Tree-based Learning for High-Fidelity Prediction of Chaos

Adam Giammarese,[1,*] Kamal Rana,[2] Erik M. Bollt,[3,4] and Nishant Malik[1]

[1] *School of Mathematics and Statistics, Rochester Institute of Technology, Rochester, NY 14623*
[2] *Chester F. Carlson Center for Imaging Science, Rochester Institute of Technology, Rochester, NY 14623*
[3] *Clarkson Center for Complex Systems Science, Clarkson University, Potsdam, NY 13699*
[4] *Department of Electrical and Computer Engineering, Clarkson University, Potsdam, NY 13699*

(Dated: March 22, 2024)

Model-free forecasting of the temporal evolution of chaotic systems is crucial but challenging. Existing solutions require hyperparameter tuning, significantly hindering their wider adoption. In this work, we introduce a tree-based approach not requiring hyperparameter tuning: TreeDOX. It uses time delay overembedding as explicit short-term memory and Extra-Trees Regressors to perform feature reduction and forecasting. We demonstrate the state-of-the-art performance of TreeDOX using the Hénon map, Lorenz and Kuramoto–Sivashinsky systems, and real-world Southern Oscillation Index.

Chaos is a ubiquitous phenomenon in nature, and its accurate prediction is challenging since it requires exact knowledge of the underlying system's governing equations and initial conditions. However, recent advances in machine learning techniques have made it possible to accurately predict the temporal evolution of chaotic systems in an entirely data-driven environment without requiring prior knowledge of the governing equations [1]. These *model-free* approaches represent a significant breakthrough in modeling complex systems and hold tremendous implications for various fields of science and technology [1–4]. In this context we present a new model-free method for forecasting chaos that outperforms existing approaches in accuracy, user-friendliness, and computational simplicity.

Deep learning techniques such as Recurrent Neural Networks (RNN) and, more specifically, Long Short-Term Memory (LSTM) provide substantial performance in forecasting chaotic systems, likely due to their ability to capture fading short-term memory [2, 5–8]. However, RNN and LSTM are both computationally expensive models, and they also require both the tuning of hyperparameters and a significant size of training data in order to provide accurate forecasts. Reservoir Computing (RC) lessens computational expense and eliminates the need for large training data sets by randomizing the recurrent pool of nodes inside an RNN (called the reservoir), which reduces training to a linear optimization problem [2, 8]. Despite the improvements to RNN and LSTM by RC, it still requires tuning numerous hyperparameters, including reservoir size, connectivity, and randomization rules.

The recently developed Next Generation Reservoir Computing (NG-RC) converts RC into a mathematically equivalent nonlinear vector autoregression (NVAR) machine, which further improves on RC by reducing the number of necessary hyperparameters and removing the need for a randomized reservoir [3, 9]. The NG-RC methodology, while promising, does not eliminate the need for hyperparameter tuning, rendering it unappealing for automated applications. Within machine learning tree-based methods are known for their versatility and robustness and involve only a few hyperparameters [10]. Furthermore, recent work demonstrates the benefits of tree-based machine learning models, such as XGBoost and Random Forests (RF), over deep learning models on tabular data, both in terms of accuracy and computational resources [11–14]. In this Letter, we introduce and investigate the efficacy of a tree-based alternative for learning chaos.

We name this tree-based method TreeDOX: **Tree**-based **D**elay **O**verembedded e**X**plicit memory learning of chaos. TreeDOX mimics the implicit fading short-term memory of RNN, LSTM, RC, and NG-RC via the use of explicit short-term memory in the form of time delay overembedding. Delay overmbedding differs from delay embedding due to its intentional usage of a higher embedding dimension than that suggested by Taken's theorem, which helps to model nonstationary dynamical systems [15–17]. TreeDOX uses an ensemble tree method called Extra Trees Regression (ETR), and leverages ETR's inherent ability to capture Gini feature importance to perform feature reduction on the delay overmbedding [18]. This reduces computational resource usage and improves generalizability. We demonstrate the efficacy of TreeDOX on a variety of chaotic systems, including the Hẽnon map, Lorenz and Kuramoto-Sivashinsky systems, and a real-world chaotic dataset: the Southern Oscillation Index (SOI). The development of TreeDOX is motivated by the desire for a user-friendly method of forecasting time series and spatiotemporal data. Ideally such a method would not be computationally expensive, would not require hyperparameter tuning, and would still retain comparable fidelity to contemporary forecasting methods, e.g. LSTM and NG-RC. While TreeDOX could certainly use the recently popular XGBoost form of tree-based regression, we opt rather for the classical Random Forest due to its typically successful performance under default

hyperparameter values, such as the number of trees in the ensemble. [19–21].

TreeDOX uses a variant of Random Forests called an Extra-Trees Regression (ETR). The ETR algorithm is an ensemble-based learning method in which each decision tree constructed using all training samples of the data, instead of bootstrapping. Unlike the standard Random Forest method, where all the attributes are used to determine the locally optimal split of a node, a random subset of features is selected for a node split in an ETR. For each feature in the random subset a set of random split values is generated within the range of the training data for the given feature, and a loss function (usually mean square error) is calculated for each random split. The feature-split pair that results in the lowest value of the given loss function is selected as the node. For every testing sample, each tree predicts the output value independently, and the final output value of the ETR is the mean of all predictions of the trees. Apart from calculating testing accuracy, ETRs also quantify the importance of each feature using mean decrease impurity and permutation importance. Another advantage to ETR is its resilience not only to correlated features but also to the value of hyperparameters such as the number of trees in the ensemble, minimum samples per leaf, or max depth of the individual trees. Oftentimes, the prescribed hyperparameters will achieve an acceptable regression; instead, the impact of such hyperparameters is felt in the space and time complexity of the algorithm, as training complexity scales linearly with both the number of trees and the number of variables seen while splitting. ETRs have two advantages over RFs: (1) lower time complexity and variance due to the randomized splits and (2) lower bias due to the lack of bootstrapping. While ETRs are more expensive to train compared to RC, they do not require an expensive grid search to find hyperparameters and tend to be just as fast when making predictions. However, unlike RC and other state-of-the-art methods ETRs do not contain explicit memory of system variables. Instead, the use of delay overembedding provides explicit memory to the model.

TreeDOX uses two such ETRs—one whose role is to calculate feature importances and another whose role is to perform predictions using reduced features. Before we formally introduce TreeDOX we will first define key concepts. First, assume that D-dimensional spatiotemporal data is in the following form: $X = \{\mathbf{x}_i\}_{i=1}^{t}$ where $\mathbf{x}_i \in \mathbb{R}^D$ and t is the length of the temporal component. A time delay overembedding of X will be denoted in the following manner: $DO(X, k, \xi) = \{[\mathbf{x}_i, \mathbf{x}_{i+\xi}, \ldots, \mathbf{x}_{i+(k-1)\xi}]\}_{i=1}^{t-(k-1)\xi}$ where $k \in \mathbb{N}$ is the overembedding dimension and $\xi \in \mathbb{N}$ is the time lag between observations in the time delay overembedding. We start by constructing the set of features and labels used in training, denoted by $F = \{\mathbf{f}_i\}_{i=1}^{t-(k-1)\xi}$ and $L = \{l_i\}_{i=1}^{t-(k-1)\xi}$, respectively. The i-th element of F, $\mathbf{f}_i$, is equivalent to the i-th element in the delay overembedding while $l_i = \mathbf{x}_{i+k\xi}$. After training the first ETR on (F, L) we request the feature importances $FI = \{FI_j\}_{j=1}^{kD}$ representing the Gini importance of the k dimensions in the time delay overembedding features. We introduce one hyperparameter, $p \in \mathbb{N}$, whose prescription is described later. We construct a set C which is filled with the indices of the p greatest elements of FI. C represents the columns of F we wish to use in final training since the ETR finds their respective time delays to hold the most predictive power. Next we construct a new set of reduced features, $F' = \{(\mathbf{f}_{ij})_{j\in C}\}_{i=1}^{t-(k-1)\xi}$, where $\mathbf{f}_{ij}$ represents the j-th column in the row vector $\mathbf{f}_i$. Lastly, we train the second ETR on (F', L) which benefits from increased generalizability due to the feature reduction.

At the beginning of the forecasting stage of TreeDOX, a vector $\mathbf{s} = [\mathbf{x}_{t-(k-1)\xi}, \mathbf{x}_{t-(k-1)\xi+1}, \ldots, \mathbf{x}_t]$ is initialized from the end of the training data and another vector $\mathbf{s}_{\text{delayed}} = (\mathbf{s}_j)_{j\in E} = [\mathbf{x}_{t-(k-1)\xi}, \mathbf{x}_{t-(k-2)\xi}, \ldots, \mathbf{x}_t]$ is collected from $\mathbf{s}$, where $E = \{1, 1+\xi, \ldots, 1+(k-1)\xi\}$. $\mathbf{s}_{\text{reduced}} = (\mathbf{s}_{\text{delayed},j})_{j\in C}$ is treated as a sample for the feature-reduced ETR from which a prediction of $\mathbf{x}_{t+1}$, $\tilde{\mathbf{x}}_{t+1}$ is extracted. $\mathbf{s}$ is updated to remove the first element and append the prediction $\tilde{\mathbf{x}}_{t+1}$ to the end. The updated $\mathbf{s}$ vector is used to repeat the process, hence the *self-evolutionary* nature of TreeDOX forecasting. Diagrams in Figs. S1 and S2 in the Supplementary Material (SM) summarize the training and forecasting stages of TreeDOX.

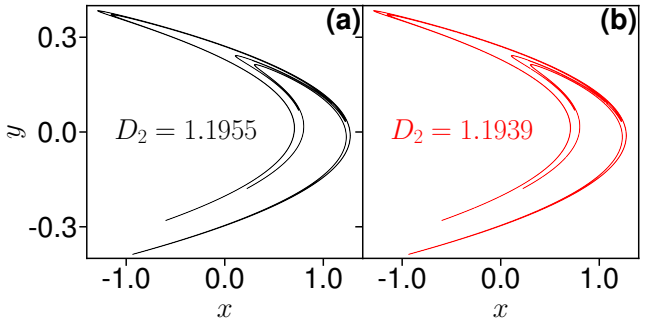

FIG. 1. (a,b) Test and predicted Hénon map attractors, respectively, with $a = 1.4$ and $b = 0.3$. Using $\xi = 1$ and a p-value of 0.05, TreeDOX selected $k = 8$ according to Eqn. 1. D_2 is the correlation dimension of the respective attractor. Here there were 100,000 and 60,000 training and testing samples used, respectively.

There exist three hyperparameters introduced in TreeDOX that are not otherwise prescribed by standard ETR implementations: dimension, k, and lag, ξ, of the delay overembedding, and the number of final features to use, p. To remove the responsibility of hyperparameter tuning from the TreeDOX user, we wish to leverage the

existing time series training data to prescribe the values of k, ξ, and p. To prescribe values to k and ξ we must investigate their role in the time delay overembedding. Since the time delay overembedding has dimension k and lag ξ one may consider that there are moving windows of length $(k-1)\xi$ (the earliest point in the time delay overembedding) used to train TreeDOX. We propose using Average Mutual Information (AMI) between the training data and its delayed copy shifted by τ states in order to determine how much information about the current state is stored in the previous states. By using a p-value (such as a standard 0.05 or 0.025) as a threshold, one may determine the critical value of $\tau_{i,crit}$ for which dimension i of a multidimensional time series does not contain significant information about the current state any longer. By choosing $(k-1)\xi$ to be the maximum value of the set of critical τ's, one may force the time delay overembedding to remain in the region where the system retains information about its next state. The choice of ξ made to specify a value for k from the prescribed $(k-1)\xi$ is a much more simple one: the smaller ξ, the more computational resources needed to train TreeDOX. Thus, if one has a powerful enough computer, one should choose $\xi = 1$ and thus $k = \max_{1\leq i\leq D}(\tau_{i,crit}) + 1$. Otherwise, choose a large enough ξ such that training is a reasonable task, and

$$k = \frac{1}{\xi}\max_{1\leq i\leq D}(\tau_{i,crit}) + 1. \tag{1}$$

Next we investigate the features fed to the ETR in order to prescribe a value of p. After training the ETR used in feature reduction we request the impurity-based feature importances (FIs) to be calculated, which ranks the k features according to their respective reduction of the specified loss function. It is important to note that FIs are normalized, meaning $\sum_{j=1}^{kD} FI_j = 1$, where FI_j is the feature importance of the j-th delay. Note that for an D-dimensional time series, there will be $k \cdot D$ features. We define a null rate of feature importance, $FI_0 = \frac{1}{kD}$. If all features were equally important their FI's would all be equal to this null rate, and otherwise it is guaranteed that some features will have greater importance and others will have less. Thus we select p to be the number of features whose importance is greater than or equal to FI_0. In all displayed TreeDOX results we use these suggested methods of prescribing k, ξ, and p. See Figs. S3 and S4 for visualizations of hyperparameter prescription in practice.

As a prototypical discrete chaotic system, the Hénon map serves to verify that TreeDOX can recreate chaotic dynamics. We generate training data from the Hénon Map, $(x_{n+1}, y_{n+1}) = (1 - ax_n^2 + y_n, bx_n)$, where $a = 1.4$ and $b = 0.3$ by evolving the iteration scheme 100,000 times with the initial condition $(x_0, y_0) = (0, 0)$ and removing 10 transient iterations, then use TreeDOX to predict the latter 60,000 test iterations. Using $\xi = 1$ and a p-value of 0.05, TreeDOX selected $k = 8$ total time delays according to Eqn. 1. Fig. 1 displays the trajectories of both the test data, $(x_n, y_n)_{n=100,001}^{160,000}$, and predicted data, $(\tilde{x}_n, \tilde{y}_n)_{n=100,001}^{160,000}$. We compare the complexity of the resulting chaotic attractors via correlation dimension, D_2 [22]. Figs. S5 and S6 show similar results for the Logistic Map.

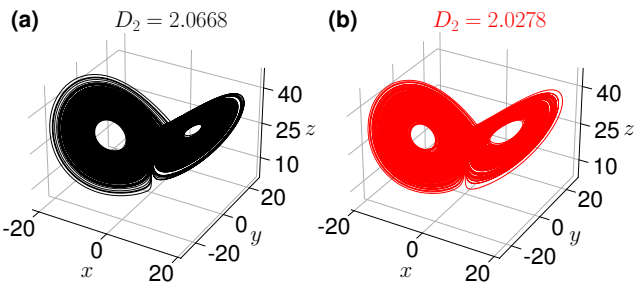

FIG. 2. (a,b) Test and predicted Lorenz system attractors, respectively, with $\sigma = 10$, $\beta = 8/3$, and $\rho = 28$. Using $\xi = 1$ and a p-value of 0.1, TreeDOX selected $k = 111$ according to Eqn. 1. D_2 is the correlation dimension of the respective attractor. Here there were 100,000 and 60,000 training and testing samples used, respectively.

The next benchmark we use is the Lorenz system, a prototypical continuous chaotic system: $(\dot{x}, \dot{y}, \dot{z}) = (\sigma(y - x), x(\rho - z) - y, xy - \beta z)$. Due to multiple nonlinear terms in the generating dynamics,

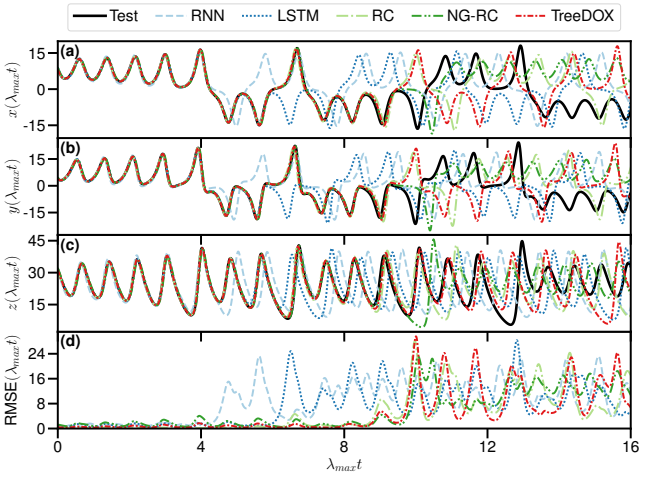

FIG. 3. Summary of Lorenz system forecasts with $\sigma = 10$, $\beta = 8/3$, $\rho = 28$, displayed using Lyapunov time, where $\lambda_{max} = 0.8739$. Using $\xi = 1$ and a p-value of 0.05, TreeDOX selected $k = 319$ according to Eqn. 1. Test data is black while RNN, LSTM, RC, NG-RC, and TreeDOX are light blue, blue, light green, green, and pink, respectively. (a,b,c) x, y, and z coordinates, respectively. (d) Root Mean Square Error (RMSE) of x, y and z forecasted versus test data. Here there were 87,815 and 2,184 training and testing samples used, respectively.

simple round-off error is enough to cause computational Lorenz system forecasts to diverge quickly, making the system difficult to numerically forecast. We select the typical parameters $\sigma = 10$, $\beta = 8/3$, and $\rho = 28$. For both Figs. 2 and 3 RK45 is used to generate training and testing data, where $[x_0, y_0, z_0] = [1, 1, 1]$ and 10,000 transient points are removed to produce signals on the chaotic attractor. Fig. 2 uses $dt = 0.005$ to produce more accurate correlation dimension estimations (both for test and predicted data), and uses 100,000 and 60,000 training and testing points, respectively. Fig. 3 uses $dt = 0.01$ and 87,815 and 2,184 training and testing points, respectively; after removing 10,000 transient points, 25 Lyapunov times worth of data is reserved for testing assuming a greatest Lyapunov exponent of $\lambda_{max} = 0.8739$ [23]. For clarity, Fig. 3 cuts off at 16 Lyapunov time. Fig. 2 demonstrates TreeDOX's ability to predict long term dynamics, while Fig. 3 portrays TreeDOX's self-evolved forecast accuracy in comparison to current methods.

Next, we test TreeDOX on a chaotic spatiotemporal system by forecasting the Kuramoto–Sivashinsky (KS) system: $u_t + u_{xxxx} + u_{xx} + uu_x = 0$, where $x \in [0, L]$. To generate the training and testing data, we used length $L = 22$ and $Q = 64$ grid points to discretize the domain, then used RK45 to evolve the system for 400,000 iterations with $\Delta t = 0.25$, random initial data, and periodic boundary conditions. After removing 2000 transient points, we use 97,441 points for training and 559 points for testing. This was done to produce 12.5 lyapunov time for the testing phase, assuming $\lambda_{max} = 0.043$ [24]. We use 1000 previous states as features for each grid point, resulting in a total of $k = 1000 \times 64 = 64000$ features, with a time delay of $\xi = 1$. Fig. 4 demonstrates promising results for predicting spatiotemporal time series such as the Kuramoto–Sivashinsky system, which are comparable to RC results [1].

To stress test TreeDOX for a real-world chaotic time-series we attempt to forecast the Southern Oscillation Index (SOI), a useful climate index with a relationship to the El Niño - La Niña climate phenomena, Walker circulation, drought, wave climate, and rainfall [25–31]. SOI is calculated as the z-score of the monthly mean sea level pressure between Tahiti and Darwin [25]. SOI encodes a high dimensional chaotic system into one dimension, resulting in unpredictable data which proves difficult for current models to predict.

Historically recorded SOI data features monthly values from January, 1866 to July, 2023 [25, 32]. January, 1866 to December, 1983 is reserved as training data, while the rest is used for testing, producing 1,416 and 465 training and testing points, respectively. See Fig. S7 for a visualization of the SOI data. Due to the difficult nature of forecasting SOI data TreeDOX and all other models tested are allowed to perform open-loop forecasting, meaning once a model predicts the next value, the correct test value is provided to the model before making the next prediction. To investigate the ability of TreeDOX and other models to make realistic forecasts each model is trained to predict a number of months in advance (denoted as 'lead' time) [33]. Fig. 5 displays the SOI prediction results for TreeDOX and other models. See Fig. S8 for results on an experiemnt of SOI predictions with varying training length, which demonstrate the beneficial accuracy and runtime scaling of TreeDOX in comparison with the other discussed models. Lastly,

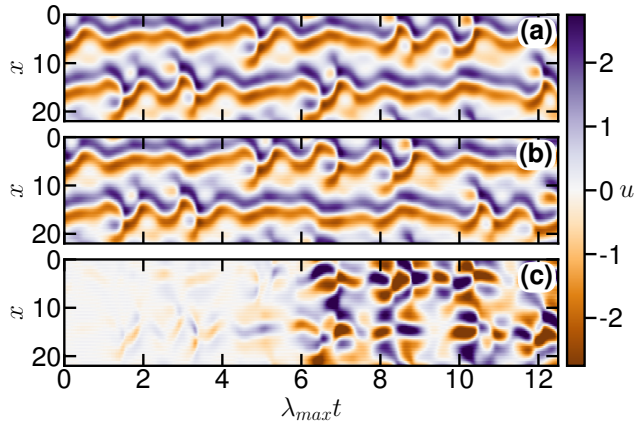

FIG. 4. Prediction of Kuramoto–Sivashinsky equation with $L = 22$ and $Q = 64$ grid points. The x-axis shows Lyapunov time, where $\lambda_{max} = 0.043$. (a,b) The test and forecasted dynamics, respectively. (c) The difference between the test and forecasted dynamics. Here there were 97,441 and 559 training and testing samples used, respectively.

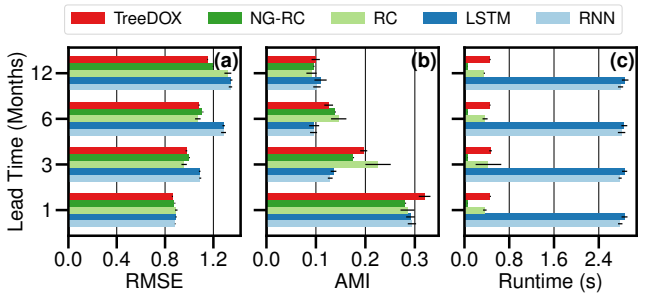

FIG. 5. Summary of SOI forecasts, where lead time is 1, 3, 6, and 12 months. RNN, LSTM, RC, NG-RC, and TreeDOX are light blue, blue, light green, green, and pink, respectively. Bars represent means and errorbars are ±1 standard deviation. With $\xi = 1$, TreeDOX selects $k = 28$ in the AMI-based prescription in Eqn. 1. (a) Root Mean Square Error (RMSE) between forecasted and test data. (b) Average Mutual Information (AMI) between forecasted and test data. (c) Runtime, in seconds, of combined model training and testing, where batch predictions were used when possible. Runtime does not include the grid search for hyperparameter tuning. Here there were 1,416 and 465 training and testing points used, respectively.

see Fig. S9 for rudimentary k-fold predictions, demonstrating the generalizability of TreeDOX.

One may observe that for both the Hènon map and Lorenz systems, TreeDOX captures their respective chaotic attractors with a similar correlation dimension. Furthermore, self-evolved TreeDOX forecasts for both the Lorenz system and Kuramoto-Sivashinsky equation show similar accuracy to state-of-the-art methods, such as LSTM and NG-RC despite its lack of hyperparameter optimization [1]. Lastly, open-loop TreeDOX matches the performance of other current models in the prediction of SOI data, with comparably lower RMSE and higher AMI, supporting the ability of TreeDOX' explicit delay-overembedded memory to capture fading memory similar to that of the implicit memory of LSTM, NG-RC, and other models.

With the increasing availability of data and computational power to analyze it, the need for effective and user-friendly time series forecasting methods is on the rise. Existing sate-of-the-art methods, such as RC and LSTM, offer a powerful ability to meet the need of time series forecasting, but can be difficult to use in practice due to its sensitivity to and required tuning of hyperparameters. We propose an alternative in the form of a time delay overembedded and Extra Tree Regressor-based algorithm for autonomous feature selection and forecasting which does not require hyperparemeter tuning.

While the development of TreeDOX was focused on ease-of-use rather than on surpassing the accuracy of modern forecasting models, after testing it on a variety of prototypical discrete, continuous, and spatiotemporal systems, we find that TreeDOX provides comparable or better performance to current methods such as RC and LSTM. We also demonstrate the efficacy of TreeDOX to predict realistic data with SOI open-loop forecasts, and again discover TreeDOX's similar performance to LSTM, NG-RC, and other state-of-the-art forecasting models.

We would like to acknowledge Research Computing at the Rochester Institute of Technology for supplying computational resources during the course of this work [34]. The research of E.B. is supported by the ONR, ARO, DARPA RSDN and the NIH and NSF CRCNS.

Data and code are available on our Github repository: `https://github.com/amg2889/TreeDOX_Tree-based_Learning_for_High-Fidelity_Prediction_of_Chaos`

* Corresponding author
amg2889@rit.edu

[1] J. Pathak, B. Hunt, M. Girvan, Z. Lu, and E. Ott, Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach, Physical Review Letters **120**, 024102 (2018).

[2] H. Jaeger and H. Haas, Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication, Science **304**, 78 (2004).

[3] D. J. Gauthier, E. Bollt, A. Griffith, and W. A. Barbosa, Next generation reservoir computing, Nature communications **12**, 5564 (2021).

[4] Z.-M. Zhai, M. Moradi, L.-W. Kong, B. Glaz, M. Haile, and Y.-C. Lai, Model-free tracking control of complex dynamical trajectories with machine learning, Nature Communications **14**, 5698 (2023).

[5] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, Learning representations by back-propagating errors, nature **323**, 533 (1986).

[6] S. Elsworth and S. Güttel, Time series forecasting using lstm networks: A symbolic approach, arXiv preprint arXiv:2003.05672 (2020).

[7] S. Hochreiter and J. Schmidhuber, Long short-term memory, Neural computation **9**, 1735 (1997).

[8] W. Maass, T. Natschläger, and H. Markram, Real-time computing without stable states: A new framework for neural computation based on perturbations, Neural computation **14**, 2531 (2002).

[9] E. Bollt, On explaining the surprising success of reservoir computing forecaster of chaos? the universal machine learning dynamical system with contrast to var and dmd, Chaos: An Interdisciplinary Journal of Nonlinear Science **31** (2021).

[10] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference and prediction*, 2nd ed. (Springer, 2009).

[11] T. Chen and C. Guestrin, Xgboost: A scalable tree boosting system, in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (2016) pp. 785–794.

[12] L. Breiman, Random forests, Machine Learning **45**, 5–32 (2001).

[13] R. Shwartz-Ziv and A. Armon, Tabular data: Deep learning is not all you need, Information Fusion **81**, 84 (2022).

[14] L. Grinsztajn, E. Oyallon, and G. Varoquaux, Why do tree-based models still outperform deep learning on typical tabular data?, Advances in Neural Information Processing Systems **35**, 507 (2022).

[15] R. Hegger, H. Kantz, L. Matassini, and T. Schreiber, Coping with nonstationarity by overembedding, Physical Review Letters **84**, 4092 (2000).

[16] P. Verdes, P. Granitto, and H. Ceccatto, Overembedding method for modeling nonstationary systems, Physical review letters **96**, 118701 (2006).

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

[18] P. Geurts, D. Ernst, and L. Wehenkel, Extremely randomized trees, Machine Learning **63**, 3 (2006).

[19] T. M. Oshiro, P. S. Perez, and J. A. Baranauskas, How many trees in a random forest?, in *Machine Learning and Data Mining in Pattern Recognition: 8th International Conference, MLDM 2012, Berlin, Germany, July 13-20, 2012. Proceedings 8* (Springer, 2012) pp. 154–168.

[20] P. Probst, M. N. Wright, and A.-L. Boulesteix, Hyperparameters and tuning strategies for random forest, Wiley Interdisciplinary Reviews: data mining and knowledge discovery **9**, e1301 (2019).

[21] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, Do we need hundreds of classifiers to solve

real world classification problems?, The journal of machine learning research **15**, 3133 (2014).
[22] P. Grassberger and I. Procaccia, Characterization of strange attractors, Physical review letters **50**, 346 (1983).
[23] B. J. Geurts, D. D. Holm, and E. Luesink, Lyapunov exponents of two stochastic lorenz 63 systems, Journal of statistical physics **179**, 1343 (2020).
[24] R. A. Edson, J. E. Bunder, T. W. Mattner, and A. J. Roberts, Lyapunov exponents of the kuramoto–sivashinsky pde, The ANZIAM Journal **61**, 270 (2019).
[25] C. F. Ropelewski and P. D. Jones, An extension of the tahiti-darwin southern oscillation index, Monthly weather review **115**, 2161 (1987).
[26] G. N. Kiladis and H. van Loon, The southern oscillation. part vii: Meteorological anomalies over the indian and pacific sectors associated with the extremes of the oscillation, Monthly weather review **116**, 120 (1988).
[27] K. E. Trenberth, The definition of el nino, Bulletin of the American Meteorological Society **78**, 2771 (1997).
[28] S. B. Power and G. Kociuba, The impact of global warming on the southern oscillation index, Climate dynamics **37**, 1745 (2011).
[29] D. Harisuseno, Meteorological drought and its relationship with southern oscillation index (soi), Civil engineering journal **6**, 1864 (2020).
[30] R. Ranasinghe, R. McLoughlin, A. Short, and G. Symonds, The southern oscillation index, wave climate, and beach rotation, Marine Geology **204**, 273 (2004).
[31] R. Chowdhury and S. Beecham, Australian rainfall trends and their relation to the southern oscillation index, Hydrological Processes: An International Journal **24**, 504 (2010).
[32] U. o. E. A. Climate Research Unit, Southern oscillation index (soi) data, `https://crudata.uea.ac.uk/cru/data/soi/`.
[33] J. Yan, L. Mu, L. Wang, R. Ranjan, and A. Y. Zomaya, Temporal convolutional networks for the advance prediction of enso, Scientific reports **10**, 8055 (2020).
[34] R. I. of Technology, Research computing services (2019).

Supplemental Material: Tree-based Learning for High-Fidelity Prediction of Chaos

Adam Giammarese,[1,*] Kamal Rana,[2] Erik M. Bollt,[3,4] and Nishant Malik[1]

[1] *School of Mathematics and Statistics, Rochester Institute of Technology, Rochester, NY 14623*
[2] *Chester F. Carlson Center for Imaging Science, Rochester Institute of Technology, Rochester, NY 14623*
[3] *Clarkson Center for Complex Systems Science, Clarkson University, Potsdam, NY 13699*
[4] *Department of Electrical and Computer Engineering, Clarkson University, Potsdam, NY 13699*

(Dated: March 22, 2024)

INTRODUCTION

This supplemental material provides additional visualizations and results on TreeDOX training, forecasting, and data from which testing was performed.

METHOD

Diagrams in Figs. S1 and S2 summarize the steps involved in the training and forecasting in TreeDOX, respectively. To prescribe a value for k (delay overembedding dimension), average mutual information (AMI) is applied to each dimension of training data, and with a given p-value selected a time delay τ_{crit} is found where the respective AMI crosses under the p-value. See the main manuscript for more information about how k is prescribed from the τ_{crit} values. Fig. S3 shows the AMI calculation for Lorenz System training data where $dt = 0.01$. One may observe the sharp drop in mutual information as the time delay τ increases, indicating the lags losing information about the current state of the system. While the p-value is arbitrary, we recommend setting a value past the plateau in AMI curves since time delays further along the plateau do not provide much more information, but rather directly increase time and space complexity of TreeDOX training. Similarly, ξ (the successive time delays in the time delay overembedding) should be selected to the smallest value possible such that the resulting time and space complexity of TreeDOX training is not too great a burden.

The number of features to use in final training, p, is prescribed by comparing the Gini feature importances, FI, from ETR #1 in Fig. S1 to a natural null rate, $FI_0 = \frac{1}{kD}$, where k is the time delay overmbedding dimension and D is the dimension of the data. Fig. S4 displays the measured FI for Lorenz System training data where $dt = 0.01$. Observe, like Fig. S3, the sharp drop-off of FI as the time delay increases. While this indicates a similar phenomena, instead of being interpreted as a drop in the amount of mutual information between a delayed state and the current state it instead may be interpreted by ETR #1 finding the delayed state to have an insignificant ability to make predictions of the current state. While delay states with the respective feature

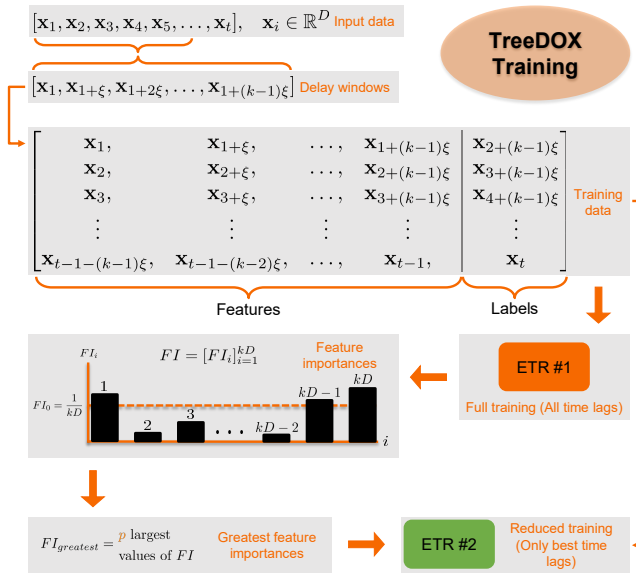

FIG. S1. Diagram summarizing the training stage for TreeDOX.

importance under FI_0 may still be considered useful for prediction, their removal provides two major advantages: reducing time and space complexity of TreeDOX training, and improving the generalizability of the model. Due to the training speedup after feature reduction, one may reasonably increase the number of estimators in the ETR ensemble to further improve results with little reduction in training performance.

DISCRETE SYSTEM

To further reinforce the ability of TreeDOX to predict discrete chaotic systems, we apply the model to the logistic map: $x_{n+1} = rx_n(1 - x_n)$ where $x \in [0, 1]$ and $r \in [0, 4]$. To start, we selected a variety of values for the parameter r - namely 0.5, 2, 3.2, 3.5, 3.56, 3.6, 3.7, 3.8, and 3.9 - and train TreeDOX with 2,000 points generated from the iteration scheme with $x_0 = 0.25$ and 25 transient points removed. Then, we test on the next 1,000 points. We use $k = 20$ delay states and $\xi = 1$ successive time delay. We plot x_{n+1} against x_n to determine

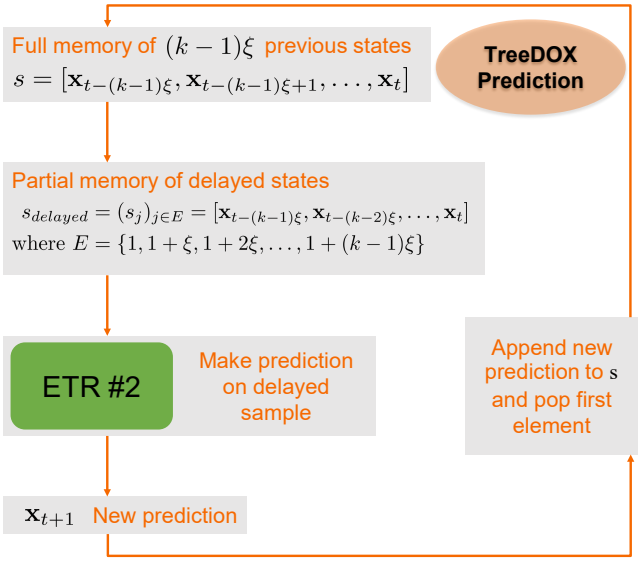

FIG. S2. Diagram summarizing the forecasting stage for TreeDOX.

if our forecasted data lies on the above parabola of the true dynamics, confirming TreeDOX has learned the underlying system. The results seen in Fig. S5 confirm the algorithm's preservation of the generating dynamics of the underlying system, since the forecasted data does, in fact, lie on the parabola $x_{n+1} = rx_n(1-x_n)$.

Lastly, we attempt to use a much larger set of training data and compare the last point in our forecasted results to the true bifurcation diagram of the logistic map. For each randomly generated value of $r \in [2.75, 4]$ we train our algorithm with a length-250 time series of the true logistic evolution with a randomly generated $x_0 \in [0, 1]$ (after removing 25 transient points). We use $k = 20$ delay states and $\xi = 1$ successive time delay. Lastly, we evolve our system for 250 points, and plot the last value in the forecasted time series. Fig. S6 confirms that our forecasting method captures the logistic dynamics well enough to recreate the bifurcation diagram with a substantial resolution.

SOUTHERN OSCILLATION INDEX

Southern Oscillation Index (SOI) is a useful climate index defined as the z-score of monthly pressure difference between Tahiti and Darwin [25]. Fig. S7 displays Southern Oscillation Index data used in the training and testing of TreeDOX and other models to which it is compared in the main manuscript. In Fig. S8 the length of training data is varied and testing is performed on the same test data seen in Fig. S7. One may observe in Fig. S7 row 1 that TreeDOX, like LSTM and RNN, tends to show less root mean square error (RMSE) than other models and the length of training data has little effect on test performance unlike NG-RC and RC. Similarly, the average mutual information (AMI) seen in Fig. S7 row 2 remains high for TreeDOX in comparison with other models and again training data length has little effect on performance. In Fig. S7 row 3, one may observe that, like RC, TreeDOX does scale in runtime with the length of training data. However, both RC and TreeDOX scale much better than LSTM and RNN. In summary, this experiment demonstrates that TreeDOX features the beneficial scaling properties of RC with the length of training data, while maintaining high accuracy across all scenarios. It is important to note that runtime here is measured using batch inputs, and

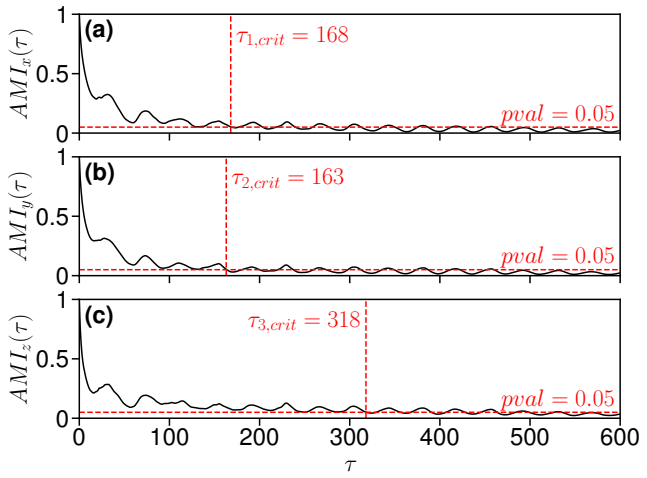

FIG. S3. Average mutual information (AMI) of Lorenz System training data where $dt = 0.01$.

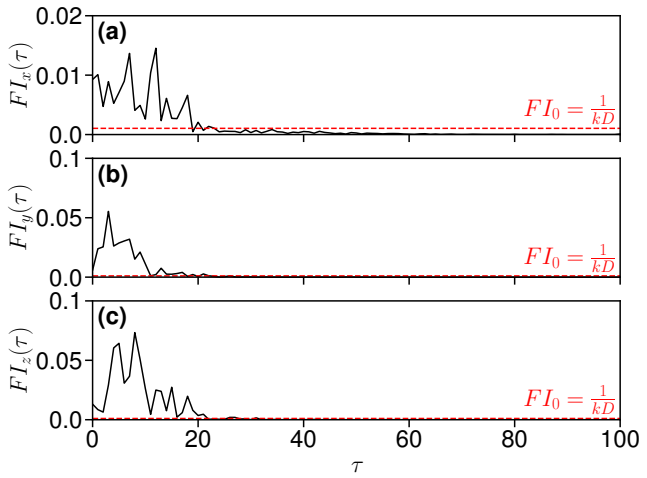

FIG. S4. Gini feature importances measured by ETR #1 in Fig. S1 for Lorenz System training data where $dt = 0.01$. For clarity, FI is unflattened to show the respective lags chosen for each of the three dimensions. $FI_0 = \frac{1}{kD}$ is the null rate, where k is the time delay overmbedding dimension and D is the dimension of the data.

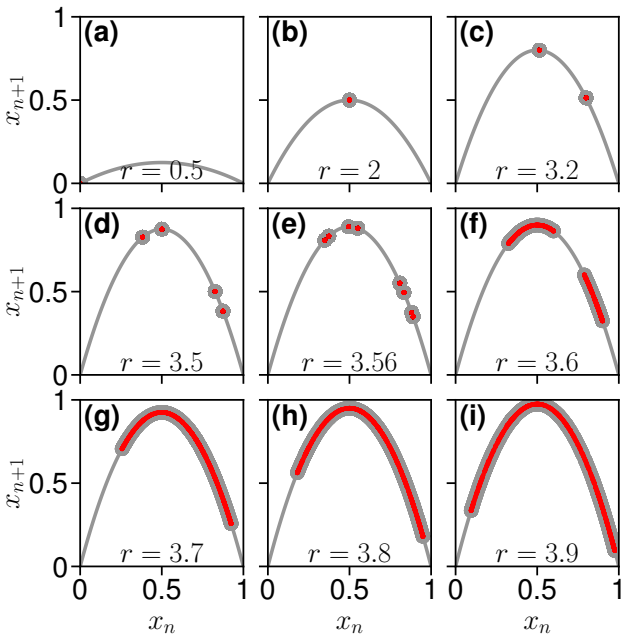

FIG. S5. Recreation of logistic map dynamics for a range of r values, where the gray parabola is the true dynamics ($x_{n+1} = rx_n(1-x_n)$) and the forecasted and test data are red and gray, respectively. (a,b,c,d,e,f,g,h,i) Test and predicted dynamics for r values of 0.5, 2, 3.2, 3.5, 3.56, 3.6, 3.7, 3.8, and 3.9, respectively.

due to the computational overhead of unbatched predictions (which is required for self-evolving forecasts rather than the open-loop SOI predictions) TreeDOX predictions may be slower in most practical scenarios. However, recent computation developlments such as Hummingbird and RAPIDS CuML show promising computational performance boosts for tree-based ensemble regression algorithms [S1, S2].

We also perform self-evolving prediction on the SOI data to demonstrate a more practical use case. However, due to the noisy nature of SOI data, we apply a Savitzky–Golay filter to reduce noise (and capture the underlying long-term dynamics) in the dataset with a window size of 121 months and a polynomial order of 1 [S3]. We also aim to demonstrate generalizability of TreeDOX, and thus perform a rudimentary k-fold cross-validation where we select a variety of windows in the SOI time series and use all data outside the window to train TreeDOX, then perform self-evolved predictions for the window. We also vary the length of these windows (from 3 months to 24 months in increments of 3 months) in order to investigate the effect of long predictions for highly chaotic time series data. To summarize the error of these predictions, we use Normalized Mean Absolute Error: $NMAE(x, \tilde{x}) = \frac{1}{N}\sum_{i=1}^{N}|x - \tilde{x}_i| / (\max(x_{train}) - \min(x_{train}))$, where x is the true value, $\tilde{x}$ is a set of forecast realizations, N is the number of forecast realizations, and x_{train} is the training data. Since the SOI data is bounded and ETRs cannot output predictions outside the training range, NMAE captures the fact that the worst possible forecast is that where the true and forecasted data are on opposite ends of the bounded range. Therefore one may interpret an NMAE of 0 to be a perfect forecast, and an NMAE of 1 to be the worst possible forecast. Fig. S9 displays the results of this experiment, where the NMAE tends to stay below 0.2 for forecast windows less than a full year.

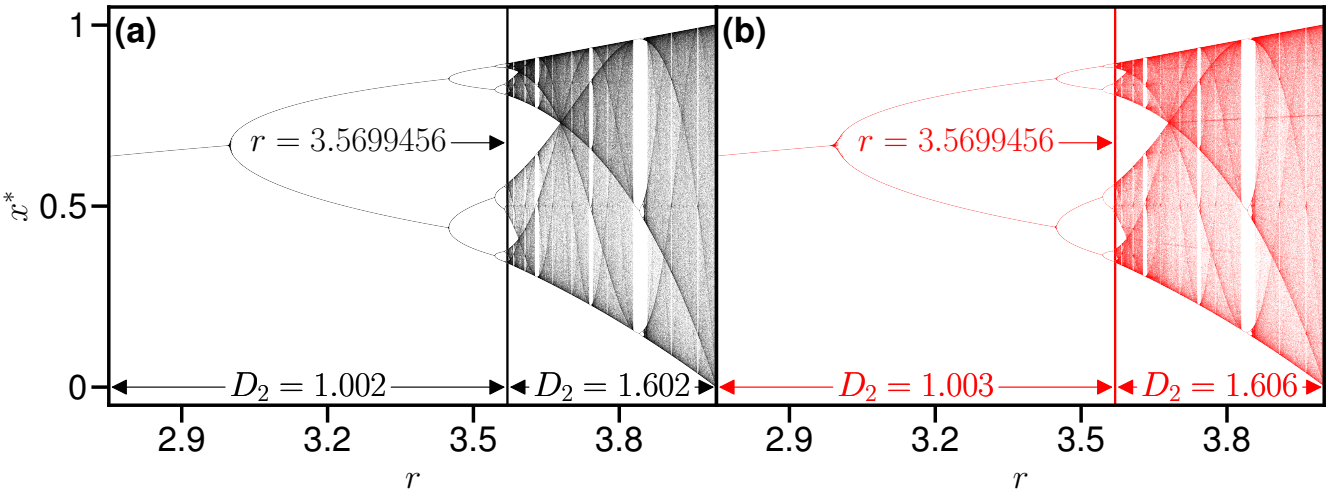

FIG. S6. Learning the bifurcation diagram of the logistic map, where the forecasted and test data are red and black, respectively. (a,b) Test and predicted bifurcation diagrams, respectively. D_2 is the correlation dimension calculated by the region of data specified by the respective arrows.

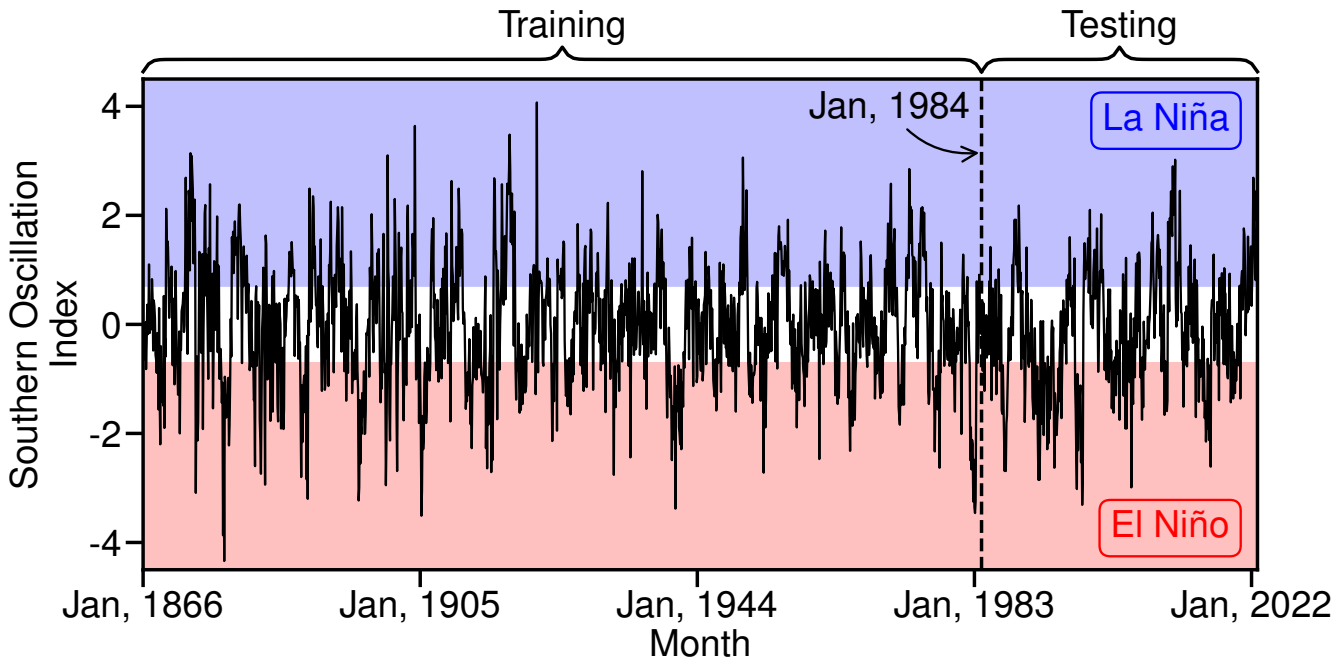

FIG. S7. Historical monthly Southern Oscillation Index data, where data prior and following January 1984 is used for training and testing, respectively, of TreeDOX and compared models.

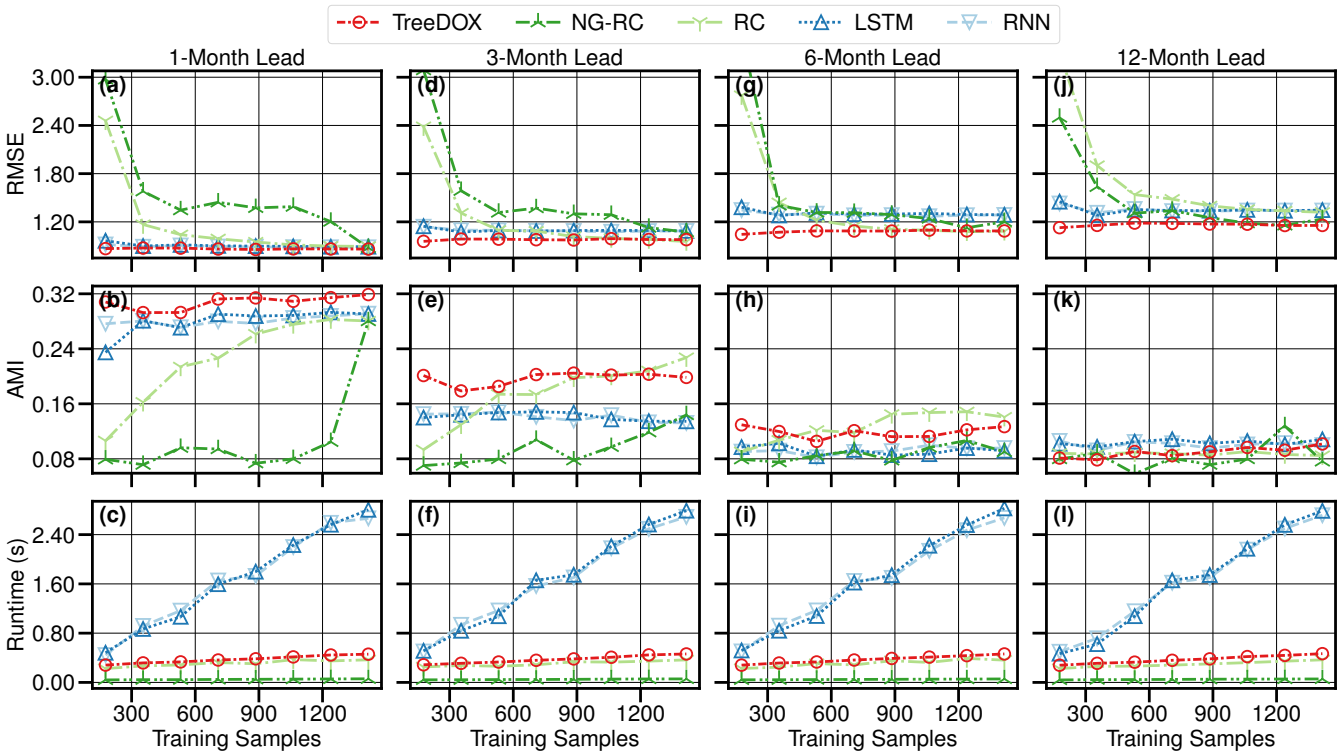

FIG. S8. Open-loop prediction results for SOI data where the length of training data is varied. Columns 1 through 4 group results by the lead (see main manuscript for details), and rows 1 and 2 display root mean square error (RMSE) and average mutual information (AMI) between test and predicted data, respectively, and row 3 shows the runtime, in seconds, of training and testing (not including hyperparameter tuning). See legend for respective color, line style, and marker style for TreeDOX, NG-RC, RC, LSTM, and RNN results.

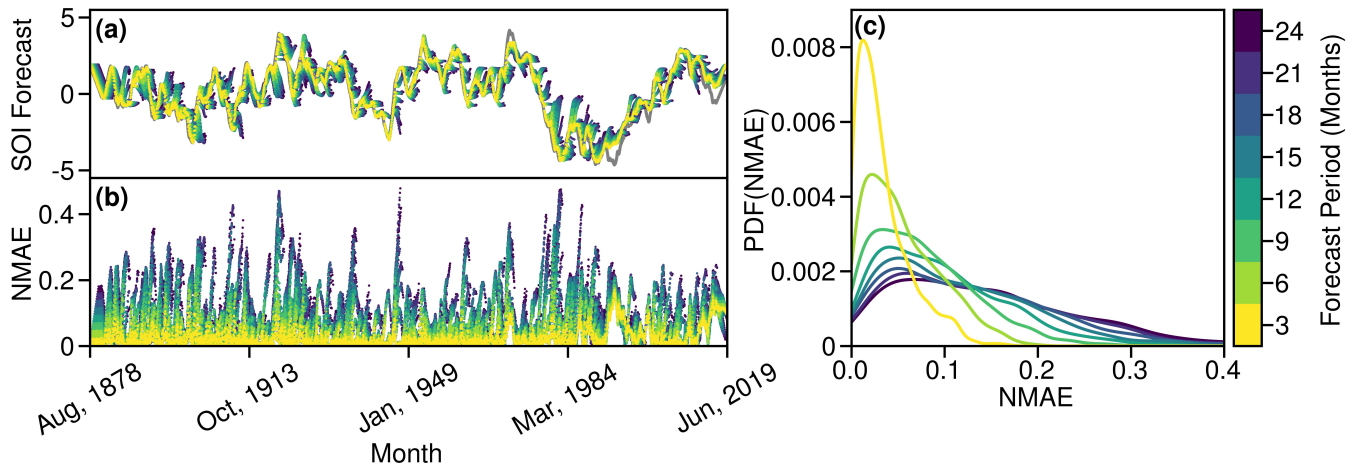

FIG. S9. A summary of k-fold cross-validation on SOI time series, where windows of various lengths (3 months, 6 months, etc.) are chosen and data outside the window is used to train TreeDOX, which then performs testing inside the window. (a) SOI tests, where the true data is gray and the tests are colored according to how long the testing window is. (b) Normalized Mean Absolute Error (NMAE) for each point in each testing window. (c) Summary statistics for the last point in the testing window for each testing window length across all windows using kernel density estimation.

* Corresponding author
amg2889@rit.edu

[S1] S. Nakandalam, K. Saur, G. Yu, K. Karanasos, C. Curino, M. Weimer, and M. Interlandi, Taming model serving complexity, performance and cost: A compilation to tensor computations approach (2020).

[S2] S. Raschka, J. Patterson, and C. Nolet, Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence, arXiv preprint arXiv:2002.04803 (2020).

[S3] A. Savitzky and M. J. Golay, Smoothing and differentiation of data by simplified least squares procedures., Analytical Chemistry **36**, 1627–1639 (1964).