{u}(t)$ has mean 0 and standard deviation 1 over the training time period.

When discussing the prediction valid time and the prediction spectra, we will do so in units of the Lyapunov time, corresponding to the inverse of the largest positive Lyapunov exponent for typical orbits of the chaotic attractor of Eq. 4.40. The Lyapunov time is characteristic time over which errors in the true chaotic system will experience an e -fold growth. For a periodicity length $L = 22$, we have computed the Lyapunov time to be $t_{Lyap} = 20.83$ using the Bennetin algorithm [131].

We have also done tests with other systems besides the Kuramoto-Sivashinky equations (e.g., the chaotic Lorenz '63 model [2]). We do not report these results here, as they result in conclusions that coincide with what follows from our tests on Eq. 4.40.

4.3.2 Prediction Test Results

We now present the result of our predictions test on the Kuramoto-Sivashinky equation. In order to test a scenario where climate stability is challenging, we have used a reservoir with 500 nodes (see Table 4.1). Though it would be computationally feasible to improve stability with a larger reservoir in this case, it might not be in scenarios with higher dimensional dynamics. In summary, we find from our tests that:

- Reservoir computers trained with no regularization or only Jacobian regularization produce predictions that are always observed to be unstable and produce predictions that have a very low valid time.
- Reservoir computers trained with only Tikhonov regularization produce predictions that are only sometimes observed to be stable, while otherwise producing predictions with a very low valid time.
- Reservoir computers trained with Jacobian and Tikhonov regularization, noise training and Tikhonov regularization, and LMNT and Tikhonov regularization are observed to always produce stable predictions if the regularization hyperparameters are chosen large enough. The latter two regularization methods result in the best prediction valid times, as well as the best mean and maximum map errors.
- Stable predictions from reservoir computers trained with only Tikhonov regularization have an average PSD that appears to match the PSD of the true system climate somewhat well, while predictions from the methods that always produce stable predictions have an average PSD that near-perfectly matches the PSD of the true system climate.

- From our variation of the regularization parameters, we find the best prediction valid time performance near the boundary between climate stability and partial climate instability, indicating that careful tuning of the regularization parameter value is needed during optimization.

4.3.2.1 Climate Stability and Valid Time

Reservoir Hyperparameters	
Number of nodes (N)	500
Average degree ($\langle d \rangle$)	3
Spectral radius (ρ)	0.6
Input weight (σ)	0.1
Input bias (θ)	0.1
Leaking rate (α)	1.0

Table 4.1: Reservoir computer hyperparameter values used for all prediction tests.

Training and Prediction Parameters	
T_{sync}	100 ($1.2 t_{Lyap}$)
T_{train}	20,000 ($240 t_{Lyap}$)
T_{pred}	Valid Time Tests: 2000 ($24 t_{Lyap}$) Climate Stability and Reproduction Tests: 16,000 ($192 t_{Lyap}$)
Number of Reservoir Ensemble Members	20
Number of Training Data Sets	10
Number of Testing Data Sets	Prediction Valid Time Tests: 35 Climate Stability and Reproduction Tests: 5
Total Number of Predictions	Prediction Valid Time Tests: 7000 Climate Stability and Reproduction Tests: 1000

Table 4.2: Parameters used for all reservoir training and tests.

Regularization Parameter Search Grids	
Regularization Type	Search Grid
Tikhonov (β_T)	0 and 10^l for $l \in \{-18, -17.5, \dots, -4.5, -4\}$
Jacobian (β_J)	10^l for $l \in \{-8, -7.8, \dots, -4.2, -4\}$
Noise Training (β_N) and LMNT (β_L)	10^l for $l \in \{-8, -7.8, \dots, -6.2, -6\}$

Table 4.3: The regularization parameters that are searched over for each reservoir prediction test.

Prediction Test Results					
Regularization Type	Regularization Parameters	Fraction of Stable Predictions	Median Valid Time (t_{Lyap})	Median $\bar{\epsilon}_{map}$	Median ϵ_{map}^{max}
None	N/A	0/1000	0.05 ± 0.01	∞	∞
Jacobian Only	$\beta_J = 10^{-7}$	0/1000	0.25 ± 0.01	∞	∞
Tikhonov Only	$\beta_T = 10^{-6}$	565/1000	0.71 ± 0.02	$(6.46 \pm 0.22) \times 10^{-1}$	5.23 ± 0.31
Noise Training Only	$\beta_N = 10^{-6.2}$	997/1000	3.67 ± 0.06	$(4.05 \pm 0.03) \times 10^{-3}$	$(2.81 \pm 0.05) \times 10^{-2}$
LMNT ($K = 4$) Only	$\beta_L = 10^{-6.4}$	990/1000	3.94 ± 0.05	$(3.69 \pm 0.03) \times 10^{-3}$	$(2.65 \pm 0.04) \times 10^{-2}$
Jacobian and Tikhonov	$\beta_J = 10^{-5.4}$ $\beta_T = 10^{-8.5}$	1000/1000	2.88 ± 0.02	$(9.16 \pm 0.04) \times 10^{-3}$	$(5.82 \pm 0.06) \times 10^{-2}$
Noise Training and Tikhonov	$\beta_N = 10^{-7.4}$ $\beta_T = 10^{-14.5}$	1000/1000	4.24 ± 0.04	$(2.77 \pm 0.02) \times 10^{-3}$	$(2.02 \pm 0.04) \times 10^{-2}$
LMNT ($K = 4$) and Tikhonov	$\beta_L = 10^{-7.4}$ $\beta_T = 10^{-16.5}$	1000/1000	4.27 ± 0.04	$(2.75 \pm 0.02) \times 10^{-3}$	$(2.03 \pm 0.04) \times 10^{-2}$

Table 4.4: Reservoir computer prediction results using different types of regularization. Bold text marks the best performance for the corresponding metric. In every case, regularization parameter values are chosen to maximize the fraction of stable predictions produced by an ensemble of reservoirs over an ensemble of training and testing data sets. If multiple regularization parameter values are found to produce the same fraction of stable predictions, then we choose the regularization parameter(s) from that subset that maximize the median prediction valid time. The $\pm$ error bounds indicate the maximum of the upper and lower 95% confidence intervals for the median [1]. In the case of the valid time, we enlarge this interval slightly to account for the Δt discretization of our predictions.

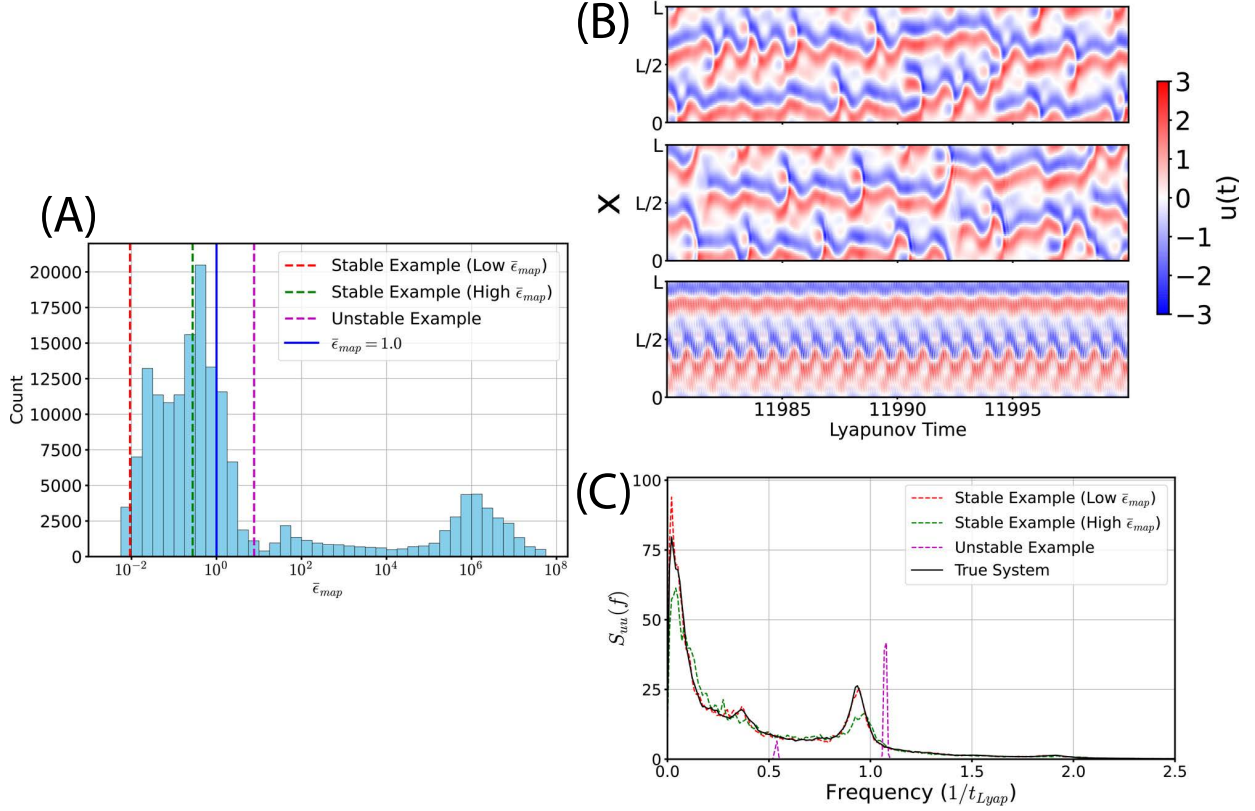


Figure 4.3: Comparing Predictions with Increasing $\bar{\epsilon}_{map}$ Values. In this figure, panel (A) shows a histogram of the mean map error values of predictions generated from reservoir computers trained with Jacobian and Tikhonov regularization using a logarithmic grid of regularization parameter values (grid given in Table 4.3). In this panel, the solid blue line marks the $\bar{\epsilon}_{map}$ climate stability threshold, while the dashed red, green, and magenta lines denote the mean map error value for three example predictions: one is a stable prediction with a low $\bar{\epsilon}_{map}$, one is stable with a high $\bar{\epsilon}_{map}$, and one is unstable, respectively. These predictions are obtained using a long $T_{pred} = 1,000,000$ ($12,000 t_{Lyap}$). The dynamics at the end of these predictions are displayed in panel (B), with the stable example prediction with a low $\bar{\epsilon}_{map}$ on the top, the stable example prediction with a high $\bar{\epsilon}_{map}$ in the middle, and the unstable example prediction on the bottom. Panel (C) displays the computed power spectral density of $u_{out,1}(t)$ and $u_{true,1}(t)$ for each of these predictions and the true test data. When computing the power spectral density using Welch's method, we used a window with 2^{13} samples ($98.304 t_{Lyap}$) and no overlap between windows.

We simulate the KS equation using reservoir computers with the hyperparameters listed in Table 4.1. For each of our tests, we train an ensemble of random reservoir realizations, each generated using a different random seed for the input coupling matrix $\mathbf{B}$ and the reservoir network adjacency matrix $\mathbf{A}$, using a training time series data set drawn from an ensemble of training

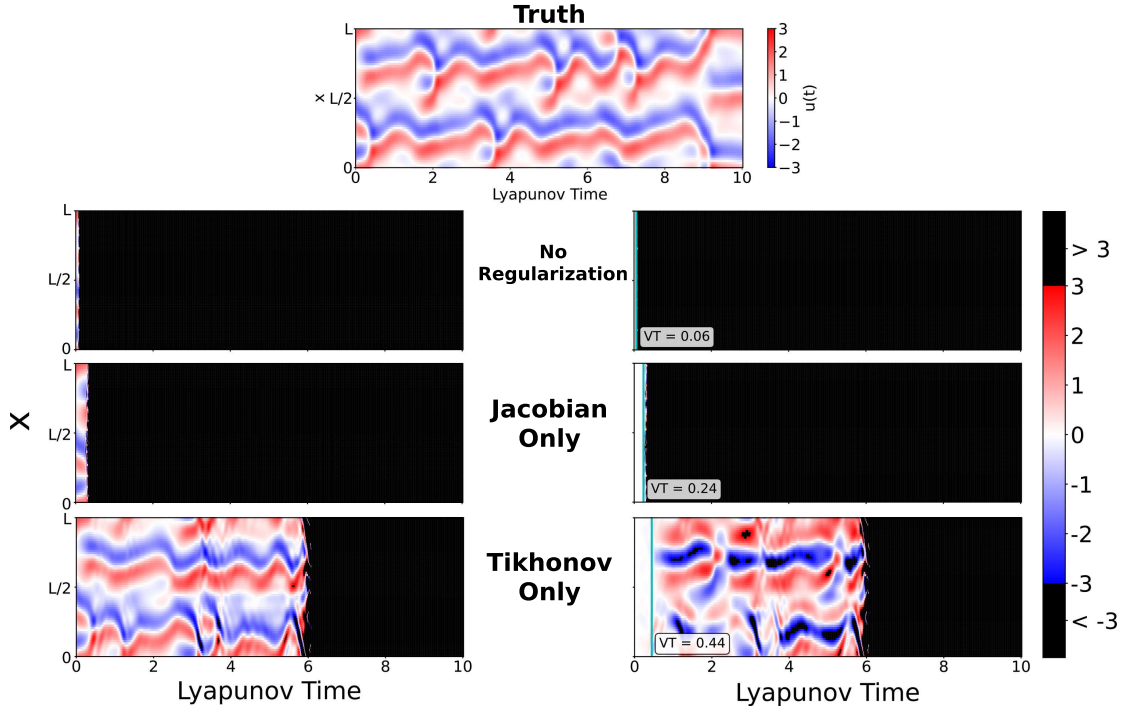


Figure 4.4: Unstable Predictions of the KS Equation. The top panel shows a testing time series $\mathbf{u}_{true}(t)$ generated from the true KS equations dynamics, while the bottom panels show results from a reservoir computer trained with the regularization type displayed between the left and right panels. Panels on the left show the reservoir prediction, $\mathbf{u}_{out}(t)$, while those on the right show the difference between the reservoir prediction and the true evolution $\mathbf{u}_{true}(t)$. In all panels, the horizontal axis denotes the time measured in Lyapunov times based on the largest positive Lyapunov exponent of the true system, while the vertical axis denotes the spatial coordinate. In the top pa and in the panels on the left, the color denotes the values of $\mathbf{u}_{true}(t)$ and $\mathbf{u}_{out}(t)$, respectively, while in panels on the right, the color denotes the difference, $\mathbf{u}_{out}(t) - \mathbf{u}_{true}(t)$, between the prediction and true dynamics. An entire prediction at a particular time being colored black indicates that this prediction has become unstable and left the region of the true attractor. In the right panels, the cyan line denotes the corresponding prediction valid time.

data sets generated using Eq. 4.40, each with a different random initial condition. Each trained reservoir makes predictions on an ensemble of testing time series data sets, which again each have a different random initial condition. The duration and number of each of these training and testing time series data sets, along with the number of reservoir realizations tested, can be found in Table 4.2. Panel (A) in Fig. 4.3 shows a histogram of the mean map error values of predictions generated from a reservoir computer trained using Jacobian and Tikhonov regularization using

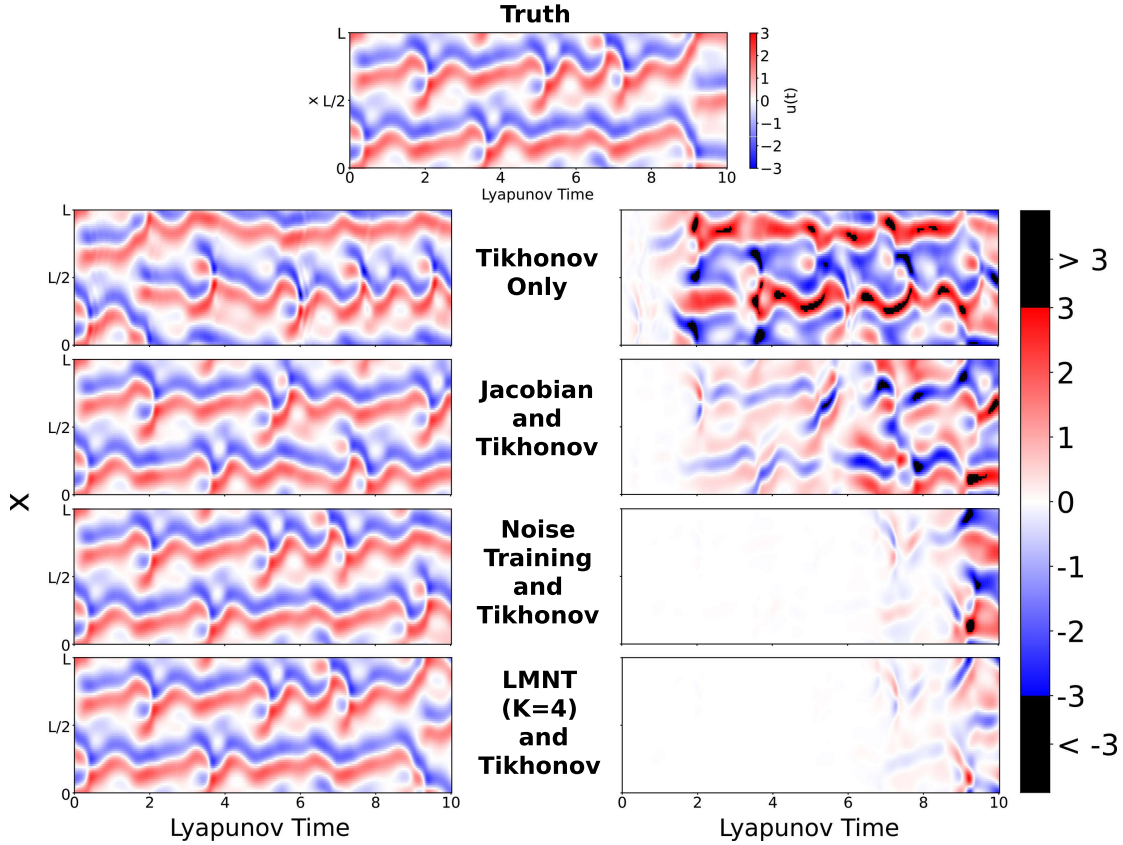


Figure 4.5: Stable Predictions of the KS Equation. The top panel shows a testing time series $\mathbf{u}_{true}(t)$ generated from the true KS equations dynamics, while the bottom panels show results from a reservoir computer trained with the regularization type displayed between the left and right panels. Panels on the left show the reservoir prediction, $\mathbf{u}_{out}(t)$, while those on the right show the difference between the reservoir prediction and the true evolution $\mathbf{u}_{true}(t)$. In all panels, the horizontal axis denotes the time measured in Lyapunov times based on the largest positive Lyapunov exponent of the true system, while the vertical axis denotes the spatial coordinate. In the top panel and in the panels on the left, the color denotes the values of $\mathbf{u}_{true}(t)$ and $\mathbf{u}_{out}(t)$, respectively, while in panels on the right, the color denotes the difference, $\mathbf{u}_{out}(t) - \mathbf{u}_{true}(t)$, between the prediction and true dynamics. Black coloring marks where the prediction (left panels) or prediction difference (right panels) is outside the range given by the color bar.

the regularization parameter values contained in Table 4.3. The histogram is multimodal, with the clearest division between cases with $\bar{\epsilon}_{map} < 10$ and $\bar{\epsilon}_{map} > 10$. Some cases with $1 < \bar{\epsilon}_{map} < 10$ stay in the general vicinity of the attractor but do not reproduce its climate, as illustrated by the 3rd example in Fig. 4.3; we have chosen to classify these as unstable. Our choice for a cutoff of $\bar{\epsilon}_{map} = 1$ between cases we classify as stable or unstable is meant to ensure that those we classify

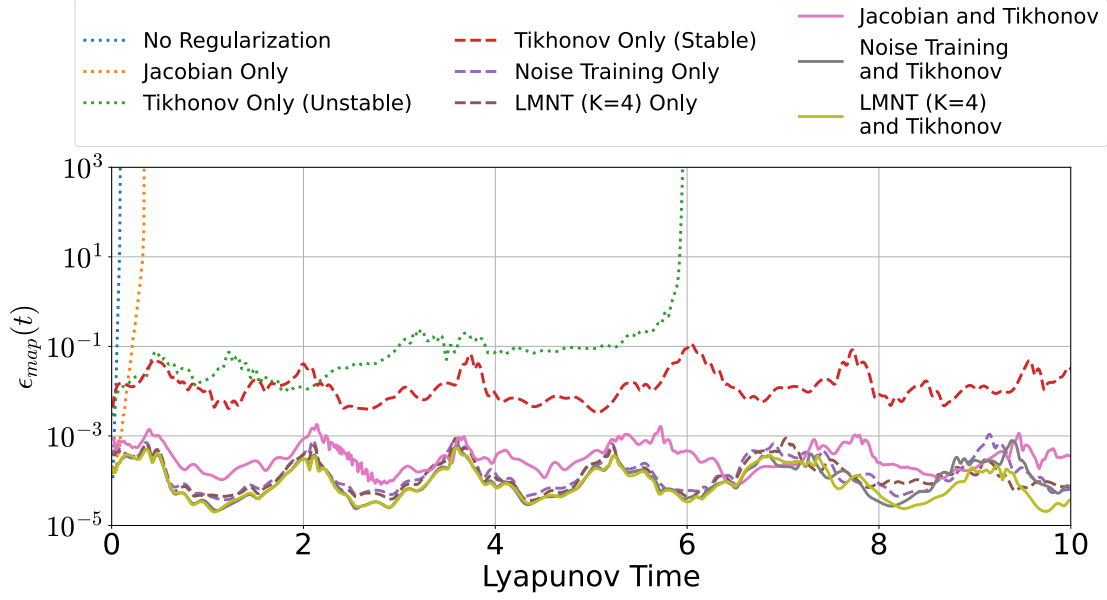


Figure 4.6: Map Error over Time During Reservoir Predictions. This figure shows the map error ϵ_{map} during each of the unstable predictions shown in Figs. 4.4 and (dotted lines), stable predictions shown in Fig. 4.5 from reservoirs trained with a single regularization type (dashed lines), and stable predictions shown in Fig. 4.5 from reservoirs trained with multiple regularization types (solid lines).

as stable do reasonably reproduce the true system climate, as illustrated by the first 2 examples in Fig. 4.3. We see that the predictions categorized as stable appear to have chaotic behavior at the end of the prediction period and have a power spectral density that matches that of the true system well, while the prediction categorized as unstable has a periodic behavior by the end of the prediction period and, as such, has a power spectral density very different from that of the true system. We additionally note that the stable prediction with a low mean map error has a power spectral density that is closer to the truth than that with a high mean map error, indicating that this metric is useful for determining how closely the prediction climate will match that of the true system.

Table 4.4 shows the results of reservoir computer predictions made using different regularization methods, ordered from worst to best performance. Figures 4.4 and 4.5 show examples

of unstable and stable predictions, respectively. Each figure uses a particular reservoir computer, training time series data set, and testing time series data set drawn from our ensemble. We note that the reservoir computer and training time series data set used in each figure are different, while the testing time series data set is the same. In addition, we want to emphasize that the predictions shown in Fig. 4.5 are observed to be stable for the entire long prediction period of $192 t_{Lyap}$. Figure 4.6 shows the map error $\epsilon_{map}(t)$ during each of the predictions shown in Figs. 4.4 and 4.5. We observe that the map error of the unstable predictions curves increases, while the map error in the stable predictions remains small throughout the entire prediction period ($192 t_{Lyap}$). We note that the map error of the predictions made using noise training, LMNT regularization, noise training and Tikhonov regularization, and LMNT and Tikhonov regularization are very similar for $t_{pred} < 7 t_{Lyap}$.

We see that reservoir computers trained without regularization or using only Jacobian regularization produce predictions with a very low median valid time (less than 30% of a Lyapunov time) and are always unstable. Reservoir computers trained with only Tikhonov regularization have a somewhat longer, but still poor prediction valid time, while just over half of the predictions remain stable. Furthermore, they require a high amount of regularization to reach this fraction of stability. When the reservoir computers are trained using only noise training or only LMNT regularization, the vast majority of the predictions are observed to be stable. Reservoir computers trained with Tikhonov regularization in addition to either Jacobian regularization, Noise Training, or LMNT produce predictions that are always observed to be stable and appear to remain stable for arbitrarily long times with appropriately chosen regularization amounts. Prediction valid times using the combination of Jacobian and Tikhonov regularization, however, are substantially lower than for those that are trained with the other stable regularization types with or

without Tikhonov regularization added. The highest median valid times are, to within uncertainty, obtained by reservoir computers trained with noise training and Tikhonov regularization or with LMNT and Tikhonov regularization.

4.3.2.2 Prediction Climate

As discussed, results for some regularization methods appear to be stable for arbitrarily long times. These are thus potential candidates for climate predictions. In Fig. 4.7, we display the PSD computed from time series data from the true system and from predictions made using these long-time stable results. We find that predictions using reservoir computers trained with only Tikhonov Regularization (using only those orbits that remain stable) give a useful, but imperfect, replication of the PSD of the true system climate. We observe in Fig. 4.7 that all of the other regularization methods that resulted in stable predictions are able to capture the PSD of the climate such that the 95% confidence interval in the computed PSD for the true system and for the ensemble of predictions overlap at all observed frequency values.

4.3.2.3 Regularization Optimization

In this section, we show how the optimal regularization parameter values used to produce the results shown in Figs. 4.4-4.7 and in Table 4.4 are obtained. Figure 4.8 displays, for two of the regularization methods, the fraction of stable predictions and the median prediction valid time for predictions made using reservoir computers trained with regularization parameter values distributed over a logarithmic grid. For each method, among all of the tested regularization parameter values, we choose those resulting in predictions that are always stable and have the

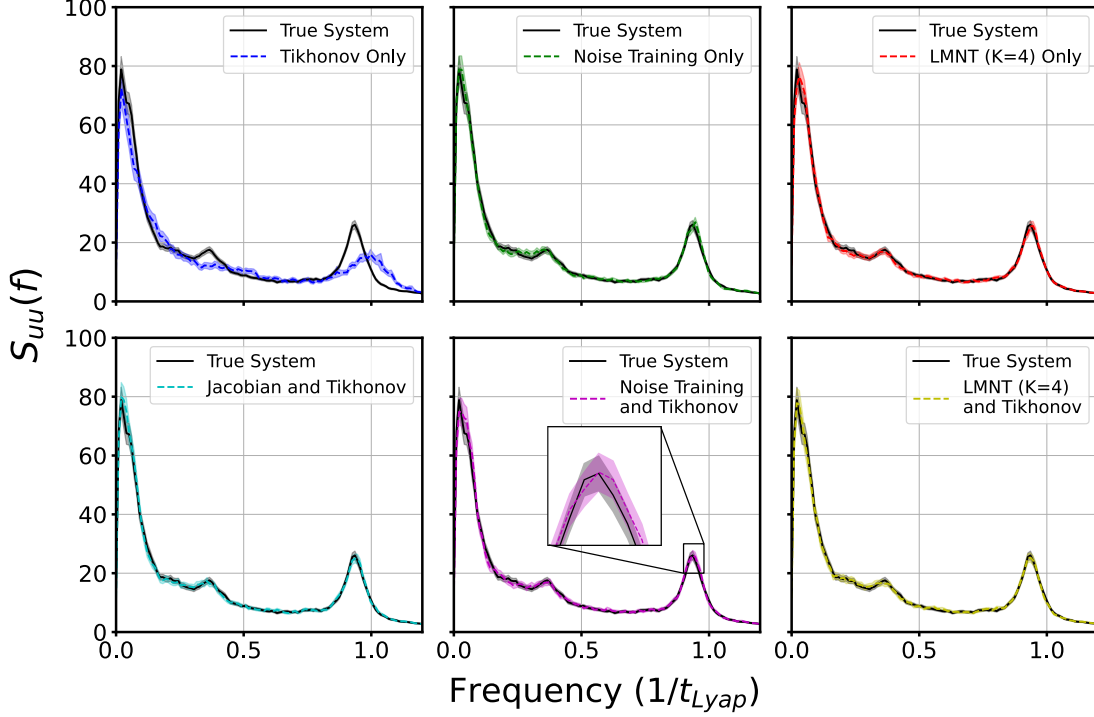


Figure 4.7: Average Predicted PSD Compared to True PSD. The true KS equation PSD is computed over a single trajectory of duration $T_{pred} = 10,000,000$ ($120,000 t_{Lyap}$), while the prediction PSD is computed as the average over predictions of duration $T_{pred} = 16,000$ ($192 t_{Lyap}$) produced by the ensemble of reservoirs, training data sets, and testing initial conditions used to test the prediction climate. When computing the PSD with Welch’s method, we used a window with 2^{13} samples ($98.304 t_{Lyap}$) and no overlapping windows. The grey filled region surrounding the PSD curve for the true system shows the 95% confidence interval in the mean spectra computed over these windows. The color-filled region surrounding the prediction PSD curves shows the 95% confidence interval in the mean spectra computed over these windows and over the different predictions. We note that for some frequency values, the line width of the plotted PSD curve is wider than the corresponding shaded region. The cutout in the bottom-center panel shows a magnified view of the PSD and 95% confidence interval between frequencies of 0.9 and $0.98 1/t_{Lyap}$.

highest prediction valid time.

In general, we find that the chosen regularization parameters are close to the boundary between parameters that produce predictions that are always stable and those that sometimes produce unstable predictions. This result demonstrates why it is advantageous be able to efficiently tune the regularization parameters used during training; the more efficiently one can tune, the closer one can explore to this stability boundary, and thus the better one’s model will be. As

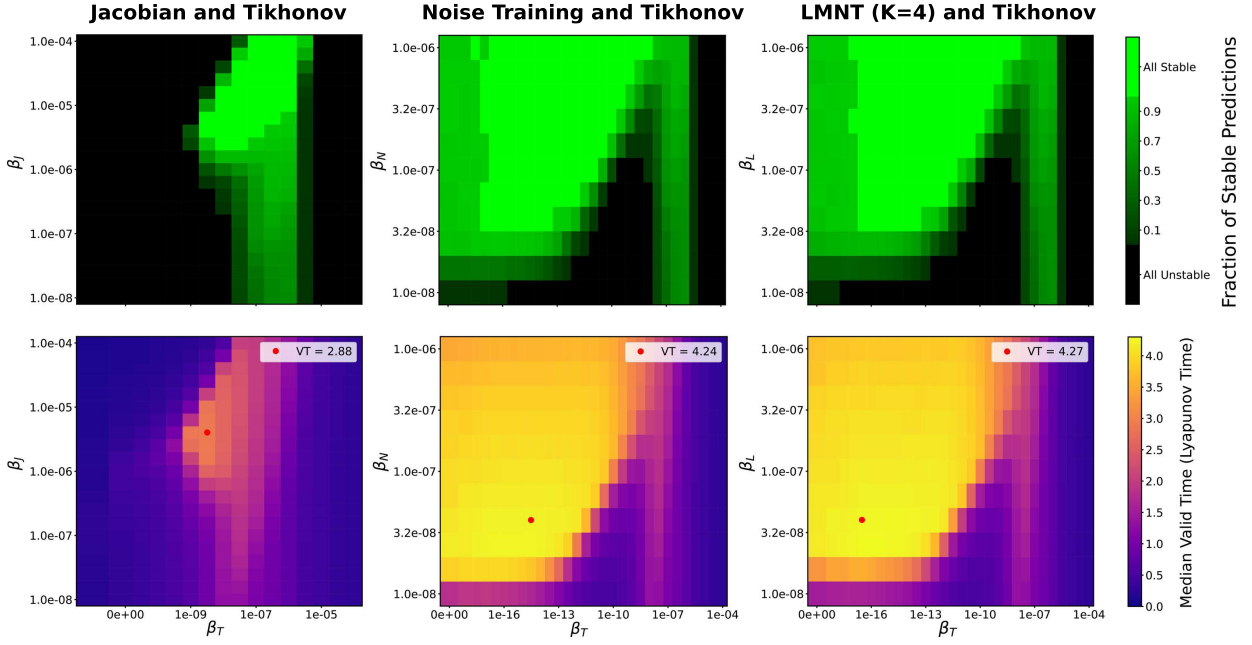


Figure 4.8: Fraction of Stable Predictions and Median Valid Time using Varying Regularization Parameter Values. Color plots displaying the fraction of stable predictions and median prediction valid time over a grid of different regularization parameters values are shown on the top and bottom, respectively. The regularization parameters used for each regularization type are plotted along the vertical and horizontal axes, respectively. The red dots in the bottom panels mark the regularization parameter values chosen to obtain the largest fraction of stable prediction and to produce the largest median prediction valid time, which is noted in the white box in the top right of the respective panel.

discussed in Sec. 4.2.3.3, the desirability of an easily-tunable regularization for reservoir computing makes LMNT preferable to noise training, which requires re-computation of the reservoir internal states each time the regularization parameter β_N is changed.

We remark that in some cases, it is possible to obtain an increased median prediction valid time at the cost of a small fraction of predictions being long-term unstable. For example, if we were to decrease β_T from the optimal value marked by the red dot in bottom left panel in Fig. 4.8, the median prediction valid time over all of the predictions would increase; however, the top left panel in Fig. 4.8 indicates that some of our predictions will now eventually become unstable. In addition, we see from the top right panel that increasing the LMNT regularization parameter value

from the optimal (e.g., to 10^{-6}) improves the robustness of our prediction stability to changes in the Tikhonov regularization parameter value. The bottom right panel shows that this comes at the cost of a decreased valid prediction time; however, if one is only able to perform a coarse hyperparameter optimization, then choosing a more robust LMNT regularization parameter value may be necessary. These cases emphasize how different choices of regularization parameter values can be optimal for different tasks.

4.4 Discussion and Conclusion

In the absence of mitigating techniques, long-term forecasting using the machine learning approach shown in Fig. 4.1 can often become unstable. Inspired by the mitigating technique of adding noise to the model input during training, we develop a novel training technique, LMNT, that introduces a new penalty term in the loss function for ML models with memory of past inputs that approximates the addition of many small, independent noise realizations. We show that in the case of reservoir computers trained to predict a paradigmatic test system, the Kuramoto-Sivashinsky equation, the addition of input noise or the use of LMNT regularization derived from our novel penalty term leads the most accurate and stable predictions. We find that reservoir computers trained using only Tikhonov regularization are only able to produce stable predictions for some reservoir realizations and training data sets. Reservoir computers trained with Jacobian and Tikhonov regularization make substantially less accurate short-term forecasts, measured by prediction valid time, than those trained with noise training or LMNT. In addition, we find that all stable predictions are able to reproduce the climate of the Kuramoto-Sivashinsky equation, though reservoir computers trained with only Tikhonov regularization did not reproduce the cli-

mate as closely as the other techniques used. In the LMNT case, the regularization matrix need not be computed again for each regularization strength value tested during the reservoir computer hyperparameter tuning, presenting a clear advantage over noise training.

While training with LMNT can lead to improved accuracy and stability, computing the LMNT penalty term can be computationally intensive due to the many matrix-matrix products involved (see Eq. 4.21 and Section 4.5). One way to speed up this computation during training would be to approximate the LMNT regularization matrix using a smaller number of training samples than is used for the rest of the training. Another, still more approximate and heuristic, possibility would be to compute the regularization using a constant input, such as the mean computed over the training data. We discuss our results using these two methods in Sec. 4.6.1.

Work remains to be completed on evaluating the performance of our new regularization technique on other systems, terrestrial climate in particular. As of the writing of this chapter, we have implemented a version of the reduced training sample LMNT in the hybrid atmospheric model described in Ref. [39]. In preliminary results, our implementation of LMNT leads to terrestrial climate predictions that are stable for over the decade-long run duration test and maintain a good climate, similar to the results achieved using input noise. Our case study using reservoir computing has shown that there can be a substantial advantage to considering, as LMNT does, the effect of many independent perturbations to model inputs further in the past than the most recent input considered in Jacobian regularization. The LMNT loss function penalty might, therefore, also be suitable for other RNN training methods, such as LSTM. Once LMNT is implemented in this context, one could also compare this technique to other regularization techniques, such as LASSO or multiple feedback training, which require that the model be trained using gradient descent or similar iterative methods.

Data and Code

The data and code that support the findings of this chapter are freely available at <https://github.com/awikner/res-noise-stabilization>.

Acknowledgements

The research completed for this article was supported by DARPA contract HR00112290035 and by ONR award N00014-22-1-2319. Alexander Wikner’s contribution to this research was also supported in part by NSF award DGE-1632976. Joseph Harvey’s contribution to this research was supported by NSF award PHY-2150399. Michelle Girvan’s contribution to this research was supported by ONR award N000142212656.

4.5 Supplemental: LMNT Performance vs. Number of Noise Steps, K

LMNT performance and implementation depend on the hyperparameter K , which represents the number of time steps over which to account for the linearized effect of input noise. In general, larger values of K should improve prediction performance, with diminishing improvement as K increases. Smaller values of K reduce the memory used to store the necessary model Jacobian matrices and the time needed to compute these Jacobian matrices and the many matrix-matrix products and sums involved (see Eq. 4.21). Ideally, one should choose the number of noise steps to be as small as possible while still resulting in near-optimal prediction stability and prediction accuracy.

LMNT Test Results					
Noise Steps, K	Regularization Parameters	Fraction of Stable Predictions	Median Valid Time (t_{Lyp})	Median $\bar{\epsilon}_{map}$	Median ϵ_{map}^{max}
1 (Jacobian)	$\beta_J = 10^{-5.4}$ $\beta_T = 10^{-8.5}$	1000/1000	2.88 ± 0.02	$(9.16 \pm 0.04) \times 10^{-3}$	$(5.82 \pm 0.06) \times 10^{-2}$
2	$\beta_L = 10^{-6.4}$ $\beta_T = 10^{-10.5}$	1000/1000	3.92 ± 0.04	$(4.10 \pm 0.02) \times 10^{-3}$	$(2.65 \pm 0.03) \times 10^{-2}$
3	$\beta_L = 10^{-6.6}$ $\beta_T = 10^{-12}$	1000/1000	4.04 ± 0.04	$(3.44 \pm 0.01) \times 10^{-3}$	$(2.39 \pm 0.03) \times 10^{-2}$
4	$\beta_L = 10^{-7.4}$ $\beta_T = 10^{-16.5}$	1000/1000	4.27 ± 0.04	$(2.75 \pm 0.01) \times 10^{-3}$	$(2.03 \pm 0.04) \times 10^{-2}$
5	$\beta_L = 10^{-7.4}$ $\beta_T = 10^{-16.5}$	1000/1000	4.27 ± 0.04	$(2.73 \pm 0.02) \times 10^{-3}$	$(2.02 \pm 0.02) \times 10^{-2}$

Table 4.5: Reservoir computer prediction results using Tikhonov regularization and LMNT regularization, where we vary the number of noise steps used to compute the LMNT regularization. For a detailed description of how we select regularization parameter values and compute uncertainty bounds, see the Table 4.4 caption.

As an example of the saturation in prediction performance as K increases, Table 4.5 displays the results of reservoirs trained with Tikhonov and LMNT regularization as we increase the number of noise steps approximated. We see that while we can select regularization parameter values such that all predictions are stable for all values of K tested, the median valid time increases and the median $\bar{\epsilon}_{map}$ and median ϵ_{map}^{max} decreases as K is increased. We find that, in this scenario, $K = 4$ gives as high a median valid time as low a median $\bar{\epsilon}_{map}$ and median ϵ_{map}^{max} as higher K ; we have therefore chosen to use $K = 4$ for all other tests of the LMNT regularization discussed in this chapter.

Finally, we discuss the computations required by LMNT, to give the reader and idea of how the complexity increases with K . To compute the LMNT penalty term for the entire training data, one must compute $\nabla_{\mathbf{s}} \mathbf{h}(\mathbf{s}_j)$, $\nabla_{\mathbf{s}} \mathbf{g}(\mathbf{u}_{j-1}, \mathbf{s}_j)$, and $\nabla_{\mathbf{u}} \mathbf{g}(\mathbf{u}_{j-1}, \mathbf{s}_j)$ for $1 \leq j \leq T_{train} - 1$. To reduce the total number of additional matrix-matrix products and conserve memory, one can

compute the sum over all training samples in order with the computation of the memory vectors also used for training. To do so, we first define $\mathcal{S}_j$ to be the result of the outer sum in Eq. 4.21 computed from $j = K$ to $j \leq T_{train} - 1$. Computation of the LMNT penalty term then proceeds as per Algorithm 1. A very similar algorithm can also be used to efficiently compute the LMNT

Algorithm 1: LMNT Penalty calculation during training

Data: Training inputs $\{\mathbf{u}_j\}$, $1 \leq j \leq T_{train}$;

Memory vectors during training $\{\mathbf{s}_j\}$, $0 \leq j \leq T_{train}$;

Noise approximation steps K

Result: LMNT Penalty term (Eq. 4.21), ℓ_{LMNT}

Compute $\nabla_{\mathbf{s}}\mathbf{h}(\mathbf{s}_K)$, $\nabla_{\mathbf{s}}\mathbf{g}(\mathbf{u}_K, \mathbf{s}_{K-1})$, and $\nabla_{\mathbf{u}}(K, K) = \nabla_{\mathbf{u}}\mathbf{g}(\mathbf{u}_K, \mathbf{s}_{K-1})$

for $k = K - 1$ **to** 1 **do**

 Compute $\nabla_{\mathbf{s}}\mathbf{g}(\mathbf{u}_k, \mathbf{s}_{k-1})$ and $\nabla_{\mathbf{u}}\mathbf{g}(\mathbf{u}_k, \mathbf{s}_{k-1})$

 Compute $\nabla_{\mathbf{u}}(K, k)$;

/* Eq. 4.22 */

end for

Compute $\mathcal{S}_K = \sum_{k=1}^K \|\nabla_{\mathbf{s}}\mathbf{h}(\mathbf{s}_K) \nabla_{\mathbf{u}}(K, k)\|_F^2$;

/* Eq. 4.21 */

for $j = K + 1$ **to** $T_{train} - 1$ **do**

 Compute $\nabla_{\mathbf{s}}\mathbf{h}(\mathbf{s}_j)$, $\nabla_{\mathbf{s}}\mathbf{g}(\mathbf{u}_j, \mathbf{s}_{j-1})$ and $\nabla_{\mathbf{u}}(j, j) = \nabla_{\mathbf{u}}\mathbf{g}(\mathbf{u}_j, \mathbf{s}_{j-1})$

for $k = j - K + 1$ **to** $j - 1$ **do**

 Compute $\nabla_{\mathbf{u}}(j, k) = \nabla_{\mathbf{s}}\mathbf{g}(\mathbf{u}_j, \mathbf{s}_{j-1}) \nabla_{\mathbf{u}}(j - 1, k)$

end for

 Compute $\mathcal{S}_j = \mathcal{S}_{j-1} + \sum_{k=1}^K \|\nabla_{\mathbf{s}}\mathbf{h}(\mathbf{s}_K) \nabla_{\mathbf{u}}(K, k)\|_F^2$

end for

Compute $\ell_{LMNT} = \frac{\beta_L}{T_{train}-K} \mathcal{S}_{T_{train}-1}$

regularization matrix (Eq. 4.33) used to train the reservoir computer.

For general ML models with memory, the K for which one expects to approach optimal prediction quality depends on the duration of the model memory. In the case of an RNN-based reservoir computer, this memory is primarily parametrized by the spectral radius, ρ , and the leaking rate, α . In general, the larger ρ is, and the closer α is to 0, the longer the reservoir computer memory will be and the larger K will need to be in order to account for perturbations to all prior inputs that substantially affect the current reservoir state vector.

4.6 Supplemental: Comparing Performance with Reduced Training Data

4.6.1 Reducing the Amount of LMNT Regularization Training Data Only

While it is most natural to compute the LMNT regularization using all of the training states (similar to how we compute the Jacobian regularization), this computation becomes expensive for long training times and large K values, as discussed in Sec. 4.5. One can substantially decrease this computational cost and potentially obtain a useful regularization matrix by either (a) computing the regularization using a size T subset of the training states, obtained by sampling the states uniformly, or (b) computing the regularization using the mean input computed over the training data and the reservoir state synchronized to this mean. The regularization matrix for the technique (a) is

$$\mathbf{R}_{L,T} = \frac{1}{T} \sum_{j=0}^{T-1} \left[\sum_{k=1+\text{floor}(j\tau)}^{K+\text{floor}(j\tau)} \nabla_{\mathbf{u}}(K + \text{floor}(j\tau), k) \nabla_{\mathbf{u}}(K + \text{floor}(j\tau), k)^{\top} \right], \quad (4.42)$$

where $\tau = (T_{\text{train}} - K)/T$, $T < T_{\text{train}} - K$ is a positive integer, and $\text{floor}(\dots)$ denotes rounding down. The regularization matrix for technique (b) is

$$\mathbf{R}_{L,0} = \sum_{k=1}^K \nabla_{\mathbf{u},0}(K, k) \nabla_{\mathbf{u},0}(K, k)^{\top}. \quad (4.43)$$

In Eq. 4.43, $\nabla_{\mathbf{u},0}(K, k)$ denotes evaluation of the right side of Eq. 4.22 at $\mathbf{u}_k = \mathbf{0}$ (the mean of our standardized training data) and $\mathbf{s}_{j-1} = \mathbf{s}_{j-2} = \dots = \mathbf{s}_k = \mathbf{s}_{\text{mean}}$, where $\mathbf{s}_{\text{mean}}$ is the memory vector synchronized to the constant mean input.

LMNT ($K = 4$) Test Results					
Training Samples, T	Regularization Parameters	Fraction of Stable Predictions	Median Valid Time (t_{Lyp})	Median $\bar{\epsilon}_{map}$	Median ϵ_{map}^{max}
Mean Input (Eq. 4.43)	$\beta_L = 10^{-7.4}$ $\beta_T = 10^{-15.5}$	1000/1000	4.30 ± 0.04	$(2.65 \pm 0.01) \times 10^{-3}$	$(2.00 \pm 0.03) \times 10^{-2}$
1	$\beta_L = 10^{-5.6}$ $\beta_T = 10^{-10}$	1000/1000	3.74 ± 0.07	$(5.57 \pm 0.03) \times 10^{-3}$	$(3.46 \pm 0.04) \times 10^{-2}$
5	$\beta_L = 10^{-7.2}$ $\beta_T = 10^{-16.5}$	1000/1000	4.21 ± 0.04	$(2.89 \pm 0.02) \times 10^{-3}$	$(2.16 \pm 0.03) \times 10^{-2}$
10	$\beta_L = 10^{-7.2}$ $\beta_T = 10^{-16}$	1000/1000	4.21 ± 0.04	$(2.89 \pm 0.02) \times 10^{-3}$	$(2.12 \pm 0.03) \times 10^{-2}$
20	$\beta_L = 10^{-7.4}$ $\beta_T = 10^{-16.5}$	1000/1000	4.26 ± 0.04	$(2.75 \pm 0.02) \times 10^{-3}$	$(2.03 \pm 0.03) \times 10^{-2}$
100	$\beta_L = 10^{-7.4}$ $\beta_T = 10^{-16.5}$	1000/1000	4.27 ± 0.04	$(2.73 \pm 0.02) \times 10^{-3}$	$(2.03 \pm 0.03) \times 10^{-2}$
19,996	$\beta_L = 10^{-7.4}$ $\beta_T = 10^{-16.5}$	1000/1000	4.27 ± 0.04	$(2.75 \pm 0.01) \times 10^{-3}$	$(2.03 \pm 0.04) \times 10^{-2}$

Table 4.6: Reservoir computer prediction results using LMNT regularization computed with different numbers of training samples. For a detailed description of how we select regularization parameter values and compute uncertainty bounds, see the Table 4.4 caption.

Table 4.6 shows the prediction results obtained from reservoir trained with LMNT, reduced training sample LMNT, and mean-input LMNT. For our particular test system, we find that training with only 20 training samples (0.1% of the available training data) is sufficient to obtain predictions that are always observed to be stable and have a median valid time equivalent to that when the LMNT regularization is computed with all of the training data. This effectiveness for a small number of training samples indicates that LMNT-like regularization would also be effective in models trained in batches using stochastic gradient descent or its derivatives. In addition, computing the LMNT regularization with the mean input and synchronized reservoir state performs as well as the full LMNT regularization. This result suggests that, in this case, the variability across the training data of the derivative matrices $\nabla_{\mathbf{u}}(j, k)$ is small enough that a small number of samples, or one representative derivative matrix, is sufficient to compute a regularization ma-

trix that performs comparably to the full LMNT. If this property holds in other applications, the computationally simpler mean-input LMNT might yield sufficient stabilization and make the full or partial LMNT computation unnecessary.

4.6.2 Reducing the Total Amount of Training Data

Regularization Parameter Search Grids	
Regularization Type	Search Grid
Tikhonov (β_T)	0 and 10^l for $l \in \{-18, -17.5, \dots, -1.5, -1\}$
Jacobian (β_J), Noise Training (β_N), and LMNT (β_L)	0 and 10^l for $l \in \{-14, -14 + 12/35, \dots, -2 - 12/35, -2\}$

Table 4.7: The regularization parameters that are searched over for each reservoir prediction test using the reduced training data.

In addition to comparing the performance of reservoir computers trained with different regularization techniques in the limit where there is sufficient training data for the resulting predictions to, in the best case, produce accurate short-predictions that are always observed to be stable, we also compared the performance of different regularization techniques in the limit of greatly reduced training data. Fig. 4.9 displays the results of training the reservoir computer with the regularization techniques which produced the best results in the original tests using only $T = 750$ training samples, with all other parameters kept the same in Tables 4.1 and 4.2. We observe that no combination of regularization techniques result in predictions with reasonable short-term accuracy, as indicated by the lack of any set of regularization parameters that result in a median prediction valid time above $0.5t_{Lyap}$. In each of the three cases, we are able to achieve universally stable predictions using an extremely high amount of Tikhonov regularization; the minuscule median prediction valid time indicates, however, that these predictions are not at all capturing the

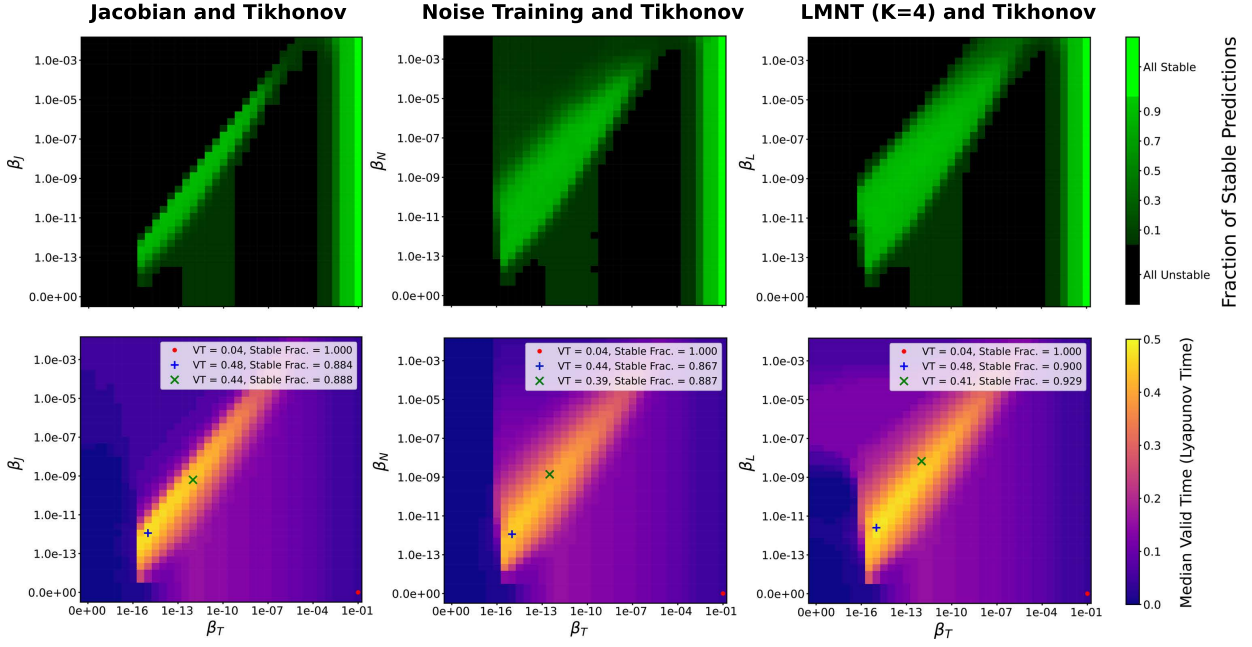


Figure 4.9: Fraction of Stable Predictions and Median Valid Time using Varying Regularization Parameter Values for Reduced Training Data . Color plots displaying the fraction of stable predictions and median prediction valid time over a grid of different regularization parameters values are shown on the top and bottom, respectively. In the median prediction valid time plots, the red dot marks the regularization parameters values that result in the largest fraction of stable predictions; if there are multiple values with the same fraction, then we mark those that result in the longest median prediction valid time. The blue crosses mark the regularization parameters values which result in the longest median prediction valid time; if there are multiple values with the same valid time, then we mark those that result in the largest fraction of stable predictions. Finally, the green x's mark the parameters values where we have performed the same optimization as for the red dot, but have excluded Tikhonov regularization parameters values above 10^{-4} . If multiple values remain after any of these optimizations, we mark the values with the lowest regularization, giving lowest priority to Tikhonov regularization.

system dynamics.

Within the region marked by the light-green coloring in the top panels and the yellow coloring in the bottom panels, we note that each combined regularization technique tested is still able to produce predictions which are mostly stable and do have a small amount of accuracy in the short term. Comparing the results with the greatest median prediction valid time and the greatest fraction of stable predictions within this region for each type of combined regularization, we find that reservoirs trained with noise training have the lowest median prediction valid

time when using the regularization parameter values optimized for median prediction valid time (see the blue crosses in the bottom panels of Fig. 4.9). In addition, when using parameters optimized for the highest fraction of stable predictions in this region (the green x's in the bottom panels of Fig. 4.9), reservoirs trained with noise training have approximately the same number of stable predictions as Jacobian regularization while having a lower median valid prediction time than those trained using either Jacobian or LMNT regularization. Reservoirs trained with LMNT regularization achieve the highest fraction of stable predictions within this region while maintaining a comparable median prediction valid time to reservoirs trained with other regularization techniques.

The comparatively poor performance of noise training in these tests indicates the effectiveness of training the reservoir with input perturbations in many different directions for each training input, as is approximated for one input in Jacobian regularization and over K past inputs in LMNT regularization, when the total amount of training data is small. In this case, it is more likely that certain regions of the attractor will be undersampled; if we would like the model to learn to return to the true attractor from a generic perturbation to one of these inputs, then we must consider many perturbations to these few cases. In the long training data case, one could instead rely on similar samples that appear later in the training data.

Chapter 5: Concluding Remarks

In summary, in this thesis we have presented and tested techniques for addressing the critical problem of forecasting complex, high-dimensional spatiotemporally chaotic systems at both short and long-time scales. In Chapter 2, we introduced and tested Combined Hybrid-Parallel Prediction (CHyPP), a hybrid technique for scalable forecasting of high-resolution chaotic dynamics that combines parallel ML models with an imperfect knowledge-based predictor. We demonstrated that this technique produces more accurate forecasts than either component model acting alone, even in the case when these component models produced poor predictions or did not capture the subgrid-scale dynamics. In Chapter 3, we developed an iterative technique for training hybrid ML models when one only has access to sparse and noisy system observations, rather than complete system states, using data assimilation. We found that this iterative technique leads to hybrid models with substantially more accurate short-term forecasts than the component knowledge-based model, and that these hybrid models can be used to substantially improve the estimation of the complete system state. Finally, in Chapter 4, we developed a training technique for mitigating instability of ML models for predicting dynamics that we call Linearized Multi-Noise Training (LMNT). We systematically compared predictions from reservoir computers trained with this technique as well as other commonly-used mitigating techniques, finding that LMNT leads to predictions that are always found to be stable while simplifying the reservoir

computer hyperparameter tuning.

Since our original tests of the CHyPP technique, CHyPP has been implemented to create a coupled atmosphere-ocean hybrid model using a relatively low-resolution global atmospheric model and many reservoir computers trained in parallel. This atmospheric-oceanic hybrid model produces more accurate short-term forecasts than the component global atmospheric model, demonstrates the longest accurate and stable climate replication of any ML earth system model at the time of this writing, and is able to capture dynamics that are not captured by the component atmospheric model [40]. In addition, the hybrid approach has been applied to non-stationary scenarios, demonstrating the ability to forecast tipping points and post-tipping point dynamics even when both component methods could not [132]. We are currently working to test our iterative technique for training on sparse and noisy observations using the previously described atmospheric-oceanic hybrid model; our current results show noticeable improvements in the estimation of the full atmospheric state. As discussed in Chapter 4, our efforts to implement LMNT into this hybrid model are ongoing.

At the moment, operational data assimilation appears to be safe from the encroachment of ML-based forecast models. The current model architectures have not yet shown sufficient accuracy when predicting at the minute-to-hour time scales required when assimilating 4D observations over the course of an analysis cycle. If this problem is solved, however, then we foresee many advantages to using ML surrogate forecast models in a data assimilation cycle. If it could be adjusted to produce forecasts with an appropriate variance, then a fast ML surrogate model would be extremely advantageous to ensemble-based DA techniques by allowing for better uncertainty estimation via increased ensemble sizes. In addition, automatic differentiation allows one to compute the adjoint to an ML surrogate model quickly and easily, which is an essential

component to 4D variational data assimilation.

Even ignoring the issues many current ML surrogate models have with stability after many autoregressive iterations, there are still some unanswered questions regarding if such models could be used for effective climate prediction. While it is possible to confirm if ML surrogate models are able to replicate the current or past terrestrial climate, the non-stationary nature of the climate and our lack of a comparable period that we could obtain validation data from means that it is not possible to validate whether these models will correctly predict the dynamics of the true non-stationary climate system directly. We note that while it has been shown that ML models can accurately recreate the dynamics of toy non-stationary systems, even when the exact form of the non-stationarity is not known [132], these techniques have not been tested yet at the climate system scale. One potentially promising method for validating ML surrogate climate models would be to use many “virtual worlds”. This involves obtaining century-long simulation data from an ensemble of physics-based climate models, each with a different set of physics and physical parametrizations, under a number of different climate change scenarios. For each of these models and scenarios, an ML surrogate model could be re-trained or fine-tuned using the early data and could then be used to produce multi-decade climate predictions. If the ML surrogate model is able to capture the future climatology for a majority of the models trained on and for a majority of the climate change scenarios explored, then it stands to reason that, if the ML model were to be trained on real data from the true changing climate, it will accurately predict the true non-stationary climate.

In the 1940’s and 50’s, the goal of using electronic computers to forecast the weather helped to jump-start the fledgling disciplines of computer science and engineering. Now, more than 70 years later, the machine learning techniques and the necessary hardware developed by these

disciplines to address problems as far afield as playing the game of Go are poised to revolutionize weather forecasting once again. Given ML surrogate weather models' current track record of accurate prediction, it is entirely possible that, in the next few years, we will see predictions from a physics-based NWP model and an ML surrogate model displayed side-by-side when we look up a forecast. We hope that the ideas and techniques discussed in this thesis will allow scientists to continue to push the boundaries of prediction, further improve the skill of ML-based weather forecasting models, and to extend these models to forecast the long-term climate under a range of climate change scenarios.

Bibliography

- [1] W J Conover. *Practical Nonparametric Statistics*. Wiley, New York, NY, USA, 3 edition, 1999.
- [2] Edward N. Lorenz. Deterministic Nonperiodic Flow. *Journal of Atmospheric Sciences*, 20(2):130–141, March 1963.
- [3] James Gleick. *Chaos: Making a New Science*. Viking, 1987.
- [4] Tien-Yien Li and James A. Yorke. Period Three Implies Chaos. *The American Mathematical Monthly*, 82(10):985–992, 1975.
- [5] S. Smale. Differentiable dynamical systems. *Bulletin of the American Mathematical Society*, 73(6):747–817, 1967.
- [6] Edward Ott, Celso Grebogi, and James A. Yorke. Controlling chaos. *Physical Review Letters*, 64(11):1196–1199, March 1990.
- [7] Cleveland Abbe. The physical basis of long-range weather forecasts. *Monthly Weather Review*, 29(12):551–561, December 1901.
- [8] Bjerknes Vilhelm. Das Problem der Wettervorhersage, betrachtet vom Standpunkte der Mechanik und der Physik. *Meteor. Z.*, 21:1–7, 1904.
- [9] J. G. Charney, R. Fjörtoft, and J. Von Neumann. Numerical Integration of the Barotropic Vorticity Equation. *Tellus*, 2(4):237–254, January 1950.
- [10] Peter Lynch. The origins of computer weather prediction and climate modeling. *Journal of Computational Physics*, 227(7):3431–3444, March 2008.
- [11] Peter Bauer, Alan Thorpe, and Gilbert Brunet. The quiet revolution of numerical weather prediction. *Nature*, 525(7567):47–55, September 2015.
- [12] Jeffrey K. Lazo, Rebecca E. Morss, and Julie L. Demuth. 300 Billion Served: Sources, Perceptions, Uses, and Values of Weather Forecasts. *Bulletin of the American Meteorological Society*, 90(6):785–798, June 2009.
- [13] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

- [14] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, December 1989.
- [15] Rajat Raina, Anand Madhavan, and Andrew Y. Ng. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09*, pages 873–880, New York, NY, USA, June 2009. Association for Computing Machinery.
- [16] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.
- [17] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical Text-Conditional Image Generation with CLIP Latents. <https://arxiv.org/abs/2204.06125v1>, April 2022.
- [18] Herbert Jaeger and Harald Haas. Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication. *Science*, 304(5667):78–80, April 2004.
- [19] 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.
- [20] Zhixin Lu, Brian R. Hunt, and Edward Ott. Attractor reconstruction by machine learning. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 28(6):061104, June 2018.
- [21] 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.
- [22] Jaideep Pathak, Alexander Wikner, Rebeckah Fussell, Sarthak Chandra, Brian R Hunt, Michelle Girvan, and Edward Ott. Hybrid forecasting of chaotic processes: Using machine learning in conjunction with a knowledge-based model. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 28(4):041101, 2018.
- [23] Richard H. Moss, Jae A. Edmonds, Kathy A. Hibbard, Martin R. Manning, Steven K. Rose, Detlef P. Van Vuuren, Timothy R. Carter, Seita Emori, Mikiko Kainuma, Tom Kram, Gerald A. Meehl, John F. B. Mitchell, Nebojsa Nakicenovic, Keywan Riahi, Steven J. Smith, Ronald J. Stouffer, Allison M. Thomson, John P. Weyant, and Thomas J. Wilbanks. The next generation of scenarios for climate change research and assessment. *Nature*, 463(7282):747–756, February 2010.

- [24] Hans Hersbach, Bill Bell, Paul Berrisford, Shoji Hirahara, András Horányi, Joaquín Muñoz-Sabater, Julien Nicolas, Carole Peubey, Raluca Radu, Dinand Schepers, Adrian Simmons, Cornel Soci, Saleh Abdalla, Xavier Abellan, Gianpaolo Balsamo, Peter Bechtold, Gionata Biavati, Jean Bidlot, Massimo Bonavita, Giovanna De Chiara, Per Dahlgren, Dick Dee, Michail Diamantakis, Rossana Dragani, Johannes Flemming, Richard Forbes, Manuel Fuentes, Alan Geer, Leo Haimberger, Sean Healy, Robin J. Hogan, Elías Hólm, Marta Janisková, Sarah Keeley, Patrick Laloyaux, Philippe Lopez, Cristina Lupu, Gabor Radnoti, Patricia de Rosnay, Iryna Rozum, Freja Vamborg, Sebastien Villaume, and Jean-Noël Thépaut. The ERA5 global reanalysis. *Quarterly Journal of the Royal Meteorological Society*, 146(730):1999–2049, 2020.
- [25] Sebastian Scher and Gabriele Messori. Weather and climate forecasting with neural networks: Using general circulation models (GCMs) with different complexity as a study ground. *Geoscientific Model Development*, 12(7):2797–2809, July 2019.
- [26] Jonathan A. Weyn, Dale R. Durran, and Rich Caruana. Can Machines Learn to Predict Weather? Using Deep Learning to Predict Gridded 500-hPa Geopotential Height From Historical Weather Data. *Journal of Advances in Modeling Earth Systems*, 11(8):2680–2693, 2019.
- [27] Jonathan A. Weyn, Dale R. Durran, and Rich Caruana. Improving Data-Driven Global Weather Prediction Using Deep Convolutional Neural Networks on a Cubed Sphere. *Journal of Advances in Modeling Earth Systems*, 12(9):e2020MS002109, 2020.
- [28] Stephan Rasp, Peter D. Dueben, Sebastian Scher, Jonathan A. Weyn, Soukayna Mouatadid, and Nils Thuerey. WeatherBench: A Benchmark Data Set for Data-Driven Weather Forecasting. *Journal of Advances in Modeling Earth Systems*, 12(11):e2020MS002203, 2020.
- [29] Stephan Rasp and Nils Thuerey. Data-Driven Medium-Range Weather Prediction With a Resnet Pretrained on Climate Simulations: A New Model for WeatherBench. *Journal of Advances in Modeling Earth Systems*, 13(2):e2020MS002405, 2021.
- [30] Ashesh Chattopadhyay, Mustafa Mustafa, Pedram Hassanzadeh, Eviatar Bach, and Karthik Kashinath. Towards physically consistent data-driven weather forecasting: Integrating data assimilation with equivariance-preserving deep spatial transformers, March 2021.
- [31] Troy Arcomano, Istvan Szunyogh, Jaideep Pathak, Alexander Wikner, Brian R. Hunt, and Edward Ott. A machine learning-based global atmospheric forecast model. *Geophysical Research Letters*, 47(9):e2020GL087776, 2020.
- [32] Ryan Keisler. Forecasting Global Weather with Graph Neural Networks, February 2022.
- [33] Jaideep Pathak, Shashank Subramanian, Peter Harrington, Sanjeev Raja, Ashesh Chattopadhyay, Morteza Mardani, Thorsten Kurth, David Hall, Zongyi Li, Kamyar Azizzadenesheli, Pedram Hassanzadeh, Karthik Kashinath, and Animashree Anandkumar. Four-CastNet: A Global Data-driven High-resolution Weather Model using Adaptive Fourier Neural Operators, February 2022.

- [34] Remi Lam, Alvaro Sanchez-Gonzalez, Matthew Willson, Peter Wirnsberger, Meire Fortunato, Ferran Alet, Suman Ravuri, Timo Ewalds, Zach Eaton-Rosen, Weihua Hu, Alexander Meroze, Stephan Hoyer, George Holland, Oriol Vinyals, Jacklynn Stott, Alexander Pritzel, Shakir Mohamed, and Peter Battaglia. GraphCast: Learning skillful medium-range global weather forecasting, August 2023.
- [35] Kaifeng Bi, Lingxi Xie, Hengheng Zhang, Xin Chen, Xiaotao Gu, and Qi Tian. Accurate medium-range global weather forecasting with 3D neural networks. *Nature*, 619(7970):533–538, July 2023.
- [36] Alexander Wikner, Jaideep Pathak, Brian Hunt, Michelle Girvan, Troy Arcomano, Istvan Szunyogh, Andrew Pomerance, and Edward Ott. Combining machine learning with knowledge-based modeling for scalable forecasting and subgrid-scale closure of large, complex, spatiotemporal systems. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 30(5):053111, May 2020.
- [37] Alexander Wikner, Jaideep Pathak, Brian R. Hunt, Istvan Szunyogh, Michelle Girvan, and Edward Ott. Using data assimilation to train a hybrid forecast system that combines machine-learning and knowledge-based components. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 31(5):053114, May 2021.
- [38] 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*, November 2023.
- [39] Troy Arcomano, Istvan Szunyogh, Alexander Wikner, Jaideep Pathak, Brian R. Hunt, and Edward Ott. A Hybrid Approach to Atmospheric Modeling That Combines Machine Learning With a Physics-Based Numerical Model. *Journal of Advances in Modeling Earth Systems*, 14(3):e2021MS002712, 2022.
- [40] Troy Arcomano, Istvan Szunyogh, Alexander Wikner, Brian R. Hunt, and Edward Ott. A Hybrid Atmospheric Model Incorporating Machine Learning Can Capture Dynamical Processes Not Captured by Its Physics-Based Component. *Geophysical Research Letters*, 50(8):e2022GL102649, 2023.
- [41] Alexander Wikner, Brian Hunt, Szunyogh Istvan, and Edward Ott. Online learning of the model state, forecast bias, and observation bias using hybrid machine learning and ensemble Kalman filtering. In *XXVIII General Assembly of the International Union of Geodesy and Geophysics (IUGG)*. GFZ German Research Centre for Geosciences, July 2023.
- [42] Dhruvit Patel and Alexander Wikner. Attention-Based Ensemble Pooling for Time Series Forecasting, October 2023.
- [43] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato,

- Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv:1609.08144 [cs]*, October 2016. arXiv: 1609.08144.
- [44] Izhar Wallach, Michael Dzamba, and Abraham Heifets. AtomNet: A Deep Convolutional Neural Network for Bioactivity Prediction in Structure-based Drug Discovery. *arXiv:1510.02855 [cs, q-bio, stat]*, October 2015. arXiv: 1510.02855.
- [45] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.
- [46] Herbert Jaeger. The “echo state” approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148, January 2001.
- [47] 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, 2002.
- [48] 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.
- [49] Masanobu Inubushi and Kazuyuki Yoshimura. Reservoir Computing Beyond Memory-Nonlinearity Trade-off. *Scientific Reports*, 7(1):10199, December 2017.
- [50] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.
- [51] 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, Doha, Qatar, October 2014. Association for Computational Linguistics.
- [52] Pantelis R. Vlachas, Jaideep Pathak, Brian R. Hunt, Themistoklis P. Sapsis, Michelle Girvan, Edward Ott, and Petros Koumoutsakos. Forecasting of Spatio-temporal Chaotic Dynamics with Recurrent Neural Networks: a comparative study of Reservoir Computing and Backpropagation Algorithms. *arXiv:1910.05266 [physics]*, October 2019. arXiv: 1910.05266.
- [53] Quentin Vinckier, François Duport, Anteo Smerieri, Kristof Vandoorne, Peter Bienstman, Marc Haelterman, and Serge Massar. High-performance photonic reservoir computer based on a coherently driven passive cavity. *Optica*, 2(5):438, May 2015.

- [54] Bogdan Penkovsky, Laurent Larger, and Daniel Brunner. Efficient design of hardware-enabled reservoir computing in FPGAs. *Journal of Applied Physics*, 124(16):162101, October 2018.
- [55] 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.
- [56] Zhong Yi Wan, Pantelis Vlachas, Petros Koumoutsakos, and Themistoklis Sapsis. Data-assisted reduced-order modeling of extreme events in complex dynamical systems. *PloS one*, 13(5):e0197704, 2018.
- [57] Herbert Jaeger. The echo state approach to analysing and training recurrent neural networks-with an erratum note. In *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, volume 148, page 34. 2001.
- [58] A.N. Tikhonov and V.I.A. Arsenin. *Solutions of ill-posed problems*. Scripta series in mathematics. Winston, 1977.
- [59] MATLAB. 9.5.0.944444 (R2018b). The MathWorks Inc., Natick, Massachusetts, 2018.
- [60] Youcef Saad and Martin H. Schultz. GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, July 1986.
- [61] Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, page 116, Banff, Alberta, Canada, July 2004. Association for Computing Machinery.
- [62] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Convolutional Networks*, page 321–361. MIT Press, 2017.
- [63] Yoshiki Kuramoto. Diffusion-Induced Chaos in Reaction Systems. *Progress of Theoretical Physics Supplement*, 64:346–367, 1978.
- [64] G. I. Sivashinsky. Nonlinear analysis of hydrodynamic instability in laminar flames—I. Derivation of basic equations. *Acta Astronautica*, 4:1177–1206, 1977.
- [65] Steven M Cox and Paul C Matthews. Exponential time differencing for stiff systems. *Journal of Computational Physics*, 176(2):430–455, 2002.
- [66] Charles Meneveau and Joseph Katz. Scale-Invariance and Turbulence Models for Large-Eddy Simulation. *Annual Review of Fluid Mechanics*, 32(1):1–32, January 2000.
- [67] Edward N. Lorenz. Designing chaotic models. *Journal of the Atmospheric Sciences*, 62(5):1574–1587, 2005.

- [68] E.N. Lorenz. Predictability: a problem partly solved. In *Seminar on Predictability, 4-8 September 1995*, volume 1, pages 1–18, Shinfield Park, Reading, 1995. ECMWF, ECMWF.
- [69] Mantas Lukoševičius. A practical guide to applying echo state networks. In *Neural Networks: Tricks of the Trade*, pages 659–686. Springer, 2012.
- [70] Marc Bocquet, Julien Brajard, Alberto Carrassi, and Laurent Bertino. Data assimilation as a learning tool to infer ordinary differential equation representations of dynamical models. *Nonlinear Processes in Geophysics*, 26(3):143–162, July 2019.
- [71] A. Chattopadhyay, P. Hassanzadeh, and D. 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, 2020.
- [72] Julien Brajard, Alberto Carrassi, Marc Bocquet, and Laurent Bertino. Combining data assimilation and machine learning to emulate a dynamical model from sparse and noisy observations: A case study with the Lorenz 96 model. *Journal of Computational Science*, 44:101171, July 2020.
- [73] Marc Bocquet, Julien Brajard, Alberto Carrassi, and Laurent Bertino. Bayesian inference of chaotic dynamics by merging data assimilation, machine learning and expectation-maximization. *Foundations of Data Science*, 2(2639-8001_2020_1_55):55, 2020.
- [74] Marc Bocquet, Alban Farchi, and Quentin Malartic. Online learning of both state and dynamics using ensemble kalman filters. *Foundations of Data Science*, 0(2639-8001_2019_0_24), 2020.
- [75] Markus Reichstein, Gustau Camps-Valls, Bjorn Stevens, Martin Jung, Joachim Denzler, Nuno Carvalhais, and Prabhat. Deep learning and process understanding for data-driven Earth system science. *Nature*, 566(7743):195–204, February 2019.
- [76] F. Tomizawa and Y. Sawada. Combining ensemble kalman filter and reservoir computing to predict spatio-temporal chaotic systems from imperfect observations and models. *Geoscientific Model Development (preprint)*, <https://doi.org/10.5194/gmd-2020-211>, in review, 2020.
- [77] Kody Law, A. Stuart, and K. Zygalakis. *Data Assimilation: A Mathematical Introduction*, volume 62. Springer International, 10 2015.
- [78] Mark Asch, Marc Bocquet, and Maëlle Nodet. *Data assimilation: Methods, Algorithms, and Applications*. Fundamentals of Algorithms. SIAM, 2016.
- [79] Steven Fletcher. *Data Assimilation for the Geosciences: From Theory to Applications*. Elsevier, 03 2017.
- [80] Franz Hamilton, Tyrus Berry, and Timothy Sauer. Ensemble Kalman Filtering without a Model. *Physical Review X*, 6(1):011021, March 2016.

- [81] Redouane Lguensat, Pierre Tandeo, Pierre Ailliot, Manuel Pulido, and Ronan Fablet. The Analog Data Assimilation. *Monthly Weather Review*, 145(10):4093–4107, October 2017.
- [82] G. V. Puskorius and L. A. Feldkamp. Neurocontrol of nonlinear dynamical systems with kalman filter trained recurrent networks. *IEEE Transactions on Neural Networks*, 5(2):279–297, 1994.
- [83] Henry D. I. Abarbanel, Paul J. Rozdeba, and Sasha Shirman. Machine Learning: Deepest Learning as Statistical Data Assimilation Problems. *Neural Computation*, 30(8):2025–2055, August 2018.
- [84] Georg A. Gottwald and Sebastian Reich. Supervised learning from noisy observations: Combining machine-learning techniques with data assimilation, 2020.
- [85] Zhijin Li and I. M. Navon. Optimality of variational data assimilation and its relationship with the kalman filter and smoother. *Quarterly Journal of the Royal Meteorological Society*, 127(572):661–683, 2001.
- [86] Brian R Hunt, Eric J Kostelich, and Istvan Szunyogh. Efficient data assimilation for spatiotemporal chaos: A local ensemble transform kalman filter. *Physica D: Nonlinear Phenomena*, 230(1-2):112–126, 2007.
- [87] Istvan Szunyogh. *Applicable Atmospheric Dynamics: Techniques for the Exploration of Atmospheric Dynamics*. World Scientific, 1 edition, 01 2014.
- [88] Geir Evensen. The Ensemble Kalman filter: Theoretical Formulation and Practical Implementation. *Ocean Dynamics*, 53(4):343–367, 2003.
- [89] P. L. Houtekamer and Herschel L. Mitchell. Data Assimilation Using an Ensemble Kalman Filter Technique. *Monthly Weather Review*, 126(3):796–811, March 1998.
- [90] Gerrit Burgers, Peter Jan van Leeuwen, and Geir Evensen. Analysis Scheme in the Ensemble Kalman Filter. *Monthly Weather Review*, 126(6):1719–1724, 1998.
- [91] Jeffrey S. Whitaker and Thomas M. Hamill. Ensemble data assimilation without perturbed observations. *Monthly Weather Review*, 130(7):1913–1924, 2002.
- [92] Craig H Bishop, Brian J Etherton, and Sharanya J Majumdar. Adaptive Sampling with the Ensemble Transform Kalman Filter. Part 1: Theoretical Aspects. *Monthly Weather Review*, 129(3):420–436, 2001.
- [93] Xuguang Wang, Craig H Bishop, and Simon J Julier. Which is better, an ensemble of positive–negative pairs or a centered spherical simplex ensemble? *Monthly Weather Review*, 132(7):1590–1605, 2004.
- [94] Yoshiki Kuramoto. Diffusion-Induced Chaos in Reaction Systems. *Progress of Theoretical Physics Supplement*, 64:346–367, February 1978.
- [95] G. I. Sivashinsky. Nonlinear analysis of hydrodynamic instability in laminar flames—I. Derivation of basic equations. *Acta Astronautica*, 4(11):1177–1206, November 1977.

- [96] Alexander McFarlane Mood. *Introduction to the Theory of Statistics*. McGraw-Hill, 1 edition, 1950.
- [97] Aly-Khan Kassam and Lloyd N. Trefethen. Fourth-Order Time-Stepping for Stiff PDEs. *SIAM Journal on Scientific Computing*, 26(4):1214–1233, January 2005.
- [98] Edward Ott, Brian R Hunt, Istvan Szunyogh, Aleksey V Zimin, Eric J Kostelich, Matteo Corazza, Eugenia Kalnay, DJ Patil, and James A Yorke. A local ensemble kalman filter for atmospheric data assimilation. *Tellus A*, 56(5):415–428, 2004.
- [99] Istvan Szunyogh, Eric J Kostelich, Gyorgyi Gyarmati, Eugenia Kalnay, Brian R Hunt, Edward Ott, Elizabeth Satterfield, and James A Yorke. A local ensemble transform kalman filter data assimilation system for the ncep global model. *Tellus A*, 60(1):113–130, 2008.
- [100] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.
- [101] 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.
- [102] 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.
- [103] Kaushik Balakrishnan and Devesh Upadhyay. Deep Adversarial Koopman Model for Reaction-Diffusion systems. *arXiv:2006.05547 [cs, eess]*, June 2020.
- [104] Stephan Rasp, Michael S. Pritchard, and Pierre Gentine. Deep learning to represent sub-grid processes in climate models. *Proceedings of the National Academy of Sciences*, 115(39):9684–9689, September 2018.
- [105] P. Gentine, M. Pritchard, S. Rasp, G. Reinaudi, and G. Yacalis. Could Machine Learning Break the Convection Parameterization Deadlock? *Geophysical Research Letters*, 45(11):5742–5751, 2018.
- [106] Ashesh Chattopadhyay, Mustafa Mustafa, Pedram Hassanzadeh, Eviatar Bach, and Karthik Kashinath. Towards physics-inspired data-driven weather forecasting: Integrating data assimilation with a deep spatial-transformer-based U-NET in a case study with ERA5. *Geoscientific Model Development*, 15(5):2221–2237, March 2022.
- [107] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhat-tacharya, Andrew Stuart, and Anima Anandkumar. Fourier Neural Operator for Parametric Partial Differential Equations, May 2021.
- [108] J. Auslander, N. P. Bhatia, and P. Seibert. Attractors in dynamical systems. Technical Report NASA-CR-59858, NASA, January 1964.

- [109] Jocelyn Sietsma and Robert J. F. Dow. Creating artificial neural networks that generalize. *Neural Networks*, 4(1):67–79, January 1991.
- [110] Guozhong An. The Effects of Adding Noise During Backpropagation Training on a Generalization Performance. *Neural Computation*, 8(3):643–674, April 1996.
- [111] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning - ICML '08*, pages 1096–1103, Helsinki, Finland, 2008. ACM Press.
- [112] Ben Poole, Jascha Sohl-Dickstein, and Surya Ganguli. Analyzing noise in autoencoders and deep networks, June 2014.
- [113] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
- [114] Klaus Greff, Rupesh K. Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM: A Search Space Odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, October 2017.
- [115] Stephen A Billings. *Nonlinear System Identification: NARMAX Methods in the Time, Frequency, and Spatio-Temporal Domains*. John Wiley & Sons, Ltd, September 2013.
- [116] Mantas Lukoševičius and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, August 2009.
- [117] Robert Tibshirani. Regression Shrinkage and Selection Via the Lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.
- [118] Judy Hoffman, Daniel A. Roberts, and Sho Yaida. Robust Learning with Jacobian Regularization, August 2019.
- [119] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.
- [120] John F. Kolen and Stefan C. Kremer. *A Field Guide to Dynamical Recurrent Networks*. John Wiley & Sons, January 2001.
- [121] Alex M Lamb, Anirudh Goyal, Ying Zhang, Saizheng Zhang, Aaron C Courville, and Yoshua Bengio. Professor Forcing: A New Algorithm for Training Recurrent Networks. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.
- [122] David Sussillo and L. F. Abbott. Generating Coherent Patterns of Activity from Chaotic Neural Networks. *Neuron*, 63(4):544–557, August 2009.
- [123] Soon Hoe Lim, N. Benjamin Erichson, Liam Hodgkinson, and Michael W Mahoney. Noisy Recurrent Neural Networks. In *Advances in Neural Information Processing Systems*, volume 34, pages 5124–5137. Curran Associates, Inc., 2021.

- [124] Louis M. Pecora and Thomas L. Carroll. Synchronization of chaotic systems. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 25(9):097611, September 2015.
- [125] I. Daubechies, M. Defrise, and C. De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics*, 57(11):1413–1457, 2004.
- [126] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 3 edition, 1999.
- [127] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.
- [128] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer New York, July 2014.
- [129] P. Welch. The use of fast Fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms. *IEEE Transactions on Audio and Electroacoustics*, 15(2):70–73, June 1967.
- [130] S. M. Cox and P. C. Matthews. Exponential Time Differencing for Stiff Systems. *Journal of Computational Physics*, 176(2):430–455, March 2002.
- [131] Galgani Bennetin, L Galgani, Antonio Giorgilli, and Jean-Marie Strelcyn. Lyapunov characteristic exponents for smooth dynamical systems and for Hamiltonian systems: A method for computing all of them. *Meccanica*, 15(9):27, 1980.
- [132] Dhruvit Patel and Edward Ott. Using machine learning to anticipate tipping points and extrapolate to post-tipping dynamics of non-stationary dynamical systems. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 33(2):023143, February 2023.