

Don't Waste Data: Transfer Learning to Leverage All Data for Machine-Learnt Climate Model Emulation

Raghul Parthipan^{1,2} & Damon J. Wischik¹

NeurIPS 2022 Workshop: Tackling Climate Change with Machine Learning

1.



2.



Everyone wants to model how the atmosphere evolves.

Goal. Speed up these models using a low-cost ML emulator.

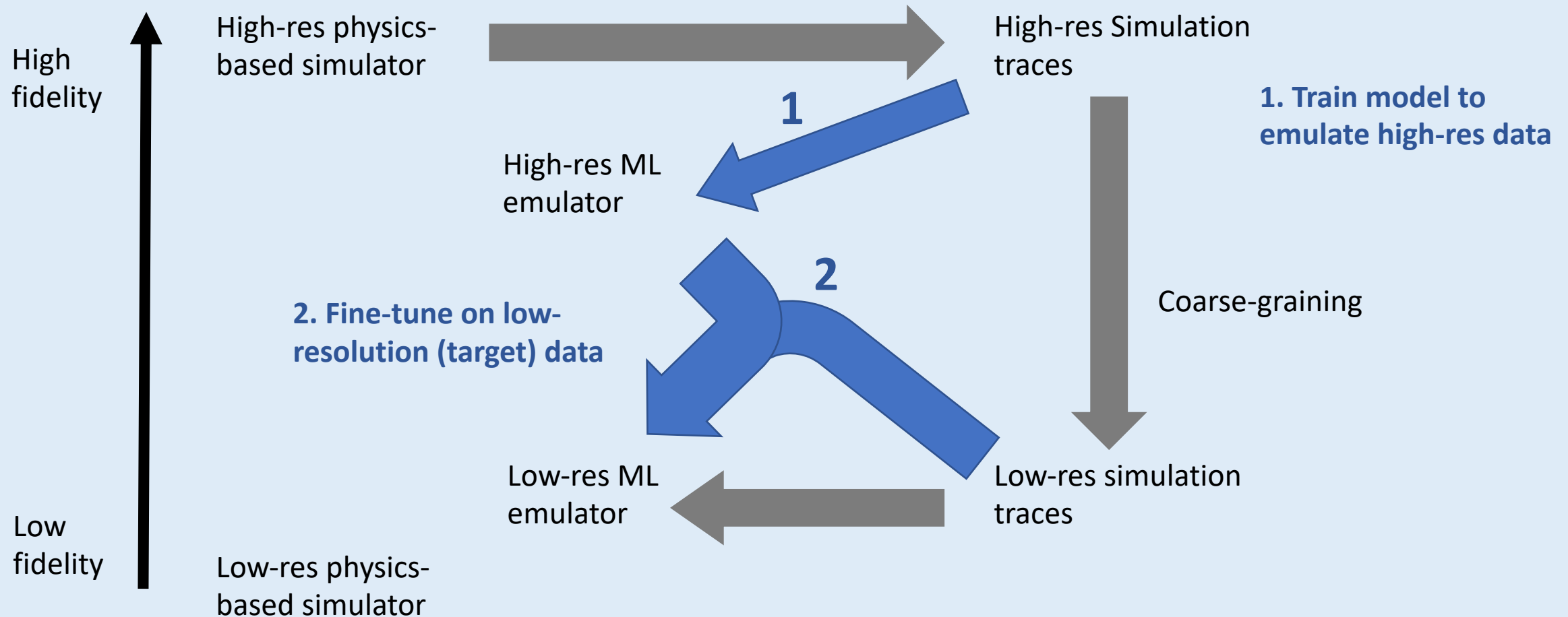


What happens next ...

- day?
- week?
- month?
- season?
- year?
- decade?
- century?

What would happen in different scenarios?

Our approach is a **two-step process** which uses all data.



We **evaluate** using three chaotic dynamical systems.

They've been used extensively in emulation studies.

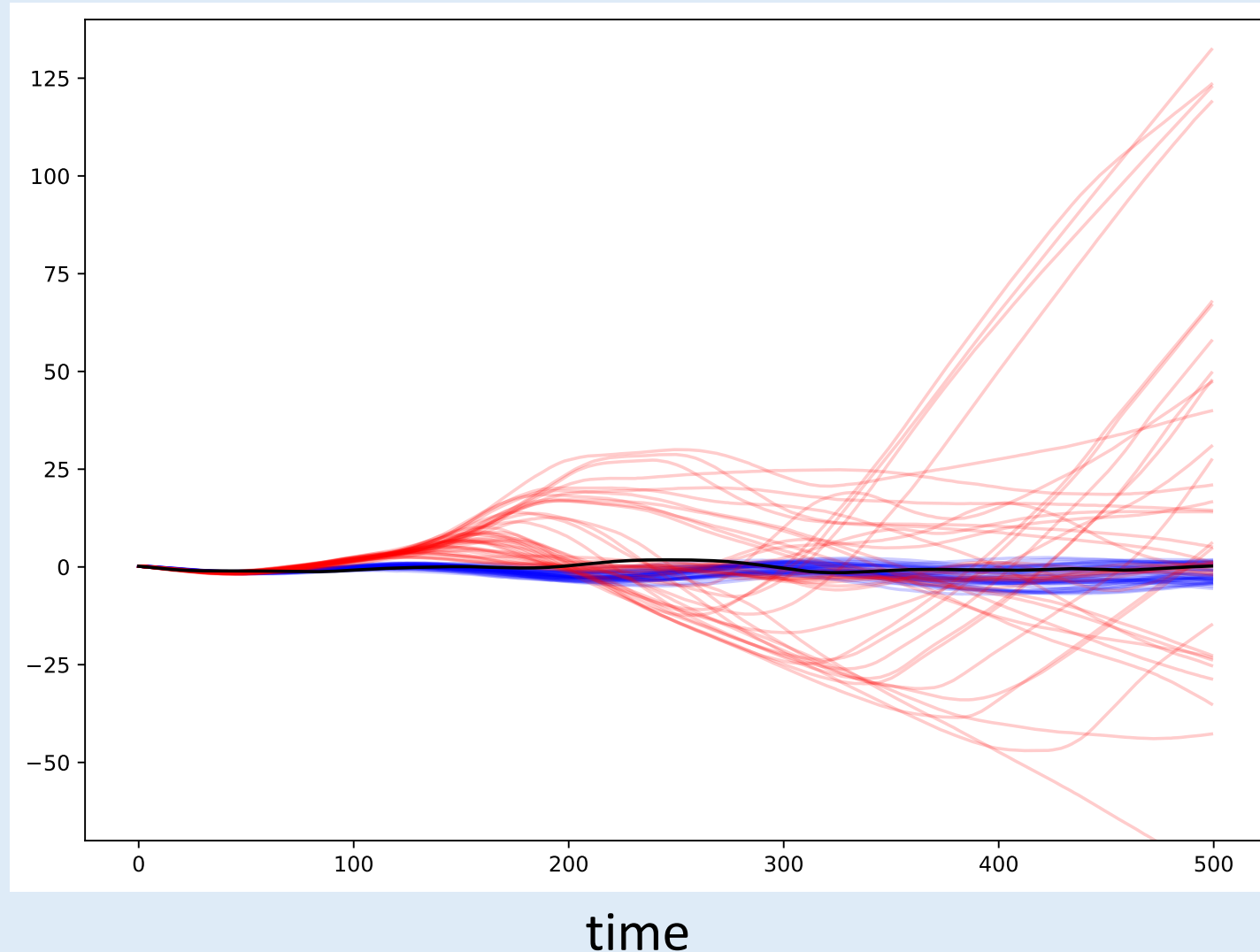
(1) Kuramoto-Sivashinsky (KS)

(2) Brusselator

(3) Lorenz 96 (L96)

Results: better forecasting skill.

One of the 8
dimensions
of low-
resolution
L96



**Coarse-grain-only
ML model**
(40 runs, same init
conditions)

Truth

**Our full-data TL
model**
(40 runs, same init
conditions)

Note: it's a
probabilistic model

TL makes better predictions.

- Robustly across multiple systems

System	Our full-data TL model	Coarse-grain-only ML model
Kuramoto-Sivashinsky	91.73	37.74
Brusselator	384.64	-555.40
Lorenz 96	11.51	4.37

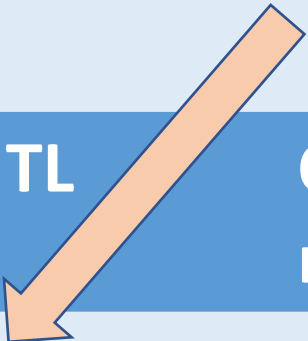
We evaluate using hold-out log-likelihood, a standard probabilistic procedure from ML.

15 randomly initialized models were trained and the average hold-out log-likelihood is shown.

TL makes better predictions.

- Robustly across multiple systems
- Gives more reliable training

Standard Errors in mean are far smaller for TL model



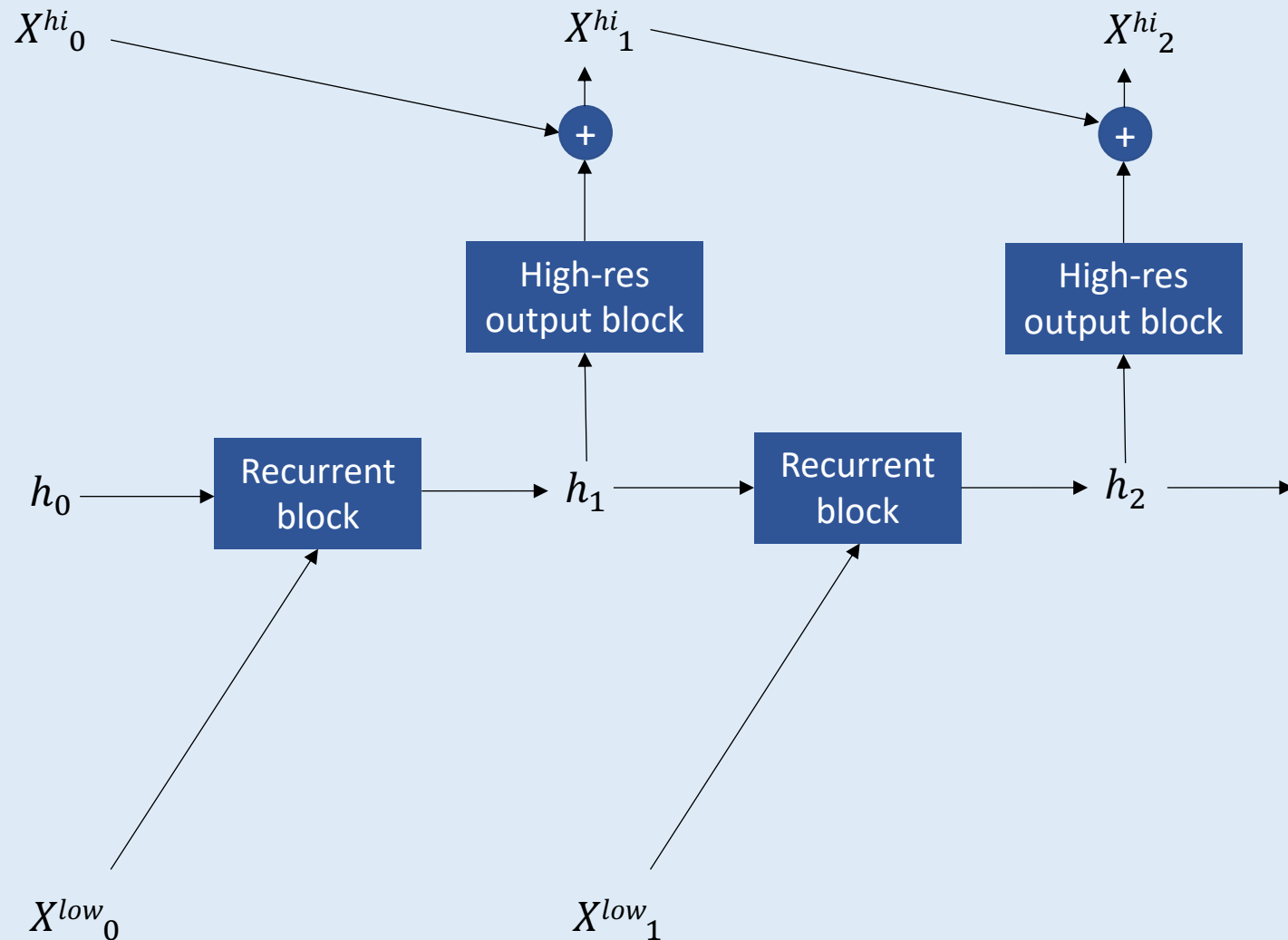
System	Our full-data TL model	Coarse-grain-only ML model
Kuramoto-Sivashinsky	91.73 ± 0.62	37.74 ± 16.50
Brusselator	384.64 ± 27.41	-555.40 ± 486.39
Lorenz 96	11.51 ± 0.64	4.37 ± 1.07

We evaluate using hold-out log-likelihood, a standard probabilistic procedure from ML.

15 randomly initialized models were trained and the average hold-out log-likelihood is shown.

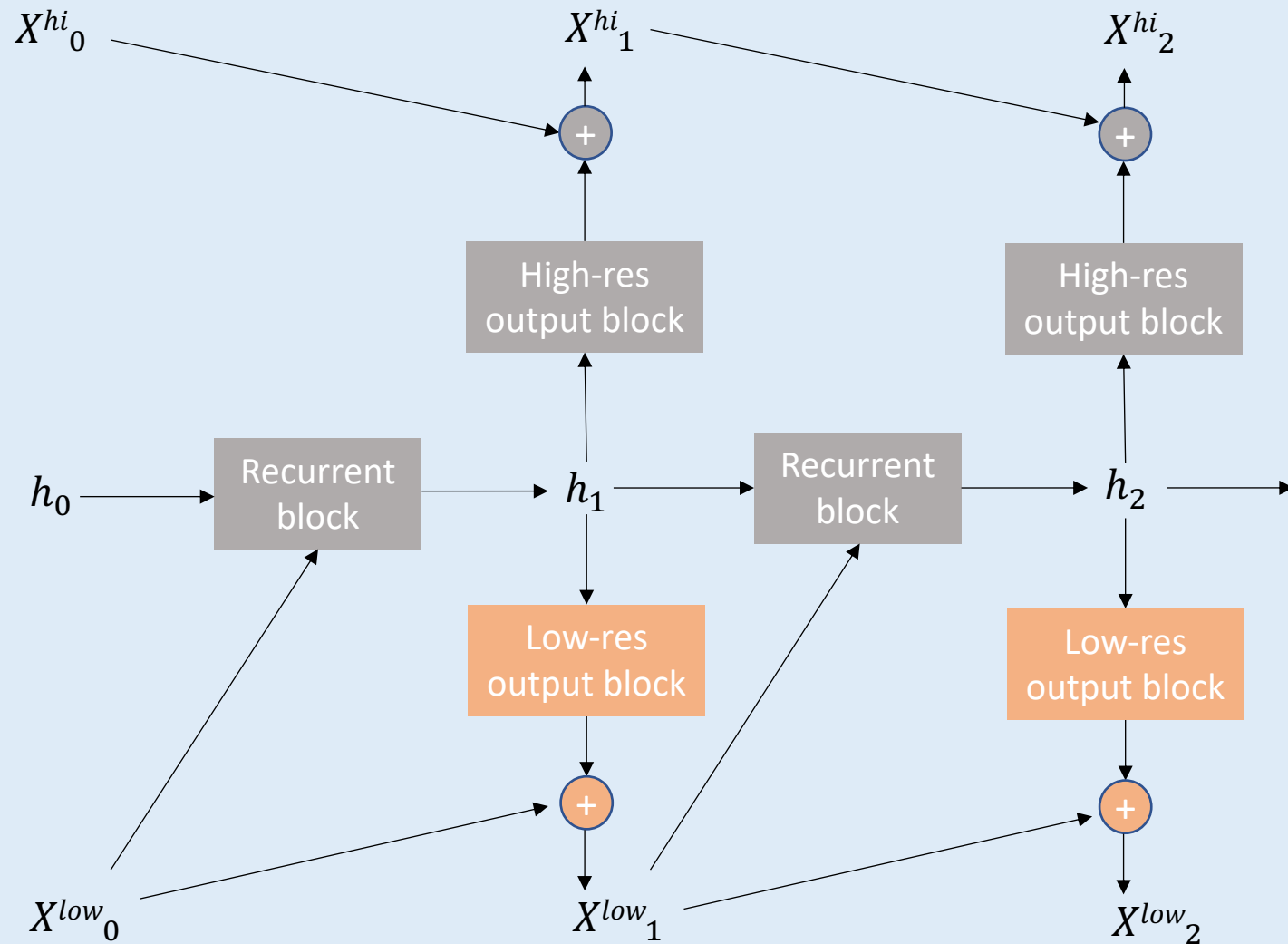
Details of approach: example with Recurrent Neural Network

Details of approach: example with Recurrent Neural Network



1. Train model to reproduce high-resolution evolution $X^{hi}_0, X^{hi}_1, X^{hi}_2 \dots$

Details of approach: example with Recurrent Neural Network



1. Train model to reproduce high-resolution evolution $X^{hi}_0, X^{hi}_1, X^{hi}_2 \dots$

2. New task-specific layers are trained to reproduce low-resolution evolution $X^{low}_0, X^{low}_1, X^{low}_2 \dots$

Takeaways

- By framing the problem as a transfer-learning task, we've shown how to learn from all high-resolution data to create low-resolution emulators.
- The approach performs particularly well in data-scarce scenarios, acting as a regularizer.
- Our simple two-step approach can be easily tested on various problems. And now must be tested on operational models. Happy to help anyone interested in applying this to their problems.
- There is still a gap between the spread and error in our forecasts, and it is unclear if that is closable by this approach.

Supporting slides

RNN model equations

For our RNN, the hidden state is shared and its evolution is described by $\mathbf{h}_{t+1} = f_{\theta}(\mathbf{h}_t, \mathbf{X}_t)$ where $\mathbf{h}_t \in \mathbb{R}^H$ and f_{θ} is a GRU cell [36]. We model the low-resolution data as

$$\mathbf{X}_{t+1} = \mathbf{X}_t + g_{\theta}(\mathbf{h}_{t+1}) + \sigma \mathbf{z}_t \quad (1)$$

and the high-resolution as

$$\mathbf{Y}_{t+1} = \mathbf{Y}_t + j_{\theta}(\mathbf{h}_{t+1}) + \rho \mathbf{w}_t \quad (2)$$

where the functions g_{θ} and j_{θ} are represented by task-specific dense layers, $\mathbf{z}_t \sim \mathcal{N}(0, I)$ and $\mathbf{w}_t \sim \mathcal{N}(0, I)$. The learnable parameters are the neural network weights θ and the noise terms $\sigma \in \mathbb{R}^1$ and $\rho \in \mathbb{R}^1$. Further details are in Appendix A.

Details on dynamical systems

For the KS and Brusselator, our low-resolution data, X_t , is a spatio-temporal averaging of the high-resolution, Y_t , as per standard coarse-graining approaches.

For the L96, the system itself makes a separation between low- and high-resolution variables.

B.1 Kuramoto-Sivashinsky

The Kuramoto-Sivashinsky equation [45, 46] models diffusive-thermal instabilities in a laminar flame front. The one-dimensional KS equation is given by the PDE

$$\frac{\partial u}{\partial t} = -\nu \frac{\partial^4 u}{\partial x^4} - \frac{\partial^2 u}{\partial x^2} - u \frac{\partial u}{\partial x} \quad (3)$$

on the domain $\Omega = [0, L]$ with periodic boundary conditions $u(0, t) = u(L, t)$ and $\nu = 1$. The case $L = 22$ is used here as it results in a chaotic attractor. Equation [3] is discretized with a grid of 100 points and solved with a time-step of $\delta t = 0.00005$ using the *py-pde* package [47]. The data are subsampled to $\Delta t = 0.002$ to create Y_t . The first 10,000 steps are discarded.

B.2 Brusselator

The Brusselator [48] is a model for an autocatalytic chemical reaction. The two-dimensional Brusselator equation is given by the PDEs

$$\frac{\partial u}{\partial t} = D_0 \nabla^2 u + a - (1 + b)u + vu^2 \quad (4)$$

$$\frac{\partial v}{\partial t} = D_1 \nabla^2 v + bu - vu^2 \quad (5)$$

on the domain $\Omega = [[0, 64], [0, 64]]$ with $D_0 = 1, D_1 = 0.1, a = 1$ and $b = 3$. The parameters lead to an unstable regime. Here, $D_0 = 1$ and $D_1 = 0.1$ are diffusivities and a and b are related to reaction rates. Equations [4-5] are discretized on a unit grid and solved with a time-step of $\delta t = 0.0002$ using the *py-pde* package [47]. The data are subsampled to $\Delta t = 0.002$ to create Y_t . The first 10,000 steps are discarded.

B.3 Lorenz 96

The Lorenz 96 is a model for atmospheric circulation [49]. We use the two-tier version which consists of two scales of variables: a large, low-frequency variable, X_k , coupled to small, high-frequency variables, $Y_{j,k}$. These evolve as follows

$$\frac{dX_k}{dt} = -X_{k-1}(X_{k-2} - X_{k+1}) - X_k + F - \frac{hc}{b} \sum_{j=J(k-1)+1}^{kJ} Y_j \quad (6)$$

$$\frac{dY_{j,k}}{dt} = -cbY_{j+1,k}(Y_{j+2,k} - Y_{j-1,k}) - cY_{j,k} - \frac{hc}{b} X_k \quad (7)$$

where in our experiments, the number of X_k variables is $K = 8$, and the number of $Y_{j,k}$ variables per X_k is $J = 32$. The value of the constants are set to $h = 1, b = 10$ and $c = 10$. These indicate that the fast variable evolves ten times faster than the slow variable and has one-tenth the amplitude. The forcing term, F , is set to $F = 20$. Equations [6-7] are solved using a fourth-order Runge-Kutta scheme with a time-step of $\delta t = 0.001$. The data are subsampled to $\Delta t = 0.005$ to create X_t and Y_t . The first 10,000 steps are discarded.