Automated Discovery of Operable Dynamics from Videos - Supplementary Information

A Details on datasets

In this section, we provide more information on the physics governing equations and the data collection processes for the four systems under study.

A.1 Spring mass

The spring mass system describes the motion of an object connected to a spring, and it is governed by the following equations of motion:

$$\dot{x} = v, \tag{1}$$

$$\dot{v} = -\frac{k}{m}x, \tag{2}$$

where x is the object's position and v is its respective velocity. The object's mass is $m = 1\text{kg}$ and the spring constant is $k = 80\text{N/m}$. The two variables x and v specify the system's state and form a set of state variables of the system.

The spring mass system is a linear oscillator whose solution is given by:

$$x(t) = x_0 \cos(\omega t) + \frac{v_0}{\omega}\sin(\omega t), \tag{3}$$

$$v(t) = -\omega x_0 \sin(\omega t) + v_0 \cos(\omega t), \tag{4}$$

where $\omega = \sqrt{\frac{k}{m}}$ is the natural frequency of the system, and (x_0, v_0) is the system's initial state. The system has only one equilibrium state at $x = 0, v = 0$ that is stable.

To collect the data, we randomly sampled the object's initial position and velocity. In total, we collected 1,200 sequences with 60fps. We used 960 of these sequences for training, 120 of them for validation, and 120 of them for testing.

A.2 Single pendulum

The single pendulum system is governed by the following equations of motion:

$$\dot{\theta} = \Omega, \tag{5}$$

$$\dot{\Omega} = -\frac{3g}{2L}\sin\theta, \tag{6}$$

where θ is the angle of the pendulum arm and Ω is its respective angular velocity. The pendulum mass is $m = 1$kg and the pendulum length is $L = 0.5$m. The two variables θ and Ω specify the system's state and form a set of state variables of the system.

The system has a stable equilibrium state at $\theta = 0, \Omega = 0$ and an unstable equilibrium state at $\theta = \pi, \Omega = 0$. Suppose that both the initial angle θ_0 and the initial angular velocity Ω_0 are small, one can solve for

$$\theta(t) = \theta_0 \cos(\omega t) + \frac{\Omega_0}{\omega}\sin(\omega t), \tag{7}$$

$$\Omega(t) = -\omega\theta_0 \sin(\omega t) + \Omega_0 \cos(\omega t), \tag{8}$$

from the linearized equations of motion, where $\omega = \sqrt{\frac{3g}{2L}}$ is the natural frequency of the system.

To collect the data, we randomly sampled the pendulum arm's initial angle and angular velocity. In total, we collected 1,200 sequences with 60fps. We used 960 of these sequences for training, 120 of them for validation, and 120 of them for testing.

A.3 Double pendulum

Our double pendulum system's data was collected from real experiments, using a two colored chaotic pendulum from 3D scientific: the first arm is black and the second arm is blue. The physical parameters of the double pendulum used for the experiments is shown in Supplementary Figure 1. Using the pivot attachment that came with the pendulum, the pendulum was installed against a brown-beige wall in the laboratory. There are 4 bearings on the pendulum. Three of them are fixed in place and one is left loose to reduce friction. We used an iPhone7 to record videos at 720p and 240fps.

Length of the first arm: 20.5 cm	 Double Pendulum
Length of the second arm: 17.9 cm	
Mass of the first arm: 0.262 kg	
Mass of the second arm: 0.110 kg	
Depth of the arms: 3.8 cm	

Supplementary Figure 1 Physical parameters of the double pendulum system

While there is no known closed-form, exact solution to the double pendulum system, numerical approximations and simplified analytical solutions for specific cases can be found using methods in classical mechanics. Let us first derive the equations of motion of the double pendulum system, with simplifying assumptions such that the pendulums are 2-dimensional rectangles of uniform density and there are no frictional forces acting on the system. Denote m_1 and m_2 the masses of the two arms of the double pendulum, L_1 and W_1 the length and width of the first arm, and L_2 and W_2 the length and width of the second arm. The momenta of inertia of the two arms are:

$$I_1 = \frac{1}{12} m_1 (L_1^2 + W_1^2) \tag{9}$$

$$I_2 = \frac{1}{12} m_2 (L_2^2 + W_2^2). \tag{10}$$

We specify the system state by the two arms' angular positions θ_1 and θ_2, and their respective angular velocities Ω_1 and Ω_2. The kinetic energy of the system is the sum of the two arms' translational and rotational kinetic energies, which is given by:

$$T = \frac{1}{2}\left(\frac{1}{4} m_1 L_1^2 + m_2 L_1^2 + I_1\right)\Omega_1^2 + \frac{1}{2}\left(\frac{1}{4} m_2 L_2^2 + I_2\right)\Omega_2^2 + \frac{1}{2} m_2 L_1 L_2 \Omega_1 \Omega_2 \cos(\theta_1 - \theta_2). \tag{11}$$

The potential energy of the system is the sum of the two arms' gravitational potential energies. Taking the configuration that both arms are horizontal as the zero point, the potential energy of the system is given by:

$$V = -\left(\frac{1}{2} m_1 + m_2\right) g L_1 \cos\theta_1 - \frac{1}{2} m_2 g L_2 \cos\theta_2. \tag{12}$$

Using the Lagrangian $L = T - V$, the equations of motion of the system can be derived as:

$$\dot{\theta}_1 = \Omega_1 \tag{13}$$

$$\dot{\theta}_2 = \Omega_2, \tag{14}$$

$$M_{\theta_1 - \theta_2} \begin{bmatrix} \dot{\Omega}_1 \\ \dot{\Omega}_2 \end{bmatrix} = \begin{bmatrix} -\frac{1}{2} L_1 L_2 \sin(\theta_1 - \theta_2)\Omega_2^2 - g L_1 \sin\theta_1 \\ \frac{1}{2} L_1 L_2 \sin(\theta_1 - \theta_2)\Omega_1^2 - \frac{1}{2} g L_2 \sin\theta_2 \end{bmatrix}, \tag{15}$$

where the matrix

$$M_{\theta_1 - \theta_2} = \begin{bmatrix} L_1^2 + \frac{I_1}{m_1} & \frac{1}{2} L_1 L_2 \cos(\theta_1 - \theta_2) \\ \frac{1}{2} L_1 L_2 \cos(\theta_1 - \theta_2) & \frac{1}{4} L_2^2 + \frac{I_2}{m_2} \end{bmatrix}. \tag{16}$$

The total energy of the system is the sum of kinetic and potential energies, which is given by:

$$E = \frac{1}{2}\left(\frac{1}{4}m_1L_1^2 + m_2L_1^2 + I_1\right)\dot{\theta}_1^2 + \frac{1}{2}\left(\frac{1}{4}m_2L_2^2 + I_2\right)\dot{\theta}_2^2 + \frac{1}{2}m_2L_1L_2\dot{\theta}_1\dot{\theta}_2\cos(\theta_1 - \theta_2)$$
$$- \left(\frac{1}{2}m_1 + m_2\right)gL_1\cos\theta_1 - \frac{1}{2}m_2gL_2\cos\theta_2. \tag{17}$$

The stable equilibrium state of the system, corresponding to the lowest energy configuration, is at $\theta_1 = \theta_2 = \Omega_1 = \Omega_2 = 0$. Linearizing the system around this equilibrium state, we obtain the linear system:

$$\dot{\theta}_1 = \Omega_1, \tag{18}$$
$$\dot{\theta}_2 = \Omega_2, \tag{19}$$
$$M_0\begin{bmatrix}\dot{\Omega}_1\\ \dot{\Omega}_2\end{bmatrix} = \begin{bmatrix}-gL_1\theta_1\\ -\frac{1}{2}gL_2\theta_2\end{bmatrix}, \tag{20}$$

where the matrix M_0 is given as

$$M_0 = \begin{bmatrix}L_1^2 + \frac{I_1}{m_1} & \frac{1}{2}L_1L_2\\ \frac{1}{2}L_1L_2 & \frac{1}{4}L_2^2 + \frac{I_2}{m_2}\end{bmatrix}. \tag{21}$$

By inverting M_0, we can write down the linear system as

$$\dot{\theta}_1 = \Omega_1, \tag{22}$$
$$\dot{\theta}_2 = \Omega_2, \tag{23}$$
$$\dot{\Omega}_1 = -\frac{1}{4}gL_1L_2^2(\theta_1 - \theta_2) - \frac{I_2}{m_2}gL_1\theta_1, \tag{24}$$
$$\dot{\Omega}_2 = \frac{1}{2}gL_1^2L_2(\theta_1 - \theta_2) - \frac{I_1}{2m_1}gL_2\theta_2. \tag{25}$$

The solution to this system is a composition of two simple oscillators with frequencies determined by the constants $g, L_1, L_2, m_1, m_2, I_1, I_2$. We numerically evaluated the frequencies as $\omega_1 = 14.007$ and $\omega_2 = 6.404$.

To construct the dataset, we collected a total of 100 videos, with an approximate length of 15 seconds for each video. We used 80 of these videos for training, 10 of them for validation, and 10 of them for testing. For better video quality, we trimmed each video to 11s in order to avoid the movement at the beginning and the end of recording caused by humans and small changes in brightness or illumination caused by the camera. Another reason is that the dynamics towards the late part of the recordings are more predictable due to the lack of energy and the loss of momentum. Afterwards, we sub-sampled the video to construct a video dataset with 60 fps to produce sufficient visual difference between subsequent frames in a prediction triplet. To feed the video frames into our visual predictive models, the images are resized to 128×128.

Since we are interested in evaluating the results of prediction from the double pendulum system, we further equalized the background of the pendulum system with a simple color filtering so that our vision algorithms can detect the position and orientation of the pendulum arms with another color filtering during the evaluation process. We performed this additional step only to the double pendulum, for the sake of evaluation alone, while other systems do not involve this extra pre-processing step.

A.4 Cylinder wake

In fluid dynamics, the flow past a circular cylinder is a well-known example that showcases the transition from a laminar steady wake to laminar periodic vortex shedding [1–3]. In the latter regime, vortexes are formed and shed from the cylinder in a regular and oscillating manner, creating a vortex street in the wake. This system is fully described by the two-dimensional incompressible Navier-Stokes equations:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \nu \nabla^2 \mathbf{u}, \tag{26}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{27}$$

where $\mathbf{u} = (u_x, u_y)$ represents the flow's velocity field, p represents the pressure divided by the constant density, $\nabla = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y})$ represents the gradient operator, and $\nu > 0$ is the kinematic viscosity.

A key quantity for describing the flow patterns is the Reynolds number $Re = U_\infty D_c/\nu$, where U_∞ is the free-stream velocity (velocity far from the cylinder), and D_c is the diameter of the cylinder. It has been demonstrated [1–3] that the solution to (26) describes a laminar steady wake when the Reynolds number Re is small and it describes laminar periodic vortex shedding when Re is large enough.

We generated our dataset by running direct numerical simulations [4, 5] on a domain $-15 \leq x \leq 35$, $-15 \leq y \leq 15$, with a unit-diameter cylinder centered at $x = 0$, $y = 0$. We set the initial velocity $\mathbf{u}^0 = (u_x^0, 0)$ and sample u_x^0 from $[0, 1]$. Thus, our dataset contains videos for flows with Reynolds numbers Re ranging from 0 to 100. Figure 7 shows the plots for trajectories in smooth neural state variables for different Reynolds numbers (in other words, different initial u_x values).

As demonstrated in [2], the system has non-unique equilibrium states, and those equilibrium states are all stable when the Reynolds number is small.

A.5 Computer vision algorithms for estimating physical quantities

In this section, we provide details on the computer vision algorithms used to estimate physical quantities from images for the four systems under study.

Spring mass position

Algorithm 1 Spring mass position estimation

Require: X (Spring Mass Image), S (scale mapping pixels to meters), (x_c, y_c) (pixel coordinate of center of image)

Ensure: position: spring mass displacement (meters)

1: $(c_x, c_y) \Leftarrow \textbf{rect}(X)$ center of fitted rectangle of mass from image

2: position $\Leftarrow S*(c_x - x_c)$ displacement from the center of the image scaled to meters

Single pendulum angle

Algorithm 2 Single pendulum angle estimation

Require: X (Single Pendulum Image), (x_c, y_c) (pixel coordinate of center of image)

Ensure: angle: single pendulum angle (degrees)

1: $(c_x, c_y) \Leftarrow \textbf{rect}(X)$ center of fitted rectangle of pendulum from image

2: $b \Leftarrow \textbf{box}(X)$ fitted box points of pendulum from image

3: $v = (v_x, v_y) \Leftarrow \textbf{LongEdge}(b)$ vector connecting box corners of longer edge

4: $v_{ref} \Leftarrow (c_x - x_c, c_y - y_c)$ reference vector

5: **if** $v \cdot v_{ref} < 0$ **then**

6: $\quad v \Leftarrow -v$

7: **end if**

8: angle $\Leftarrow \textbf{rad2Angle}(\arctan\left(\frac{-v_y}{v_x}\right) + \frac{\pi}{2})$ counter clockwise angle from downward direction between 0 and 360

Double pendulum angles

Algorithm 3 Double pendulum angles estimation

Require: X (Double Pendulum Image), (x_c, y_c) (pixel coordinate of center of image)
Ensure: angle 1: upper pendulum angle (degrees), angle 2: lower pendulum angle (degrees)

$(c_x^u, c_y^u) \Leftarrow$ **rect upper**(X) center of fitted rectangle of upper pendulum from image
$b^u \Leftarrow$ **box upper**(X) fitted box points of upper pendulum from image
$v^u = (v_x^u, v_y^u) \Leftarrow$ **LongEdge**(b^u) vector connecting box corners of longer edge
$v_{ref}^u \Leftarrow (c_x^u - x_c, c_y^u - y_c)$ reference vector for the upper pendulum
if $v^u \cdot v_{ref}^u < 0$ **then**
    $v^u \Leftarrow -v^u$
end if
angle 1$\Leftarrow$ **rad2Angle**$(\arctan\left(\frac{-v_y^u}{v_x^u}\right) + \frac{\pi}{2})$ counter clockwise angle from downward direction between 0 and 360
$(c_x^l, c_y^l) \Leftarrow$ **rect lower**(X) center of fitted rectangle of lower pendulum from image
$b^l \Leftarrow$ **box lower**(X) fitted box points of lower pendulum from image
$v^l = (v_x^l, v_y^l) \Leftarrow$ **LongEdge**(b^l) vector connecting box corners of longer edge
$v_{ref}^l \Leftarrow (c_x^l - 2 * c_x^u + x_c, c_y^l - 2 * c_y^u + y_c)$ reference vector for lower pendulum
if $v^l \cdot v_{ref}^l < 0$ **then**
    $v^l \Leftarrow -v^l$
end if
angle 2$\Leftarrow$ **rad2Angle**$(\arctan\left(\frac{-v_y^l}{v_x^l}\right) + \frac{\pi}{2})$ counter clockwise angle from downward direction between 0 and 360

Cylinder wake normalized energy

Algorithm 4 Cylinder wake normalized energy estimation

Require: X (Cylinder Wake Image), ρ_N (density such that maximum velocities leads to $E_N = 1$), $c2idx$ (pixel color to $[0, 1]$ scale), $v_x^{\min}$ (minimum x velocity), $v_x^{\max}$ (maximum x velocity), $v_y^{\min}$ (minimum y velocity), $v_y^{\max}$ (maximum y velocity)
Ensure: E_N: energy normalized to [0,1]

$V_x \Leftarrow v_x^{\min} + (v_x^{\max} - v_x^{\min}) * c2idx(X)$ pixels mapped to x velocities
$V_y \Leftarrow v_y^{\min} + (v_y^{\max} - v_y^{\min}) * c2idx(X)$ pixels mapped to y velocities
$E_N \Leftarrow \sum \frac{1}{2}\rho(V_x^2 + V_y^2)$

B Details on results

B.1 Intrinsic dimension estimation

In Supplementary Table 1, we provide the estimated intrinsic dimensions for the studied systems, compared to their ground truth values. The intrinsic dimension for each system was estimated by averaging the results of the Levina-Bickel's algorithm on 64 and 8192 dimensional latent vectors, using three different random seeds. The standard errors are also shown for each averaged value. We also provide the rounded values of these estimated intrinsic dimensions to their nearest integers.

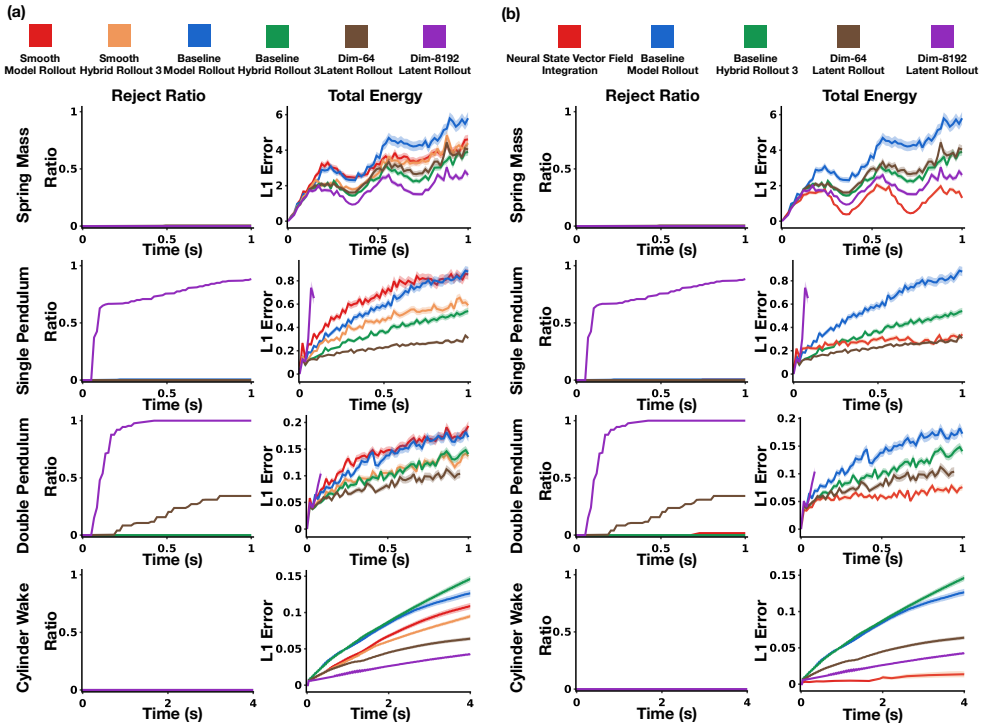
Spring Mass			**Single Pendulum**		
Estimated	Rounded	Ground Truth	Estimated	Rounded	Ground Truth
1.96(±0.01)	2	2	2.10(±0.01)	2	2
Double Pendulum			**Cylinder Wake**		
Estimated	Rounded	Ground Truth	Estimated	Rounded	Ground Truth
4.39(±0.05)	4	4	2.52(±0.02)	3	Unknown

Supplementary Table 1 The estimated intrinsic dimensions for the studied systems

B.2 Discovery of smooth neural state variables and neural state vector fields

We demonstrate the long-term prediction stability and accuracy for our smooth neural state variables, where we compare the computer vision based energy estimation values for the decoded images against the estimated values from their ground truth images. The reject ratio shows the ratio of images that the computer vision estimation fails to derive a value, and it is scaled to remove the cases in which the computer vision estimation fails in the ground truth images themselves. In Supplementary Figure 2(a), we show the long term accuracy of the model rollout and hybrid rollout procedures as described in [6]. Note that the smooth neural state variables allow similar level of accuracy and stability to non smooth neural state variables. Furthermore, compared to the dim-64 and dim-8192 prediction schemes, neural state variables, both smooth and non-smooth, allow stable long term predictions with no rejections. In Supplementary Figure 2(b), we show the long term accuracy and stability of our neural state vector field based predictions. Note that we can achieve a better level of accuracy than the rollout methods, with similar level of stability. In Supplementary Figure 2(c), we show the neural state variable auto-encoders' pixel reconstruction error, highlighting the similar level of accuracy even when trained with smoothness constraints. Finally in Supplementary Figure 2(d), we show the single step pixel reconstruction error of the neural state vector fields trained on smooth and baseline neural state variables. Neural state vector fields trained on smooth neural state variables are consistently more

accurate than those trained on baseline neural state variables in the lower dimensional systems, but we see a slight decrease in accuracy for the cylinder wake system.



(c)

Neural State Variable Auto-encoder Pixel Reconstruction Error

Neural State Variable Type	Smooth	Baseline
Spring Mass	1.850e-04 (±3.298e-05)	1.026e-04 (±3.103e-06)
Single Pendulum	2.874e-04 (±2.898e-05)	1.491e-04 (±1.180e-05)
Double Pendulum	6.316e-04 (±4.931e-06)	5.441e-04 (±1.328e-05)
Cylinder Wake	7.884e-06 (±9.618e-07)	5.373e-06 (±1.596e-07)

(d)

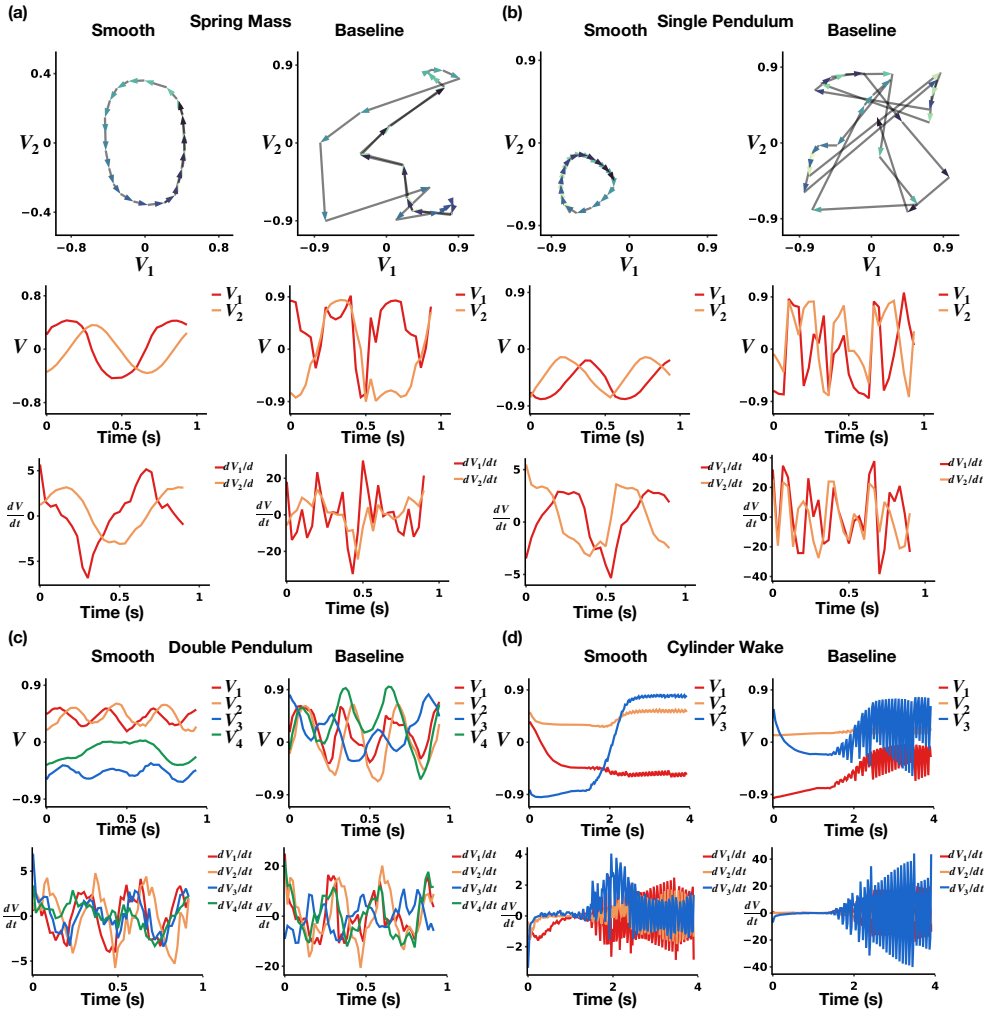
Neural State Vector Field Integration Single Step Pixel Reconstruction Error

Neural State Variable Type	Smooth	Baseline
Spring Mass	3.086e-04 (±4.629e-05)	1.024e-03 (±2.034e-04)
Single Pendulum	3.300e-04 (±4.594e-05)	3.482e-03 (±3.823e-04)
Double Pendulum	8.086e-04 (±3.890e-05)	1.551e-03 (±1.360e-04)
Cylinder Wake	1.373e-04 (±7.388e-05)	4.527e-05 (±3.204e-05)

Supplementary Figure 2 (a) Long-term prediction accuracy from rollout predictions (b) Long-term prediction accuracy through neural state vector field integration (c) Neural State Variable Auto-encoder Pixel Reconstruction Error (d) Neural State Vector Field Integration Single Step Pixel Reconstruction Error

While preserving similar levels of predictive accuracy, smooth neural state variables display significantly smoother trajectories. A more detailed comparison of the trajectories shown in Figure 2 are given in Supplementary Figure 3, where we also

show the derivative $\frac{\mathrm{d}\mathbf{V}_t}{\mathrm{d}t}$ of each trajectory. We observe that the derivatives are an order of magnitude greater for the baseline neural state variable trajectories compared to our smooth neural state variable trajectories.



Supplementary Figure 3 Comparison between smooth and non-smooth neural state variable trajectories. For each system, the same image sequence is encoded to neural state variable trajectories with and without smoothness constraints. Each trajectory is plotted in their respective neural state variable state space as well as a time series against time. We also show the finite difference between neighboring states divided by the time step plotted against time.

To further demonstrate the improved smoothness of our extracted variables, we compared smooth and baseline neural state variables through a family of quantitative

smoothness metrics. Let us denote $\mathbf{V} \in \mathbb{R}^d$ the neural state variable with intrinsic dimension d. For a continuous-in-time trajectory $\mathbf{V}(t)$, we can form a generalized metric to measure its smoothness:

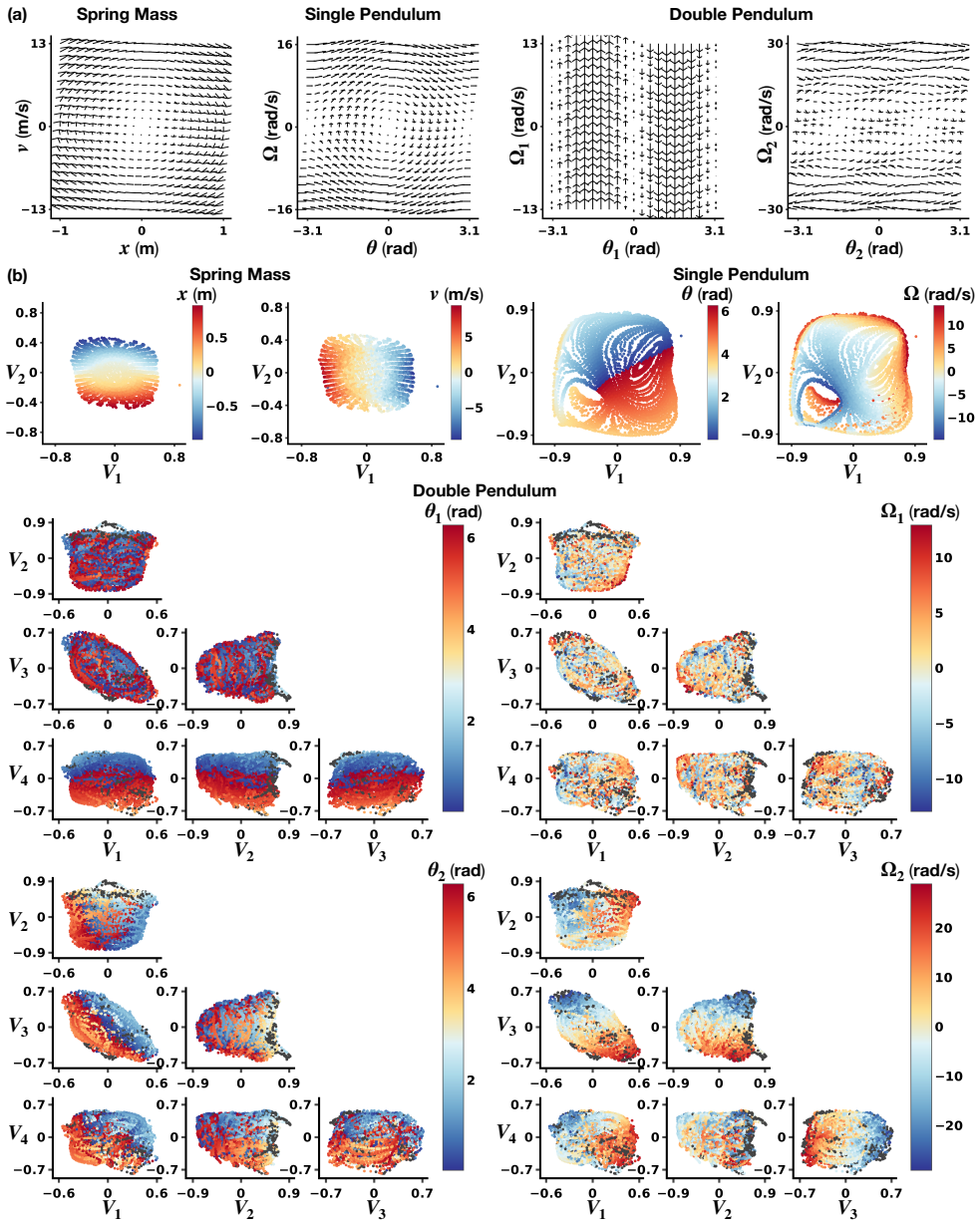
$$\mathrm{SM}_{k,p}(\mathbf{V}(t)) = \begin{cases} \left(\int_0^{(N-1)dt} \left\| \frac{d^k \mathbf{V}(t)}{dt^k} \right\|^p dt \right)^{\frac{1}{p}}, & 1 \leq p < \infty, \\ \max_{t \in [0,(N-1)dt]} \left\| \frac{d^k \mathbf{V}(t)}{dt^k} \right\|, & p = \infty, \end{cases} \tag{28}$$

where k is the order of differentiation on $\hat{\mathbf{V}}$ and p is usually chosen as 1, 2, or ∞. As shown in Supplementary Table 2, the results show a consistent trend that our smooth neural state variables exhibit smoother trajectories than the baseline neural state variables. All the above comparisons consistently demonstrate the improved smoothness of our discovered smooth neural state variables compared to the unconstrained baseline neural state variables, and underscore the necessity of enforcing smoothness and the effectiveness of applying smoothness constraints through our method.

	Spring Mass		**Single Pendulum**	
	Smooth	**Baseline**	**Smooth**	**Baseline**
$\mathbf{SM_{1,1}}$	3.89(±0.05)	5.69(±0.10)	2.10(±0.04)	5.70(±0.13)
$\mathbf{SM_{2,1}}$	401.36(±22.30)	820.38(±17.83)	136.01(±7.49)	940.81(±30.36)
$\mathbf{SM_{1,\infty}}$	10.28(±0.31)	36.92(±0.64)	7.41(±0.68)	42.82(±0.95)
$\mathbf{SM_{2,\infty}}$	1579.85(±92.00)	5214.81(±100.00)	947.09(±96.00)	5977.84(±155.00)
	Double Pendulum		**Cylinder Wake**	
	Smooth	**Baseline**	**Baseline**	**Baseline**
$\mathbf{SM_{1,1}}$	3.27(±0.04)	5.86(±0.09)	4.49(±0.12)	31.36(±0.92)
$\mathbf{SM_{2,1}}$	267.45(±6.82)	417.35(±12.51)	530.40(±17.05)	5021.17(±156.04)
$\mathbf{SM_{1,\infty}}$	7.41(±0.25)	21.86(±0.51)	2.17(±0.03)	28.23(±0.66)
$\mathbf{SM_{2,\infty}}$	1134.83(±65.00)	2756.03(±103.00)	368.63(±10.00)	4967.85(±135.00)

Supplementary Table 2 Comparison between the calculated average and standard error of the smoothness metrics for smooth and baseline neural state variable trajectories, each averaged over all trajectories in the test data across three random seeds. The derivative norms were further normalized by the range of the neural state variables to ensure a fair comparison.

In Supplementary Figure 4, the discovered state spaces are compared to the ground truth phase spaces for the spring mass, single pendulum, and double pendulum systems. This figure demonstrates that our discovered state spaces closely mirror the structure of the ground truth phase spaces and display smooth variation in positions and velocities as states change.



Supplementary Figure 4 The discovered state space is compared to the ground truth state space. (a) The vector fields for the spring mass, single pendulum, and double pendulum systems (centered around the stable equilibrium) according to classical mechanics are visualized. As the double pendulum system dataset is collected from real observations, the corresponding dynamics are only approximate, and do not account for any non-conservative forces that may exist in the dataset. (b) The discovered neural state variables are visualized against the corresponding physical variables identified by the computer vision algorithms.

B.3 Near-equilibrium analysis

We describe the equilibrium identification process in further detail in Algorithm 5. Given our set of neural state variables encoded from our test data, the following algorithm returns a set of identified equilibrium states, each labeled with their long term stability. We further show the stability analysis process in Algorithm 6. As per the Lyapunov definition of stability, we used an additional parameter $\mathcal{E}$ for all the ϵ to test the stability with. For our experiments on the spring mass, single pendulum, and double pendulum systems, we used $C = 10$, $n_d = 10$, $n_e = 10$, $\mathcal{E} = \{0.5\%, 1\%, 3\%, 5\%, 10\%\}$, and $T = 300$, which is five times longer than the length of the sequences in our datasets. Δt used for trajectory integration is the same time interval from our datasets.

Algorithm 5 Equilibrium states identification

Require: C (number of candidate equilibrium states), N (number of elements per dimension to perform grid search), n_d (number of random directions to sample from for each ϵ), n_e (number of sub-samples for each ϵ), T (length of sampled trajectories in number of time steps), $\mathcal{E}$ (list of ϵ to test for stability)

Ensure: $\mathbf{V}^{\text{eq}}$ the set of identified equilibrium states and their stability

1: $\mathbf{V}^{\text{test}} \Leftarrow \{\mathbf{V}_0, \mathbf{V}_{\Delta t}, \mathbf{V}_{2\Delta t}, \cdots\}$ encoded from sequences in test data
2: $\mathbf{D} \Leftarrow$ the smallest rectangle covering all states in $\mathbf{V}^{\text{test}}$
3: $\mathbf{V}^{\text{candidates}} \Leftarrow$ The C states in $\mathbf{V}^{\text{test}}$ with the lowest $\left\|\hat{F}(\mathbf{V})\right\|$ values
4: **if** $\mathbf{V}$ has dimension > 2 **then**
5: $\quad \mathbf{V}^{\text{candidates}} \Leftarrow \mathbf{V}^{\text{candidates}} \cup$ C states from N^d uniform grid covering $\mathbf{D}$ with the lowest $\left\|\hat{F}(\mathbf{V})\right\|$ values
6: **end if**
7: $\mathbf{V}^{\text{eq}} \Leftarrow \{\}$
8: **for** $\mathbf{V}_{\text{candidate}}$ in $\mathbf{V}^{\text{candidates}}$ **do**
9: $\quad$ **if** the root-solving algorithm with $\mathbf{V}_{\text{candidate}}$ as the initial guess successfully returns $\mathbf{V}_{\text{eq}} \in \mathbf{D}$ **then**
10: $\quad\quad$ stab $\Leftarrow$ Stability($\mathbf{V}_{\text{eq}}$, n_d, n_e, T, $\mathcal{E}$)
11: $\quad\quad$ $\mathbf{V}^{\text{eq}} \Leftarrow \mathbf{V}^{\text{eq}} \cup \{(\mathbf{V}_{\text{eq}},$ stab$)\}$
12: $\quad$ **end if**
13: **end for**

The cylinder wake system does not have a unique stable equilibrium state, unlike the other three systems studied. We thus used a different candidate generation process than that shown in Algorithm 5 line 3. Instead, we utilized the observed attraction towards an equilibrium state from Section 2.4, and derived a candidate equilibrium state from each trajectory in the test set. These candidate states were generated by taking the average of the final ten states in each trajectory. Figure 4 shows more identified stable equilibrium states for the cylinder wake system.

Algorithm 6 Stability check

Require: $\mathbf{V}_{\text{eq}}$ (identified equilibrium state), n_d (number of random directions to sample from for each ϵ), n_e (number of sub-samples for each ϵ), T (length of sampled trajectories in number of time steps), $\mathcal{E}$ (list of ϵ to test for stability)

Ensure: stable: True ($\mathbf{V}_{\text{eq}}$ is a stable equilibrium state) or False ($\mathbf{V}_{\text{eq}}$ is an unstable equilibrium state)

$\mathbf{V}^{\text{samples}} \Leftarrow \{\}$
for ϵ in $\mathcal{E}$ **do**
  for $i = 1, 2, \cdots, n_d$ **do**
    $r_i \Leftarrow$ unit vector with random direction
    $\epsilon_e \Leftarrow \epsilon / n_e$
    for $j = 1, 2, \cdots, n_e$ **do**
      $\mathbf{V}_{i,j} \Leftarrow \mathbf{V}_{\text{eq}} + j \times \epsilon_e \times r_i$
      $\mathbf{V}_{i,j}^{\text{traj}} \Leftarrow$ trajectory by integrating $\hat{F}$ from $\mathbf{V}_{i,j}$ for $T-1$ steps
      $d_{\text{ini}} \Leftarrow \|\mathbf{V}_i - \mathbf{V}_{\text{eq}}\| = j \times \epsilon_e$
      $d_{\max} \Leftarrow \max_{0 \le k < T} \|\mathbf{V}_{i,j}^{\text{traj}}(k\Delta t) - \mathbf{V}_{\text{eq}}\|$
      $\mathbf{V}^{\text{samples}} \Leftarrow \mathbf{V}^{\text{samples}} \cup \{(d_{\text{ini}}, d_{\max})\}$
    end for
  end for
end for
$\mathcal{E}_{stability} \Leftarrow \{\epsilon : \text{False } \forall \epsilon \in \mathcal{E}\}$
for ϵ in $\mathcal{E}$ **do**
  if $\exists (d^*_{\text{ini}}, d^*_{\max}) \in \mathbf{V}^{\text{samples}} : d^*_{\max} < \epsilon \wedge \forall (d_{\text{ini}}, d_{\max}) \in \mathbf{V}^{\text{samples}},\ d_{\text{ini}} \le d^*_{\text{ini}} \Rightarrow d_{\max} < \epsilon$ **then**
    $\mathcal{E}_{stability}[\epsilon] \Leftarrow$ True
  end if
end for
stable $\Leftarrow (\forall \epsilon \in \mathcal{E} \quad \mathcal{E}_{stability}[\epsilon] \leftrightarrow \text{True})$

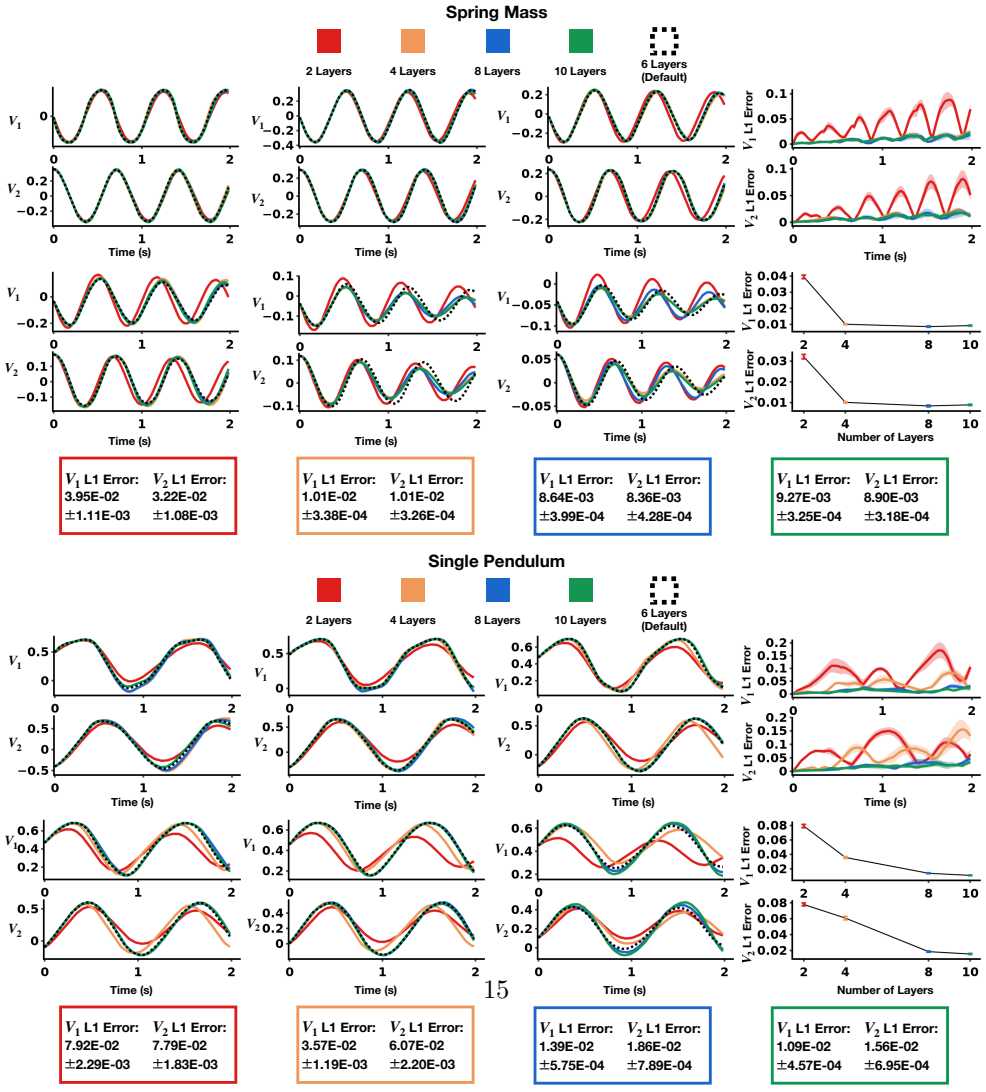
B.4 Non-equilibrium analysis

The rendered videos of the integrated trajectories shown in Figure 5(a) are provided as Supplementary videos. The corresponding filenames for the trajectories of each color are shown in Supplementary Table 3.

The quality of the periodic orbits in the integrated trajectories shown in Figure 5(a) were analyzed in relation to the complexity of the neural network used to approximate the neural state vector field. The results in Supplementary Figure 5 show that the complexity of the neural network has little effect on the quality of identified periodic orbits. While having too few layers may reduce the model's prediction stability, adding additional layers show minimal difference in integrated trajectories from those from our utilized networks.

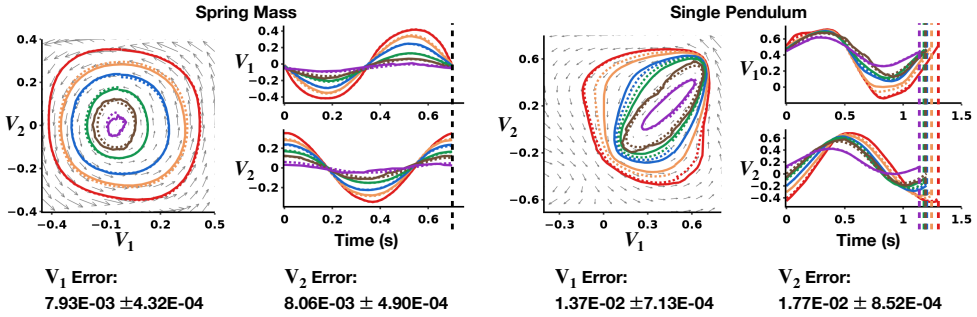
Spring Mass		
Purple	**Brown**	**Green**
springMass-traj0.m4v	springMass-traj1.m4v	springMass-traj2.m4v
Blue	**Yellow**	**Red**
springMass-traj3.m4v	springMass-traj4.m4v	springMass-traj5.m4v
Single Pendulum		
Purple	**Brown**	**Green**
singlePendulum-traj0.m4v	singlePendulum-traj1.m4v	singlePendulum-traj2.m4v
Blue	**Yellow**	**Red**
singlePendulum-traj3.m4v	singlePendulum-traj4.m4v	singlePendulum-traj5.m4v

Supplementary Table 3 The rendered videos of the integrated trajectories for the spring mass and single pendulum systems shown in Figure 5(a) are included as Supplementary Material with their filenames presented in this table. The colors of the trajectories ordered from smallest to largest amplitudes are: purple, brown, green, blue, yellow, and red.



Supplementary Figure 5 Periodic trajectories from identical initial states as in Figure 5(a) are integrated using neural networks of varying depth for the neural state vector field. Deeper neural networks also utilize wider layers with maximum widths increasing in powers of 2.

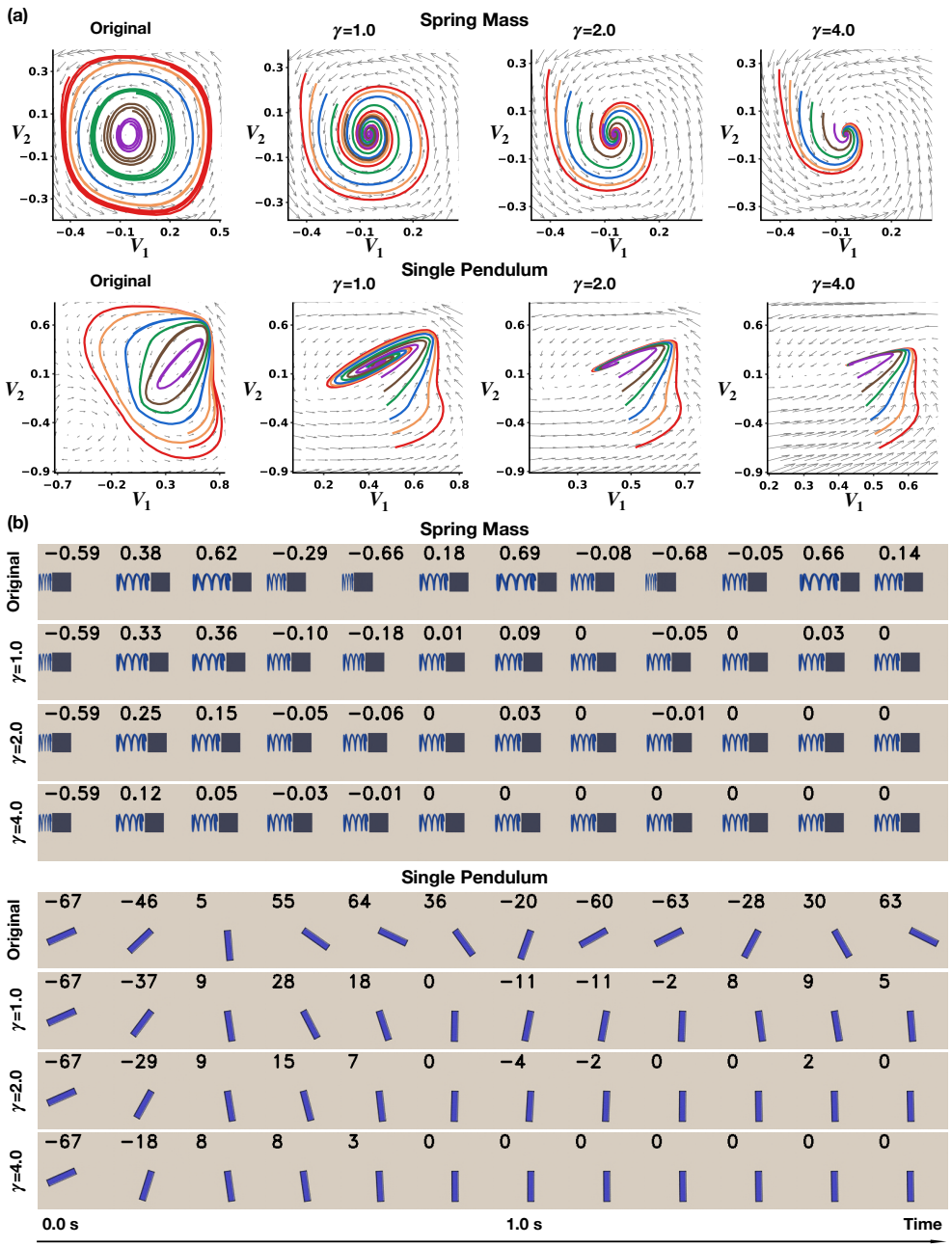
Although the integrated trajectories do not form perfectly closed orbits, the identified periodic behavior can further be verified by fitting a boundary value problem using our neural state vector field. In Supplementary Figure 6, the fitted periodic orbits for each trajectory from Figure 5(a) are plotted (solid line) in conjunction to the sampled trajectories (dotted line). The converged periodic orbits are nearly identical to the original integrated trajectories, and we are able to see the similar pattern of uniform period for the spring mass and varying periods for the single pendulum. The mean and margin of error for the L1 errors of the periodic orbits with respect to the sampled trajectories are also plotted below the trajectory plots for each system.



Supplementary Figure 6 A boundary value problem was solved by fitting an initial guess of one period of the sampled trajectories shown in Figure 5(a). The fitted curves are shown with a solid curve, and the integrated trajectories are shown with a dotted curve of the same color. The length of the period was approximated by finding the first point within the integrated trajectory with the minimum distance to the initial state. The periods are marked with dotted vertical lines.

B.5 Synthesizing new data with parameterized novel dynamics

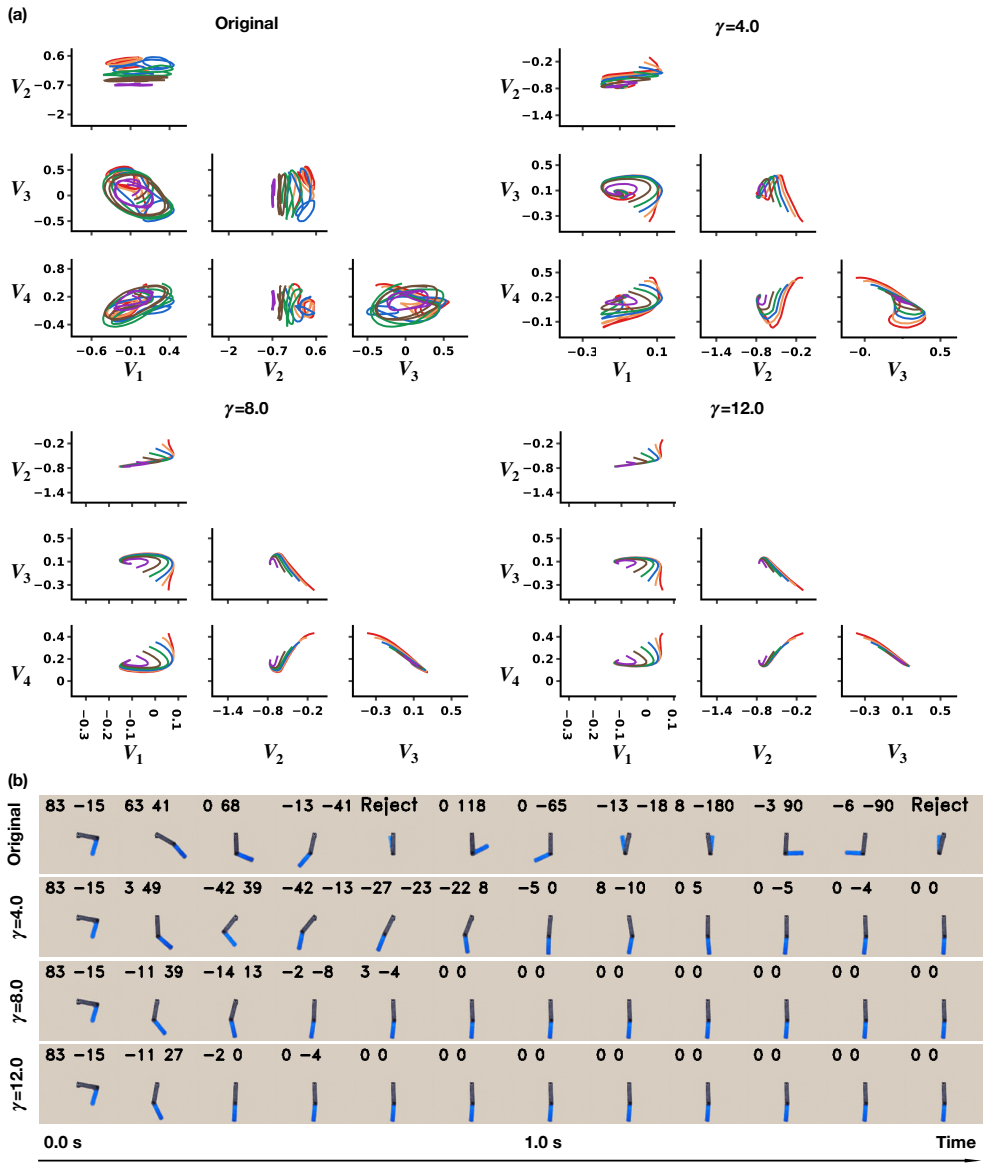
In this section, we further demonstrate our framework's ability to generate videos that show novel dynamics. Particularly, the novel behaviors are fully controllable through the damping factor γ which parameterize the strength of the perturbation in Equation (4).



Supplementary Figure 7 Synthesizing new data sequences with parameterized novel dynamics. (a) Plots depict trajectories in the smooth neural state variable space for the spring mass and single pendulum systems with varying damping factors. (b) Video frames demonstrate the effects corresponding to increasing damping factors.

Supplementary Figure 7(a) shows trajectories of the new dynamics, as described by Equation (4), in the smooth neural state variable space for the spring mass and single pendulum systems. These trajectories were generated by randomly sampling initial states and integrating the damped system with the damping factor γ ranging from zero (original dynamics) to 4.0. The plots reveal that, as the damping factor γ increases, the trajectories converge to the equilibrium state more rapidly. The same behavior is exemplified for the double pendulum system in Supplementary Figure 8(a).

In Supplementary Figure 7(b), we present video frames generated from a trajectory of the spring mass and single pendulum systems shown in Supplementary Figure 7(a), sampled at 6 frames per second (fps). The video frames show that while the original dynamics are approximately periodic, the damped dynamics eventually become stationary. Moreover, as the damping factor γ is increased, the damping effect becomes more pronounced, causing the system to converge to the stationary state more rapidly. This behavior is similarly observed in the video frames generated from a trajectory of the double pendulum system in Supplementary Figure 8(b).

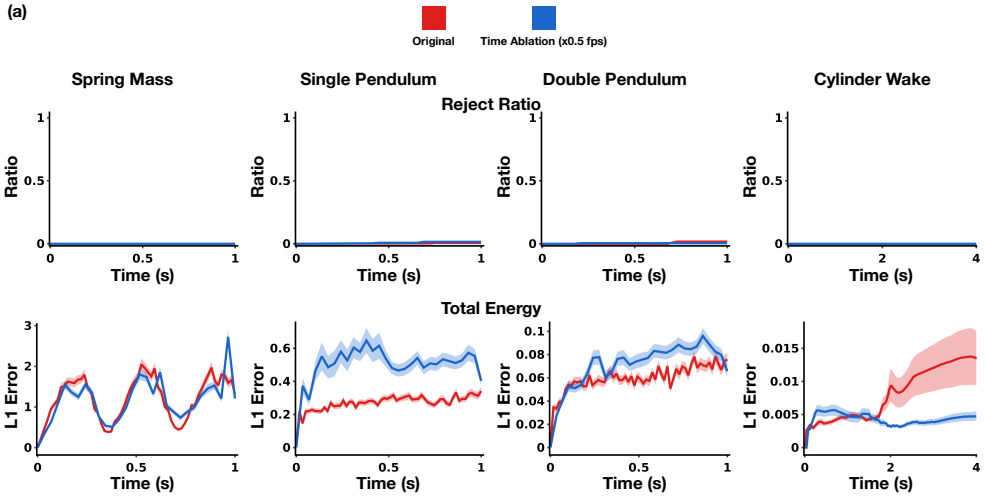


Supplementary Figure 8 Synthesizing new data sequences with parameterized novel dynamics. (a) Plots depict trajectories in the smooth neural state variable space for the double pendulum system with varying damping factors. (b) Video frames demonstrate the effects corresponding to increasing damping factors.

B.6 Effects of temporal resolution

To further analyze the effects of temporal resolution of the provided dataset on model performance, we have run ablation experiments testing our framework on data at a lower temporal resolution (i. e., $\Delta t \rightarrow 2\Delta t$, $\times 0.5$ fps). As shown in Supplementary Figure 9, our framework achieves comparable predictive accuracy even when data are at a lower temporal resolution, despite velocity information being harder to learn.

(a)



(b)

Neural State Vector Field Integration Single Step Pixel Reconstruction Error

System	Original	Time Ablation (x0.5 fps)
Spring Mass	3.086e-04 (±4.629e-05)	1.194e-03 (±2.061e-04)
Single Pendulum	3.300e-04 (±4.594e-05)	1.869e-03 (±4.979e-04)
Double Pendulum	8.086e-04 (±3.890e-05)	2.272e-03 (±3.645e-04)
Cylinder Wake	1.373e-04 (±7.388e-05)	3.738e-04 (±1.305e-04)

Supplementary Figure 9 Our framework is applied to datasets at a lower time resolution ($\Delta t \rightarrow 2\Delta t$, $\times 0.5$ fps), resulting in comparable accuracies despite the lower temporal resolution. (a) Long-term prediction accuracy through neural state vector field integration; (b) Pixel reconstruction error of a single-step neural state vector field integration.

C Implementation details

C.1 Auto-encoders for extracting neural state variables

In this section, we provide details about our compression models. We used the same architectures for the auto-encoders $g = (g_E, g_D)$ and $h = (h_E, h_D)$ as implemented in [6].

Auto-encoder g

The auto-encoder $g = (g_E, g_D)$ is a neural network with specific parameters listed in Supplementary Table 4 (with the latent dimension $N = 64$) and Supplementary Table 5 (with the latent dimension $N = 8192$). All convolutional or transposed

Layer	Kernel Size	#Filters	Stride	Padding	Activation
Conv1	4×4	32	2	1	ReLU
Conv2	4×4	32	2	1	ReLU
Conv3	4×4	64	2	1	ReLU
Conv4	4×4	64	2	1	ReLU
Conv5	4×4	64	2	1	ReLU
Conv6	4×4	64	2	1	ReLU
Conv7	4×4	64	2	1	ReLU
Conv8	3×4	64	(1,2)	1	ReLU
Deconv8	3×4	64	(1,2)	1	ReLU
Deconv7	4×4	64	2	1	ReLU
Deconv6	4×4	64	2	1	ReLU
Deconv5	4×4	64	2	1	ReLU
Deconv4	4×4	64	2	1	ReLU
Deconv3	4×4	32	2	1	ReLU
Deconv2	4×4	16	2	1	ReLU
Deconv1	4×4	3	2	1	ReLU

Supplementary Table 4 The architecture of the auto-encoder g with the latent dimension $N = 64$

convolutional layers are accompanied with a batch normalization layer and a specified activation function. For the encoder network, after each "Conv" layer as shown in Supplementary Table 4 and Supplementary Table 5, we attached another convolutional layer with the same number of filters as the current convolutional layer but with 3×3 kernel and 1 as stride. For the decoder network, along with each "Deconv" layer as shown in Supplementary Table 4 and Supplementary Table 5 except for the last one, the input is also passed through a transposed convolutional layer with the kernel size 4×4, 2 as stride, and a Sigmoid activation function. The output of this branch is then concatenated with each "Deconv" layer along the feature dimension as the input of the next "Deconv" layer.

All models g in our experiments were trained over 1000 epochs, with an initial learning rate of 0.001. The learning rate was scaled by $\gamma = 0.5$ at the start of epochs 20, 50, 100, and 300. The model from the epoch with the lowest reconstruction error on the validation set was used for subsequent experiments.

Layer	Kernel Size	#Filters	Stride	Padding	Activation
Conv1	4×4	32	2	1	ReLU
Conv2	4×4	32	2	1	ReLU
Conv3	4×4	64	2	1	ReLU
Conv4	4×4	128	2	1	ReLU
Conv5	3×4	128	(1,2)	1	ReLU
Deconv5	3×4	64	(1,2)	1	ReLU
Deconv4	4×4	64	2	1	ReLU
Deconv3	4×4	32	2	1	ReLU
Deconv2	4×4	16	2	1	ReLU
Deconv1	4×4	3	2	1	ReLU

Supplementary Table 5 The architecture of the auto-encoder g with the latent dimension $N = 8192$

Auto-encoder h

The auto-encoder $h = (h_E, h_D)$ is also a neural network, with specific parameters listed in Supplementary Table 6. Each layer is a linear layer accompanied with a sine activation function, and d refers to the system's intrinsic dimension. The intermediate features whose dimension is d are identified as our smooth neural state variables. We used the latent vectors produced from the auto-encoder g as outlined in Supplementary Table 4 with $N = 64$ as the input to h.

Encoder Layer	#Units	Activation	Decoder Layer	#Units	Activation
Layer1	128	Sine	Layer5	32	Sine
Layer2	64	Sine	Layer6	64	Sine
Layer3	32	Sine	Layer7	128	Sine
Layer4	d	Sine	Layer8	64	None

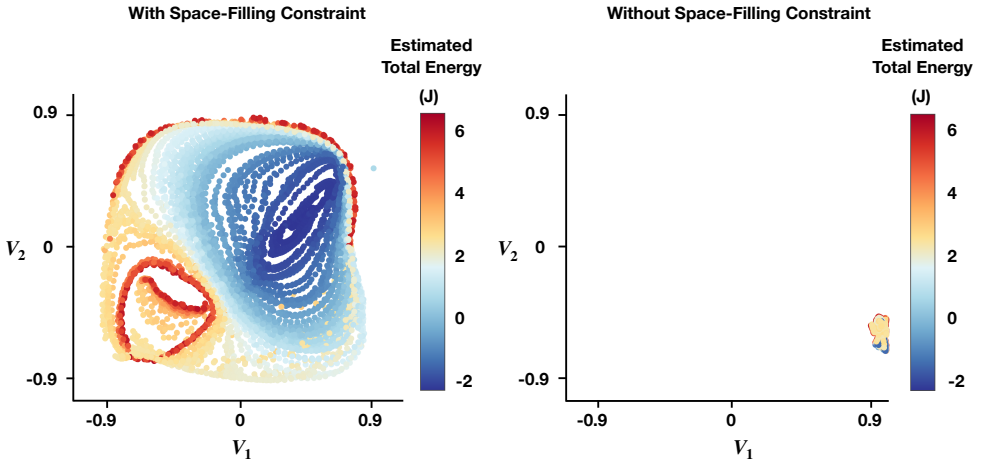
Supplementary Table 6 The architecture of the auto-encoder h

All models h in our experiments were trained over 1000 epochs, with an initial learning rate of 0.0005. The learning rate was scaled by $\gamma = 0.5$ at the start of epochs 15, 30, 100, 300, and 500. The model from the epoch with the lowest validation loss was used for subsequent experiments, where the validation loss was calculated using Equation (5) on the validation set with a fixed $\beta = 1$.

C.2 Smoothness regularization

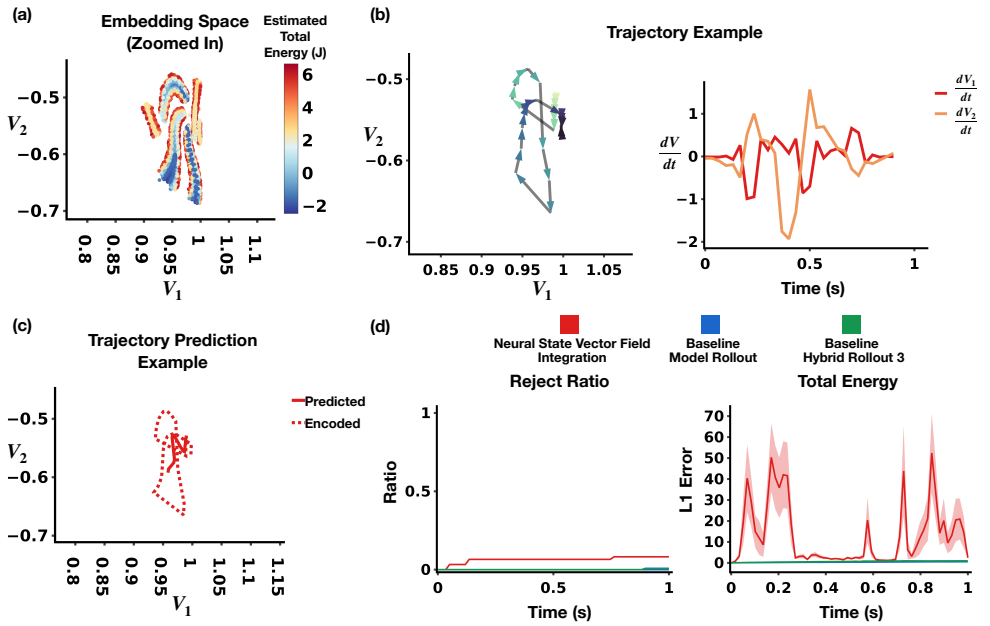
In Section 3.2, we introduced Equation (6), designed to enforce smoothness on the encoded state space by penalizing the neighboring distance between consecutive states.

However, without further regularization, the smoothness loss may dominate the the optimization procedure, leading to undesirable effects on the resulting state space. The necessity of the space-filling loss is demonstrated in Supplementary Figure 10, where we show two plots to visualize the distribution of data points over the smooth neural state variable space. The left plot was produced from the auto-encoder h trained with the space-filling loss, while the right one was produced by the auto-encoder h trained without it. When the space-filling loss is included in the loss function, the data points are well spread out, whereas they collapse into a very small region in the smooth neural state variable space if only the smoothness constraint is enforced.



Supplementary Figure 10 The test data encoded to smooth neural state variables for the single pendulum system, colored by the total energy calculated from computer vision estimated position and velocity from the respective images. When the space-filling loss is not included in the loss function, the embedding space collapses to a small region in the domain, as seen on the right.

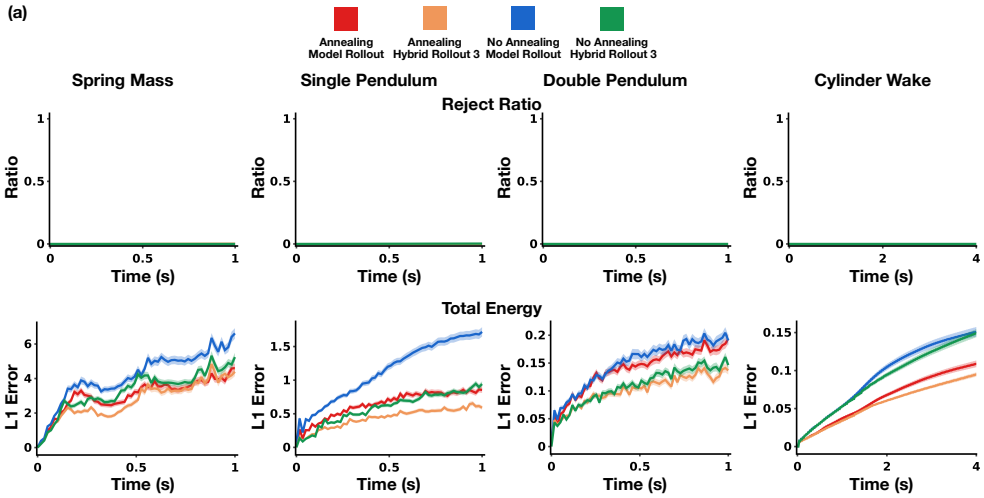
The collapsed state space cannot be utilized to extract accurate dynamics, even when normalization is applied to scale the state space, as demonstrated in Supplementary Figure 11. A NeuralODE was trained on the normalized state variables, scaled by the standard deviation and biased by the mean of the training data. These results demonstrate that both the smoothness loss and space-filling loss are necessary for extracting smooth neural state variables with continuous trajectories that enable learning continuous dynamics via a NeuralODE.



Supplementary Figure 11 Collapsed Space Normalization Results (a) A zoomed in visualization of the collapsed state space. (b) A sample trajectory that is plotted in the zoomed in state space and its finite-difference approximated first order derivatives plotted against time. (c) The predicted trajectory plotted against the example trajectory. (d) Long term prediction error of the trained NeuralODE is worse than even the model rollout prediction scheme from a baseline neural state variable autoencoder.

To reduce the effects of additional constraints on the reconstruction accuracy, we introduced an annealing method during training. We cyclically increased and decreased the weights of the smoothness and space-filling constraints by altering β of Equation (5). We utilized an annealing schedule of 5 cycles with each cycle spanning 200 epochs, where β in Equation (5) was linearly increased from 0 to 1 over the first 100 epochs and kept at 1 for the next 100 epochs. Supplementary Figure 12 demonstrates the improved accuracy when annealing is applied during training. Supplementary Figure 12(a) shows the long-term stability and accuracy for the smooth neural state variable rollout predictions, where we compared the computer vision based energy estimation values for the decoded images against those estimated from the ground truth images. Supplementary Figure 12(b) shows the auto-encoder pixel reconstruction mean square error of the smooth neural state variables with their respective standard error bounds. Smooth neural state variables trained with annealing show generally better reconstruction accuracy.

(a)



(b)

Neural State Variable Auto-encoder Pixel Reconstruction Error

Training Method	Annealing	No Annealing
Spring Mass	1.850e-04 (±3.298e-05)	6.548e-04 (±8.258e-05)
Single Pendulum	2.874e-04 (±2.898e-05)	5.338e-04 (±5.104e-05)
Double Pendulum	6.316e-04 (±4.931e-06)	7.360e-04 (±3.449e-05)
Cylinder Wake	7.884e-06 (±9.618e-07)	1.176e-05 (±2.143e-06)

Supplementary Figure 12 Accuracy Comparison between smooth models trained with and without annealing: (a) Long Term Prediction Stability and Accuracy for Smooth Neural State Variables trained with and without annealing. (b) Auto-encoder Pixel reconstruction error mean and standard error bounds for smooth neural state variable models trained with and without annealing.

The effects of the smoothness and space-filling constraints are balanced by tuning the weights of the sub-loss terms w_{smooth}, w_{space}, while keeping L_0 fixed to a reasonably small value. In our experiments, we used a fixed value of $L_0 = 0.05$ and tuned the loss weights by optimizing both the reconstruction accuracy and the smoothness metric as defined in Supplementary Equation (28). The random seeds and corresponding loss weights utilized for each system are shared in Supplementary Table 7.

	Spring Mass			Single Pendulum		
Random Seed	1	3	4	1	2	3
w_{smooth}	64	64	64	64	128	64
w_{space}	16	8	16	32	128	64
	Double Pendulum			**Cylinder Wake**		
Random Seed	1	2	3	1	2	3
w_{smooth}	16	16	16	1024	1024	1024
w_{space}	4	8	8	0.5	0.5	0.5

Supplementary Table 7 Loss weights used in our experiments. All values were tuned to optimize the smoothness metric as introduced in Supplementary Equation (28) along with the reconstruction accuracy.

C.3 Implementation of neural state vector fields

The neural state vector field $\hat{F}$ is implemented as a multi-layer perceptron (MLP), with specific parameters listed in Supplementary Table 8 (with 6 layers) and Supplementary Table 9 (with 8 layers), where the number of layers were chosen based on the intrinsic dimension d of the system (e.g. 6 layers when $d \leq 2$ and 8 layers when $d > 2$). For both MLPs, each layer is accompanied with the ReLU activation function, except for the last layer.

Layer	#Units	Activation
Layer1	32	ReLU
Layer2	64	ReLU
Layer3	128	ReLU
Layer4	64	ReLU
Layer5	32	ReLU
Layer6	d	None

Supplementary Table 8 The architecture of the neural state vector field $\hat{F}$

All models $\hat{F}$ in our experiments were trained over 1000 epochs, with an initial learning rate of 0.0003. The learning rate was scaled by $\gamma = 0.5$ at the start of epochs 15, 30, 100, 300, and 500. The model from the epoch with the lowest validation loss was used for subsequent experiments, where the validation loss was calculated using Equation (7) on the validation set with a fixed $\rho = 0.5$.

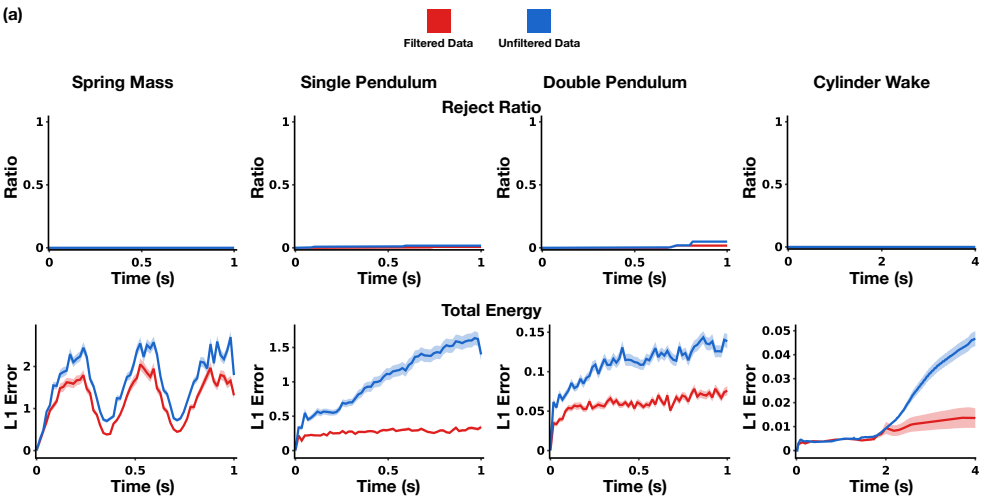
To improve the accuracy of the trained neural state vector field, we applied filtering to our neural state variable trajectories before training. We removed data sequences that contain trajectories where the distance between any consecutive state is greater

Layer	#Units	Activation
Layer1	32	ReLU
Layer2	64	ReLU
Layer3	128	ReLU
Layer4	256	ReLU
Layer5	128	ReLU
Layer6	64	ReLU
Layer7	32	ReLU
Layer8	d	None

Supplementary Table 9 The architecture of the neural state vector field $\hat{F}$

than the 99 -percentile distribution within the respective dataset subset (e.g. training, validation or test). Supplementary Figure 13 demonstrates the improved accuracy when filtering is applied before training the neural state vector field. Supplementary Figure 13(a) shows the long-term prediction stability and accuracy for the neural state vector field integrated neural state variable trajectories, where we compared the computer vision based energy estimation values for the decoded images against the energy estimated from the ground truth images. Supplementary Figure 13(b) shows the single step pixel reconstruction mean square error of the neural state vector field predictions with their respective standard error bounds. Using filtered trajectories shows consistent improvement in both cases. Supplementary Figure 13(c) shows the remaining number of trajectories after filtering for the training, validation and test sets, averaged over three random seeds along with their respective standard error bounds.

(a)



(b)

Neural State Vector Field Integration Single Step Pixel Reconstruction Error

Dataset	Filtered Data	Unfiltered Data
Spring Mass	3.086e-04 (±4.629e-05)	3.366e-04 (±2.146e-05)
Single Pendulum	3.300e-04 (±4.594e-05)	9.471e-04 (±1.334e-04)
Double Pendulum	8.086e-04 (±3.890e-05)	1.193e-03 (±8.073e-05)
Cylinder Wake	1.373e-04 (±7.388e-05)	1.374e-04 (±6.135e-05)

(c)

Number of Remaining Trajectories

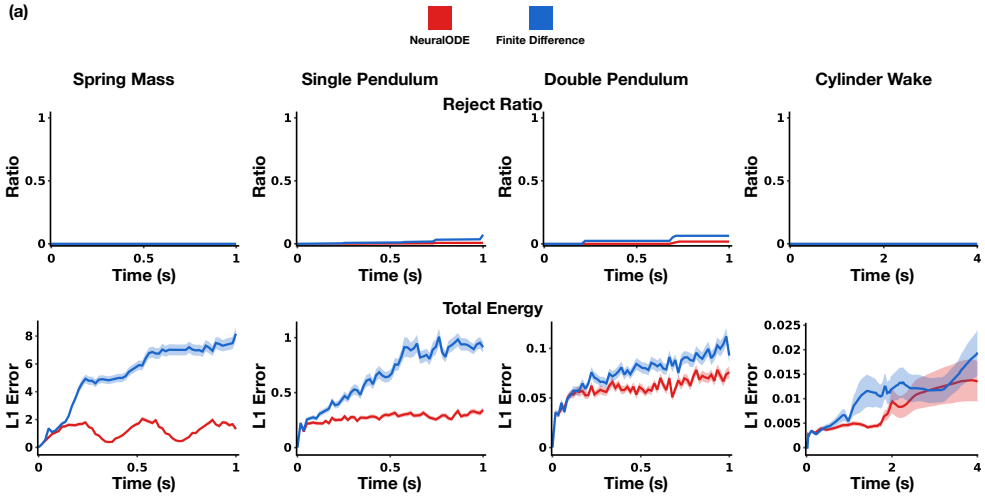
Dataset	Train	Validation	Test
Spring Mass	610.33 (±27.93) / 960	78.00 (±4.03) / 120	77.00 (±3.86) / 120
Single Pendulum	754.00 (±14.24) / 960	93.00 (±3.30) / 120	93.00 (±3.30) / 120
Double Pendulum	428.33 (±16.98) / 879	55.33 (±3.41) / 110	59.00 (±1.70) / 110
Cylinder Wake	171.33 (±25.26) / 960	23.33 (±2.68) / 120	21.33 (±2.60) / 120

Supplementary Figure 13 Accuracy Comparison between neural state vector fields trained with and without filtering: (a) Long Term Prediction Stability and Accuracy for Neural State Vector Fields trained with and without filtering. (b) Neural State Vector Field integration single step pixel reconstruction error mean and standard error bounds when trained and tested on filtered and unfiltered data. (c) Average number of trajectories remaining after filtering, reported with standard error bounds.

Furthermore, we implemented the training procedure such that the neural state vector field is trained as a NeuralODE [7, 8], allowing us to optimize for the weighted reconstruction accuracy over longer time horizons, rather than explicitly fitting the outputs of the neural state vector field to reproduce the finite difference approximated rate of change for a single step. The relative weights of latter states in the trajectories were annealed over 4 cycles of 250 epochs each, where ρ in Equation (7) was linearly increased from 0.1 to 0.9 over the first 125 epochs and kept at 0.9 over the remaining 125 epochs. Supplementary Figure 14 demonstrates the improved accuracy when utilizing the NeuralODE paradigm for training. Supplementary Figure 14(a) shows the long-term prediction stability and accuracy for the neural state vector field

integrated neural state variable trajectories, where we compare the computer vision based energy estimation values for the decoded images against the energy estimated from the ground truth images. Supplementary Figure 14(b) shows the single step pixel reconstruction mean square error of the neural state vector field predictions with their respective standard error bounds. Using the NeuralODE training method shows consistent improvement in long term prediction accuracy.

(a)



(b)

Neural State Vector Field Integration Single Step Pixel Reconstruction Error

Training Method	NeuralODE	Finite Difference
Spring Mass	3.086e-04 (±4.629e-05)	5.312e-04 (±9.364e-05)
Single Pendulum	3.300e-04 (±4.594e-05)	3.528e-04 (±3.707e-05)
Double Pendulum	8.086e-04 (±3.890e-05)	7.556e-04 (±3.926e-05)
Cylinder Wake	1.373e-04 (±7.388e-05)	1.315e-04 (±7.796e-05)

Supplementary Figure 14 Accuracy Comparison between neural state vector fields trained as a NeuralODE and neural state vector fields trained to reconstruct the finite difference between consecutive states: (a) Long Term Prediction Stability and Accuracy for Neural State Vector Fields trained as neuralODE and Neural State Vector Fields trained to reconstruct the approximated rate of change using finite difference. (b) Neural State Vector Field integration single step pixel reconstruction error mean and standard error bounds when trained as a NeuralODE compared to when trained to reconstruct the finite difference between consecutive states.

References

[1] Dušek, J., Le Gal, P., Fraunié, P.: A numerical and theoretical study of the first hopf bifurcation in a cylinder wake. Journal of Fluid Mechanics **264**, 59–80 (1994)

[2] Barkley, D.: Linear analysis of the cylinder wake mean flow. Europhysics Letters **75**(5), 750 (2006)

[3] Noack, B.R., Afanasiev, K., Morzyński, M., Tadmor, G., Thiele, F.: A hierarchy of low-dimensional models for the transient and post-transient cylinder wake. Journal of Fluid Mechanics **497**, 335–363 (2003)

[4] Taira, K., Colonius, T.: The immersed boundary method: a projection approach. Journal of Computational Physics **225**(2), 2118–2137 (2007)

[5] Colonius, T., Taira, K.: A fast immersed boundary method using a nullspace approach and multi-domain far-field boundary conditions. Computer Methods in Applied Mechanics and Engineering **197**(25-28), 2131–2146 (2008)

[6] Chen, B., Huang, K., Raghupathi, S., Chandratreya, I., Du, Q., Lipson, H.: Automated discovery of fundamental variables hidden in experimental data. Nature Computational Science **2**(7), 433–442 (2022) https://doi.org/10.1038/s43588-022-00281-6 . Accessed 2024-03-27

[7] Chen, R.T.Q., Rubanova, Y., Bettencourt, J., Duvenaud, D.: Neural Ordinary Differential Equations (2019). https://arxiv.org/abs/1806.07366

[8] Poli, M., Massaroli, S., Yamashita, A., Asama, H., Park, J., Ermon, S.: Torchdyn: Implicit models and neural numerical methods in pytorch