Neural Networks 170 (2024) 94–110

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Full Length Article

Stabilizing machine learning prediction of dynamics: Novel noise-inspired regularization tested with reservoir computing

Alexander Wikner [a,*], Joseph Harvey [d,1], Michelle Girvan [a], Brian R. Hunt [b], Andrew Pomerance [e], Thomas Antonsen [a,c], Edward Ott [a,c]

[a] Department of Physics, University of Maryland, 4150 Campus Dr, 20742, College Park, United States
[b] Department of Mathematics, University of Maryland, 4176 Campus Dr, 20742, College Park, United States
[c] Department of Electrical and Computer Engineering, University of Maryland, 8223 Paint Branch Dr, 20742, College Park, United States
[d] Hillsdale College, 33 E College St, 49242, Hillsdale, United States
[e] Potomac Research LLC, 801 N Pitt St, 22341, Alexandria, United States

ARTICLE INFO

Dataset link: https://github.com/awikner/res-noise-stabilization

Keywords:
Chaotic dynamics
Prediction
Climate
Stability
Reservoir computing
Regularization

ABSTRACT

Recent work has shown that machine learning (ML) models can skillfully forecast the dynamics of unknown chaotic systems. Short-term predictions of the state evolution and long-term predictions of the statistical patterns of the dynamics ("climate") can be produced by employing a feedback loop, whereby the model is trained to predict forward only one time step, then the model output is used as input for multiple time steps. In the absence of mitigating techniques, however, this feedback can result in artificially rapid error growth ("instability"). One established mitigating technique is to add noise to the ML model training input. Based on this technique, we formulate a new penalty term in the loss function for ML models with memory of past inputs that *deterministically* approximates the effect of many small, independent noise realizations added to the model input during training. We refer to this penalty and the resulting regularization as Linearized Multi-Noise Training (LMNT). We systematically examine the effect of LMNT, input noise, and other established regularization techniques in a case study using reservoir computing, a machine learning method using recurrent neural networks, to predict the spatiotemporal chaotic Kuramoto–Sivashinsky equation. We find that reservoir computers trained with noise or with LMNT produce climate predictions that appear to be indefinitely stable and have a climate very similar to the true system, while the short-term forecasts are substantially more accurate than those trained with other regularization techniques. Finally, we show the deterministic aspect of our LMNT regularization facilitates fast reservoir computer regularization hyperparameter tuning.

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 (Arcomano et al., 2020; Bi et al., 2023; Lam et al., 2023; Pathak et al., 2022; Rasp et al., 2020; Rasp & Thuerey, 2021). 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 Balakrishnan & Upadhyay, 2020; Pathak, Hunt, Girvan, Lu, & Ott, 2018; Vlachas, Byeon, Wan, Sapsis, & Koumoutsakos, 2018; Vlachas et al., 2020), 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" (Gentine, Pritchard, Rasp, Reinaudi, & Yacalis, 2018; Rasp, Pritchard, & Gentine, 2018), 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 Bi et al. (2023), Chattopadhyay, Mustafa, Hassanzadeh, Bach, and Kashinath (2022), Lam et al. (2023), Scher and Messori (2019) and Vlachas et al. (2020). We call this situation a "climate instability"; such an instability might or might not degrade the accuracy of a short term forecast.

* Corresponding author.
E-mail address: awikner1@umd.edu (A. Wikner).
[1] Joseph Harvey is currently employed at Aunalytics in South Bend, IN 46601.

https://doi.org/10.1016/j.neunet.2023.10.054
Received 12 December 2022; Received in revised form 19 October 2023; Accepted 30 October 2023
Available online 7 November 2023

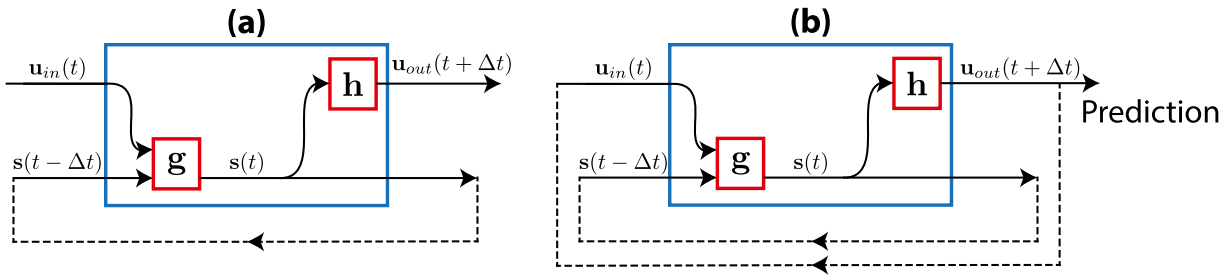

Fig. 1. Machine Learning Model with Memory Used for Predicting Dynamics. The above diagrams show an ML model with memory for prediction in the (a) "open-loop" configuration (for training) and (b) "closed-loop" configuration (for prediction). In panels (a) and (b), the dashed line indicates that the output is fed back into the model as the next input.

1.1. Machine learning prediction of dynamics and the issue of climate stability

In this article, 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. 1. The model output is given by the memory evolution function, $\mathbf{g}$, and the output function, $\mathbf{h}$, as follows:

$$\begin{aligned}\mathbf{s}(t) &= \mathbf{g}(\mathbf{u}_{in}(t), \mathbf{s}(t-\Delta t)),\\ \mathbf{u}_{out}(t+\Delta t) &= \mathbf{h}(\mathbf{s}(t)).\end{aligned} \tag{1}$$

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. 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. 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. 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 $n = 1, 2, 3, \ldots$, 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 Arcomano et al. (2020), Li et al. (2021), Pathak et al. (2018, 2022), Rasp et al. (2020), Rasp and Thuerey (2021) and Vlachas et al. (2018, 2020) 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 Auslander, Bhatia, & Seibert, 1964), the closed-loop ML model plausibly has an invariant set that approximates this attractor (Lu, Hunt, & Ott, 2018). 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 (Pathak, Lu, Hunt, Girvan, & Ott, 2017). 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 $\{\mathbf{u}(t)\}$ (Lu et al., 2018). 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 Pathak et al., 2017). 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.

1.2. Stabilization

One of our primary goals in this article 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 (An, 1996; Goodfellow, Bengio, & Courville, 2016; Greff, Srivastava, Koutník, Steunebrink, & Schmidhuber, 2017; Poole, Sohl-Dickstein, & Ganguli, 2014; Sietsma & Dow, 1991; Vincent, Larochelle, Bengio, & Manzagol, 2008). The addition of noise as regularization is also an essential component of NARMAX models for nonlinear system identification (Billings, 2013). Adding input noise is a commonly used technique in reservoir computing (see, e.g., Jaeger, 2001; Lukoševičius & Jaeger, 2009; Vlachas et al., 2020). 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. 1(a)), while the prediction uses the closed-loop system (Fig. 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.

1.3. Outline and main results

Our article is structured as follows: In Section 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 Section 2.1 our implementation of reservoir computing using a recurrent neural network (RNN), describe the process of training in Section 2.2, and discuss different regularization techniques, whose performance we will test, in Section 2.3. We then describe our LMNT technique and the resulting regularization in Section 2.4. In Section 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 Section 3. We discuss alternative implementations of LMNT and give concluding remarks in Section 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. 1(a). Since this relates to the subject of this article, we discuss some of this past background. These techniques include regularization, such as Tikhonov regularization (Tikhonov & Arsenin, 1977) (also known as ridge regression), LASSO regression (Tibshirani, 1996), and Jacobian regularization (Hoffman, Roberts, & Yaida, 2019). Training using a dropout scheme (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014) is another method which introduces random noise to improve model robustness. In RNN models, teacher forcing (Kolen & Kremer, 2001), the more sophisticated professor forcing (Lamb et al., 2016), or the online weight adjustment technique known as FORCE learning (Sussillo & Abbott, 2009) 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 (Lim, Erichson, Hodgkinson, & Mahoney, 2021). In the case where an ML prediction model runs in a closed-loop configuration (as in our Fig. 1(b)), using a loss function that incorporates multiple feedback loops, which can be done simultaneously, as in Lam et al. (2023), Rasp et al. (2020), or iteratively, as in Pathak et al. (2022) (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 (Arcomano et al., 2022; Wikner et al., 2020). 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.

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, $\mathbf{u}(t) = [u_1(t), u_2(t), \ldots, u_M(t)]^\intercal$, obtained from time t_0 to time t_0+t_{train} with a sampling time-step of Δt. We standardize $\mathbf{u}$ by applying, for each k, a linear transformation to the kth 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. 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 $t_0 \le t \le t_0 + t_{\text{train}} - \Delta t$. 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. 1(b). After receiving input up to time $t_{init} \ge t_0 + t_{train}$ 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. 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.

2.1. Reservoir computing

We next describe the particular ML technique with memory used in this article, 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 article, 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 ith element of the reservoir state vector $\mathbf{r}(t)$ represents the scalar state of the ith 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 ($\mathbf{A} \to [\text{constant}]\times \mathbf{A}$) 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 (Jaeger, 2001) 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:

$$\mathbf{r}(t) = (1 - \alpha)\mathbf{r}(t - \Delta t) + \alpha \tanh(\mathbf{A}\mathbf{r}(t - \Delta t) + \mathbf{B}\mathbf{u}_{in}(t) + \mathbf{C}), \tag{2}$$

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:

$$\mathbf{s}(t) = \mathbf{f}(\mathbf{r}(t), \mathbf{u}_{in}(t)) = \begin{bmatrix} 1 \\ \mathbf{u}_{in}(t) \\ \mathbf{r}(t) \\ \mathbf{r}(t)^2 \end{bmatrix}. \tag{3}$$

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

$$\mathbf{u}_{out}(t + \Delta t) = \mathbf{h}(\mathbf{s}(t)) = \mathbf{W}\mathbf{s}(t). \tag{4}$$

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 Pathak et al. (2018), 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 Sections 2.2–2.4) are "hyperparameters" (see Section 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 Section 2.3.

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

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:

$$\begin{aligned} \{\mathbf{u}(t)\} = \{\, & \mathbf{u}(0), \mathbf{u}(\Delta t), \dots, \\ & \mathbf{u}((T_{sync} + T_{train} - 1)\Delta t), \mathbf{u}((T_{sync} + T_{train})\Delta t) \,\}. \end{aligned} \tag{5}$$

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:

$$\mathbf{s}(t) = \mathbf{g}_o(\mathbf{u}(t), \mathbf{s}(t - \Delta t)) = \begin{bmatrix} 1 \\ \mathbf{u}(t) \\ \mathbf{g}_r(\mathbf{u}(t), \mathbf{s}(t - \Delta t)) \\ \left(\mathbf{g}_r(\mathbf{u}(t), \mathbf{s}(t - \Delta t))\right)^2 \end{bmatrix}, \tag{6}$$

where

$$\begin{aligned} \mathbf{g}_r(\mathbf{u}(t), \mathbf{s}(t - \Delta t)) = {} & (1 - \alpha)\left[\mathbf{0}_{N\times(1+M)}, \mathbf{I}_{N\times N}, \mathbf{0}_{N\times N}\right]\mathbf{s}(t - \Delta t) \\ & + \alpha \tanh\left(\left[\mathbf{0}_{N\times(1+M)}, \mathbf{A}, \mathbf{0}_{N\times N}\right]\mathbf{s}(t - \Delta t) + \mathbf{B}\mathbf{u}(t) + \mathbf{C}\right). \end{aligned} \tag{7}$$

In Eq. (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 (Pecora & Carroll, 2015). 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, $0 \le j \le T_{train} - 1$ 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:

$$\ell(\mathbf{W}) = \frac{1}{T_{train}} \sum_{j=0}^{T_{train}-1} \|\mathbf{W}\mathbf{s}_j - \mathbf{v}_j\|_2^2 + \sum_i \beta_i \mathrm{Tr}\,(\mathbf{W}\mathbf{R}_i\mathbf{W}^\intercal). \tag{8}$$

In Eq. (8) and in future equations, $\mathrm{Tr}(\dots)$ 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 article and their associated penalties in the loss function in Sections 2.3–2.4. 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 jth 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. (8) with respect to $\mathbf{W}$ to zero:

$$\mathbf{W}\left(\frac{1}{T_{train}}\mathbf{S}\mathbf{S}^\intercal + \sum_i \beta_i \mathbf{R}_i\right) = \frac{1}{T_{train}}\mathbf{V}\mathbf{S}^\intercal. \tag{9}$$

While all of the regularization methods which we use in this article may be incorporated into the matrix solution, we note that this is not the case for all regularization types (e.g., LASSO regularization Tibshirani, 1996). In such cases, it would be necessary to use a method other than the matrix solution to minimize Eq. (8) (e.g., proximal gradient methods for LASSO Daubechies, Defrise, & De Mol, 2004). We solve for $\mathbf{W}$ in Eq. (9) using the `gesv` function in LAPACK (Anderson et al., 1999), implemented via `numpy.linalg.solve` (Harris et al., 2020) in Python.

2.3. Regularization

We now discuss the regularization techniques to improve an ML model's climate stability that we will test in this article. 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. (8). Noise training, which replaces the vectors $\mathbf{s}_j$ in Eq. (8) with perturbed vectors $\tilde{\mathbf{s}}_j$, is the one exception to this general form discussed in this article.

2.3.1. Tikhonov regularization

Tikhonov regularization (Tikhonov & Arsenin, 1977), 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 article, 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

$$\beta_T \|\mathbf{W}\|_F^2 = \beta_T \mathrm{Tr}\,(\mathbf{W}\mathbf{W}^\intercal) \tag{10}$$

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. (8), we define a Tikhonov regularization matrix as

$$\mathbf{R}_T = \mathbf{I}_{(1+M+2N)\times(1+M+2N)}, \tag{11}$$

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

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 (Hoffman et al., 2019). 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. 1 and Eq. (1), the Jacobian regularization penalty term for a single training sample can be expressed as

$$\beta_J \|\nabla_{\mathbf{s}}\mathbf{h}(\mathbf{s}_j)\nabla_{\mathbf{u}}\mathbf{g}(\mathbf{u}_j, \mathbf{s}_{j-1})\|_F^2. \tag{12}$$

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

$$\frac{\beta_J}{T_{train}-1}\sum_{j=1}^{T_{train}-1} \|\mathbf{W}\nabla_{\mathbf{u}}\mathbf{g}_o(\mathbf{u}_j, \mathbf{s}_{j-1})\|_F^2 = \beta_J \mathrm{Tr}\,(\mathbf{W}\mathbf{R}_\mathbf{J}\mathbf{W}^\intercal), \tag{13}$$

where the Jacobian regularization matrix is

$$\mathbf{R}_\mathbf{J} = \frac{1}{T_{train}-1}\sum_{j=1}^{T_{train}-1} \nabla_{\mathbf{u}}\mathbf{g}_o(\mathbf{u}_j, \mathbf{s}_{j-1},)\nabla_{\mathbf{u}}\mathbf{g}_o(\mathbf{u}_j, \mathbf{s}_{j-1})^\intercal. \tag{14}$$

We have introduced a normalizing factor of $1/(T_{train}-1)$ so that the scaling of $\mathbf{R}_\mathbf{J}$ is approximately independent of T_{train}. We compute $\nabla_{\mathbf{u}}\mathbf{g}_o(\mathbf{u}_j, \mathbf{s}_{j-1})$ from Eqs. (6) and (7) as follows.

$$\nabla_{\mathbf{u}}\mathbf{g}_o(\mathbf{u}_j, \mathbf{s}_{j-1}) = \begin{bmatrix} 0 \\ \mathbf{I}_{M\times M} \\ \alpha\,\mathrm{diag}(\mathbf{d}(\mathbf{r}_{j-1}, \mathbf{u}_j))\mathbf{B} \\ \alpha\,\mathrm{diag}(2\mathbf{r}_j)\,\mathrm{diag}(\mathbf{d}(\mathbf{r}_{j-1}, \mathbf{u}_j))\mathbf{B} \end{bmatrix}, \quad \text{where} \tag{15}$$

$$\mathrm{diag}(\mathbf{d}) = \begin{bmatrix} d_1 & & 0 \\ & \ddots & \\ 0 & & d_N \end{bmatrix}, \quad \text{and} \tag{16}$$

$$\mathbf{d}(\mathbf{r}_{j-1}, \mathbf{u}_j) = \quad \mathrm{sech}^2\,(\mathbf{A}\mathbf{r}_{j-1} + \mathbf{B}\mathbf{u}_j + \mathbf{C}), \tag{17}$$

where the sech^2 in Eq. (17) is the derivative of $\tanh$. We note that Eq. (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 $\mathbf{u}_j$.

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:

$$\mathbf{u}_{in}(t) = \mathbf{u}(t) + \sqrt{\beta_N}\boldsymbol{\gamma}(t), \tag{18}$$

where $\boldsymbol{\gamma}(t)$ is the noise vector and β_N is the noise variance. We generate the set of noise vectors, $\{\boldsymbol{\gamma}_j\}$, to be added during training for each of the M components of $\boldsymbol{\gamma}(t)$ 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:

$$\ell(\mathbf{W}) = \frac{1}{T_{train}}\sum_{j=1}^{T_{train}} \|\mathbf{W}\tilde{\mathbf{s}}_j - \mathbf{v}_j\|_2^2 + \sum_i \beta_i \mathrm{Tr}\,(\mathbf{W}\mathbf{R}_i\mathbf{W}^\intercal), \tag{19}$$

where we include potential additional regularization terms as well. In practice, Tikhonov regularization is often used in conjunction with noise training. We minimize Eq. (19) as we did Eq. (8): by constructing a matrix $\tilde{\mathbf{S}}$, where the jth column of $\tilde{\mathbf{S}}$ is $\tilde{\mathbf{s}}_j$, and solving the following linear system:

$$\mathbf{W}\left(\frac{1}{T_{train}}\tilde{\mathbf{S}}\tilde{\mathbf{S}}^\intercal + \sum_i \beta_i\mathbf{R}_i\right) = \frac{1}{T_{train}}\mathbf{V}\tilde{\mathbf{S}}^\intercal. \tag{20}$$

We note that while we can solve Eq. (20) using the same method used for Eq. (9), changing the noise scaling β_N requires us to re-compute the matrix $\tilde{\mathbf{S}}$ before we can solve Eq. (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. (9) again. This re-computation generally makes tuning the noise scaling more computationally costly than the other regularization parameters.

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. (1) as

$$\ell_{LMNT} = \beta_L \frac{1}{T_{train}-K}\sum_{j=K}^{T_{train}-1}\sum_{k=j-K+1}^{j} \|\nabla_{\mathbf{s}}\mathbf{h}(\mathbf{s}_j)\nabla_{\mathbf{u}}(j,k)\|_F^2, \tag{21}$$

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 $k \le j$:

$$\begin{aligned}\nabla_{\mathbf{u}}(j,k) = \nabla_{\mathbf{s}}\mathbf{g}(\mathbf{u}_j, \mathbf{s}_{j-1})\nabla_{\mathbf{s}}\mathbf{g}(\mathbf{u}_{j-1}, \mathbf{s}_{j-2})\dots \\ \nabla_{\mathbf{s}}\mathbf{g}(\mathbf{u}_{k+1}, \mathbf{s}_k)\nabla_{\mathbf{u}}\mathbf{g}(\mathbf{u}_k, \mathbf{s}_{k-1}).\end{aligned} \tag{22}$$

Here $\{\mathbf{u}_j\}$ and $\{\mathbf{s}_j\}$ are the training inputs and the resulting memory vectors, as in Section 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 Section 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 Section 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 $\{\boldsymbol{\gamma}_{i,1}\}$; then, beginning the computation again, we compute the memory vectors for another set of noise vectors $\{\boldsymbol{\gamma}_{i,2}\}$; 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

$$\ell(\mathbf{W}) = \frac{1}{P}\sum_{p=1}^{P}\frac{1}{T_{train}}\sum_{j=0}^{T_{train}-1} \|\mathbf{W}\tilde{\mathbf{s}}_{j,p} - \mathbf{v}_j\|_2^2 \tag{23}$$

where we have added a factor of $1/P$ to normalize the loss function to have values similar to Eq. (8). We perform a bias–variance decomposition (James, Witten, Hastie, & Tibshirani, 2014) on the loss function,

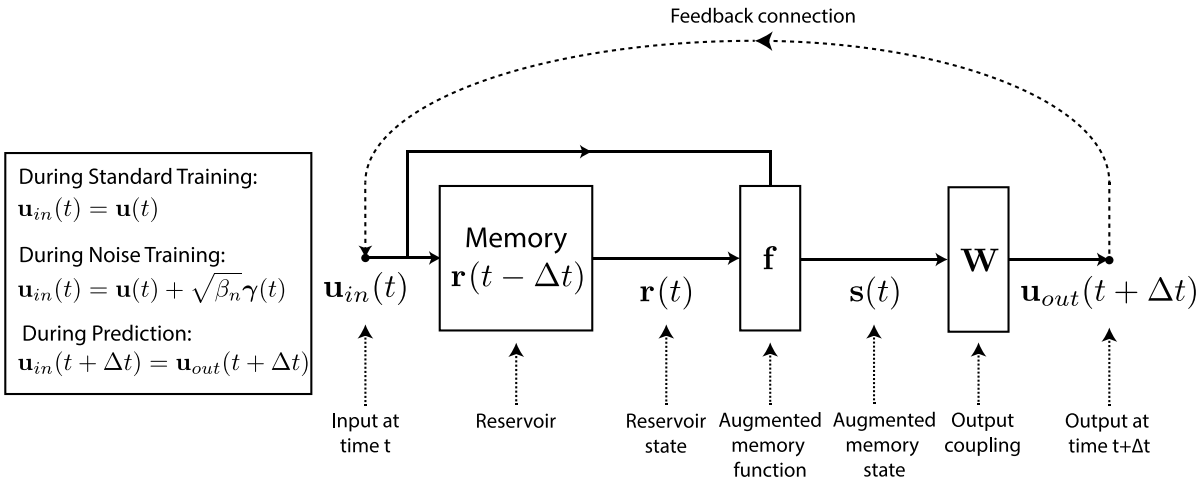

Fig. 2. Diagram of our Reservoir Computer Model. The reservoir is trained in the "open-loop" training phase (analogous to Fig. 1(a)). During this phase, the dashed line representing the feedback loop is inactive. Once training is complete, the "closed-loop" prediction phase (analogous to Fig. 1(b)) may 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 Section 2.2) and for noise training (discussed in Section 2.3.3), as well as the feedback input during prediction (discussed in Section 2.5).

obtaining:

$$\frac{1}{P}\sum_{p=1}^{P}\frac{1}{T_{train}}\sum_{j=0}^{T_{train}-1}\|\mathbf{W}\tilde{\mathbf{s}}_{j,p}-\mathbf{v}_j\|_2^2 = \underbrace{\frac{1}{T_{train}}\sum_{j=0}^{T_{train}-1}\|\mathbf{W}\bar{\mathbf{s}}_j-\mathbf{v}_j\|_2^2}_{\text{bias}} + \underbrace{\frac{1}{P}\sum_{p=1}^{P}\frac{1}{T_{train}}\sum_{j=0}^{T_{train}-1}\|\mathbf{W}\mathbf{q}_{j,p}\|_2^2}_{\text{variance}} \tag{24}$$

In Eq. (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. (24) by assuming that: (a) $P \to \infty$; (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. (24) as:

$$\begin{aligned}&\lim_{P\to\infty}\frac{1}{P}\sum_{p=1}^{P}\frac{1}{T_{train}}\sum_{j=0}^{T_{train}-1}\|\mathbf{W}\mathbf{q}_{j,p}\|_2^2\\ &=\lim_{P\to\infty}\frac{1}{T_{train}}\sum_{j=0}^{T_{train}-1}\mathrm{Tr}\left[\mathbf{W}\Big(\frac{1}{P}\sum_{p=1}^{P}\mathbf{q}_{j,p}\mathbf{q}_{j,p}^{\intercal}\Big)\mathbf{W}^{\intercal}\right]\\ &=\frac{1}{T_{train}}\sum_{j=0}^{T_{train}-1}\mathrm{Tr}[\mathbf{W}\boldsymbol{\Sigma}_j\mathbf{W}^{\intercal}],\end{aligned} \tag{25}$$

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

$$\tilde{\mathbf{s}}_{j,p} \approx \mathbf{s}_j + \sqrt{\beta_N}\sum_{k=j-K+1}^{j}\nabla_{\mathbf{u}}(j,k)\boldsymbol{\gamma}_{k,p} + \mathcal{O}(\beta_N), \tag{26}$$

where $\mathbf{s}_j$ is the feature vector for noiseless training and $\nabla_{\mathbf{u}}(j,k)$ is computed from Eq. (21) with $\mathbf{g} = \mathbf{g}_o$ from Eq. (6). Then, since $\boldsymbol{\gamma}_{k,p}$ is chosen from a distribution with mean zero,

$$\lim_{P\to\infty}\bar{\mathbf{s}}_j = \mathbf{s}_j + \mathcal{O}(\beta_N), \tag{27}$$

and for large P,

$$\mathbf{q}_{j,p} \approx \sqrt{\beta_N}\sum_{k=j-K+1}^{j}\nabla_{\mathbf{u}}(j,k)\boldsymbol{\gamma}_{k,p} + \mathcal{O}(\beta_N). \tag{28}$$

We use Eq. (27) to approximate the bias term in Eq. (24), and Eq. (28) to approximate the variance term. We assume that the $\mathcal{O}(\beta_N)$ term in Eq. (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. (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 $\|\mathbf{W}\bar{\mathbf{s}}_j - \mathbf{v}_j\|_2^2 - \|\mathbf{W}\mathbf{s}_j - \mathbf{v}_j\|_2^2 \approx 2\mathbf{W}\mathcal{O}(\beta_N)(\mathbf{W}\mathbf{s}_j - \mathbf{v}_j)^{\intercal}$ after summing over j.

Next, since the noise vectors $\boldsymbol{\gamma}_{j,p}$ are independent of each other and each has covariance $\mathbf{I}$, we approximate $\boldsymbol{\Sigma}_j$ in Eq. (25) as

$$\boldsymbol{\Sigma}_j = \sum_{k=j-K+1}^{j}\nabla_{\mathbf{u}}(j,k)\big(\beta_N\mathbf{I}\big)\nabla_{\mathbf{u}}(j,k)^{\intercal} = \beta_N\sum_{k=j-K+1}^{j}\nabla_{\mathbf{u}}(j,k)\nabla_{\mathbf{u}}(j,k)^{\intercal}. \tag{29}$$

Combining Eqs. (25) and (22), we obtain the LMNT regularization function for the reservoir computer:

$$\beta_L\frac{1}{T_{train}-K}\sum_{j=K}^{T_{train}-1}\sum_{k=j-K+1}^{j}\|\mathbf{W}\nabla_{\mathbf{u}}(j,k)\|_F^2, \tag{30}$$

where we have replaced β_N by β_L. Notice that Eq. (30) agrees with Eq. (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. (22). In Eq. (22), $\mathbf{u}_k$ and $\mathbf{s}_k$ are computed without any addition of noise, making this regularization deterministic. We use Eq. (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. (6) and shown in Fig. 2:

$$\nabla_{\mathbf{s}}\mathbf{g}_o(\mathbf{u}_j,\mathbf{s}_{j-1}) = \begin{bmatrix}\mathbf{0}_{(1+M)\times(1+M+2N)}\\ \nabla_{\mathbf{s}}\mathbf{g}_r(\mathbf{s}_{j-1},\mathbf{u}_j)\\ \mathrm{diag}(2\mathbf{r}_j)\,\nabla_{\mathbf{s}}\mathbf{g}_r(\mathbf{s}_{j-1},\mathbf{u}_j)\end{bmatrix}, \quad \text{where} \tag{31}$$

$$\begin{aligned}\nabla_{\mathbf{s}}\mathbf{g}_r(\mathbf{s}_{j-1},\mathbf{u}_j) = {}& \alpha\,\mathrm{diag}(\mathbf{d}(\mathbf{r}_{j-1},\mathbf{u}_j))\begin{bmatrix}\mathbf{0}_{N\times(1+M)}, & \mathbf{A}, & \mathbf{0}_{N\times N}\end{bmatrix}\\ &+\begin{bmatrix}\mathbf{0}_{N\times(1+M)}, & (1-\alpha)\mathbf{I}_{N\times N}, & \mathbf{0}_{N\times N}\end{bmatrix}\end{aligned} \tag{32}$$

and where $\mathrm{diag}(\dots)$ and $\mathbf{d}(\mathbf{r}_{j-1},\mathbf{u}_j)$ are the same as in Eqs. (16) and (17), respectively. When computed using the inputs and internal reservoir states during training, the resulting regularization matrix is:

$$\mathbf{R}_L = \frac{1}{T_{train}-K}\sum_{j=K}^{T_{train}-1}\left[\sum_{k=j-K+1}^{j}\nabla_{\mathbf{u}}(j,k)\nabla_{\mathbf{u}}(j,k)^{\intercal}\right]. \tag{33}$$

In the case where $K = 1$, this regularization matrix is identical to the Jacobian regularization matrix, $\mathbf{R}_J$. For all results in this article, 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 Appendix A. 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 Appendix B.1, we discuss computing the LMNT regularization with a greatly reduced number of training samples, while in Appendix B.2, we compare prediction stability and accuracy when the total amount of training samples is greatly reduced.

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. 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. (3)). We then activate the feedback loop shown in Fig. 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

$$\mathbf{g}_c(\mathbf{s}(t-\Delta t))=\begin{bmatrix}1\\ \mathbf{W}\mathbf{s}(t-\Delta t)\\ \mathbf{g}_s(\mathbf{s}(t-\Delta t))\\ \left(\mathbf{g}_s(\mathbf{s}(t-\Delta t))\right)^2\end{bmatrix} \tag{34}$$

and

$$\begin{aligned}\mathbf{g}_s(\mathbf{s}(t-\Delta t))=&(1-\alpha)\left[\mathbf{0}_{N\times(1+M)},\ \mathbf{I}_{N\times N},\ \mathbf{0}_{N\times N}\right]\mathbf{s}(t-\Delta t)\\ &+\alpha\tanh\left(\left[\mathbf{0}_{N\times 1},\ \mathbf{BW},\ \mathbf{A},\ \mathbf{0}_{N\times N}\right]\mathbf{s}(t-\Delta t)+\mathbf{C}\right).\end{aligned} \tag{35}$$

The right side of Eq. (35) is obtained by substituting $\mathbf{u}(t)=\mathbf{W}\mathbf{s}(t+\Delta t)$ into the right side of Eq. (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
$$VT=\min_{T_{init}\Delta t\leq t\leq(T_{init}+T_{pred})\Delta t}\left\{t\,\middle|\,\frac{\|\mathbf{u}_{out}(t)-\mathbf{u}_{true}(t)\|_2}{\overline{E}}>\epsilon_{VT}\right\}-\Delta t. \tag{36}$$
Here, ϵ_{VT} is the valid time error threshold, and $\overline{E}$ 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 paper, 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
$$\epsilon_{map}(t)=\frac{\|\mathbf{u}_{out}(t)-\mathbf{F}(\mathbf{u}_{out}(t-\Delta t),t-\Delta t,t)\|_2}{\overline{E}_{map}}. \tag{37}$$
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,
$$\overline{E}_{map}=\overline{\|\mathbf{u}_{j+1}-\mathbf{u}_j\|_2}, \tag{38}$$
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 Section 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,
$$\epsilon_{map}^{max}=\max_{T_{init}\Delta t\leq t\leq(T_{init}+T_{pred})\Delta t}\epsilon_{map}(t). \tag{39}$$
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 (Welch, 1967).

3. Results

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 (Kuramoto, 1978; Sivashinsky, 1977) with periodic boundary conditions:

$$\frac{\partial y(x,t)}{\partial t} + y(x,t)\frac{\partial y(x,t)}{\partial x} + \frac{\partial^2 y(x,t)}{\partial x^2} + \frac{\partial^4 y(x,t)}{\partial x^4} = 0, \tag{40}$$

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 Section 2.5, we simulate Eq. (40) on a spatial grid consisting of 64 grid points equally-spaced at intervals of $\Delta x = L/64$ using the ETRK4 method (Cox & Matthews, 2002; Kassam & Trefethen, 2005) to integrate the system at a time step of $\Delta t = 0.25$. The resulting discretized system state is:

$$\mathbf{y}(t) = [y(0,t), y(\Delta x,t), \ldots, y((M-1)\Delta x,t)]^{\mathsf{T}}, \tag{41}$$

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. (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. (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 (Bennetin, Galgani, Giorgilli, & Strelcyn, 1980).

We have also done tests with other systems besides the Kuramoto–Sivashinsky equations (e.g., the chaotic Lorenz '63 model Lorenz, 1963). We do not report these results here, as they result in conclusions that coincide with what follows from our tests on Eq. (40).

3.2. Prediction test results

We now present the result of our predictions test on the Kuramoto–Sivashinsky equation. In order to test a scenario where climate stability is challenging, we have used a reservoir with 500 nodes (see Table 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.

Table 1
Reservoir computer hyperparameter values used for all prediction tests.

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

- 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.

3.2.1. Climate stability and valid time

We simulate the KS equation using reservoir computers with the hyperparameters listed in Table 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 data sets generated using Eq. (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 2. Panel (A) in Fig. 3 shows a histogram of the mean map error values of predictions generated from a reservoir computer trained using Jacobian and Tikhonov regularization using the regularization parameter values contained in Table 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. 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 as stable do reasonably reproduce the true system climate, as illustrated by the first 2 examples in Fig. 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 shows the results of reservoir computer predictions made using different regularization methods, ordered from worst to best

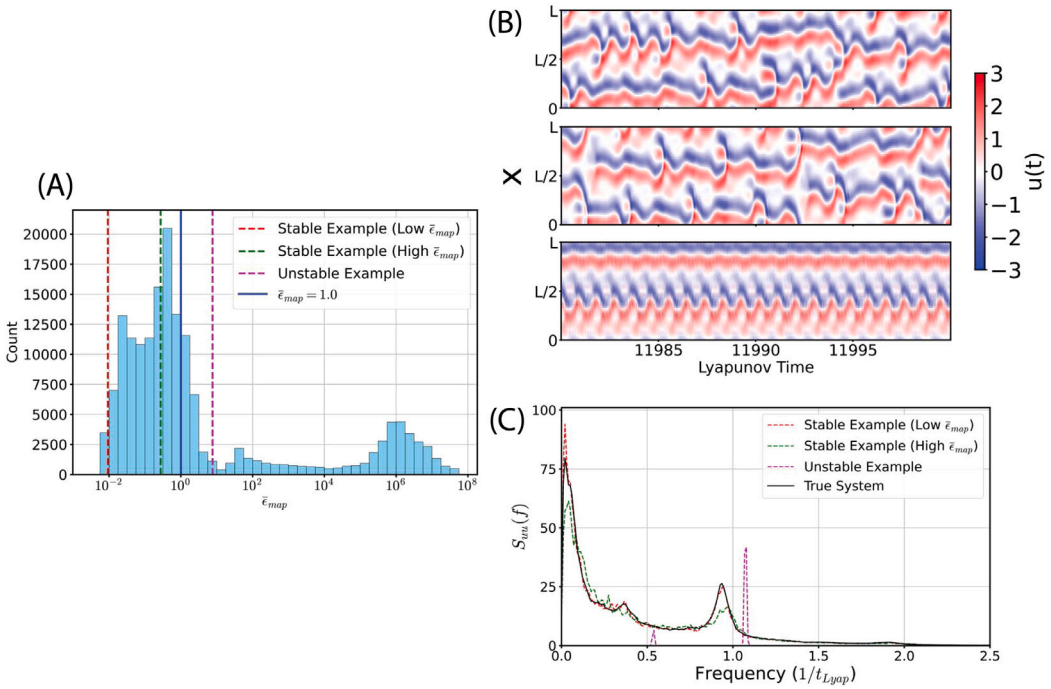

Fig. 3. Comparing Predictions with Increasing $\bar{\epsilon}_{map}$ Values. In this figure, panel (A) shows a histogram of the mean map error values of predictions generated from reservoir computers trained with Jacobian and Tikhonov regularization using a logarithmic grid of regularization parameter values (grid given in Table 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.

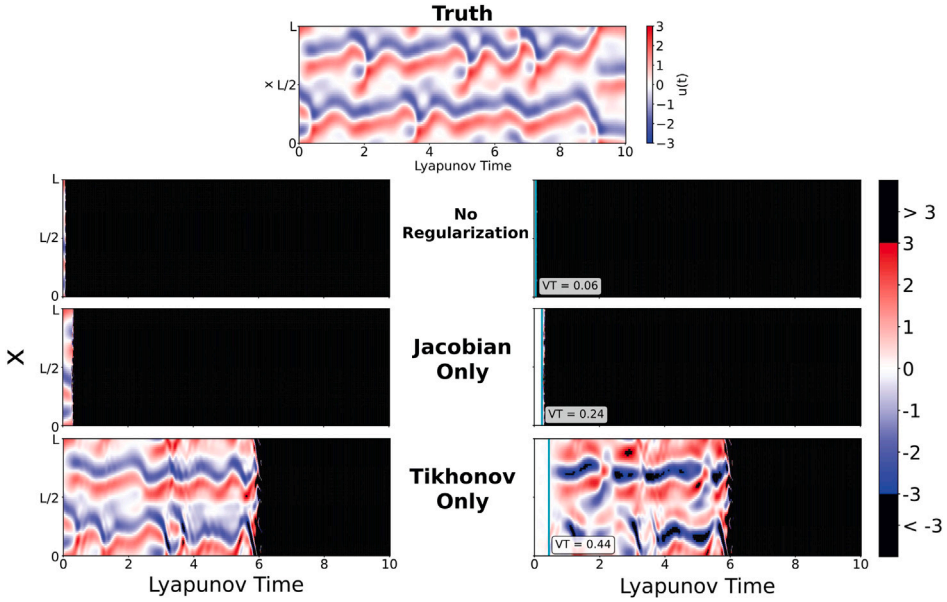

Fig. 4. Unstable Predictions of the KS Equation. The top panel shows a testing time series $\mathbf{u}_{true}(t)$ generated from the true KS equations dynamics, while the bottom panels show results from a reservoir computer trained with the regularization type displayed between the left and right panels. 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.

Table 2
Parameters used for all reservoir training and 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 3
The regularization parameters that are searched over for each reservoir prediction test.

Regularization parameter search grids	
Regularization type	Search grid
Tikhonov (β_T)	0 and 10^l for $l \in \{-18, -17.5, \ldots, -4.5, -4\}$
Jacobian (β_J)	10^l for $l \in \{-8, -7.8, \ldots, -4.2, -4\}$
Noise Training (β_N) and LMNT (β_L)	10^l for $l \in \{-8, -7.8, \ldots, -6.2, -6\}$

Table 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 ± error bounds indicate the maximum of the upper and lower 95% confidence intervals for the median (Conover, 1999). In the case of the valid time, we enlarge this interval slightly to account for the Δt discretization of our predictions.

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**	$\mathbf{4.24 \pm 0.04}$	$\mathbf{(2.77 \pm 0.02) \times 10^{-3}}$	$\mathbf{(2.02 \pm 0.04) \times 10^{-2}}$
LMNT ($K = 4$) and Tikhonov	$\beta_L = 10^{-7.4}$ $\beta_T = 10^{-16.5}$	**1000/1000**	$\mathbf{4.27 \pm 0.04}$	$\mathbf{(2.75 \pm 0.02) \times 10^{-3}}$	$\mathbf{(2.03 \pm 0.04) \times 10^{-2}}$

performance. Figs. 4 and 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. 5 are observed to be stable for the entire long prediction period of $192\, t_{Lyap}$. Fig. 6 shows the map error $\epsilon_{map}(t)$ during each of the predictions shown in Figs. 4 and 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.

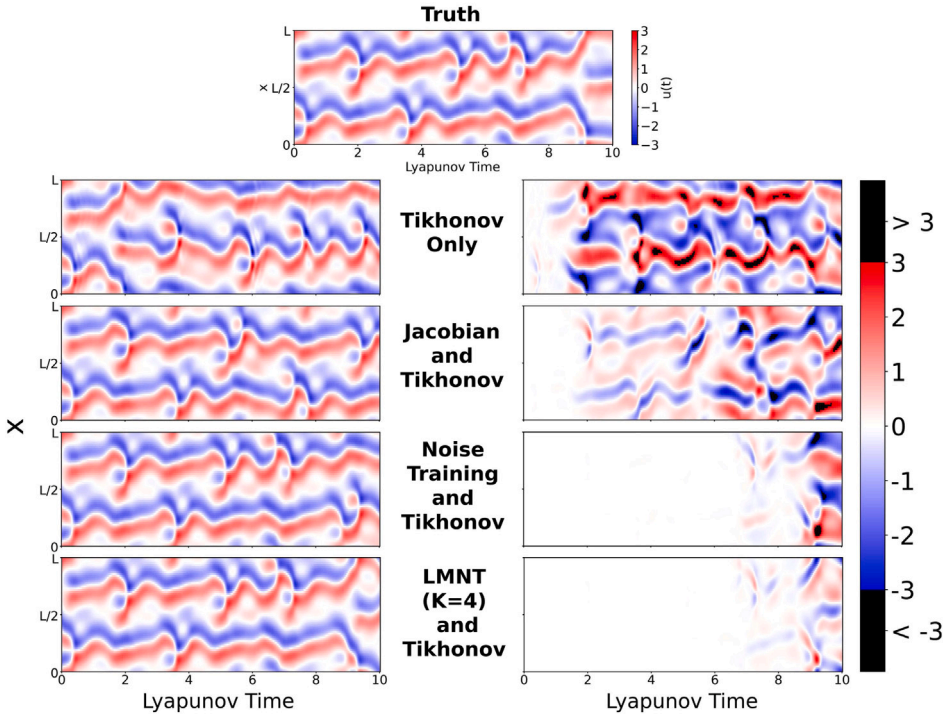

Fig. 5. Stable Predictions of the KS Equation. The top panel shows a testing time series $\mathbf{u}_{true}(t)$ generated from the true KS equations dynamics, while the bottom panels show results from a reservoir computer trained with the regularization type displayed between the left and right panels. 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.

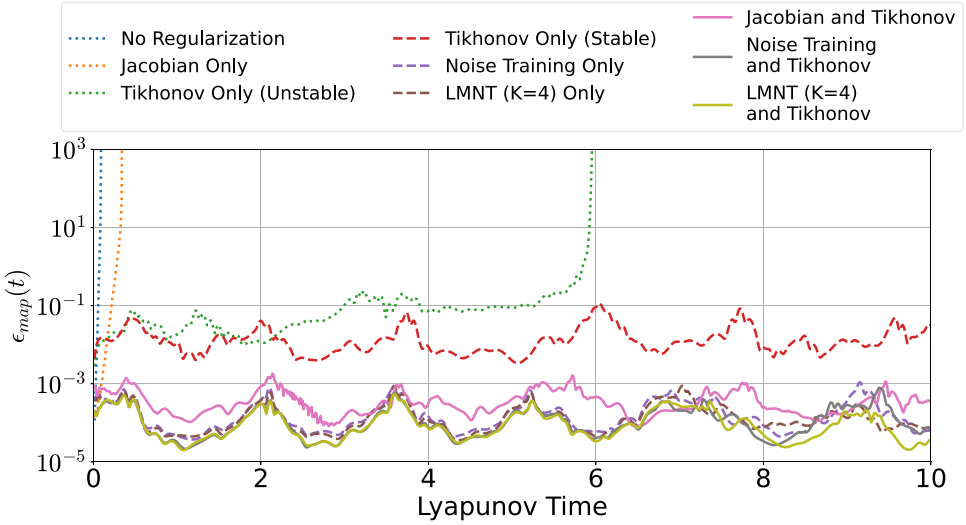

Fig. 6. Map Error over Time During Reservoir Predictions. This figure shows the map error ϵ_{map} during each of the unstable predictions shown in Figs. 4 and (dotted lines), stable predictions shown in Fig. 5 from reservoirs trained with a single regularization type (dashed lines), and stable predictions shown in Fig. 5 from reservoirs trained with multiple regularization types (solid lines).

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. 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. 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.

3.2.3. Regularization optimization

In this section, we show how the optimal regularization parameter values used to produce the results shown in Figs. 4–7 and in Table 4 are obtained. Fig. 8 displays, for two of the regularization methods, the

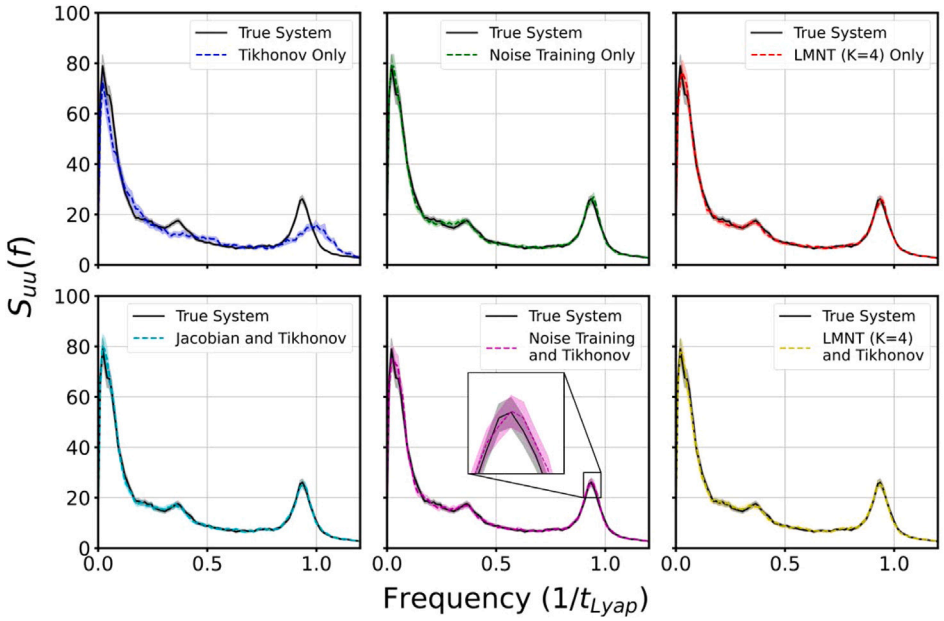

Fig. 7. Average Predicted PSD Compared to True PSD. The true KS equation PSD is computed over a single trajectory of duration T_{pred} = 10,000,000 (120,000 t_{Lyap}), while the prediction PSD is computed as the average over predictions of duration T_{pred} = 16,000 (192 t_{Lyap}) produced by the ensemble 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 $1/t_{Lyap}$.

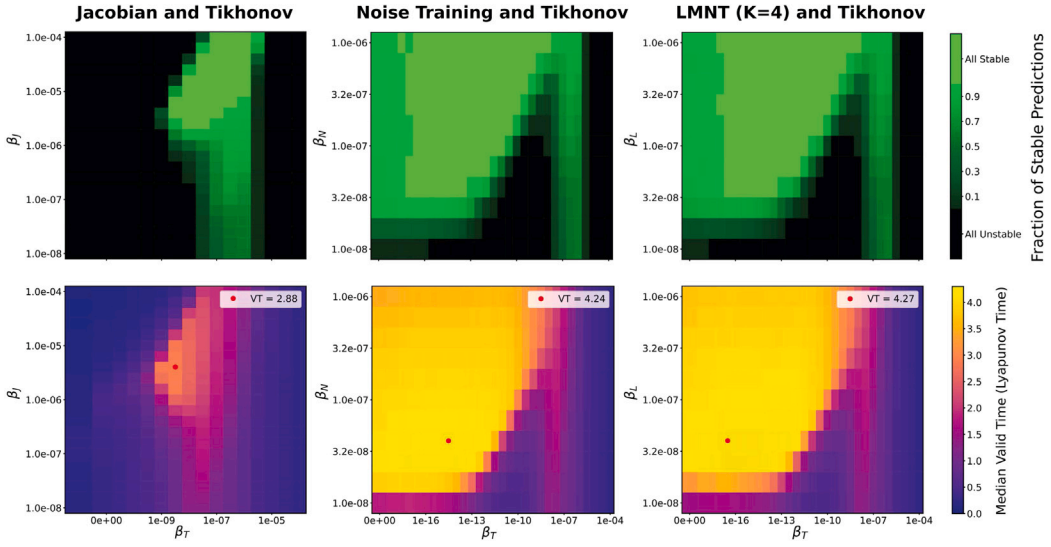

Fig. 8. Fraction of Stable Predictions and Median Valid Time using Varying Regularization Parameter Values. Color plots displaying the fraction of stable predictions and median prediction valid time over a grid of different regularization parameters values are shown on the top and bottom, respectively. 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.

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 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 discussed in Section 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. 8, the median prediction valid time over all of the predictions would increase; however, the top left panel in Fig. 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. Discussion and conclusion

In the absence of mitigating techniques, long-term forecasting using the machine learning approach shown in Fig. 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 climate 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. (21) and Appendix A). 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 Appendix B.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 article, we have implemented a version of the reduced training sample LMNT in the hybrid atmospheric model described in Arcomano et al. (2022). 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.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

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

Acknowledgments

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

Appendix A. 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 lienarized 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. (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.

As an example of the saturation in prediction performance as K increases, Table A.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 article.

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 S_j to be the result of the outer sum in Eq. (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 regularization matrix (Eq. (33)) used to train the reservoir computer.

Table A.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 caption.

LMNT test results					
Noise Steps, K	Regularization parameters	Fraction of stable predictions	Median valid time (t_{Lyap})	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	$(2.73 \pm 0.02) \times 10^{-3}$	$(2.02 \pm 0.02) \times 10^{-2}$

Algorithm 1: LMNT Penalty calculation during training

Data: Training inputs $\{\mathbf{u}_j\}$, $1 \le j \le T_{train}$;
Memory vectors during training $\{\mathbf{s}_j\}$, $0 \le j \le T_{train}$;
Noise approximation steps K
Result: LMNT Penalty term (Eq. (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. (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. (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 $\ell_{LMNT} = \frac{\beta_L}{T_{train}-K}\mathcal{S}_{T_{train}-1}$

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.

Appendix B. Comparing performance with reduced training data

B.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 Appendix A. 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

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

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

$$\mathbf{R}_{L,0} = \sum_{k=1}^{K} \nabla_{\mathbf{u},\mathbf{0}}(K,k)\nabla_{\mathbf{u},\mathbf{0}}(K,k)^{\intercal}. \quad \text{(B.2)}$$

In Eq. (B.2), $\nabla_{\mathbf{u},\mathbf{0}}(K,k)$ denotes evaluation of the right side of Eq. (22) at $\mathbf{u}_k = \mathbf{0}$ (the mean of our standardized training data) and $\mathbf{s}_{j-1} = \mathbf{s}_{j-2} = \cdots = \mathbf{s}_k = \mathbf{s}_{mean}$, where $\mathbf{s}_{mean}$ is the memory vector synchronized to the constant mean input.

Table B.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 matrix 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.

B.2. Reducing the total amount of 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. B.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. All reservoir hyperparameters, training, and prediction parameters are kept the same as

Table B.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 caption.

LMNT ($K = 4$) test results					
Training samples, T	Regularization parameters	Fraction of stable predictions	Median valid time (t_{Lyap})	Median $\bar{\epsilon}_{map}$	Median ϵ_{map}^{max}
Mean Input (Eq. (B.2))	$\beta_L = 10^{-7.4}$ $\beta_T = 10^{-15.5}$	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 B.7
The regularization parameters that are searched over for each reservoir prediction test using the reduced training data.

Regularization parameter search grids	
Regularization type	Search grid
Tikhonov (β_T)	0 and 10^l for $l \in \{-18, -17.5, \ldots, -1.5, -1\}$
Jacobian (β_J), Noise Training (β_N), and LMNT (β_L)	0 and 10^l for $l \in \{-14, -14 + 12/35, \ldots, -2 - 12/35, -2\}$

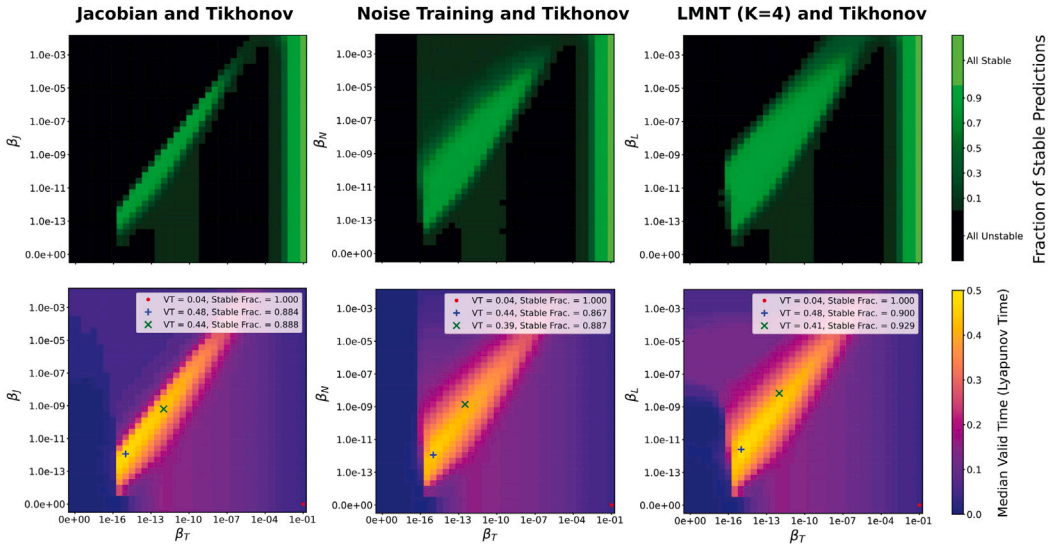

Fig. B.9. Fraction of Stable Predictions and Median Valid Time using Varying Regularization Parameter Values for Reduced Training Data . Color plots displaying the fraction of stable predictions and median prediction valid time over a grid of different regularization parameters values are shown on the 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.

in Tables 1 and 2, while the regularization parameter values tested are shown in Table B.7. 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 $0.5\, t_{Lyap}$. 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 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. B.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. B.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.

References

An, G. (1996). The effects of adding noise during backpropagation training on a generalization performance. *Neural Computation*, *8*(3), 643–674. http://dx.doi.org/10.1162/neco.1996.8.3.643.

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., et al. (1999). *LAPACK users' guide* (3rd ed.). Philadelphia, PA: Society for Industrial and Applied Mathematics.

Arcomano, T., Szunyogh, I., Pathak, J., Wikner, A., Hunt, B. R., & Ott, E. (2020). A machine learning-based global atmospheric forecast model. *Geophysical Research Letters*, *47*(9), Article e2020GL087776. http://dx.doi.org/10.1029/2020GL087776.

Arcomano, T., Szunyogh, I., Wikner, A., Pathak, J., Hunt, B. R., & Ott, E. (2022). 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), Article e2021MS002712. http://dx.doi.org/10.1029/2021MS002712.

Auslander, J., Bhatia, N. P., & Seibert, P. (1964). *Attractors in dynamical systems: Tech. rep. NASA-CR-59858*, NASA.

Balakrishnan, K., & Upadhyay, D. (2020). Deep adversarial koopman model for reaction-diffusion systems. arXiv:2006.05547 [cs, eess].

Bennetin, G., Galgani, L., Giorgilli, A., & Strelcyn, J.-M. (1980). Lyapunov characteristic exponents for smooth dynamical systems and for Hamiltonian systems: A method for computing all of them. *Meccanica*, *15*(9), 27.

Bi, K., Xie, L., Zhang, H., Chen, X., Gu, X., & Tian, Q. (2023). Accurate medium-range global weather forecasting with 3D neural networks. *Nature*, *619*(7970), 533–538. http://dx.doi.org/10.1038/s41586-023-06185-3.

Billings, S. A. (2013). *Nonlinear system identification: NARMAX methods in the time, frequency, and spatio-temporal domains*. John Wiley & Sons, Ltd.

Chattopadhyay, A., Mustafa, M., Hassanzadeh, P., Bach, E., & Kashinath, K. (2022). 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. http://dx.doi.org/10.5194/gmd-15-2221-2022.

Conover, W. J. (1999). *Practical nonparametric statistics* (3). New York, NY, USA: Wiley.

Cox, S. M., & Matthews, P. C. (2002). Exponential time differencing for stiff systems. *Journal of Computational Physics*, *176*(2), 430–455. http://dx.doi.org/10.1006/jcph.2002.6995.

Daubechies, I., Defrise, M., & De Mol, C. (2004). An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics*, *57*(11), 1413–1457. http://dx.doi.org/10.1002/cpa.20042.

Gentine, P., Pritchard, M., Rasp, S., Reinaudi, G., & Yacalis, G. (2018). Could machine learning break the convection parameterization deadlock? *Geophysical Research Letters*, *45*(11), 5742–5751. http://dx.doi.org/10.1029/2018GL078202.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2017). LSTM: a search space Odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, *28*(10), 2222–2232. http://dx.doi.org/10.1109/TNNLS.2016.2582924.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., et al. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. http://dx.doi.org/10.1038/s41586-020-2649-2.

Hoffman, J., Roberts, D. A., & Yaida, S. (2019). Robust learning with Jacobian regularization. http://dx.doi.org/10.48550/arXiv.1908.02729, arXiv:1908.02729.

Jaeger, H. (2001). *148, 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.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2014). An introduction to statistical learning: with applications in r. Springer New York.

Kassam, A.-K., & Trefethen, L. N. (2005). Fourth-order time-stepping for stiff PDEs. *SIAM Journal on Scientific Computing*, *26*(4), 1214–1233. http://dx.doi.org/10.1137/S1064827502410633.

Kolen, J. F., & Kremer, S. C. (2001). *A field guide to dynamical recurrent networks*. John Wiley & Sons.

Kuramoto, Y. (1978). Diffusion-induced chaos in reaction systems. *Progress of Theoretical Physics. Supplement*, *64*, 346–367. http://dx.doi.org/10.1143/PTPS.64.346.

Lam, R., Sanchez-Gonzalez, A., Willson, M., Wirnsberger, P., Fortunato, M., Alet, F., et al. (2023). GraphCast: learning skillful medium-range global weather forecasting. http://dx.doi.org/10.48550/arXiv.2212.12794, arXiv:2212.12794.

Lamb, A. M., Goyal, A., Zhang, Y., Zhang, S., Courville, A. C., & Bengio, Y. (2016). Professor forcing: a new algorithm for training recurrent networks. In *Advances in neural information processing systems, vol. 29*. Curran Associates, Inc..

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., et al. (2021). Fourier neural operator for parametric partial differential equations. http://dx.doi.org/10.48550/arXiv.2010.08895, arXiv:2010.08895.

Lim, S. H., Erichson, N. B., Hodgkinson, L., & Mahoney, M. W. (2021). Noisy recurrent neural networks. In *Advances in neural information processing systems, vol. 34* (pp. 5124–5137). Curran Associates, Inc..

Lorenz, E. N. (1963). Deterministic Nonperiodic Flow. *Journal of Atmospheric Sciences*, *20*(2), 130–141. http://dx.doi.org/10.1175/1520-0469(1963)020<0130:DNF>2.0.CO;2.

Lu, Z., Hunt, B. R., & Ott, E. (2018). Attractor reconstruction by machine learning. *Chaos. An Interdisciplinary Journal of Nonlinear Science*, *28*(6), Article 061104. http://dx.doi.org/10.1063/1.5039508.

Lukoševičius, M., & Jaeger, H. (2009). Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, *3*(3), 127–149. http://dx.doi.org/10.1016/j.cosrev.2009.03.005.

Pathak, J., Hunt, B., Girvan, M., Lu, Z., & Ott, E. (2018). Model-free prediction of large spatiotemporally chaotic systems from data: a reservoir computing approach. *Physical Review Letters*, *120*(2), Article 024102. http://dx.doi.org/10.1103/PhysRevLett.120.024102.

Pathak, J., Lu, Z., Hunt, B. R., Girvan, M., & Ott, E. (2017). Using machine learning to replicate chaotic attractors and calculate Lyapunov exponents from data. *Chaos. An Interdisciplinary Journal of Nonlinear Science*, *27*(12), Article 121102. http://dx.doi.org/10.1063/1.5010300.

Pathak, J., Subramanian, S., Harrington, P., Raja, S., Chattopadhyay, A., Mardani, M., et al. (2022). FourCastNet: a global data-driven high-resolution weather model using adaptive Fourier neural operators. http://dx.doi.org/10.48550/arXiv.2202.11214, arXiv:2202.11214.

Pecora, L. M., & Carroll, T. L. (2015). Synchronization of chaotic systems. *Chaos. An Interdisciplinary Journal of Nonlinear Science*, *25*(9), Article 097611. http://dx.doi.org/10.1063/1.4917383.

Poole, B., Sohl-Dickstein, J., & Ganguli, S. (2014). Analyzing noise in autoencoders and deep networks. http://dx.doi.org/10.48550/arXiv.1406.1831, arXiv:1406.1831.

Rasp, S., Dueben, P. D., Scher, S., Weyn, J. A., Mouatadid, S., & Thuerey, N. (2020). WeatherBench: a benchmark data set for data-driven weather forecasting. *Journal of Advances in Modeling Earth Systems*, *12*(11), Article e2020MS002203. http://dx.doi.org/10.1029/2020MS002203.

Rasp, S., Pritchard, M. S., & Gentine, P. (2018). Deep learning to represent subgrid processes in climate models. *Proceedings of the National Academy of Sciences*, *115*(39), 9684–9689. http://dx.doi.org/10.1073/pnas.1810286115.

Rasp, S., & Thuerey, N. (2021). 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), Article e2020MS002405. http://dx.doi.org/10.1029/2020MS002405.

Scher, S., & Messori, G. (2019). 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. http://dx.doi.org/10.5194/gmd-12-2797-2019.

Sietsma, J., & Dow, R. J. F. (1991). Creating artificial neural networks that generalize. *Neural Networks*, *4*(1), 67–79. http://dx.doi.org/10.1016/0893-6080(91)90033-2.

Sivashinsky, G. I. (1977). Nonlinear analysis of hydrodynamic instability in laminar flames—i. Derivation of basic equations. *Acta Astronautica*, *4*(11), 1177–1206. http://dx.doi.org/10.1016/0094-5765(77)90096-0.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, *15*(1), 1929–1958.

Sussillo, D., & Abbott, L. F. (2009). Generating coherent patterns of activity from chaotic neural networks. *Neuron*, *63*(4), 544–557. http://dx.doi.org/10.1016/j.neuron.2009.07.018.

Tibshirani, R. (1996). Regression Shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society. Series B. Statistical Methodology*, *58*(1), 267–288. http://dx.doi.org/10.1111/j.2517-6161.1996.tb02080.x.

Tikhonov, A., & Arsenin, V. (1977). *Scripta Series in Mathematics, Solutions of Ill-Posed Problems*. Winston.

Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on machine learning* ICML '08, (pp. 1096–1103). Helsinki, Finland: ACM Press, http://dx.doi.org/10.1145/1390156.1390294.

Vlachas, P. R., Byeon, W., Wan, Z. Y., Sapsis, T. P., & Koumoutsakos, P. (2018). 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), Article 20170844. http://dx.doi.org/10.1098/rspa.2017.0844.

Vlachas, P. R., Pathak, J., Hunt, B. R., Sapsis, T. P., Girvan, M., Ott, E., et al. (2020). Backpropagation algorithms and Reservoir Computing in Recurrent Neural Networks for the forecasting of complex spatiotemporal dynamics. *Neural Networks*, *126*, 191–217. http://dx.doi.org/10.1016/j.neunet.2020.02.016.

Welch, P. (1967). 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. http://dx.doi.org/10.1109/TAU.1967.1161901.

Wikner, A., Pathak, J., Hunt, B., Girvan, M., Arcomano, T., Szunyogh, I., et al. (2020). 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), Article 053111. http://dx.doi.org/10.1063/5.0005541.