Supplementary Information for "Data-driven discovery of spatiotemporal dynamical systems with sparse interpretable neural networks"

Siyuan Xing, Qingyu Han, Efstathios G. Charalampidis, Ying-Cheng Lai

July 4, 2025

Contents

Supplementary Note 1: Training of SREINet

Algorithm 1 Training of SREINet (sparse regression embedded interpretable network)

Data dimension: d; early-stopping loss value: δ; maximum epochs per dimension: n_{max}; minimum pruning for early stopping: Δ_{min}.

Generate a piecewise-continuous pruning schedule F with N periods.
for $i = 1, 2, \ldots, d$ **do**
  for epoch $= 1, 2, \ldots, n_{max}$ **do**
    Δ=F(epoch) ▷ Δ: current pruning threshold
    $\boldsymbol{B}$=*generate_binary_masks*(Δ) ▷ $\mathbf{B}$: binary masks
    for $i = 1, 2, \ldots, l$ **do** ▷ l: number of layers
      $W[i] = W[i] * \boldsymbol{B}[i]$ ▷ elementwise product to zero the weights less than Δ
    end for
    for $(\mathcal{X}_i, \dot{\mathcal{X}}_i)$ in training_set **do** ▷ Mini-batch training
      train_loss[i] = *train_step*($\mathcal{X}_i$, $\dot{\mathcal{X}}_i$, $\boldsymbol{B}$)
    end for
    for $(\mathcal{X}_i, \dot{\mathcal{X}}_i)$ in validation_set **do**
      val_loss[i] = $\|\mathcal{NN}(\mathcal{X}_i) - \dot{\mathcal{X}}_i\|_2$
    end for
    if *mean*(train_loss) $\leq \delta$ and $\Delta \geq \Delta_{min}$ **then**
      break ▷ Stop the training of the ith dimension
    end if
  end for
end for

Algorithm 1 presents the pseudocode of the proposed training method. Similar to standard training of neural networks (NNs), the method splits data into training and validation sets using an 80-20 ratio. To start, we generate a N-period pruning schedule, where each period is characterized by a piecewise-continuous function with three length-tunable segments: a zero segment, a transition segment, and a maximum segment. Introducing the transition and periodicity to the pruning schedule serves to mitigate the risk of accidentally pruning dominant terms before their weights grow beyond the threshold. Inside the main loop, each dimension is individually trained for a maximum of n_{max} steps. During each step, a new threshold is selected according to the pruning schedule, which will then be used to generate binary masks, i.e., 0-1 matrices that match the shapes of the weight matrices. Afterwards, these binary masks are element-wise multiplied with the corresponding weight matrices to nullify the weights below the current threshold. After the nullification, the mini-batch training is deployed to minimize the loss function, along with the evaluation of the loss values on the validation set. When the training-loss mean falls below a minimum value and the pruning value exceeds its minimum threshold, training for this dimension is deemed complete. The process is repeated for each dimension until training is done for all dimensions.

Algorithm 2 Generate binary masks: $masks = generate\_binary\_masks(\Delta)$

Input: pruning threshold: Δ.
binary_masks = [ ]
for $\boldsymbol{W}$ in model.trainable_variables **do**
  mask = $|W| \geq \Delta$ ▷ Create a mask where each element is 1 if $|\boldsymbol{W}_{ij}| \geq \Delta$, otherwise 0
  mask[0,0] = 1 ▷ The weights connecting constant neurons cannot be pruned
  binary_masks.append(mask)
end for
return binary_masks

The two key pseudocodes for the generation of binary masks and mini-batch training with binary masking,

Algorithm 3 Train step with pruning: $loss = train\_step[\mathcal{X},\ \dot{\mathcal{X}},\ \boldsymbol{B}]$

Input: measurement data $\mathcal{X}$, velocity-field data $\dot{\mathcal{X}}$, binary masks $\boldsymbol{B}$
$\hat{\dot{\mathcal{X}}} = \mathcal{NN}(\mathcal{X})$ ▷ Estimate the velocity field by SREINet
loss= $\|\hat{\dot{\mathcal{X}}} - \dot{\mathcal{X}}\|_2$ ▷ Calculate loss value
gradients = calculate_gradient(loss, model.trainable_weights)
for $i = 1, 2, \ldots, l$ **do** ▷ l: the number of layers
    masked_gradients[i] = gradients[i]$*\boldsymbol{B}[i]$ ▷ $*$: element-wise product
end for
optimizer.apply_gradients(masked_gradients, model.trainable_weights)
return loss

are detailed in Algorithms 2 and 3. The first pseudocode generates binary masks (matrices) by iterating through the trainable variables of the network, assigning ones to those with a absolute value greater than the current threshold and zeros to those below. Note that the weights connecting neurons with constant 1 cannot be pruned, because SREINet has only one path for the constant term, so pruning any weight on this path will eliminate the constant term. The second pseudocode involves multiplying the binary masks with the gradients during backpropagation, preventing the gradients of pruned weights from contributing to the gradient update.

Supplementary Note 2: Numerical experiments

We describe more training details and numerical results from the examples presented in the main text.

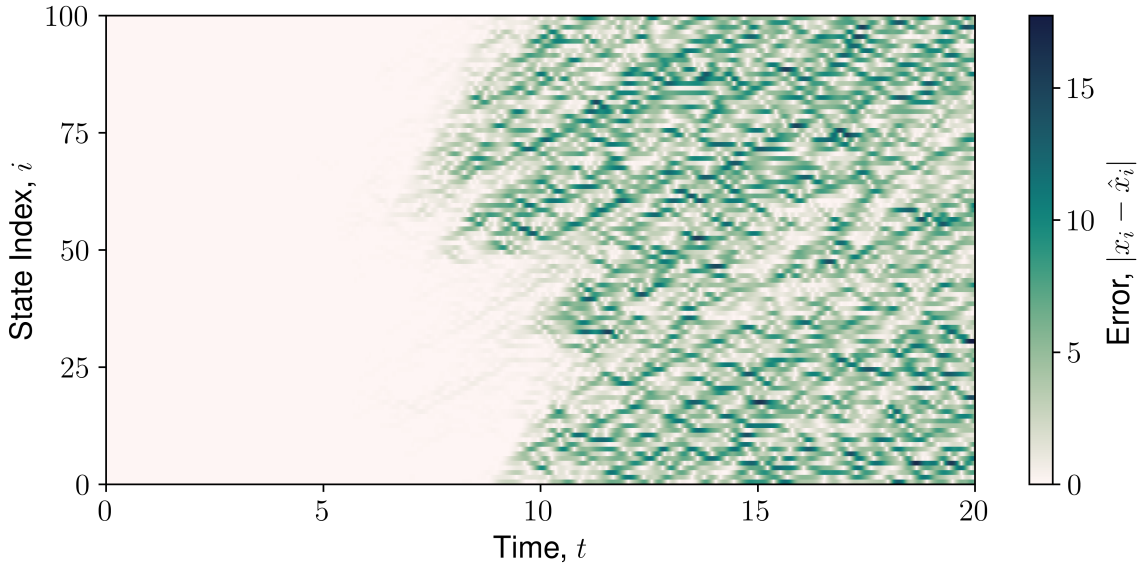
Lorenz-96 model

The differential equation of the Lorenz-96 system reads

$$\dot{x}_j = x_{j-1}(Ax_{j+1} - Bx_{j-2}) - Cx_j + F, \quad j = 1, 2, \ldots, n, \tag{1}$$

where $x_{-1} = x_{n-1}$, $x_0 = x_n$, and $x_{n+1} = x_1$. In numerical experiments, we choose $n = 100$, which divides a latitude circle equally into 100 sectors. To generate the data, we use an 8th order Runge-Kutta method, DOP853 to simulate the dynamics from $t = 0$ to $t = 10$ and produce 6 trajectories originated from random initial conditions. With a time step $dt = 10^{-3}$, 60,000 points are collected in total. This dataset is then shuffled and split into training and validation sets with the 80-20 ratio. To construct SREINet for the Lorenz-96 system, we adopt univariate first-order monomial candidate functions in each layer, assuming that the highest polynomial order is two. This yields a two-layer SREINet with 101 neurons per layer (comprising a constant function $\varphi = 1$ and 100 monomials of input variables). The network weights are initialized with a random distribution with zero mean and standard deviation 10^{-3}. Training is performed using the Adam optimizer at the fixed learning rate of 10^{-3} and mini-batch size of 128. This size balances the training speed and accuracy with the given data volume. We set the maximum pruning threshold to 0.3. The minimum loss value for early stopping of each dimension is fixed at 10^{-10}, with the early-stopping pruning at 0.1. The pruning schedule has a period of 30 epochs, beginning with the first 15% of the epochs without pruning, followed by a 75% transition period using a shifted sine function. After the training, we perform numerical simulation from the final time of the training data to further validate the network's extrapolation capability.

As the approximation of vector field has been discussed in the main text, we focus herein on the error evolution of the state variables, as shown in the Supplementary Fig. 1, where a good agreement between the states of identified model and of the actual system can be seen for $t < 6$. For $t > 6$, the effect of chaos intrinsic to the system can be seen, where the trajectories from the SREINet-discovered model and the actual system diverge. The chaotic behavior is in fact shared by all systems studied with random initial conditions. The divergent behavior can be further seen through the time series of four sampled states x_{20}, x_{40}, x_{60}, x_{80}, as shown in Supplementary Fig. 2.



Supplementary Figure 1: Evolution of absolute solution error in the Lorenz-96 system with random initial conditions. The trajectory of the identified model closely matches the ground truth for $t < 6$. Due to the system's chaotic nature, small errors in the vector field prediction eventually cause the trajectory from the discovered model to diverge over time from that of the actual system.

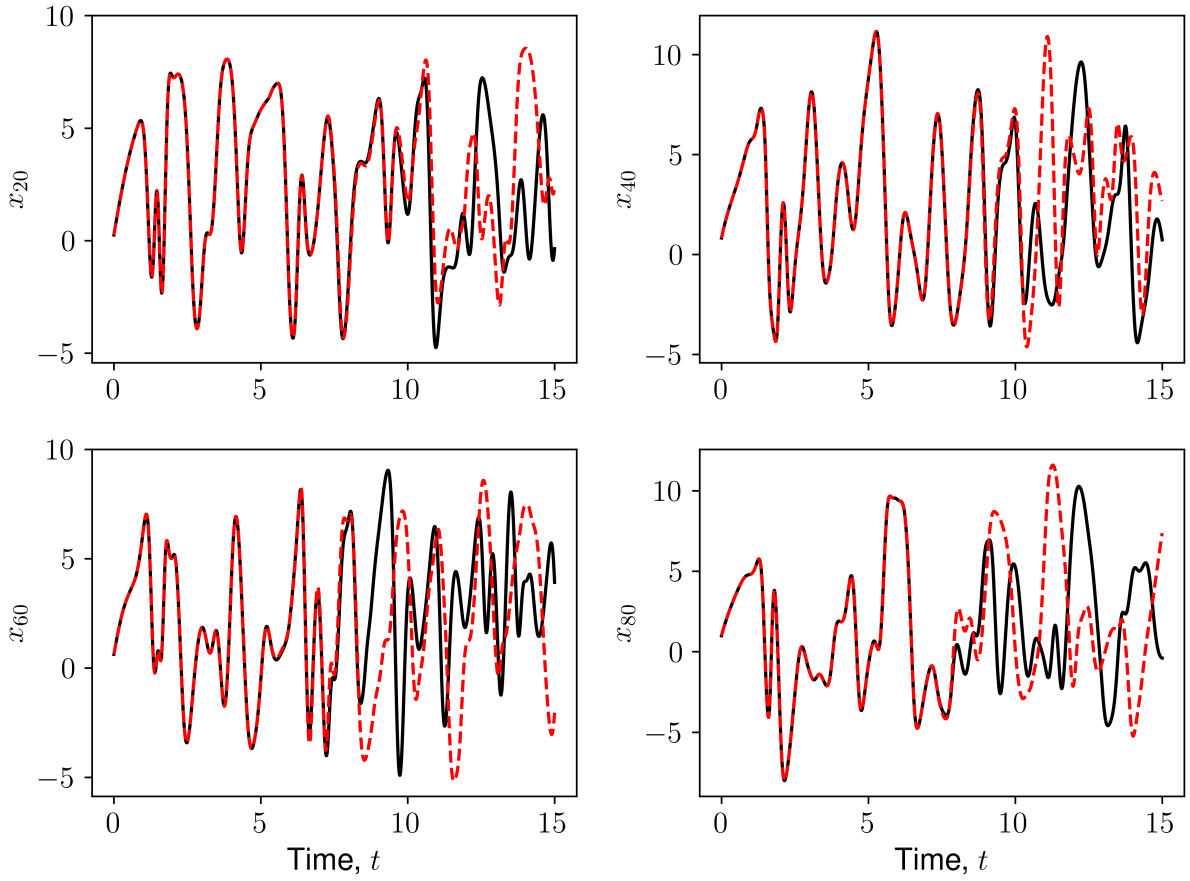
Kuramoto model

The Kuramoto model under external forcing has the form

$$\dot{\theta}_i = \omega_i + \frac{K}{n}\sum_{j=1}^{n}\sin(\theta_j - \theta_i) + h\sin(\theta_i) \quad i = 1, 2, \ldots, n \tag{2}$$

where ω_i is the natural frequency of the i-th oscillator, K is the coupling strength, and h is the coefficient of external forcing. To be concrete, we set the system size to be $n = 60$. The Kuramoto network structure is in fact equivalent to that of, e.g., a 120-dimension Lorenz-96 system. We assume the oscillators' nature frequencies are evenly distributed in $[-5, 5]$, use $h = 0.2$ and focus on the weak coupling regime by setting $K = 2$. We collect 10 trajectories emanating from random initial conditions with a normal distribution of $\mathcal{N} \sim (0,\ 10^{-4})$. Each trajectory is simulated using a backward differentiation scheme for the time duration of $T = 500$, and sampled 5000 points with time step $dt = 0.1$. In total, we produce a dataset with $50,000$ points, then shuffle and split them into training and validation sets. We construct a two-layer SREINet with cosine and sine activation functions in each layer. With an additional neuron $\varphi = 1$, each layer possesses 121 neurons, yielding a SREINet with $121^2 = 14,641$ trainable weights. During the training, we use the Adam optimizer at the constant learning rate of 10^{-3} with the mini-batch size of 128. The trained SREINet includes terms $\sin^2(\theta_i)$ and $\cos^2(\theta_i)$ that sum to constant values. We use a post-processing code to combine such terms into constants when recovering the governing equations. The pruning period is 50 with the maximum pruning threshold of 0.15. The loss value and pruning threshold for early stopping is set to 10^{-10} and 0.05, respectively.

Supplementary Fig. 3 presents the time evolution of the absolute phase difference between the original and identified systems. The results are produced by employing a backward differentiation scheme with the initial conditions of the first trajectory in the training set. The phases of the identified system closely matches with the ground truth for $t < 25$ and deviations occur afterwards due to chaos. The growth of the error can also been seen from the snapshots of sampled oscillators at distinct time instants, as shown in Supplementary Fig. 4.



Supplementary Figure 2: Time series of four distinct states in the Lorenz-96 system: (a) x_{20}, (b) x_{40}, (c) x_{60}, (d) x_{80}. Black curve: ground truth; red dash curve: simulation of the SREINet-discovered system. In each case, the time series from the discovered model agrees with that of the actual model for $t < 6$. The apparently divergence for $t > 6$ is the result of chaos.

Discrete ϕ^4 model

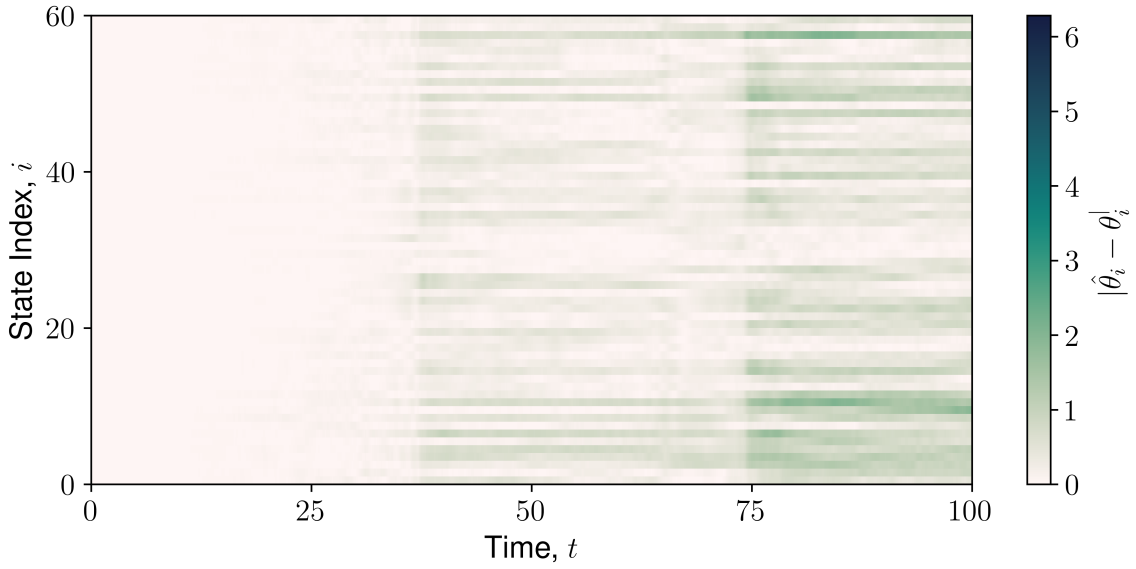
The ϕ^4 system with a cubic nonlinearity is given by

$$\ddot{u}_i = C(u_{i+1} + u_{i-1} - 2u_i) + 2(u_i - u_i^3),\ i = -n/2, -n/2+1, \ldots, n/2-1, \tag{3}$$

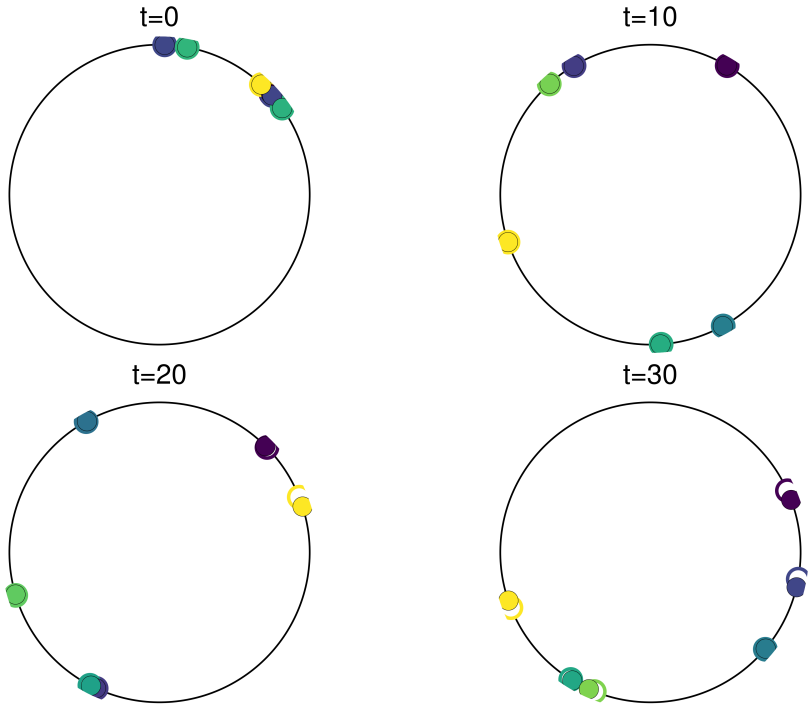
where C is the coupling strength between adjacent nodes. This model describes the dynamics of a 1D discrete lattice with N nodes, where the coupling term $(u_{n+1} + u_{n-1} - 2u_n)$ represents the discrete Laplacian. We impose free boundary conditions: $u_{n/2} = u_{n/2-1}$ and $u_{-n/2-1} = u_{-n/2}$. Let $x_i = u_i$. The corresponding state-space form of Eq. (3) is

$$\begin{aligned} \dot{x}_i &= x_{i+n}, \\ \dot{x}_{i+n} &= C(x_{i+1} + x_{i-1} - 2x_i) + 2(x_i - x_i^3). \end{aligned} \tag{4}$$

We set the system size to $n = 50$ and $C = 2$, and employ the numerical integrator DOP853 to generate 6 trajectories with a time span of $T = 500$ and random initial conditions drawn from $N\ (0, 0.1)$. We sample the data with the time step $dt = 0.1$, yielding in total $50,000$ points. For convenience, we group the data of the velocity equations into the first 50 dimension and the data of the acceleration equations into the last 50 dimensions. The data are split into training and validation sets with the $80 - 20$ ratio. We construct a 3-layer SREINet with monomial candidate functions, resulting in 101 candidate functions per layer with



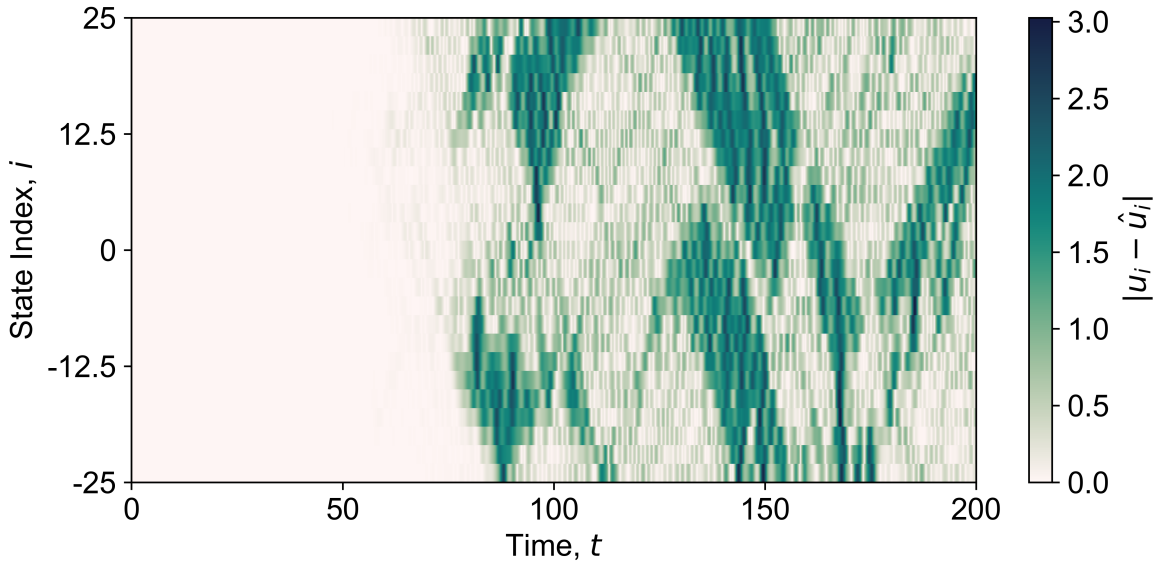
Supplementary Figure 3: Evolution of prediction errors in the Kuramoto System. The inference error of the vector field remain below 4×10^{-4}. Despite of this small error, the trajectories of states diverge for $t > 20$ due to chaos.



Supplementary Figure 4: Snapshots of six Kuramoto oscillators on a ring. The oscillator indices are $i \in 0, 10, 20, 30, 40, 50, 60$. Top left: $t = 0$, top right: $t = 10$, bottom left: $t = 20$, bottom right: $t = 30$.

$2 \times 101^2 = 20,402$ trainable weights in total. We adopt the Adam optimizer at the learning rate of 0.001

with the mini-batch size 128. The pruning schedule has 60 epochs per period, with the maximum pruning threshold of 0.5. The early-stopping loss pruning thresholds are 10^{-10} and 0.15, respectively. Supplementary Fig. 5 shows that the state prediction error is negligible with $t < 60$, as further illustrated in Supplementary Fig. 5. The trajectory from the SREINet identified system diverges from that from the original system for $t > 60$.



Supplementary Figure 5: Error evolution in the ϕ_4 system.

DNLS model

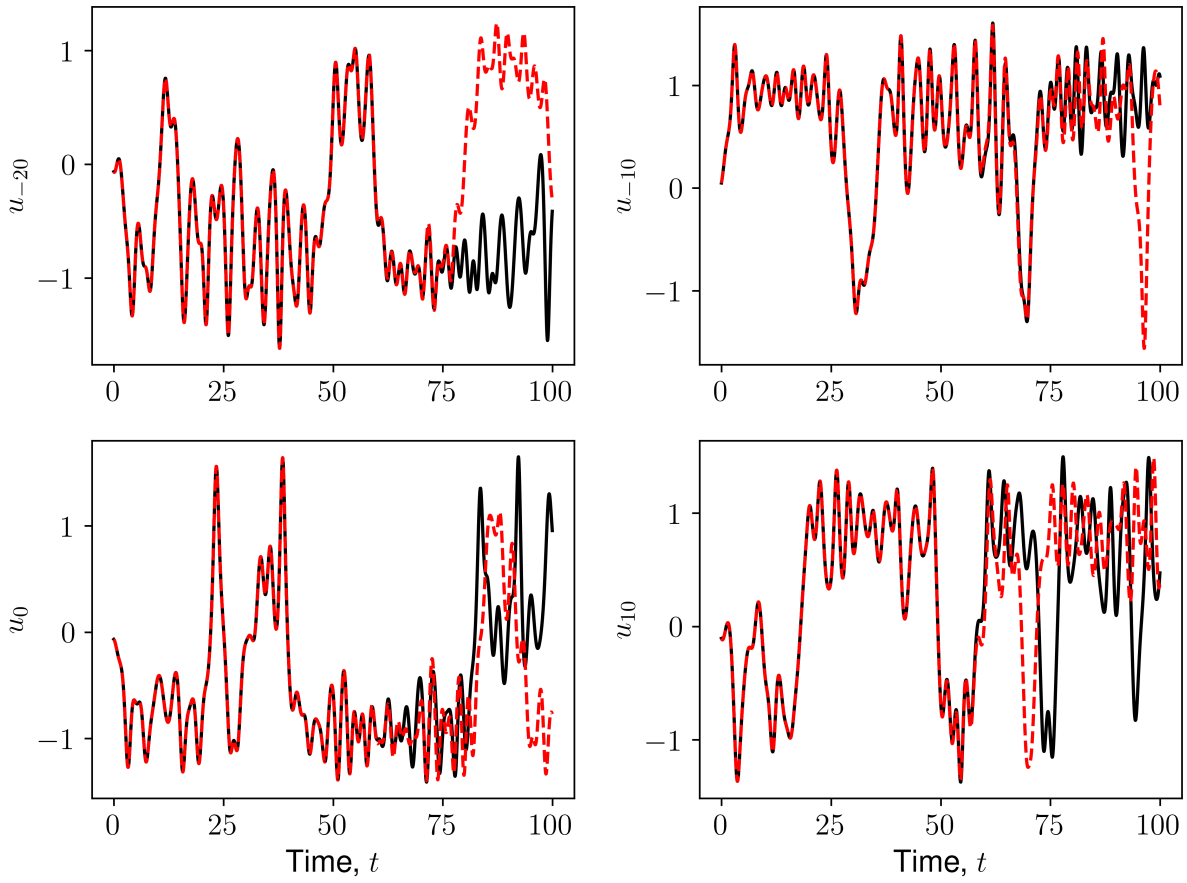
The discrete nonlinear Schrödinger equation (DNLS) has the form

$$i\dot{u}_j = -\frac{C}{2}(u_{j+1} + u_{j-1} - 2u_j) - |u_j|^2 u_j, \tag{5}$$

where the dynamical variable u_j is complex, C is the coupling constant and the coupling term $u_{j+1}+u_{j-1}-2u_j$ represents discrete Laplacian. We choose $n = 50$ sites, and set $C = 4$. The one-dimensional lattice is supplemented with periodic boundary conditions: $u_{-26} = u_{24}$ and $u_{25} = u_{-25}$. we apply the numerical scheme "DOP853" to simulate 16 trajectories for the duration of $T = 100$. The initial amplitudes are randomly selected while the initial phases are uniformly distributed in $[0, 2\pi]$. We generate a dataset of $160,000$ samples with the time step $dt = 0.01$ to ensure convergence. For convenience, we write $u_j = \alpha_j + i\beta_j$ and obtain the differential equations for the real and imaginary parts:

$$\begin{aligned} \dot{\alpha}_j &= -2(\alpha_{j+1} + \beta_{j-1} - 2\beta_j) - \beta_j(\alpha_j^2 + \beta_j^2), \\ \dot{\beta}_j &= 2(\alpha_{j+1} + \alpha_{j-1} - 2\alpha_j) + \alpha_j(\alpha_j^2 + \beta_j^2). \end{aligned} \tag{6}$$

We split the real and imaginary parts of data and concatenate them to reconstruct the dataset. This doubles the number of dimensions to 100. We shuffle the dataset and divide it into training and validation sets using the 80-20 ratio. We apply a 3-layer SREINet with monomial candidate functions, identical to the SREINet structure that we applied to the ϕ^4 model. We train the SREINet using the Adam optimizer at the fixed learning rate of 5×10^{-3} with the mini-batch size 512. We set the maximum pruning threshold to 0.15 with the period of 40 epochs. Early stopping is triggered when the training loss falls below 10^{-10} and the pruning value is greater than 0.05. The predictions are produced using the initial conditions of the first trajectory in the dataset. As illustrated in Supplementary Fig. 7, the SREINet discovered model can yield trajectories that match the ground truth for $t < 50$.



Supplementary Figure 6: Time histories of sampled nodes in the ϕ_4 system. Top left: u_{10}, top right: u_{20}, bottom left: u_{30}, bottom right: u_{40}. The black solid curve corresponds to the ground truth and the red dash curve is from the SREINet discovered system.

AL model

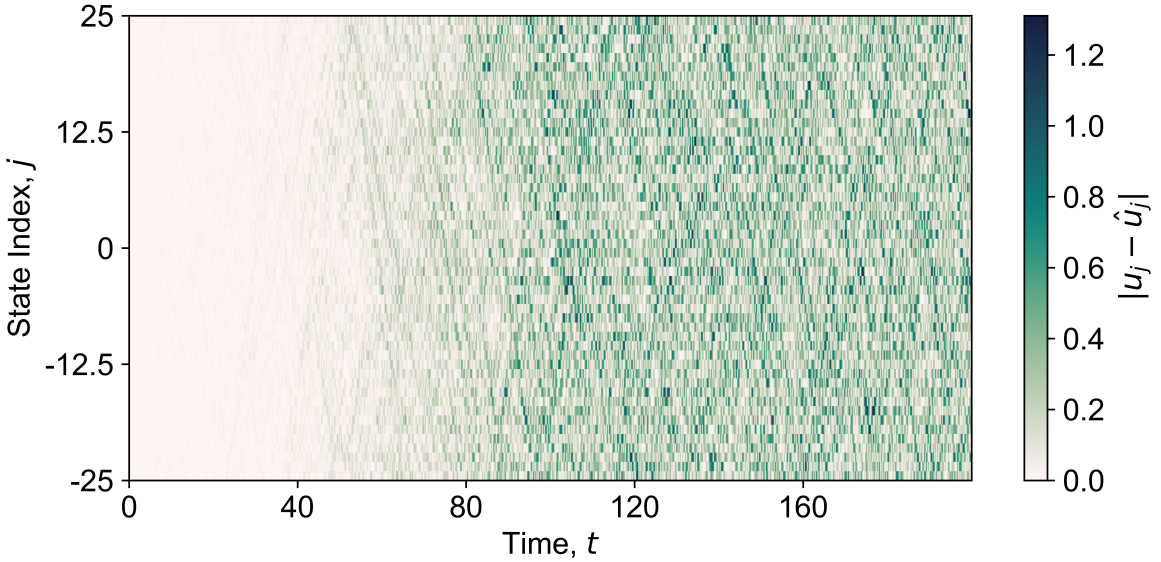
The Ablowitz-Ladik model is given by

$$i\dot{u}_j = -\frac{1}{2}(u_{j+1} - 2u_j + u_{j-1}) - \frac{1}{2}|u_j|^2(u_{j+1} + u_{j-1}), \tag{7}$$

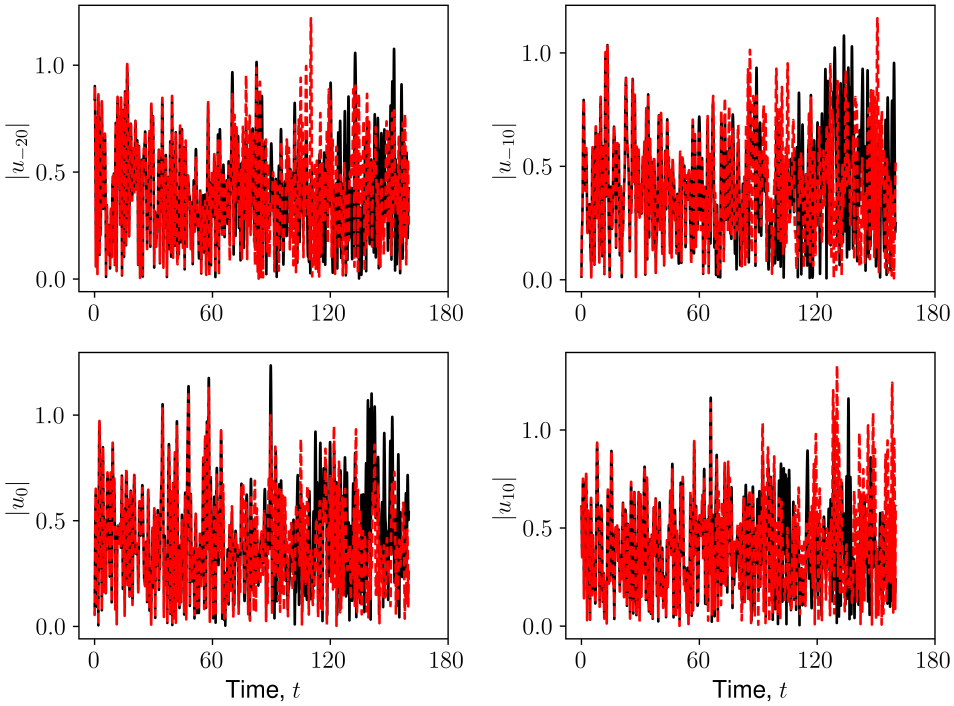
with the periodic boundary conditions $u_{-1} = u_n$, and $u_{n+1} = u_0$. We simulate four trajectories with the time span of $T = 500$, where the initial amplitude of the nodal dynamics are randomly selected and the initial phases are uniformly distributed in $[0, 2\pi]$. The dataset is produced by sampling each trajectory with the time step $dt = 0.1$, with $200,000$ points. Similar to the DNLS, we set $u_j = \alpha_j + i\beta_j$ and get

$$\begin{aligned}
\dot{\alpha}_j &= \frac{1}{2}(-\beta_{j+1} + 2\beta_j - \beta_{j-1}) + \frac{1}{2}(\alpha_j^2 + \beta_j^2)(-\beta_{j+1} - \beta_{j-1}), \\
\dot{\beta}_j &= \frac{1}{2}(\alpha_{j+1} - 2\alpha_j + \alpha_{j-1}) + \frac{1}{2}(\alpha_j^2 + \beta_j^2)(\alpha_{j+1} + \alpha_{j-1}).
\end{aligned} \tag{8}$$

We concatenate the data for the real and imaginary parts and apply the 80-20 ratio to split it into training and validation sets. We generate a 3-layer SREINet with 129 neurons per layer. The candidate functions of each layer are $\mathbf{\Phi} = [1, x_1, x_2, \ldots, x_{128}]$, leading to a network with $2 \times 129 \times 129 = 33,282$ trainable weights. The learning rate of the Adam optimizer is set as 10^{-3}, with the mini-batch size 512. The early-stopping loss threshold is 10^{-10} while the early-stopping pruning threshold is 0.03. The maximum pruning value is 0.10.

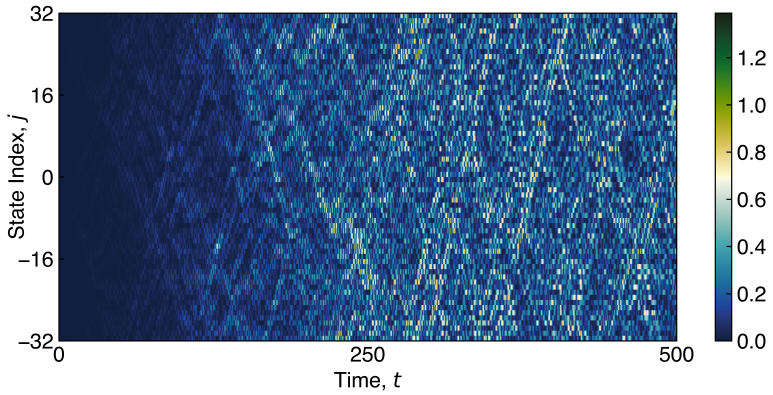


Supplementary Figure 7: Error evolution of the DNLS with random initial conditions. The dynamics of the SREINet identified system accurately match the ground truth for $t < 40$ and divergence occurs afterwards due to chaos.

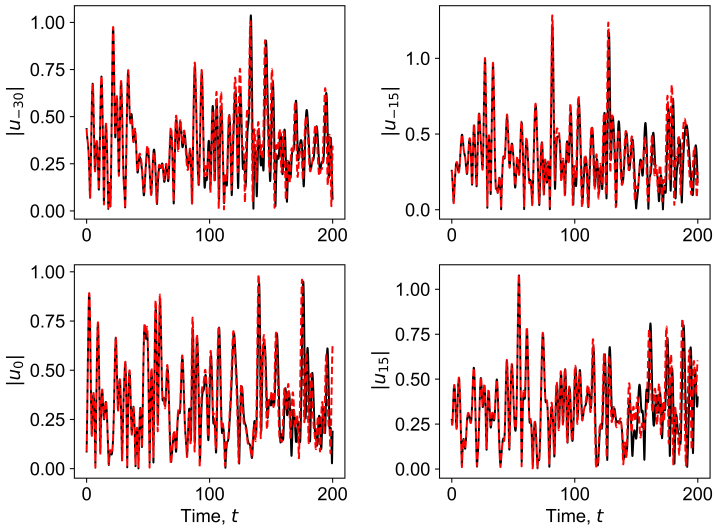


Supplementary Figure 8: Representative time series from the SREINet identified DNLS system in comparison with the ground truth. Shown are the time series at four different spatial sites: $|u_{10}|$, $|u_{15}|$, $|u_{20}|$, and $|u_{30}|$. The SREINet identified system accurately predicts wave propagation in the DNLS system.

The pruning profile has 40 epochs per period. As shown in Supplementary Fig. 9, the SREINet discovered model can precisely predict the evolution of states for $t < 100$. The large prediction errors afterwards are mainly caused by the phase difference, as can be verified through the amplitude histories of, e.g., u_{-30}, u_{-15}, u_0, u_{15} in Supplementary Fig. 10. The agreement between the prediction (red) and ground truth (black) is reasonable, in spite of the discrepancies in $150 < t < 200$.



Supplementary Figure 9: Error evolution of the AL model with random initial conditions. The dynamics generated by the SREINet identified system agree well with the ground truth for $t < 100$.



Supplementary Figure 10: Representative time series generated by the SREINet identified system for the AL model. Shown are the time series from four spatial sites: $|u_{-30}|$, $|u_{-15}|$, $|u_0|$, and $|u_{15}|$. The SREINet discovered system can accurately predict wave propagation in the AL system.

Supplementary Note 3: Computational complexity

Space complexity

The memory consumption of SREINet is estimated by summing three components: weights, layer outputs, and input batch data. Additional memory overhead for gradients and optimizer are neglected as they do not affect the overall space complexity. During training, the total number of floating-point numbers needed by the network is

$$\underbrace{(p-1)(n+1)^2+(n+1)}_{weights}+\underbrace{pm_b(n+1)}_{\substack{layer\\outputs}} \tag{9}$$

where n is the data dimension (network width is argumented by 1 due to the constant term), p is the number of layers, and m_b is the mini-batch size. The first term represents the total number of weights (trainable and fixed). The second term characters the memory required to store layer outputs during forward propagation, which are also needed for gradient computation in backpropagation. The final term represents the memory for mini-batch input. In summary, the space complexity [1] of SREINet is

$$\mathcal{O}(pn^2+pnm_b). \tag{10}$$

In contrast, the space complexity of a recent NN structure [2] is $\mathcal{O}(pn^3+pn^2m_b)$, and that of a matrix-formulation algorithm (e.g., SINDy [3]) is $\mathcal{O}(m(n+p)!/(n!p!))$ (see Supplementary Table 1), where m denotes the total number of data points.

Time complexity

We analyze the time complexity [1] of SREINet during training through floating-point operations (FLOPs) of forward and backward propagation, neglecting other computational overhead.

During forward propagation, each layer (except the last one) performs matrix-vector product, yielding $(n+1)^2m_b$ FLOPs for multiplications and $(n+1)nm_b$ FLOPs for additions. The total cost per layer is therefore

$$F_{fp}=\underbrace{(p-1)(n+1)(2n+1)m_b}_{\substack{matrix-vector\\product}}+\underbrace{(p-1)(n+1)m_b}_{\substack{Hadamard\\product}}+\underbrace{(2n+1)m_b}_{\substack{last\\layer}}, \tag{11}$$

where the second term represents the FLOPs of the Hadamard product, and the third term indicates the FLOPs of the output layer.

To evaluate the FLOPs of backpropagation, we recall that the output of SREINet is denoted by $\mathcal{N}(\boldsymbol{x})$, abbreviated as $\mathcal{N}$ in the following analysis. In addition, the output of the l-th layer is represented by $\boldsymbol{y}^{[l]}$. The gradients of the network output with respect to the parameters and activations of the l-th layer are

$$\frac{\partial\mathcal{N}}{\partial W_{ij}^{[l]}}=\frac{\partial\mathcal{N}}{\partial y_i^{[l+1]}}\frac{\partial y_i^{[l+1]}}{\partial W_{ij}^{[l]}}, \tag{12}$$

$$\frac{\partial\mathcal{N}}{\partial y_i^{[l]}}=\left[\frac{\partial\mathcal{N}}{\partial\boldsymbol{y}^{[l+1]}}\right]^T\frac{\partial\boldsymbol{y}^{[l+1]}}{\partial y_i^{[l]}}, \tag{13}$$

where $\frac{\partial\mathcal{N}}{\partial y_i^{[l+1]}}$ (and $\frac{\partial\mathcal{N}}{\partial\boldsymbol{y}^{[l+1]}}$) is obtained through the backpropagation from the (l+1)-th layer, and computational cost of evaluating $\frac{\partial y_i^{[l+1]}}{\partial W_{ij}^{[l]}}$ and $\frac{\partial\boldsymbol{y}^{[l+1]}}{\partial y_i^{[l]}}$ using automatic differentiation is negligible [4]. Consequently, the FLOPs required for backpropagation through the l-th layer is $\left[(n+1)^2+(2n+1)(n+1)\right]m_b$. The total FLOPs for backpropagation across all layers is therefore

$$F_{bp}=(p-1)\left[(n+1)^2+(2n+1)(n+1)\right]m_b+(n+1)m_b, \tag{14}$$

where $(n+1)m_b$ is the backpropagation FLOPs of the output layer, which involves only addition operations. In summary, combining forward and backward propagation, the time complexity of SREINet per training step is

$$F=F_{fp}+F_{bp}=\mathcal{O}(pn^2m_b). \tag{15}$$

On the contrary, the time complexity per straining step of the recent NN [2] is $\mathcal{O}(pn^3m_b)$. The SINDy algorithm [3] employs Sequential Thresholded Least Squares (STLSQ), which involves solving a least-squares problem as its core computational step. The time complexity of this step only is $\mathcal{O}(ms^2)$, where $s = (n+p)!/(p!n!)$ represents the number of candidate functions in the library.

Supplementary Note 4: Learning with empirical data

We use the triple-pendulum dataset from [5], which contains three experimental datasets of free swing dynamics. The first two datasets are employed for identifying the governing equations of the triple-pendulum system, while the third dataset is used for validation.

The original position data was acquired using optical encoders at a sampling frequency of 10 kHz over a 60-second duration, with velocity data computed via finite difference methods. Applying numerical differentiation again to the velocity data would significantly amplify noise in the acceleration estimations. To mitigate this issue, we downsample the datasets with a step size of 100. Subsequently, we apply smoothed numerical differentiation using a Savitzky-Golay filter with a window length of 7 and polynomial order of 2.

The input variables are defined as $\boldsymbol{x} = [\theta_1, \theta_2, \theta_3, \dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3]$, while the outputs correspond to $\dot{\boldsymbol{x}}$. The training dataset comprises input data constructed by concatenating position and velocity measurements, whereas the output data is formed by concatenating velocity and acceleration data.

The training dataset is partitioned into mini-batches of size 128, and we adopt a learning rate of 0.01. We construct a 3-layer SREINet, where each layer employs candidate functions

$$\boldsymbol{\Phi}(\boldsymbol{x}) = [1, \boldsymbol{x}^T, \cos(\boldsymbol{x})^T, \sin(\boldsymbol{x})^T, \cos(2\boldsymbol{x})^T, \sin(2\boldsymbol{x})^T]^T,$$

with $\cos(\boldsymbol{x}) = [\cos(x_1), \cos(x_2), \ldots, \cos(x_n)]^T$. We train the first three output dimensions (velocities) and the last three output dimensions (accelerations) separately, using different hyperparameters. For the first three dimensions, we employ a pruning threshold of 0.6, an early stopping pruning threshold of 0.4, and an early stopping loss threshold of 10^{-6}. These dimensions are trained for 240 epochs per period with a maximum of 10 periods.

For the last three dimensions, we use a pruning threshold of 0.1, an early stopping pruning threshold of 0.05, and an early stopping loss threshold of 0.2. These dimensions are trained for 200 epochs per period with a maximum of 5 periods. The final loss values for the six dimensions are 4.1×10^{-15}, 1.7×10^{-8}, 2.4×10^{-14}, 5.5×10^{-1}, 1.4, and 3.5, respectively. Despite the relatively large errors in the last three dimensions, the identified system can still provide a reasonable approximation of the original vector field.

Supplementary Table 1: Memory consumption of SINDy

		$\boldsymbol{\Theta}$		$\boldsymbol{\Xi}$	
p	n	**Size** ($m \times r$)	**Memory** (GB)	**Size** ($r \times n$)	**Memory** (GB)
2	10	2000×66	4.92×10^{-4}	66×10	2.46×10^{-6}
	100	2000×5151	3.83×10^{-2}	5151×100	1.9×10^{-3}
	1000	2000×501501	3.74	501501×1000	1.87
3	10	2000×286	2.13×10^{-3}	286×10	1.065×10^{-5}
	100	2000×176851	1.32	176851×100	6.59×10^{-2}
	1000	$2000\times1.67\times10^{8}$	1249.23	$1.67\times10^{8}\times1000$	624.61
5	10	2000×3003	0.022	3003×10	1.12×10^{-4}
	100	$2000\times9.95\times10^{7}$	719.43	$9.95\times10^{7}\times100$	35.97
	1000	$2000\times8.46\times10^{12}$	6.30×10^{7}	$8.46\times10^{12}\times1000$	3.15×10^{7}

Table 1: Nominal memory consumption of SINDy with respect to order of nonlinearity p and number of dimension n. The size and memory requirement of two SINDy data structures: $\boldsymbol{\Theta}$ and $\boldsymbol{\Xi}$, grows exponentially as the dimension n and the polynomial order p increase. Sample points: $m = 2000$. The number of monomials up to the pth order: $r = \binom{n+p}{n}$. Each element in the matrices is a single-precision floating-point number (4 bytes).

Supplementary Table 2: Memory comparison of SREINet and a recent NN

Memory Cost (GB)	SREINet (n)			A recent NN structure [2] (n)		
p	10	100	1000	10	100	1000
2	5.3e-6	8.6e-5	4.2e-3	5.3e-5	8.6e-3	4.2
3	5.7e-6	1.2e-4	7.9e-3	5.7e-5	1.2e-2	7.9
5	6.6e-6	2.0e-4	0.015	6.6e-5	2.0e-2	15.4

Table 2: Memory consumption of SREINet in comparison with that of a recent neural-network structure [2] for various dimensions n and nonlinearity orders p during mini-batch training. SREINet significantly reduces the memory usage, enabling the identification of system dynamics with thousands of dimensions on a desktop computer. In comparison to the matrix-based formulation such as SINDy, SREINet also reduces the memory storage by orders of magnitude for high dimensions. The mini-batch size is 128.

Supplementary Table 3: Training performance and computational cost analysis

Model	Dimensions	Pruning epochs per period	Average epochs	Time/dim [s]	Total time [min]
Lorenz-96	100	30	81.19	30.11	50.18
Kuramoto	60	50	87.55	35.25	35.25
ϕ^4	100	60	40.05	18.83	31.38
DNLS	100	40	101.53	62.07	103.45
AL	128	60	102.10	93.23	198.89

Table 3: Comparison of pruning periods, average training epochs, and computational time of the five models studied in the main text. Except for ϕ^4, the rest four models converge with more than one pruning period. This pruning mechanism has greatly reduced the effort required for hyperparameter tuning and, therefore, significantly enhanced the algorithm's robustness.

Supplementary Table 4: Effect of noise

Noise Level	Mean	Std	25% (Q1)	50% (Median)	75% (Q3)
1%	0.006	0.005	0.002	0.004	0.008
5%	0.033	0.027	0.008	0.025	0.052
10%	0.100	0.075	0.027	0.095	0.159
15%	0.225	0.166	0.062	0.215	0.358

Table 4: Statistics of coefficient errors at different noise levels.

Supplementary Table 5: Comparison with MANDy

	MANDy		SREINet	
Points	Mean [Sec]	Std [Sec]	Mean [Sec]	Std [Sec]
3,000	44.96	2.01	N/A	N/A
4,000	95.78	0.26	94.24	17.44
5,000	200.70	8.30	86.20	18.09
6,000	287.58	2.50	99.84	18.32
7,000	431.67	11.81	83.63	8.48
8,000	587.86	14.23	79.22	17.33
9,000	788.33	5.69	100.54	21.90
10,000	1115.93	0.16	118.30	35.28

Table 5: Computation time of MANDy and SREINet versus data size.

Supplementary Table 6: Training hyperparameters

Model	Learning rate	Loss threshold	Pruning threshold	Pruning period/	Epochs per period	Zero percent	Transition percent	Transition function
Lorenz-96	0.001	$1e-10$	0.3	50	30	15%	75%	Sine in $[-\pi, \pi]$
Kuramoto	0.001	$1e-10$	0.015	50	50	15%	75%	Sine in $[-\pi, \pi]$
ϕ^4	0.001	$1e-10$	0.5	10	60	15%	75%	Sine in $[-\pi, \pi]$
DNLS	0.005	$1e-10$	0.15	20	40	15%	75%	Sine in $[-\pi, \pi]$
AL	0.001	$1e-10$	0.1	20	60	15%	75%	Sine in $[-\pi, \pi]$

Table 6: Hyperparameters for the selected models.

Supplementary Note 5: Identified paradigmatic spatiotemporal dynamic systems

Lorenz-96.

Ground truth:

$$\dot{x}_j = x_{j-1}(x_{j+1} - x_{j-2}) - x_j + 8, \quad j = 1, 2, \ldots, n,$$

with periodic boundary conditions $x_{-1} = x_{n-1}$, $x_0 = x_n$, and $x_{n+1} = x_1$.

Identified system (program output):

(x1)'= 7.999988556 - 0.9999985695*x1 + 0.9999999404*x100*x2 - 0.9999997318*x100*x99
(x2)'= 7.999989033 - 0.9999985695*x2 + x1*x3 - 0.9999999106*x1*x100
(x3)'= 7.999988556 - 0.9999984205*x3 - 0.9999997616*x1*x2 + 1.00000006*x2*x4
(x4)'= 7.999989986 - 0.9999988675*x4 - 0.9999997616*x2*x3 + 0.9999999702*x3*x5
(x5)'= 7.999988556 - 0.9999985695*x5 - 0.9999998212*x3*x4 + 0.9999999702*x4*x6
(x6)'= 7.999989033 - 0.9999984205*x6 - 0.9999996722*x4*x5 + 0.999999851*x5*x7
(x7)'= 7.99998951 - 0.9999987185*x7 - 0.9999996722*x5*x6 + 0.9999999404*x6*x8
(x8)'= 7.999989986 - 0.9999988079*x8 - 0.999999702*x6*x7 + 0.9999997318*x7*x9
(x9)'= 7.99999094 - 0.9999988675*x9 - 0.9999998212*x7*x8 + 0.9999999702*x10*x8
(x10)'= 7.99999094 - 0.9999989569*x10 - 0.999999851*x8*x9 + 0.9999997914*x11*x9
(x11)'= 7.99998951 - 0.9999984205*x11 - 0.999999851*x10*x9 + 0.9999999106*x10*x12
(x12)'= 7.99998951 - 0.9999987185*x12 - 0.9999997318*x10*x11 + 0.9999999106*x11*x13
(x13)'= 7.999990463 - 0.9999989867*x13 - 0.999999702*x11*x12 + 0.999999851*x12*x14
(x14)'= 7.999990463 - 0.9999986887*x14 - 0.9999997318*x12*x13 + 0.9999999702*x13*x15
(x15)'= 7.99998951 - 0.9999986589*x15 - 0.9999998808*x13*x14 + 0.9999999702*x14*x16
(x16)'= 7.999990463 - 0.9999989271*x16 - 0.9999997616*x14*x15 + 0.9999999702*x15*x17
(x17)'= 7.999989986 - 0.9999986887*x17 - 0.999999851*x15*x16 + 0.9999999404*x16*x18
(x18)'= 7.999988556 - 0.9999985695*x18 - 0.9999997318*x16*x17 + 0.999999851*x17*x19
(x19)'= 7.99999094 - 0.9999990463*x19 - 0.999999702*x17*x18 + x18*x20
(x20)'= 7.999989986 - 0.9999986291*x20 - 0.9999997914*x18*x19 + 0.9999999702*x19*x21
(x21)'= 7.999989033 - 0.9999985695*x21 - 0.9999998212*x19*x20 + 0.9999998808*x20*x22
(x22)'= 7.999988556 - 0.9999986291*x22 - 0.9999997616*x20*x21 + 0.999999851*x21*x23
(x23)'= 7.99998951 - 0.9999988079*x23 - 0.999999702*x21*x22 + 1.00000003*x22*x24
(x24)'= 7.99998951 - 0.9999987185*x24 - 0.999999851*x22*x23 + 0.9999997914*x23*x25
(x25)'= 7.999989986 - 0.9999989271*x25 - 0.9999997616*x23*x24 + 0.9999999106*x24*x26
(x26)'= 7.999989986 - 0.9999987483*x26 - 0.9999997914*x24*x25 + 0.9999999106*x25*x27
(x27)'= 7.999989986 - 0.9999989569*x27 - 0.9999999106*x25*x26 + 0.9999999106*x26*x28
(x28)'= 7.99999094 - 0.9999988675*x28 - 0.9999999106*x26*x27 + x27*x29
(x29)'= 7.999988556 - 0.9999986887*x29 - 0.9999999404*x27*x28 + 1.000000119*x28*x30
(x30)'= 7.999988079 - 0.9999997616*x28*x29 + 0.9999999702*x29*x31 - 0.9999984503*x30
(x31)'= 7.999988556 - 0.9999985695*x31 - 0.9999999404*x29*x30 + 0.9999999702*x30*x32
(x32)'= 7.99998951 - 0.9999988675*x32 - 0.9999997914*x30*x31 + 0.9999997914*x31*x33
(x33)'= 7.999988556 - 0.9999985099*x33 - 0.9999998212*x31*x32 + x32*x34
(x34)'= 7.999988556 - 0.9999984503*x34 - 0.999999702*x32*x33 + 0.9999999404*x33*x35
(x35)'= 7.999989033 - 0.9999985099*x35 - 0.9999997318*x33*x34 + 0.999999851*x34*x36
(x36)'= 7.999989986 - 0.9999987185*x36 - 0.999999851*x34*x35 + 0.9999999106*x35*x37
(x37)'= 7.999989986 - 0.9999989271*x37 - 0.9999997914*x35*x36 + 0.9999999404*x36*x38
(x38)'= 7.99998951 - 0.9999986887*x38 - 0.9999997616*x36*x37 + 0.9999999404*x37*x39
(x39)'= 7.99998951 - 0.9999985397*x39 - 0.9999998212*x37*x38 + 1.000000089*x38*x40
(x40)'= 7.99998951 - 0.9999988377*x40 - 0.999999851*x38*x39 + 0.9999999106*x39*x41
(x41)'= 7.999988556 - 0.9999985695*x41 - 0.9999997616*x39*x40 + 0.9999998212*x40*x42
(x42)'= 7.999990463 - 0.9999988973*x42 - 0.9999999106*x40*x41 + 1.00000006*x41*x43
(x43)'= 7.999988556 - 0.9999988079*x43 - 0.9999997616*x41*x42 + 1.00000003*x42*x44
(x44)'= 7.999989033 - 0.9999985695*x44 - 0.9999998212*x42*x43 + 0.9999999702*x43*x45
(x45)'= 7.999987125 - 0.9999983311*x45 - 0.9999998808*x43*x44 + 0.9999998212*x44*x46

(x46)'= 7.99998951 - 0.9999987483*x46 - 0.9999997914*x44*x45 + 0.9999999404*x45*x47
(x47)'= 7.99998951 - 0.9999987781*x47 - 0.9999997318*x45*x46 + x46*x48
(x48)'= 7.999990463 - 0.9999987483*x48 - 0.999999851*x46*x47 + 1.00000006*x47*x49
(x49)'= 7.999989033 - 0.9999988675*x49 - 0.9999997914*x47*x48 + 0.9999998808*x48*x50
(x50)'= 7.999989033 - 0.9999988377*x50 - 0.9999997914*x48*x49 + 1.00000003*x49*x51
(x51)'= 7.999989033 - 0.9999985695*x51 - 0.9999997914*x49*x50 + 0.9999999404*x50*x52
(x52)'= 8.000012398 - 1.00000149*x52 - 1.000000238*x50*x51 + x51*x53
(x53)'= 7.999988079 - 0.9999983609*x53 - 0.999999702*x51*x52 + 1.00000003*x52*x54
(x54)'= 7.999990463 - 0.9999986887*x54 - 0.9999998808*x52*x53 + 0.999999851*x53*x55
(x55)'= 7.999989986 - 0.9999989867*x55 - 0.999999702*x53*x54 + 1.00000003*x54*x56
(x56)'= 7.999990463 - 0.9999989271*x56 - 0.9999998808*x54*x55 + 1.00000006*x55*x57
(x57)'= 7.99998951 - 0.9999986291*x57 - 0.9999997616*x55*x56 + 0.9999999404*x56*x58
(x58)'= 7.99998951 - 0.9999987483*x58 - x56*x57 + 0.9999999404*x57*x59
(x59)'= 7.999989033 - 0.9999986887*x59 - 0.9999998212*x57*x58 + 0.9999997318*x58*x60
(x60)'= 7.99998951 - 0.9999988079*x60 - 0.9999998808*x58*x59 + 1.00000003*x59*x61
(x61)'= 7.99998951 - 0.9999986887*x61 - 0.9999997914*x59*x60 + 1.00000003*x60*x62
(x62)'= 7.999989033 - 0.9999987483*x62 - 0.9999997318*x60*x61 + 0.9999999404*x61*x63
(x63)'= 7.999989986 - 0.9999988377*x63 - 0.9999995828*x61*x62 + 0.9999999106*x62*x64
(x64)'= 7.999989033 - 0.9999984503*x64 - 0.9999997616*x62*x63 + 0.9999999404*x63*x65
(x65)'= 7.999989033 - 0.9999987185*x65 - 0.9999998212*x63*x64 + 0.9999997914*x64*x66
(x66)'= 7.99998951 - 0.9999986887*x66 - 0.9999997318*x64*x65 + 0.999999851*x65*x67
(x67)'= 7.999989986 - 0.9999988079*x67 - 0.9999997318*x65*x66 + 1.000000149*x66*x68
(x68)'= 7.999989986 - 0.9999988675*x68 - 0.9999997914*x66*x67 + 0.9999999702*x67*x69
(x69)'= 7.999990463 - 0.9999988079*x69 - 0.9999999106*x67*x68 + 0.9999999702*x68*x70
(x70)'= 7.999988079 - 0.9999982715*x70 - 0.9999998212*x68*x69 + x69*x71
(x71)'= 7.999990463 - 0.9999986887*x71 - 0.9999997914*x69*x70 + 0.9999999702*x70*x72
(x72)'= 7.999989986 - 0.9999985397*x72 - 0.9999997914*x70*x71 + 0.9999999702*x71*x73
(x73)'= 7.999988556 - 0.9999986887*x73 - 0.9999997616*x71*x72 + 1.000000149*x72*x74
(x74)'= 7.999988556 - 0.9999986589*x74 - 0.9999996126*x72*x73 + x73*x75
(x75)'= 7.999990463 - 0.9999988973*x75 - 0.9999999106*x73*x74 + x74*x76
(x76)'= 7.999989986 - 0.9999987185*x76 - 0.9999999106*x74*x75 + 1.00000006*x75*x77
(x77)'= 7.99998951 - 0.9999986887*x77 - 0.9999997318*x75*x76 + x76*x78
(x78)'= 7.999989033 - 0.9999988377*x78 - 0.9999997914*x76*x77 + 0.9999999702*x77*x79
(x79)'= 7.99999094 - 0.9999990165*x79 - 0.9999999106*x77*x78 + 0.9999999404*x78*x80
(x80)'= 7.999989033 - 0.9999985993*x80 - 0.9999996722*x78*x79 + 0.9999999106*x79*x81
(x81)'= 7.999988079 - 0.9999985397*x81 - 0.9999997914*x79*x80 + 0.9999999106*x80*x82
(x82)'= 7.999989986 - 0.9999988377*x82 - 0.9999997318*x80*x81 + 0.9999998212*x81*x83
(x83)'= 7.999989986 - 0.9999987781*x83 - 0.9999997616*x81*x82 + x82*x84
(x84)'= 7.999989986 - 0.9999987781*x84 - 0.9999997616*x82*x83 + x83*x85
(x85)'= 7.999988079 - 0.9999983907*x85 - 0.9999997318*x83*x84 + 0.999999851*x84*x86
(x86)'= 7.99998951 - 0.9999988377*x86 - 0.999999702*x84*x85 + 0.9999999404*x85*x87
(x87)'= 7.999989986 - 0.9999988675*x87 - 0.9999998212*x85*x86 + 0.9999999702*x86*x88
(x88)'= 7.999989033 - 0.9999984205*x88 - 0.999999851*x86*x87 + 0.9999999106*x87*x89
(x89)'= 7.999987125 - 0.9999983907*x89 - 0.9999996722*x87*x88 + 1.00000003*x88*x90
(x90)'= 7.999989033 - 0.9999985993*x90 - 0.9999997914*x88*x89 + 0.9999998808*x89*x91
(x91)'= 7.99998951 - 0.9999987483*x91 - 0.9999997318*x89*x90 + 1.00000003*x90*x92
(x92)'= 7.99999094 - 0.999999851*x90*x91 + 0.9999999404*x91*x93 - 0.9999988675*x92
(x93)'= 7.99998951 - 0.9999983907*x93 - 0.9999998212*x91*x92 + x92*x94
(x94)'= 7.999989986 - 0.9999987483*x94 - 0.9999997318*x92*x93 + 1.00000003*x93*x95
(x95)'= 7.99999094 - 0.9999989867*x95 - 0.9999997914*x93*x94 + 0.9999999702*x94*x96
(x96)'= 7.99999094 - 0.9999988079*x96 - 0.9999998212*x94*x95 + 0.999999851*x95*x97
(x97)'= 7.999988079 - 0.9999982417*x97 - 0.999999851*x95*x96 + 1.000000089*x96*x98
(x98)'= 7.99998951 - 0.9999987185*x98 - 0.9999997914*x96*x97 + 0.9999999702*x97*x99
(x99)'= 7.999991417 - 0.9999989569*x99 - 0.9999998212*x97*x98 + 0.9999999106*x100*x98

(x100)'= 7.999990463 - 0.9999989867*x100 + 0.9999999106*x1*x99 - 0.9999997914*x98*x99

ϕ^4

Ground truth (state space form):

$$\dot{x}_i = x_{i+n},$$
$$\dot{x}_{i+n} = 2(x_{i+1} + x_{i-1} - 2x_i) + 2(x_i - x_i^3),$$

with free boundary conditions $x_1 = x_0$, $x_{50} = x_{51}$.

Identified system (program output):
(x1)'= 1.000000047*x51
(x2)'= 1.000000106*x52
(x3)'= 0.999999816*x53
(x4)'= 1.000000002*x54
(x5)'= 1.000000049*x55
(x6)'= 0.9999999962*x56
(x7)'= 0.9999999984*x57
(x8)'= 0.9999999509*x58
(x9)'= 0.999999985*x59
(x10)'= 1.000000035*x60
(x11)'= 0.9999999969*x61
(x12)'= 1.00000002*x62
(x13)'= 1.000000001*x63
(x14)'= 0.9999999619*x64
(x15)'= 1.000000018*x65
(x16)'= 0.9999999979*x66
(x17)'= 0.9999999739*x67
(x18)'= 0.999999983*x68
(x19)'= 1.000000038*x69
(x20)'= 1.000000051*x70
(x21)'= 1.000000031*x71
(x22)'= 1.000000057*x72
(x23)'= 1.000000007*x73
(x24)'= 1.000000059*x74
(x25)'= 1.000000046*x75
(x26)'= 1.000000008*x76
(x27)'= 1.000000009*x77
(x28)'= 0.9999999341*x78
(x29)'= 1.000000007*x79
(x30)'= 1.000000000x80
(x31)'= 1.000000000x81
(x32)'= 1.000000001*x82
(x33)'= 0.9999999549*x83
(x34)'= 1.000000038*x84
(x35)'= 0.9999999983*x85
(x36)'= 1.00000004*x86
(x37)'= 1.000000007*x87
(x38)'= 1.000000057*x88
(x39)'= 1.000000002*x89
(x40)'= 0.9999999955*x90
(x41)'= 1.000000005*x91
(x42)'= 1.000000015*x92

(x43)'= 0.9999999808*x93
(x44)'= 0.9999998647*x94
(x45)'= 1.000000031*x95
(x46)'= 1.000000012*x96
(x47)'= 1.000000024*x97
(x48)'= 1.00000002*x98
(x49)'= 1.000000029*x99
(x50)'= 1.000000027*x100
(x51)'= - 1.999999992*x1^3 + 2.000000052*x2
(x52)'= 2.00000017*x1 - 1.999999968*x2 - 1.999999843*x2^3 + 1.999999766*x3
(x53)'= 2.000000189*x2 - 1.999999985*x3 - 1.999999927*x3^3 + 1.999999985*x4
(x54)'= 2.000000182*x3 - 1.99999997*x4 + 2.000000182*x5 - 2.000000193*x4^3
(x55)'= 2.000000026*x4 - 1.999999817*x5 + 2.000000026*x6 - 2.000000039*x5^3
(x56)'= 2.000000033*x5 - 2.000000033*x6 + 2.000000033*x7 - 2.000000034*x6^3
(x57)'= 1.999999978*x6 - 1.999999978*x7 - 1.999999964*x7^3 + 1.999999978*x8
(x58)'= 2.000000028*x7 - 2.000000221*x8 + 2.000000028*x9 - 1.99999996*x8^3
(x59)'= 2.000000481*x8 - 2.000000481*x9 + 2.00000043*x10 - 2.00000024*x9^3
(x60)'= 1.999999876*x9 - 1.999999876*x10 - 2.000000226*x10^3 + 1.999999876*x11
(x61)'= 1.99999992*x10 - 1.99999992*x11 + 2.000000123*x12 - 2.000000228*x11^3
(x62)'= 1.999999917*x11 - 1.999999917*x12 - 2.000000106*x12^3 + 2.000000117*x13
(x63)'= 1.999999982*x12 - 1.999999982*x13 + 1.999999982*x14 - 1.999999913*x13^3
(x64)'= 2.000000156*x13 - 1.999999967*x14 + 1.999999967*x15 - 1.999999755*x14^3
(x65)'= 2.000000022*x14 - 1.99999983*x15 + 2.000000022*x16 - 1.999999828*x15^3
(x66)'= 1.999999841*x15 - 2.000000037*x16 + 1.999999841*x17 - 2.000000215*x16^3
(x67)'= 1.999999976*x16 - 1.999999976*x17 - 2.000000097*x17^3 + 1.999999976*x18
(x68)'= 1.999999712*x17 - 2.000000268*x18 + 2.000000083*x19 - 2.000000339*x18^3
(x69)'= 2.000000115*x18 - 2.000000115*x19 + 2.000000115*x20 - 2.000000114*x19^3
(x70)'= 2.000000119*x19 - 2.000000119*x20 + 1.99999992*x21 - 2.000000136*x20^3
(x71)'= 1.999999837*x20 - 2.000000053*x21 - 1.999999774*x21^3 + 1.999999837*x22
(x72)'= 2.000000146*x21 - 1.999999945*x22 + 1.999999743*x23 - 2.000000133*x22^3
(x73)'= 2.000000208*x22 - 2.000000005*x23 - 1.999999935*x23^3 + 2.000000208*x24
(x74)'= 1.999999934*x23 - 1.999999745*x24 - 2.00000011*x24^3 + 1.999999934*x25
(x75)'= 2.000000028*x24 - 2.000000228*x25 + 2.000000028*x26 - 1.999999942*x25^3
(x76)'= 1.999999966*x25 - 2.000000056*x26 - 1.99999996*x26^3 + 2.000000056*x27
(x77)'= 1.999999896*x26 - 1.999999896*x27 - 2.000000095*x27^3 + 2.000000099*x28
(x78)'= 2.000000062*x27 - 2.000000062*x28 + 2.000000062*x29 - 2.000000014*x28^3
(x79)'= 2.000000044*x28 - 1.999999664*x29 - 2.000000224*x29^3 + 1.999999854*x30
(x80)'= 2.000000097*x29 - 2.000000097*x30 + 2.000000097*x31 - 2.000000069*x30^3
(x81)'= 1.999999884*x30 - 2.000000277*x31 - 2.000000012*x31^3 + 2.000000277*x32
(x82)'= 2.000000116*x31 - 1.999999935*x32 - 2.000000046*x32^3 + 1.999999935*x33
(x83)'= - 2.000000201*x33 + 2.000000103*x32 - 1.999999993*x33^3 + 2.000000103*x34
(x84)'= 1.999999933*x33 - 2.000000126*x34 - 2.000000377*x34^3 + 2.000000126*x35
(x85)'= 1.999999875*x34 - 1.999999875*x35 - 1.999999947*x35^3 + 1.999999875*x36
(x86)'= 2.000000063*x35 - 2.000000269*x36 - 1.999999866*x36^3 + 2.000000063*x37
(x87)'= 1.999999839*x36 - 2.000000013*x37 - 1.999999782*x37^3 + 1.999999839*x38
(x88)'= 2.000000035*x37 - 2.000000234*x38 + 2.000000035*x39 - 1.999999808*x38^3
(x89)'= 2.000000031*x38 - 2.000000031*x39 + 1.999999816*x40 - 2.000000069*x39^3
(x90)'= 1.99999996*x39 - 1.999999944*x40 - 1.999999978*x40^3 + 1.999999944*x41
(x91)'= 2.000000059*x40 - 2.000000059*x41 - 2.000000052*x41^3 + 2.000000059*x42
(x92)'= 1.999999844*x41 - 2.000000248*x42 - 1.999999918*x42^3 + 1.999999844*x43
(x93)'= 1.999999945*x42 - 2.000000144*x43 - 1.999999915*x43^3 + 2.000000144*x44
(x94)'= 1.999999832*x43 - 1.999999832*x44 - 2.000000116*x44^3 + 2.00000003*x45
(x95)'= 2.000000163*x44 - 1.999999964*x45 + 2.000000163*x46 - 1.999999941*x45^3
(x96)'= 1.999999943*x45 - 2.000000123*x46 + 1.999999943*x47 - 1.99999986*x46^3

(x97)'= 1.999999823*x46 - 1.999999823*x47 + 2.000000229*x48 - 1.999999758*x47^3
(x98)'= 2.000000077*x47 - 2.000000268*x48 + 2.000000077*x49 - 2.000000493*x48^3
(x99)'= 1.999999972*x48 - 1.999999972*x49 + 1.999999765*x50 - 1.999999919*x49^3
(x100)'= 1.999999989*x49 - 2.000000133*x50^3

DNLS

Ground truth:

$$\begin{aligned}\dot{\alpha}_j &= -2(\beta_{j+1} + \beta_{j-1} - 2\beta_j) - \beta_j(\alpha_j^2 + \beta_j^2),\\ \dot{\beta}_j &= 2(\alpha_{j+1} + \alpha_{j-1} - 2\alpha_j) + \alpha_j(\alpha_j^2 + \beta_j^2),\end{aligned} \tag{16}$$

where $u_j = \alpha_j + i\beta_j$. We assume periodic boundary conditions: $u_0 = u_{50}$, $u_{51} = u_1$.

Identified system (program output):

(a1)'= - 1.999999906*b2 + 3.99999988*b1 - 0.9999998209*a1^2*b1 - 0.9999998414*b1^3 - 2.000000154*b50
(a2)'= - 2.000000007*b1 + 4.000000015*b2 - 2.000000007*b3 - 0.9999999576*a2^2*b2 - 0.9999999743*b2^3
(a3)'= - 2.0000001*b2 + 4.000000199*b3 - 2.0000001*b4 - 1.000000404*a3^2*b3 - 0.9999998923*b3^3
(a4)'= - 1.000000008*a4^2*b4 - 1.999999949*b3 + 3.999999898*b4 - 1.000000008*b4^3 - 1.999999949*b5
(a5)'= - 2.000000019*b4 + 4.000000038*b5 - 2.000000019*b6 - 1.000000037*a5^2*b5 - 1.000000065*b5^3
(a6)'= - 1.999999987*b5 + 3.999999974*b6 - 1.999999987*b7 - 0.9999998763*a6^2*b6 - 0.9999999521*b6^3
(a7)'= - 1.999999926*b6 + 3.999999853*b7 - 1.000000012*a7^2*b7 - 1.000000014*b7^3 - 1.999999926*b8
(a8)'= - 0.9999999124*a8^2*b8 - 1.999999995*b7 + 3.999999989*b8 - 0.9999999601*b8^3 - 1.999999995*b9
(a9)'= - 2.000000028*b8 + 4.000000056*b9 - 2.000000028*b10 - 1.000000982*a9^2*b9 - 0.9999998735*b9^3
(a10)'= - 0.9999995406*a10^2*b10 - 2.000000101*b9 + 4.000000201*b10 - 0.9999995903*b10^3 - 1.999999744*b11
(a11)'= - 2.00000001*b10 + 4.00000002*b11 - 2.00000001*b12 - 1.00000001*a11^2*b11 - 0.999999996*b11^3
(a12)'=4.000000011*b12 - 2.000000058*b11 - 1.000000202*a12^2*b12 - 0.9999999382*b12^3 - 1.999999883*b13
(a13)'= - 2.000000082*b12 + 4.000000163*b13 - 2.000000082*b14 - 1.000000674*a13^2*b13 - 1.000000006*b13^3
(a14)'= - 1.000000058*a14^2*b14 - 2.000000031*b13 + 4.000000063*b14 - 1.000000118*b14^3 - 2.000000031*b15
(a15)'= - 0.9999998546*a15^2*b15 - 2.00000002*b14 + 4.00000004*b15 - 1.000000101*b15^3 - 2.00000002*b16
(a16)'= - 1.000000206*a16^2*b16 - 2.000000001*b15 + 4.000000003*b16 - 0.9999999832*b16^3 - 2.000000001*b17
(a17)'= - 2.000000043*b16 + 3.999999916*b17 - 2.000000043*b18 - 1.000000303*a17^2*b17 - 1.000000001*b17^3
(a18)'= - 2.00000009*b17 + 3.999999769*b18 - 0.9999999846*a18^2*b18 - 0.9999996906*b18^3 - 1.999999885*b19
(a19)'= - 2.000000047*b18 + 4.000000093*b19 - 2.000000047*b20 - 0.9999999497*a19^2*b19 - 0.9999998784*b19^3
(a20)'= - 1.000000654*a20^2*b20 - 2.000000225*b19 + 4.000000093*b20 - 1.000000135*b20^3 - 2.000000225*b21
(a21)'= - 2.000000034*b20 + 3.999999946*b21 - 2.000000034*b22 - 1.000000086*a21^2*b21 - 0.9999999848*b21^3
(a22)'= - 1.999999973*b21 + 3.999999946*b22 - 1.000000007*a22^2*b22 - 0.9999998598*b22^3 - 1.999999973*b23
(a23)'= - 1.999999962*b22 + 3.999999924*b23 - 1.999999962*b24 - 1.000000109*a23^2*b23 - 0.9999999955*b23^3
(a24)'= - 0.9999994674*a24^2*b24 - 2.000000079*b23 + 3.999999771*b24 - 0.9999995415*b24^3 - 2.000000079*b25
(a25)'= - 1.000000059*a25^2*b25 - 2.00000006*b24 + 4.000000119*b25 - 1.000000068*b25^3 - 2.00000006*b26
(a26)'= - 2.000000002*b25 + 4.000000003*b26 - 2.000000002*b27 - 0.9999998949*a26^2*b26 - 0.9999999*b26^3
(a27)'= - 1.000000069*a27^2*b27 - 2.000000036*b26 + 4.000000073*b27 - 1.000000165*b27^3 - 2.000000036*b28
(a28)'= - 1.999999957*b27 + 4.000000155*b28 - 1.999999957*b29 - 1.000000215*a28^2*b28 - 1.000000048*b28^3
(a29)'= - 1.000000138*a29^2*b29 - 2.000000042*b28 + 4.000000083*b29 - 0.9999997001*b29^3 - 2.000000042*b30
(a30)'= - 1.000000274*a30^2*b30 - 2.000000013*b29 + 4.000000026*b30 - 0.9999999443*b30^3 - 2.000000013*b31
(a31)'= - 2.000000088*b30 + 4.000000175*b31 - 2.000000088*b32 - 0.9999999692*a31^2*b31 - 0.9999999848*b31^3
(a32)'= - 2.00000004*b31 + 4.000000079*b32 - 2.00000004*b33 - 0.9999999704*a32^2*b32 - 1.000000014*b32^3
(a33)'=3.999999968*b33 - 2.000000091*b32 - 0.9999999376*a33^2*b33 - 0.9999999925*b33^3 - 2.000000091*b34
(a34)'=4.000000013*b34 - 0.9999998633*a34^2*b34 - 1.999999971*b33 - 0.9999999821*b34^3 - 1.999999971*b35
(a35)'= - 2.00000002*b34 + 4.000000039*b35 - 2.00000002*b36 - 1.00000001*a35^2*b35 - 0.999999968*b35^3
(a36)'= - 1.000000211*a36^2*b36 - 2.000000002*b35 + 4.000000003*b36 - 0.9999999279*b36^3 - 2.000000002*b37
(a37)'= - 2.000000101*b36 + 4.000000203*b37 - 2.000000101*b38 - 0.9999999557*a37^2*b37 - 1.00000003*b37^3
(a38)'= - 0.9999999482*a38^2*b38 - 1.999999969*b37 + 3.999999939*b38 - 0.9999998896*b38^3 - 1.999999969*b39
(a39)'= - 1.000000033*a39^2*b39 - 1.999999967*b38 + 3.999999934*b39 - 1.00000006*b39^3 - 1.999999967*b40
(a40)'= - 2.000000062*b39 + 4.000000124*b40 - 2.000000062*b41 - 0.999999942*a40^2*b40 - 0.9999999463*b40^3

(a41)'= - 1.999999959*b40 + 3.999999919*b41 - 0.9999998242*a41^2*b41 - 0.9999998912*b41^3 - 1.999999959*b42
(a42)'= - 1.999999986*b41 + 4.000000049*b42 - 1.999999986*b43 - 1.000000051*a42^2*b42 - 1.000000074*b42^3
(a43)'= - 2.000000003*b42 + 4.000000006*b43 - 2.000000003*b44 - 1.000000055*a43^2*b43 - 1.000000011*b43^3
(a44)'= - 2.000000078*b43 + 4.000000155*b44 - 2.000000078*b45 - 1.000000179*a44^2*b44 - 0.9999999475*b44^3
(a45)'= - 1.000000187*a45^2*b45 - 1.999999929*b44 + 3.999999858*b45 - 1.000000067*b45^3 - 1.999999929*b46
(a46)'= - 2.000000042*b45 + 4.000000084*b46 - 2.000000042*b47 - 0.9999997128*a46^2*b46 - 1.000000081*b46^3
(a47)'= - 1.000000092*a47^2*b47 - 1.999999974*b46 + 3.999999948*b47 - 0.9999997161*b47^3 - 1.999999974*b48
(a48)'= - 2.000000092*b47 + 4.000000183*b48 - 2.000000092*b49 - 1.000000135*a48^2*b48 - 1.000000043*b48^3
(a49)'= - 2.000000057*b48 + 4.000000113*b49 - 2.000000057*b50 - 1.000000151*a49^2*b49 - 0.999999959*b49^3
(a50)'= - 2.00000016*b1 - 1.999999964*b49 + 3.999999928*b50 - 0.9999999898*a50^2*b50 - 0.9999997811*b50^3
(b1)'= - 3.999999889*a1 + 1.000000005*a1^3 + 1.000000202*a1*b1^2 + 1.999999944*a2 + 1.999999775*a50
(b2)'=1.999999961*a1 - 3.999999921*a2 + 0.9999998528*a2^3 + 1.999999961*a3 + 0.9999998844*a2*b2^2
(b3)'=2.000000023*a2 - 4.000000046*a3 + 2.000000023*a4 + 0.9999999787*a3^3 + 1.000000034*a3*b3^2
(b4)'=1.99999996*a3 - 3.99999992*a4 + 1.000000017*a4^3 + 1.000000031*a4*b4^2 + 1.99999996*a5
(b5)'=2.000000003*a4 - 4.000000005*a5 + 2.000000003*a6 + 1.00000002*a5^3 + 1.000000046*a5*b5^2
(b6)'=2.000000115*a5 - 3.999999872*a6 + 0.9999999546*a6^3 + 1.999999936*a7 + 1.000000059*a6*b6^2
(b7)'=1.999999989*a6 - 4.000000049*a7 + 1.999999992*a8 + 1.000000059*a7^3 + 1.000000102*a7*b7^2
(b8)'=1.999999903*a7 - 3.999999805*a8 + 1.00000009*a8^3 + 0.9999999788*a8*b8^2 + 1.999999903*a9
(b9)'=2.000000047*a8 - 4.000000094*a9 + 2.000000047*a10 + 1.000000023*a9^3 + 0.9999999685*a9*b9^2
(b10)'= - 4.000000304*a10 + 1.999999974*a9 + 1.000000829*a10^3 + 1.999999974*a11 + 0.9999996891*a10*b10^2
(b11)'=2.000000278*a10 - 4.000000168*a11 + 0.9999998725*a11^3 + 0.9999999655*a11*b11^2 + 2.000000084*a12
(b12)'=2.000000015*a11 - 4.000000031*a12 + 0.9999999954*a12^3 + 1.000000107*a12*b12^2 + 2.000000015*a13
(b13)'=2.000000114*a12 - 4.000000227*a13 + 2.000000114*a14 + 1.00000003*a13^3 + 0.9999999543*a13*b13^2
(b14)'=2.000000288*a13 - 4.000000174*a14 + 0.9999999756*a14^3 + 2.000000087*a15 + 0.9999999193*a14*b14^2
(b15)'=1.999999958*a14 - 3.999999916*a15 + 1.999999958*a16 + 0.9999999728*a15^3 + 0.9999997729*a15*b15^2
(b16)'=2.000000011*a15 - 4.000000023*a16 + 0.9999996481*a16^3 + 0.999999668*a16*b16^2 + 2.000000011*a17
(b17)'=2.000000003*a16 - 4.000000007*a17 + 2.000000003*a18 + 1.000000148*a17^3 + 0.9999993176*a17*b17^2
(b18)'=2.000000013*a17 - 4.000000026*a18 + 0.9999999362*a18^3 + 2.000000013*a19 + 1.000000193*a18*b18^2
(b19)'=2.000000039*a18 - 4.000000078*a19 + 2.000000039*a20 + 0.9999999438*a19^3 + 1.000000036*a19*b19^2
(b20)'=2.000000069*a19 - 4.000000137*a20 + 1.000000295*a20^3 + 1.00000016*a20*b20^2 + 2.000000069*a21
(b21)'=2.00000004*a20 - 4.000000081*a21 + 2.00000004*a22 + 0.9999999717*a21^3 + 1.000000212*a21*b21^2
(b22)'=2.000000076*a21 - 4.000000153*a22 + 0.9999996205*a22^3 + 2.000000076*a23 + 0.9999998329*a22*b22^2
(b23)'=2.00000005*a22 - 4.000000101*a23 + 2.00000005*a24 + 1.00000001*a23^3 + 1.000000067*a23*b23^2
(b24)'=2.000000078*a23 - 3.999999775*a24 + 0.9999995971*a24^3 + 2.000000078*a25 + 0.9999997633*a24*b24^2
(b25)'=1.999999951*a24 - 3.999999901*a25 + 0.9999999934*a25^3 + 1.999999951*a26 + 0.9999999253*a25*b25^2
(b26)'=2.000000099*a25 - 3.999999826*a26 + 0.9999999922*a26^3 + 2.000000099*a27 + 0.9999996216*a26*b26^2
(b27)'=2.000000001*a26 - 4.000000001*a27 + 2.000000001*a28 + 1.00000001*a27^3 + 0.9999997886*a27*b27^2
(b28)'=2.000000006*a27 - 4.000000011*a28 + 0.9999999849*a28^3 + 1.000000129*a28*b28^2 + 2.000000006*a29
(b29)'=2.000000073*a28 - 4.000000145*a29 + 2.000000073*a30 + a29^3 + 0.9999999797*a29*b29^2
(b30)'=1.999999932*a29 - 3.999999863*a30 + 1.0000001*a30^3 + 1.000000034*a30*b30^2 + 1.999999932*a31
(b31)'= - 3.999999933*a31 + 1.999999978*a30 + 0.9999999015*a31^3 + 1.999999978*a32 + 0.9999999574*a31*b31^2
(b32)'=2.000000045*a31 - 4.000000079*a32 + 2.000000045*a33 + 0.9999999495*a32^3 + 0.9999999286*a32*b32^2
(b33)'=1.999999975*a32 - 4.000000305*a33 + 1.000000013*a33^3 + 1.000002627*a33*b33^2 + 1.999999975*a34
(b34)'=1.999999939*a33 - 3.999999879*a34 + 0.9999999883*a34^3 + 0.9999999919*a34*b34^2 + 1.999999939*a35
(b35)'=2.000000015*a34 - 4.00000003*a35 + 2.000000015*a36 + 0.9999999845*a35^3 + 0.9999999537*a35*b35^2
(b36)'=1.999999985*a35 - 3.99999997*a36 + 0.999999978*a36^3 + 0.9999999683*a36*b36^2 + 1.999999985*a37
(b37)'=1.99999997*a36 - 4.000000058*a37 + 1.99999997*a38 + 0.9999998759*a37^3 + 0.9999999368*a37*b37^2
(b38)'=2.000000057*a37 - 4.000000113*a38 + 0.9999999736*a38^3 + 2.000000057*a39 + 1.000000019*a38*b38^2
(b39)'=2.000000027*a38 - 4.000000054*a39 + 1.00000008*a39^3 + 2.000000027*a40 + 0.9999998972*a39*b39^2
(b40)'=1.999999966*a39 - 3.999999932*a40 + 1.999999966*a41 + 0.9999999548*a40^3 + 1.000000067*a40*b40^2
(b41)'=1.99999996*a40 - 3.99999992*a41 + 0.9999997698*a41^3 + 1.99999996*a42 + 1.000000053*a41*b41^2
(b42)'=1.999999997*a41 - 3.999999994*a42 + 1.999999997*a43 + 0.9999999761*a42^3 + 1.000000061*a42*b42^2
(b43)'=1.999999983*a42 - 3.999999965*a43 + 0.9999999391*a43^3 + 1.999999983*a44 + 0.999999967*a43*b43^2
(b44)'=2.000000081*a43 - 4.000000161*a44 + 0.9999999164*a44^3 + 1.000000038*a44*b44^2 + 2.000000081*a45

(b45)'=2.000000083*a44 - 4.000000167*a45 + 2.000000083*a46 + 0.999999838*a45^3 + 1.000000596*a45*b45^2
(b46)'= - 3.999999985*a46 + 2.000000001*a45 + 0.9999999732*a46^3 + 1.000000013*a46*b46^2 + 2.000000001*a47
(b47)'= - 3.999999957*a47 + 2.000000102*a46 + 0.9999999699*a47^3 + 0.9999999476*a47*b47^2 + 1.999999918*a48
(b48)'=2.000000068*a47 - 4.000000136*a48 + 2.000000068*a49 + 0.9999999219*a48^3 + 1.000000046*a48*b48^2
(b49)'=2.000000088*a48 - 4.000000176*a49 + 2.000000088*a50 + 0.9999999621*a49^3 + 0.9999999467*a49*b49^2
(b50)'= - 3.999999923*a50 + 1.999999971*a1 + 1.999999971*a49 + 0.9999999349*a50^3 + 0.9999998079*a50*b50^2

AL model

The identified AL model is not presented herein due to the length of equations. Interested readers may consult the notebook *SREINet_AL.ipynb*.

Kuramoto model

The identified Kuramoto model is not presented herein due to the length of equations. Interested readers may consult the notebook *SREINet_Kuramoto.ipynb*.

References

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. *Introduction to Algorithms* (MIT Press, 2022), 4 edn.

[2] Xing, S., Han, Q. & Charalampidis, E. CombOpNet: a neural-network accelerator for SINDy. *J. Vib. Test. Sys. Dyn.* **9**, 1–20 (2025).

[3] Brunton, S. L., Proctor, J. L. & Kutz, J. N. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences* **113**, 3932–3937 (2016).

[4] Baydin, A. G., Pearlmutter, B. A., Radul, A. A. & Siskind, J. M. Automatic differentiation in machine learning: a survey. *J. Mach. Learn. Res.* **18**, 5595–5637 (2017).

[5] Kaheman, K. *et al.* The experimental multi-arm pendulum on a cart: A benchmark system for chaos, learning, and control. *HardwareX* **15**, e00465 (2023).