HELMHOLTZ AI | ARTIFICIAL INTELLIGENCE COOPERATION UNIT

m-dml.org

Helmholtz-Zentrum hereon

Deep Differentiable Emulators

David Greenberg
Model-driven Machine Learning Group
Helmholtz Centre Hereon

Deep Learning and Optimization
IMT Atlantique
November 22, 2022

Outline

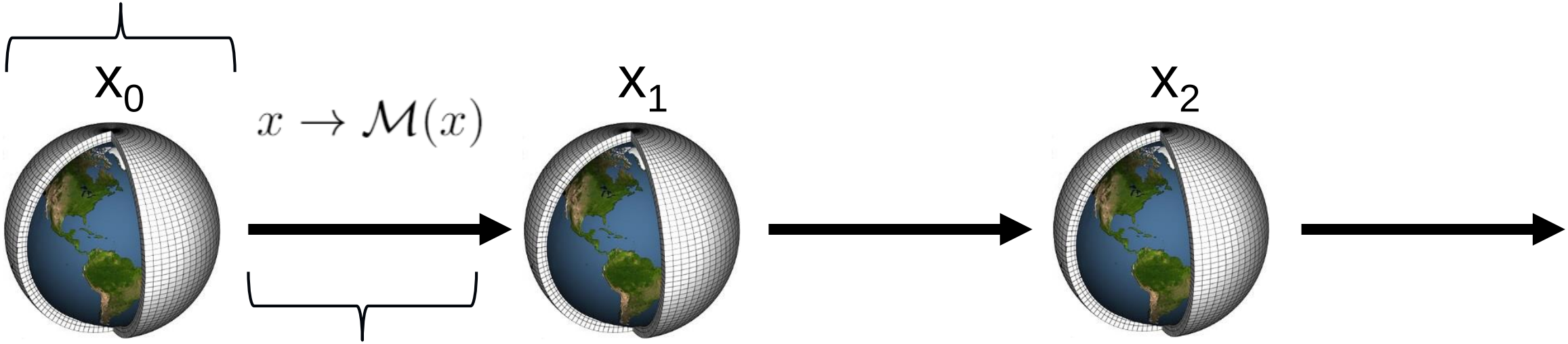
1 Deep Differentiable Emulators for Inverse Problems in the Geosciences

2 Hybrid Emulators

3 ML Models as Earth System Model Components

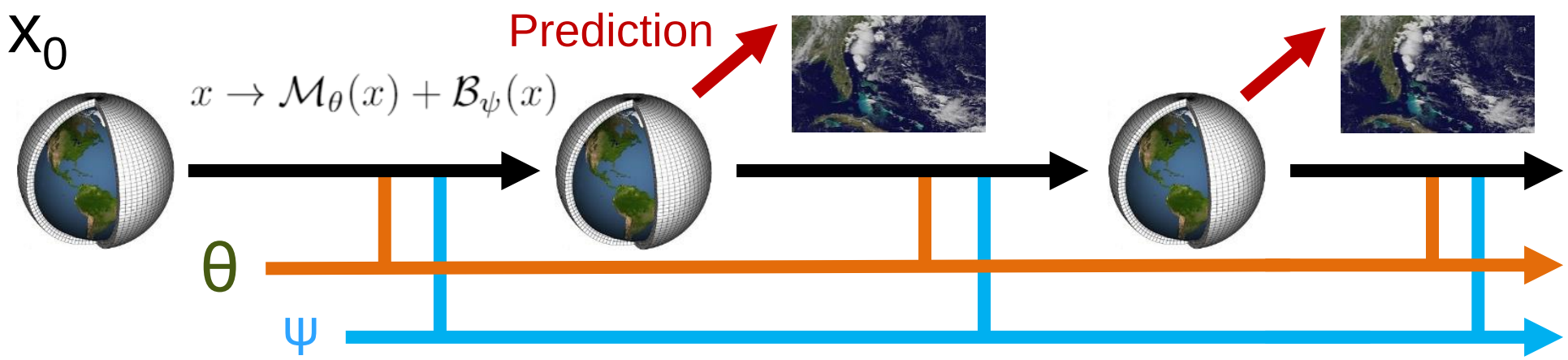
Simulators as State space models

System state
- Physical variables (pressure, temperature, moisture, …)
- Snapshot for one time point



State update function
- Fluid dynamics
- Local physical processes (radiation, turbulence, phase changes, …)
- Advection of tracers (droplets, aerosols, etc.)

3 optimization tasks in the geosciences



1 **Data assimilation**: choose x_0 so predictions match observations

2 **Model tuning**: choose θ so predictions match observations

3 **Learning corrective terms**: choose ψ so predictions match observations

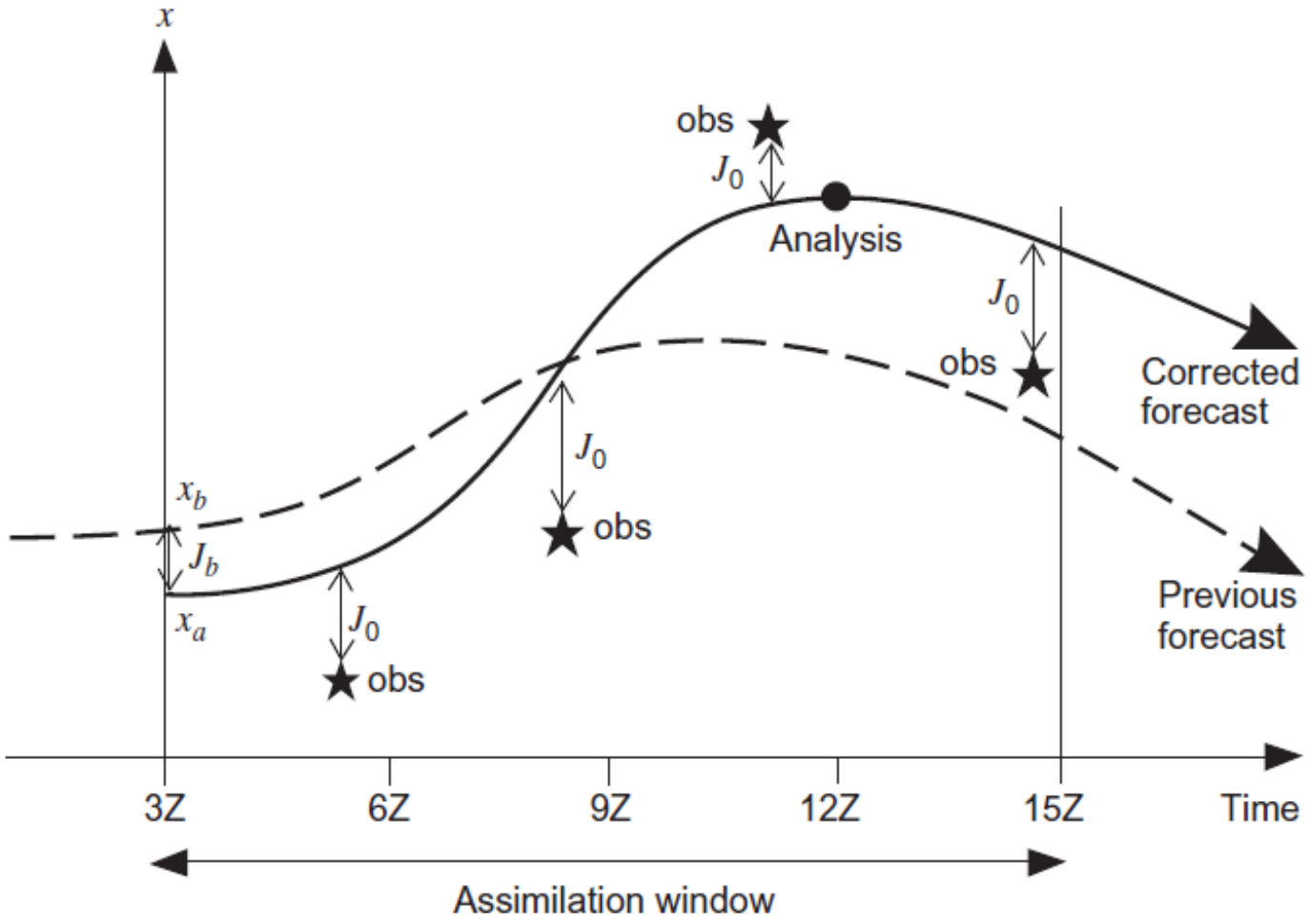
Because: unknown physics, sub-grid-scale processes

For all 3 problems, we need $\frac{d\mathcal{M}}{dx}$ in order to optimize our predictions over x_0, θ or ψ.

But these derivatives are not available for most simulators!

Data Assimilation

Goal: find a sequence of system states that matches both model and data



Holton, 2012

Strong Constraint 4DVar

How can we formalize our desire to match both model and data?

Assumption 1: initial state x_0 has a Gaussian prior with mean $\mathbf{x_f}$ and covariance **B**.

Assumption 2: for state **x**, observations **y** are Gaussian with mean **H(x),** covariance **R**

Now let's carry out **maximum a posteriori estimation likelihood**: we find the initial state x_0 with the highest the posterior probability given observations and prior assumptions. This is equivalent to minimizing an objective function:

$$\mathcal{L}_{DA}(x_0) = -\log p(y_1, \ldots, y_W | x_0) - \log p(x_0 | x_f) = -\sum_{n=1} \log p(y_n | x_0) - \log p(x_0 | x_f)$$

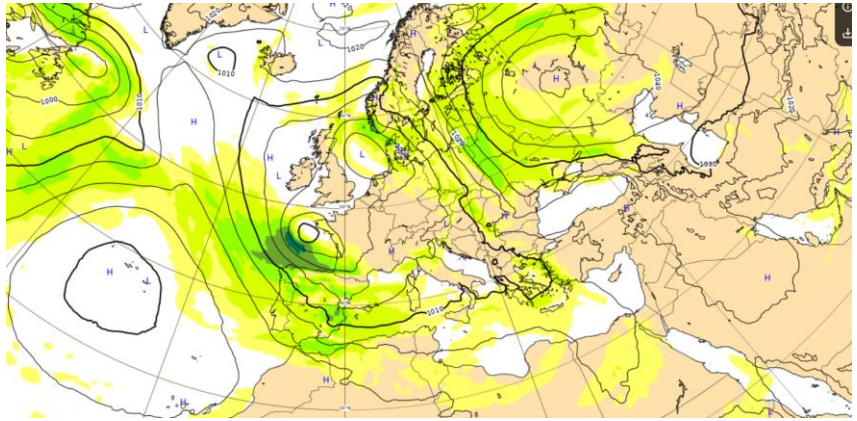
$$\propto \sum_{n=1}^{W} \left(y_n - \mathcal{H}\mathcal{M}^{(n)}(x_0)\right)^\top R^{-1} \left(y_n - \mathcal{H}\mathcal{M}^{(n)}(x_0)\right) + (x_0 - x_f)^\top B^{-1} (x_0 - x_f)$$

Strategy A: hand-written simulator gradient routines

We can minimize $\mathcal{L}_{DA}(\boldsymbol{x_0})$ by optimization if we can calculate its derivatives with respect to $\mathbf{x_0}$.

To do this we require a computer program that calculates $d\mathcal{M}(x)/dx$ for any **x**, where $x_{n+1} = \mathcal{M}(x_n)$.

Once we have written such a derivative-calculating program, we can use it in an **adjoint** scheme to differentiate $\mathcal{L}_{DA}(\boldsymbol{x_0})$ (details in Bouttier & Courtier, 1999).



Example: ECMWF Integrated Forecast System

Graph shows part of **x** corresponding to see mean sea level pressure and wind speed at 850 hPa at 12:00 on 07.12.2020.

y consists of ~25 Million observations in a 12 hour window

Strategy B: finite differences

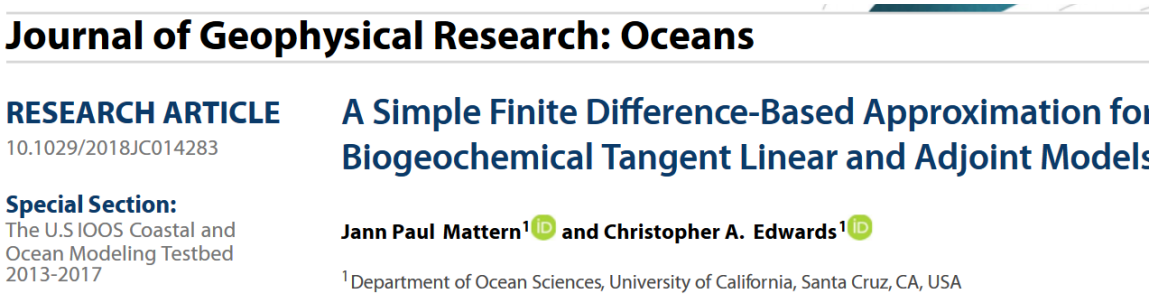
What if we don't have a hand-written gradient routine for $\mathcal{M}$?

We want to calculate the vector of derivatives $\frac{d\mathcal{L}_{DA}(x_0)}{dx_0}$. By definition this is simply

$$\left[\frac{d\mathcal{L}_{\mathrm{DA}}(x_0)}{dx_0}\right]_k = \lim_{h\to 0} \frac{\mathcal{L}_{\mathrm{DA}}(x_0 + he_k) - \mathcal{L}_{\mathrm{DA}}(x_0)}{h}$$

where $\mathbf{e_k}$ is the **k**-th coordinate unit vector.

We can approximate this limit using a small, nonzero value for **h**. While simple, this method is easy to implement and sometimes used effectively with low-D system states!


Journal of Geophysical Research: Oceans

RESEARCH ARTICLE
10.1029/2018JC014283

Special Section:
The U.S IOOS Coastal and Ocean Modeling Testbed 2013-2017

A Simple Finite Difference-Based Approximation for Biogeochemical Tangent Linear and Adjoint Models

Jann Paul Mattern[1] and Christopher A. Edwards[1]

[1]Department of Ocean Sciences, University of California, Santa Cruz, CA, USA

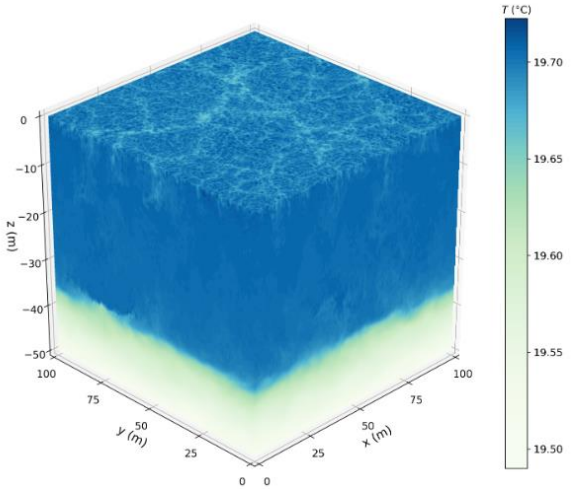
Strategy C: automatic differentiation

Instead of writing a computer program to calculate **dM(x) / dx**, we can calculate this derivative automatically.

For simulators implemented in modern machine learning frameworks (PyTorch, Julia, JAX, etc.), derivatives can be calculated for all implemented functions automatically with no extra programming.

Added bonus: GPU support & frequent updates!

Oceanic boundary layer turbulence forced by surface cooling
Oceananigans.jl in Julia
Ramadhan et al., 2020



What about the vast majority of Earth science simulators, which are programmed in environments (e.g. Fortran) that don't provide autodiff?

ADIFOR—Generating Derivative Codes from Fortran Programs*

CHRISTIAN BISCHOF[1], ALAN CARLE[2], GEORGE CORLISS[1], ANDREAS GRIEWANK[1], AND PAUL HOVLAND[1]

[1]*Mathematics and Computer Science Division, Argonne National Laboratory, 9700 S. Cass Avenue, Argonne, IL 60439*

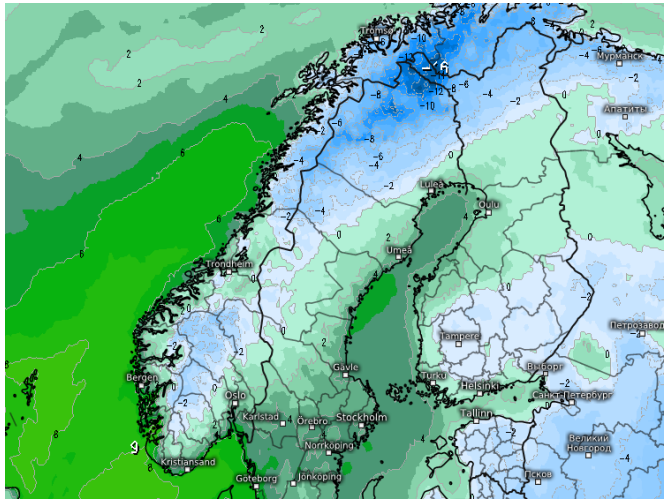
[2]*Center for Research on Parallel Computation, Rice University, P. O. Box 1892, Houston, TX 77251*

Some tools for automatically writing differentiation routines do exist, but are mainly based on inefficient static code analysis, and rarely used.

How to minimize $\mathcal{L}_{DA}(x_0)$?

Strategy D: Ensemble Kalman Filter (EnKF)

- Generates simulation **ensemble** with different system states
- Ensemble members nudged toward the observations, increasing **p(y | x)**
- Full mathematical details beyond our scope, see Roth et al. 2017 for a review.
- Approximates $p(x_i \mid y)$ as Gaussian, less ideal for highly nonlinear **M** / non-Gaussian obs.
- Differentiation not required, but large systems require hyperparameter tuning, tricks and assumptions that depending on spatial resolution etc.
- Strictly speaking doesn't minimize $\mathcal{L}_{DA}(x_0)$, but end effect is similar

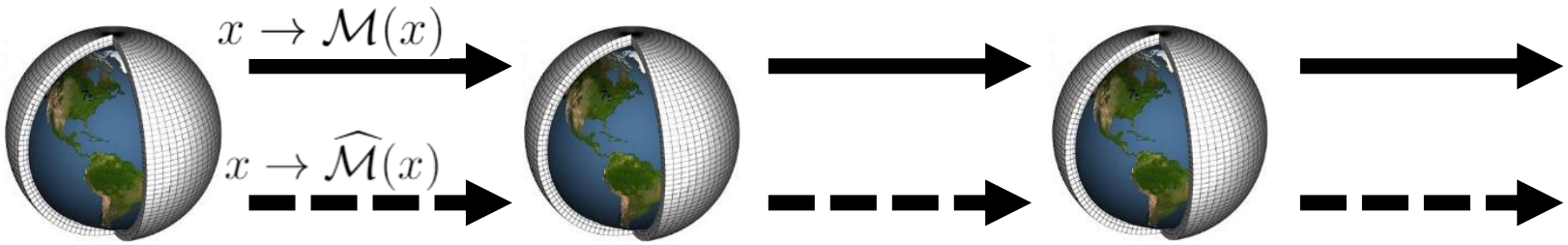


Temperature prediction over northern Europe, 12:00 on 07.12.20
ICON, run by the German Weather Service using EnKF

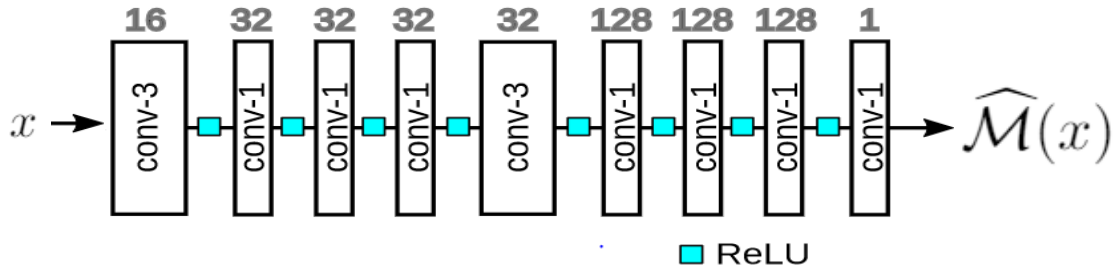
Differentiating Simulators with ML

How can we obtain derivatives of the state update function $\mathcal{M}$?

1. Generate simulations

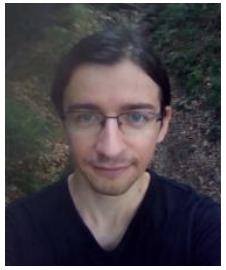


2. Train a neural network to *emulate* the state update function



3. Differentiate the neural network's outputs w.r.t its inputs to obtain:

$$\frac{d\widehat{\mathcal{M}}}{dx} \approx \frac{d\mathcal{M}}{dx}$$



Marcel Nonnenmacher

We've used the differentiability of $\widehat{\mathcal{M}}$ twice:
once to train it, once to estimate simulator gradients.

Nonnenmacher & Greenberg, 2020, *JAMES*

Objective Function for Emulator Training

Objective function

Next system state

Emulator state update function

$$\mathcal{L}_{\mathrm{DYN}}(\phi) = \sum_{n} ||\mathbf{x}_{n+1} - \widehat{\mathcal{M}}_{\phi}(\mathbf{x}_n)||_2^2 = ||M(\mathbf{x}_n) - \widehat{\mathcal{M}}_{\phi}(\mathbf{x}_n)||_2^2$$

Neural network parameters

Time index

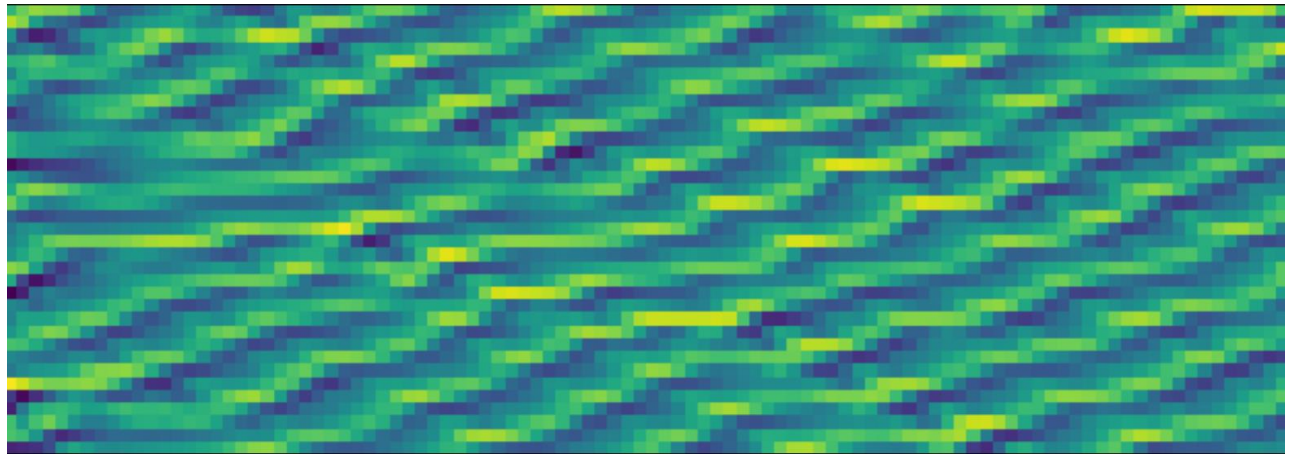
Simulator state update function

Model System: Lorenz `96

$$\frac{dx_k}{dt} = -x_{k-1}\left(x_{k-2} - x_{k+1}\right) - x_k + F$$

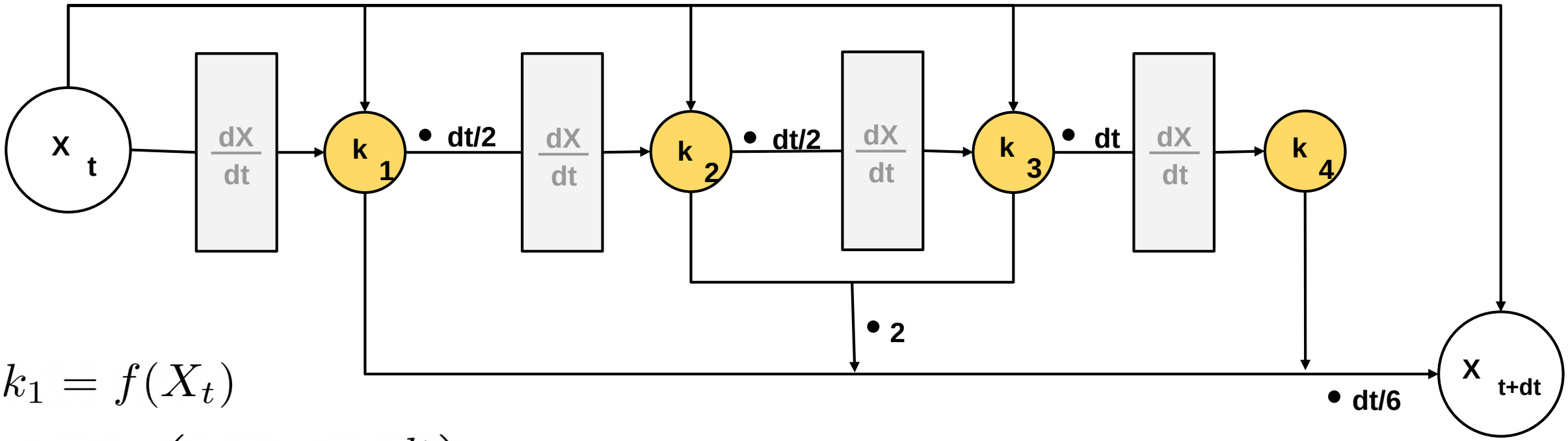
Lorenz, 1996

- 40 coupled nonlinear differential equations
- Chaotic dynamics (Lyapunov time 1.67 for F = 8)



Time

Runge-Kutta Integration



$$k_1 = f(X_t)$$

$$k_2 = f\left(X_t + k_1 \cdot \frac{dt}{2}\right)$$

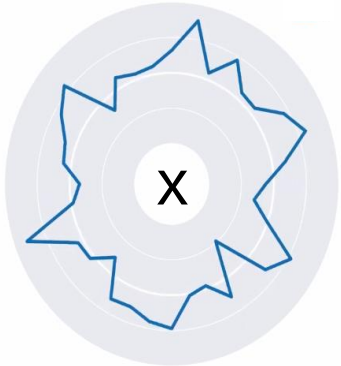
$$k_3 = f\left(X_t + k_2 \cdot \frac{dt}{2}\right)$$

$$k_4 = f\left(X_t + k_3 \cdot dt\right)$$

$$X_{t+dt} = X_t + \frac{dt}{6}(k_1 + 2(k_2 + k_3) + k4)$$

Choosing a Neural Network Architecture for our Emulator

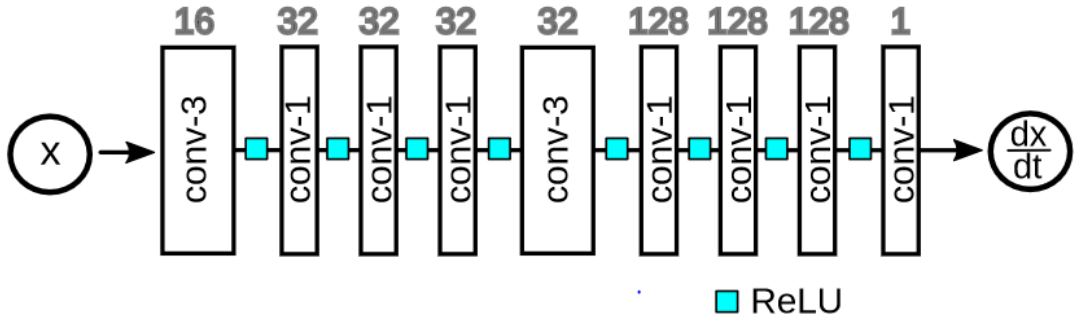
$$\frac{dx_k}{dt} = -x_{k-1}\left(x_{k-2} - x_{k+1}\right) - x_k + F$$



How should we design the network architecture of the emulator?

Decisions to be made:

- Number of layers
- Number of units in each layer
- Type of layers (convolutional, recurrent, …)
- Activation function (Relu, tanh, Elu …)

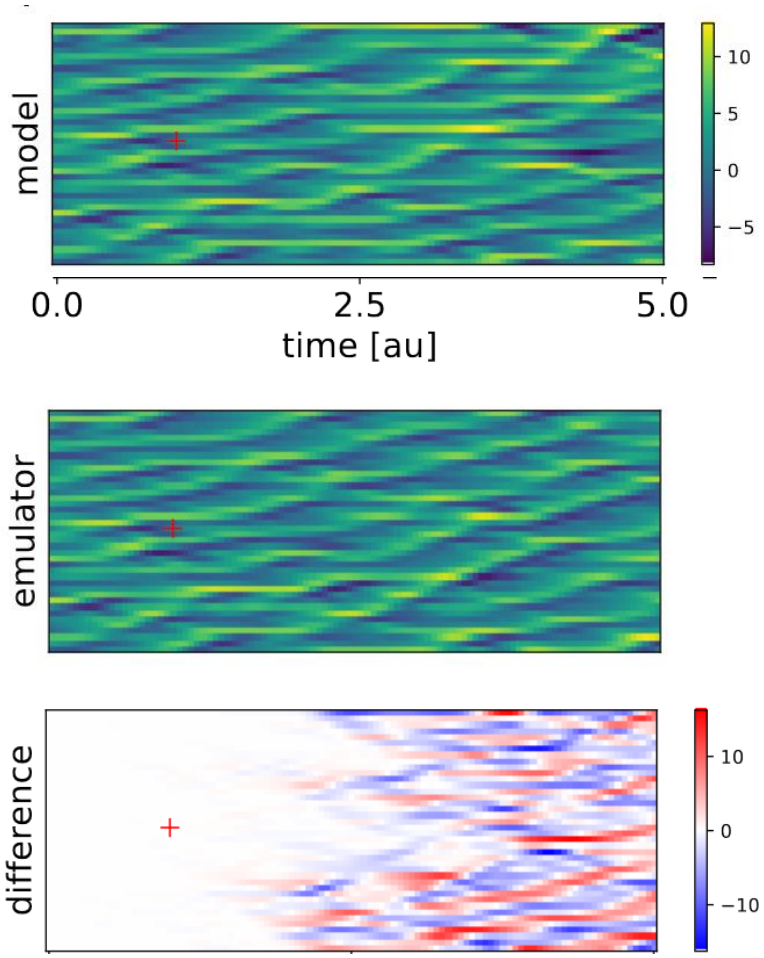


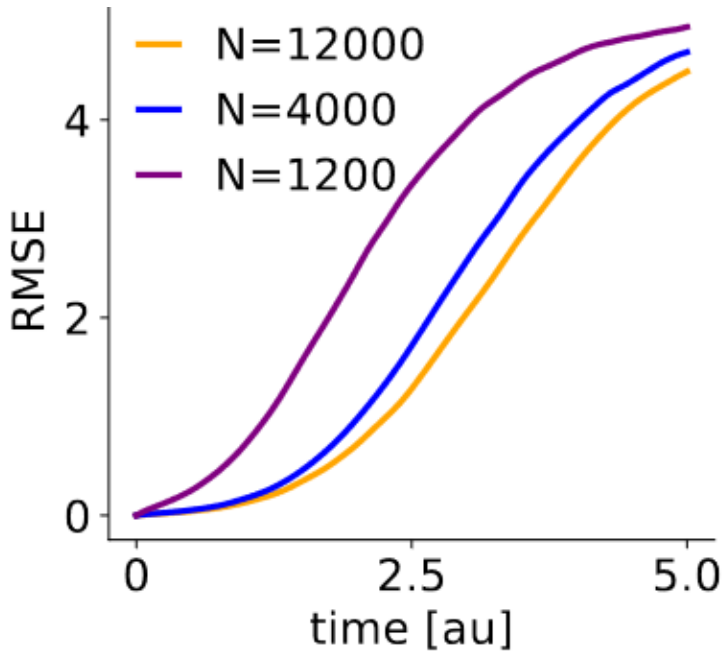
For L96:

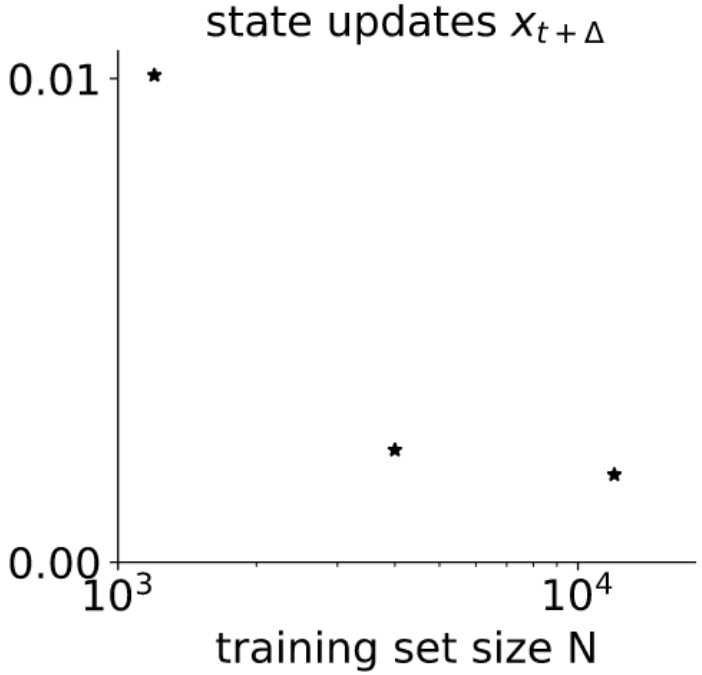
- We use 1D convolutional layers to exploit **spatial structure** and spatial **invariance**
- We use **periodic** convolutions to match information flow in the state space
- We use a limited number of layers with size 3 kernels, and the rest operate on each spatial location independently to preserve **causal structure** for any network depth
- We could also have tried to **match the mathematical operations** in the simulator (e.g. bilinear layers)

Overall: we can use **partial knowledge** in architecture design

Accuracy of emulators trained on Lorenz `96

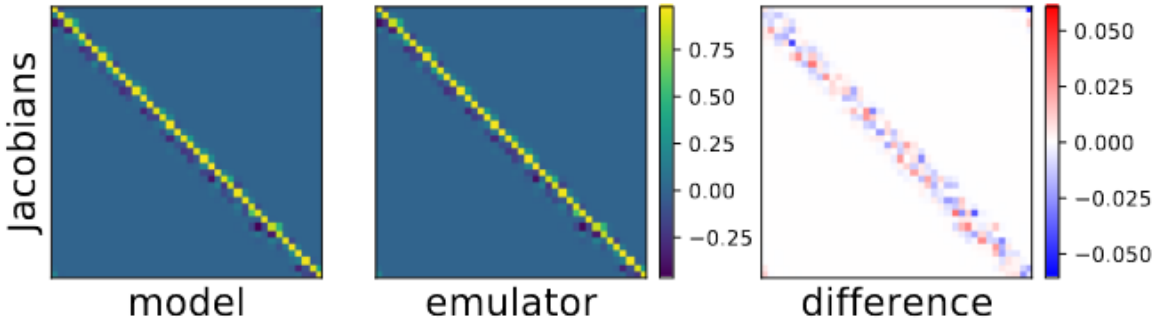




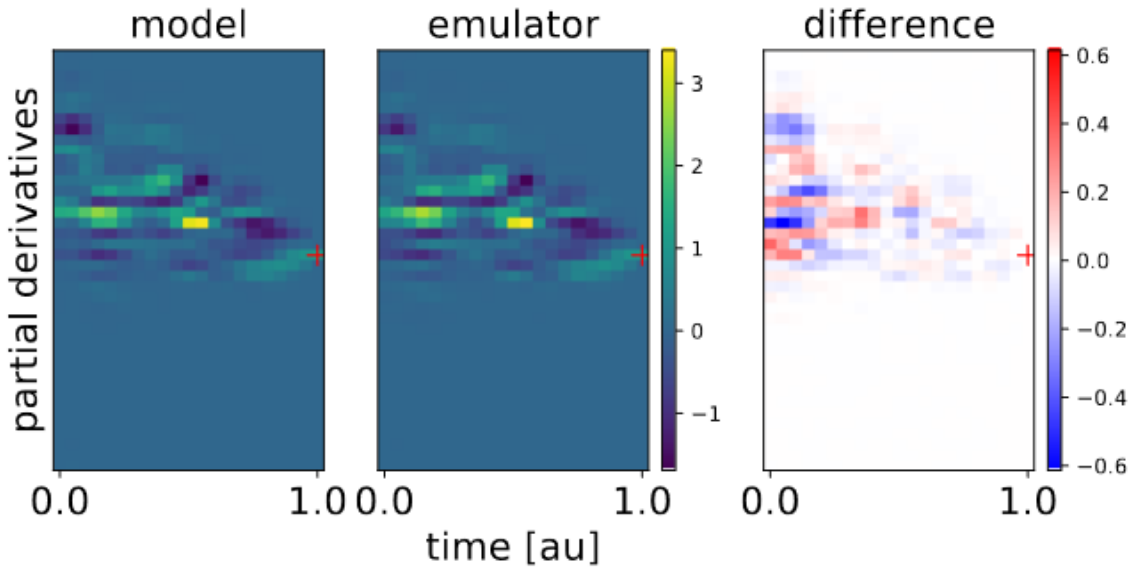


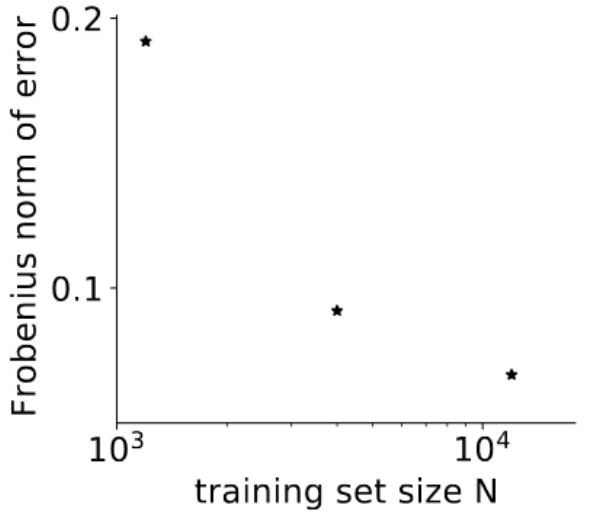
How Accurate are Estimated Input Derivatives?

…for a single time step?

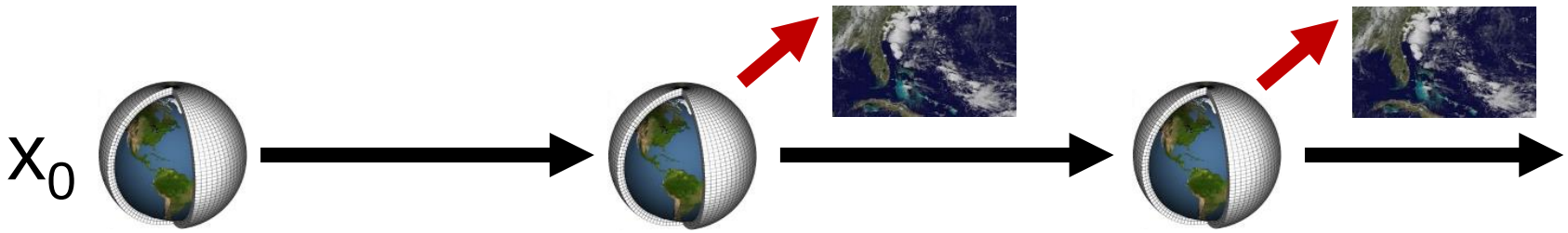


…for longer simulations?





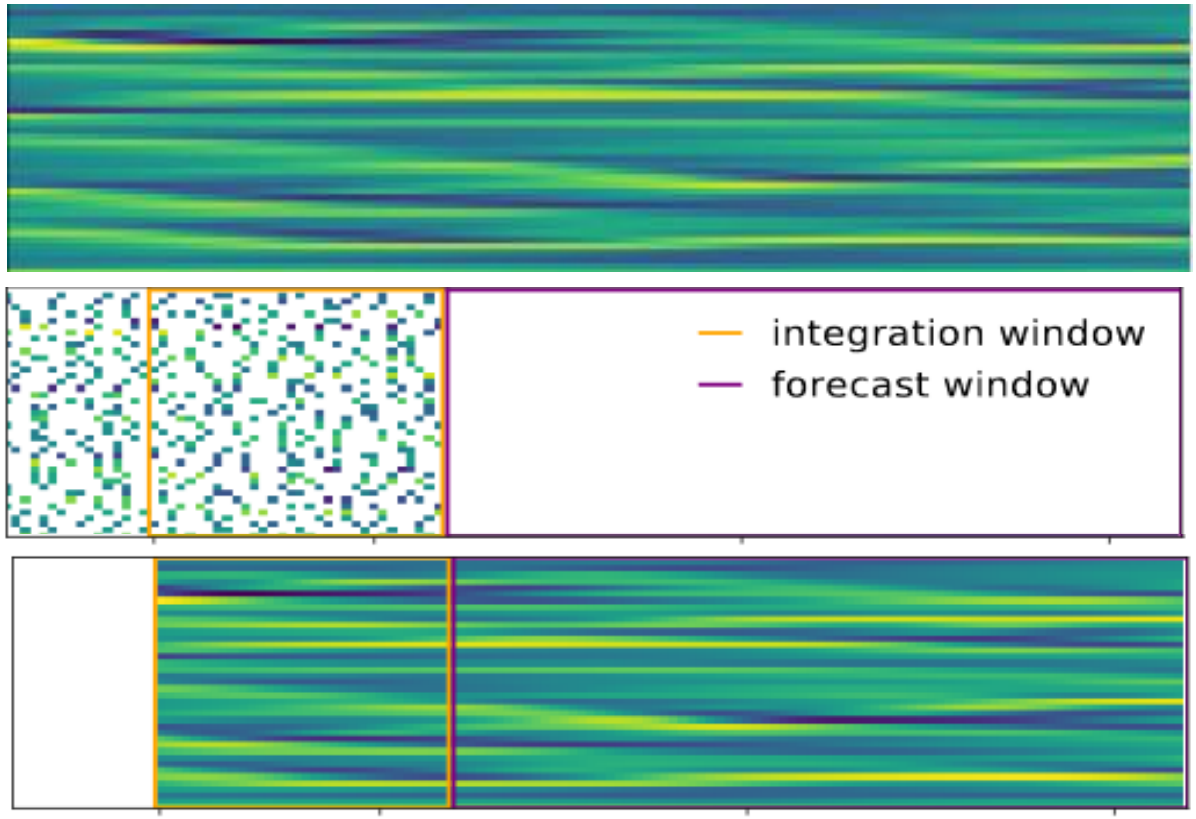
Data Assimilation for L96



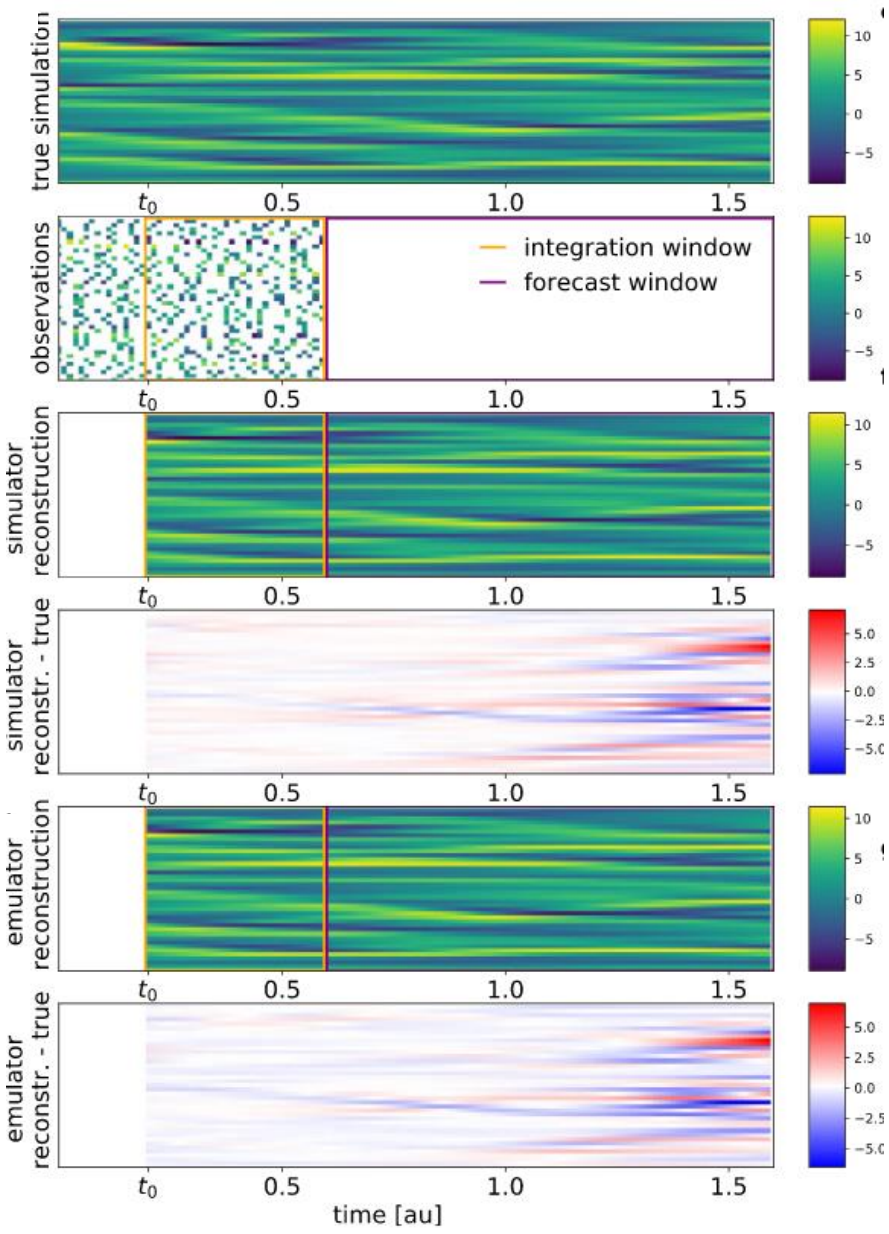
Ground truth

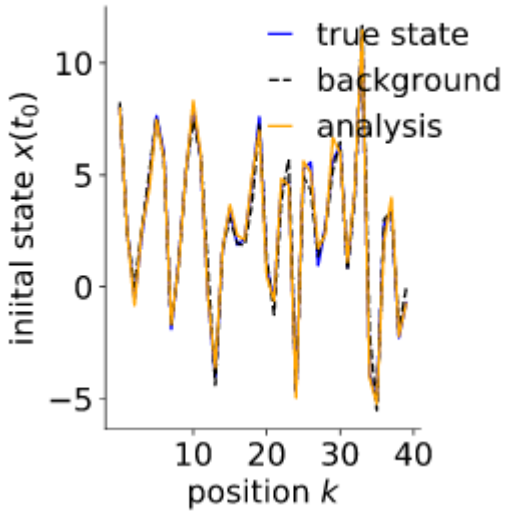
Noisy & incomplete observations

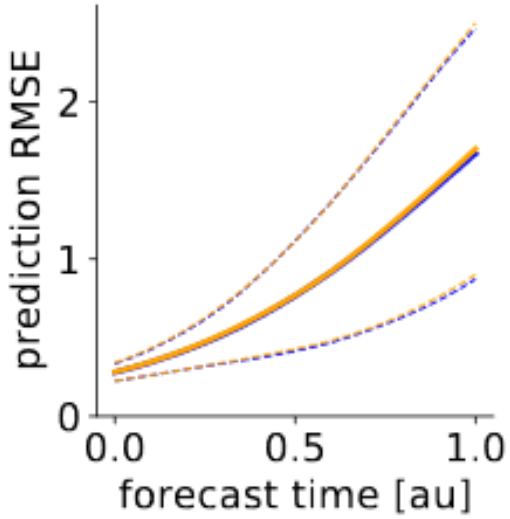
Emulator reconstruction

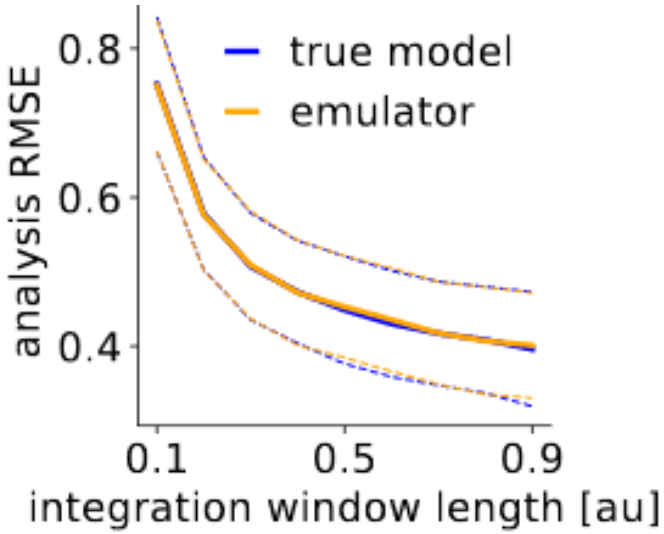


Data Assimiliation with True vs. Estimated Simulator Derivatives

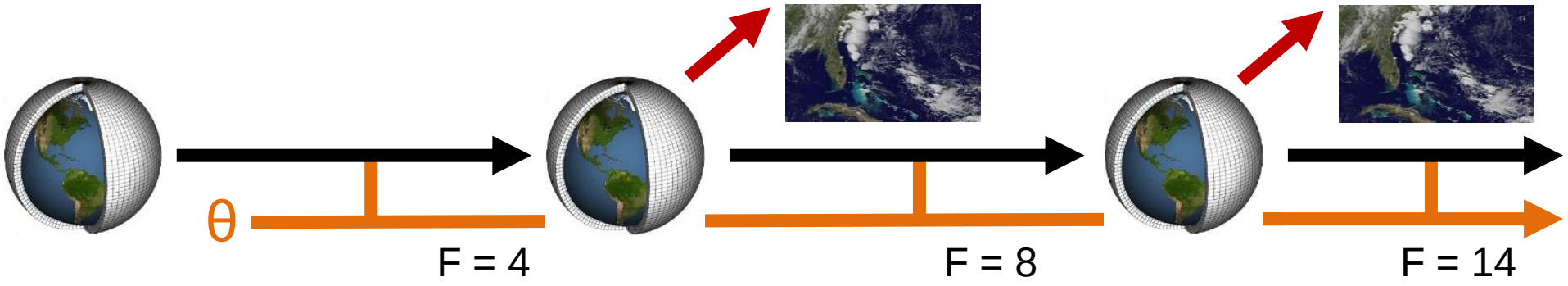






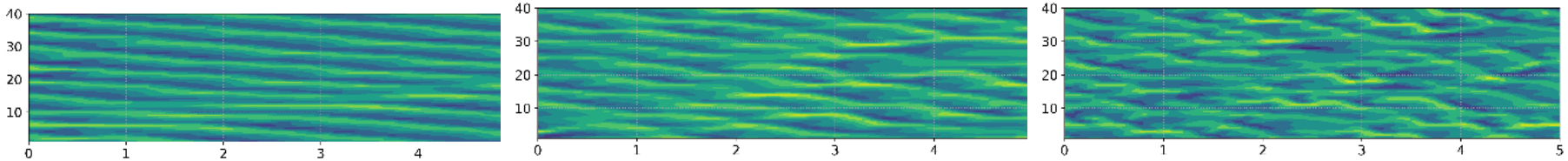


Application 2: Model Tuning

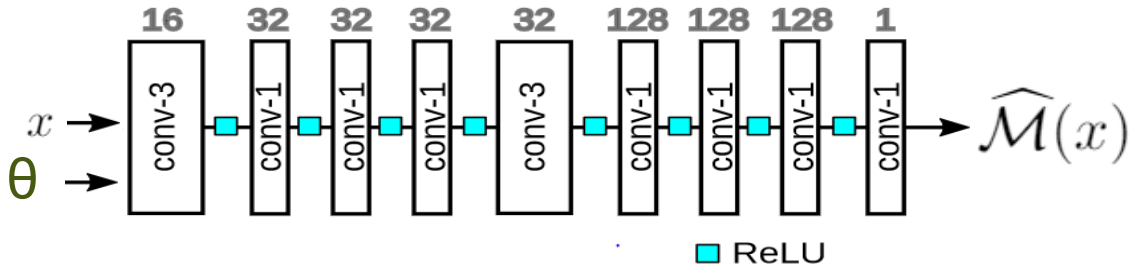


Simulator

$$\frac{dx_k}{dt} = -x_{k-1}\left(x_{k-2} - x_{k+1}\right) - x_k + F$$



Conditional emulator

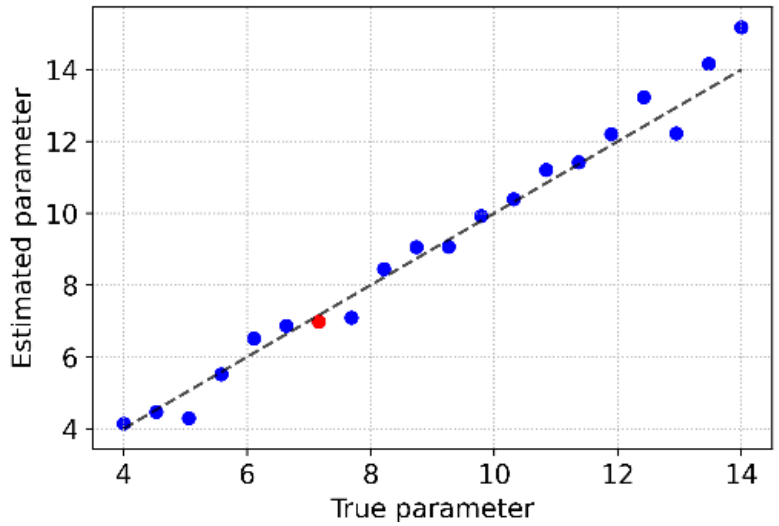


Vadim Zinchenko

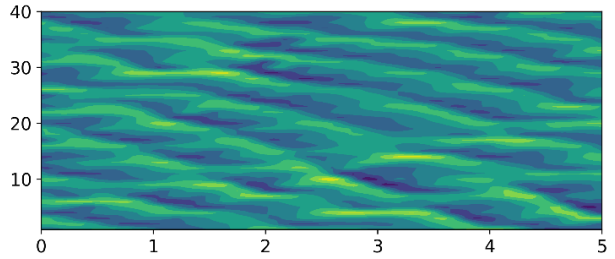
Application 2: Model Tuning

Task: estimate parameter from noisy, incomplete observations

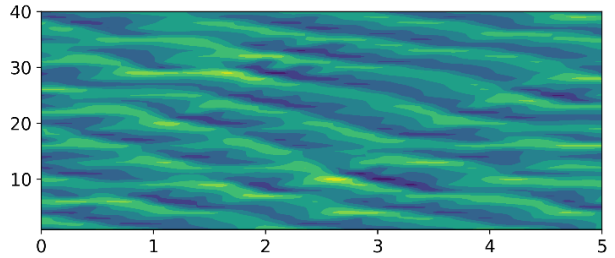
Strategy: estimate simulator parameters by optimizing over the input to a conditional emulator



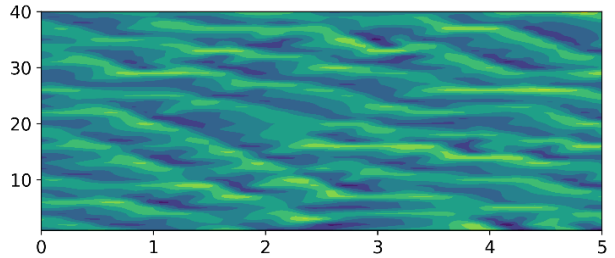
Simulator
F = 7.1



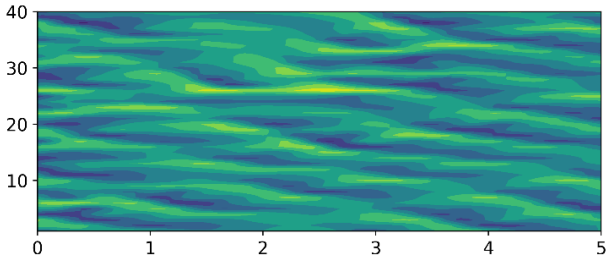
Conditional emulator
F = 7.0 (optimized)



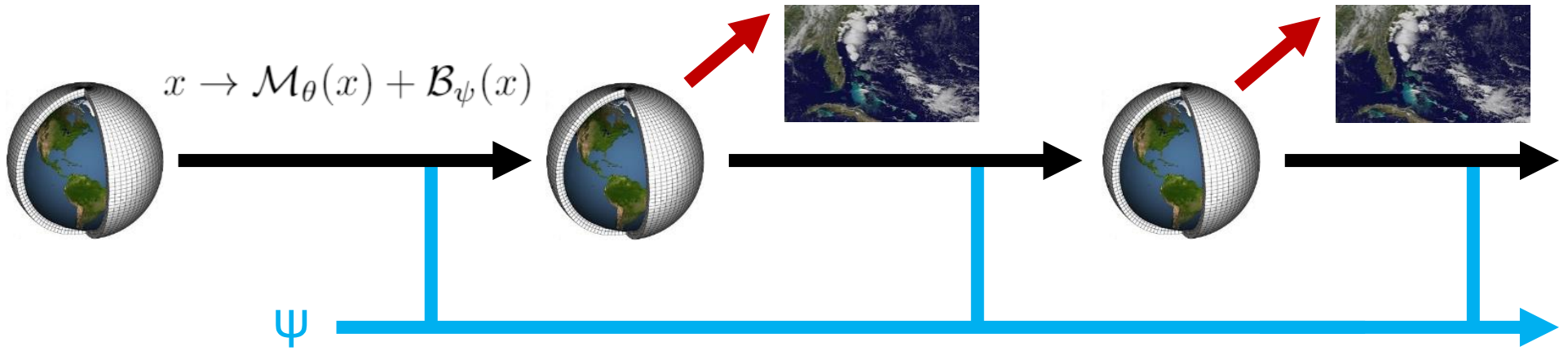
Conditional emulator
F = 5.0 (too low)



Conditional emulator
F = 9.0 (too high)



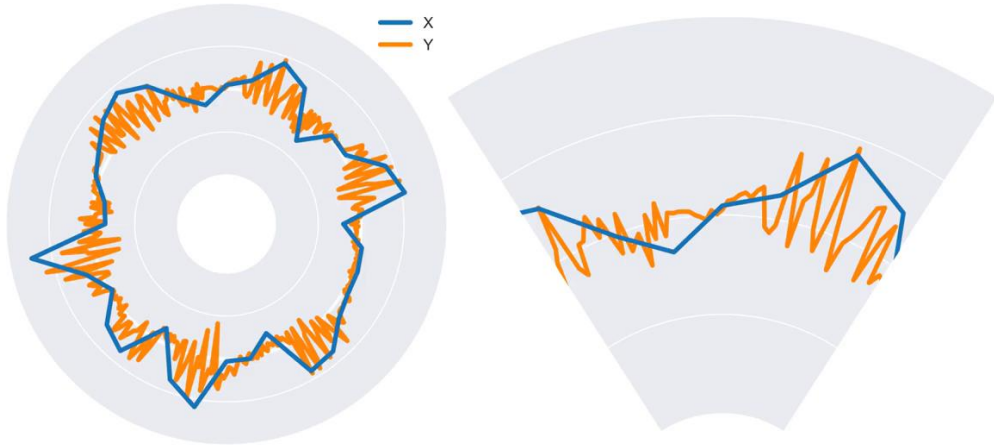
Application 3: Learning To Parameterize Sub-grid-scale Processes



2-level L96 model

$$\frac{dx_k}{dt} = -x_{k-1}\left(x_{k-2} - x_{k+1}\right) - x_k + F - hc\mathbb{E}_j[z_{j,k}]$$

$$\frac{1}{c}\frac{dz_{j,k}}{dt} = -bz_{j+1,k}\left(z_{j+2,k} - z_{j-1,k}\right) - z_{j,k} + \frac{h}{J}x_k$$



Animation: Stephan Rasp

Task: learn a correction to 1-level L96 so that it behaves like the 2-level model

Application 3: Learning To Parameterize Sub-grid-scale Processes

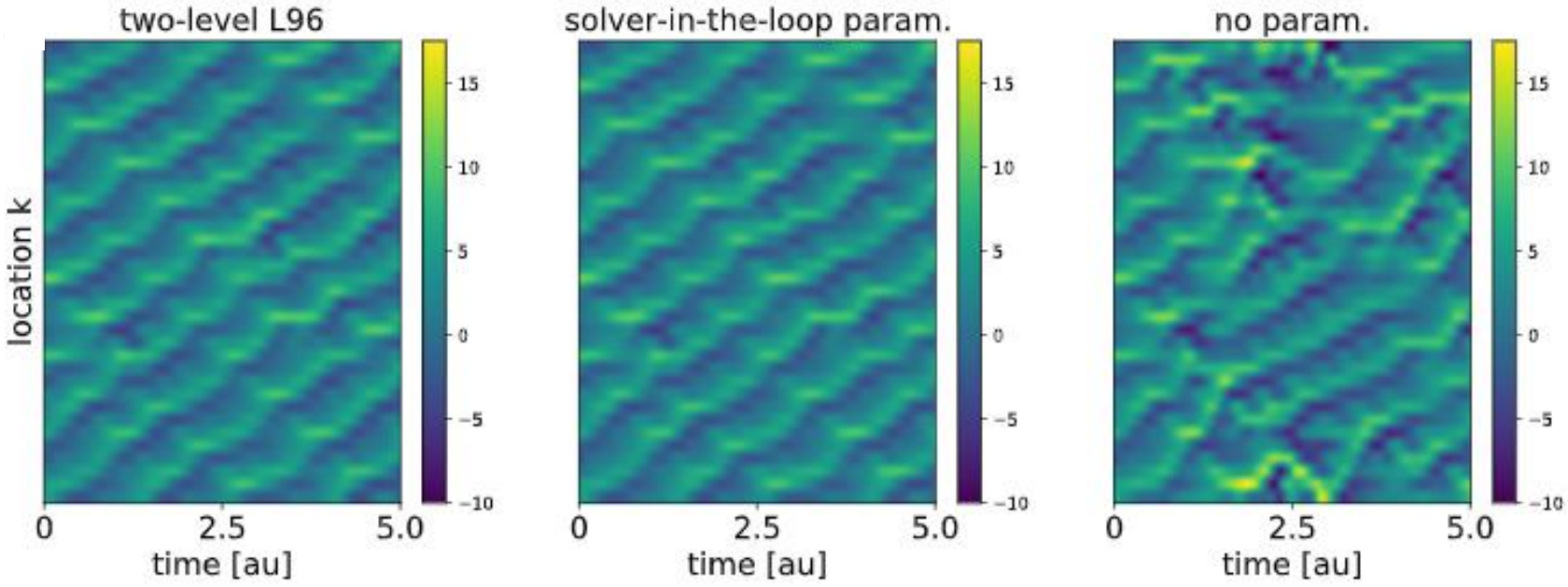
Fine-scale model

$$\frac{dx_k}{dt} = -x_{k-1}\left(x_{k-2} - x_{k+1}\right) - x_k + F - hc\mathbb{E}_j[z_{j,k}]$$

$$\frac{1}{c}\frac{dz_{j,k}}{dt} = -bz_{j+1,k}\left(z_{j+2,k} - z_{j-1,k}\right) - z_{j,k} + \frac{h}{J}x_k$$

Coarse-scale model with corrective term

$$\frac{dx_k}{dt} = -x_{k-1}\left(x_{k-2} - x_{k+1}\right) - x_k + F + \mathcal{B}_\psi(x_k)$$



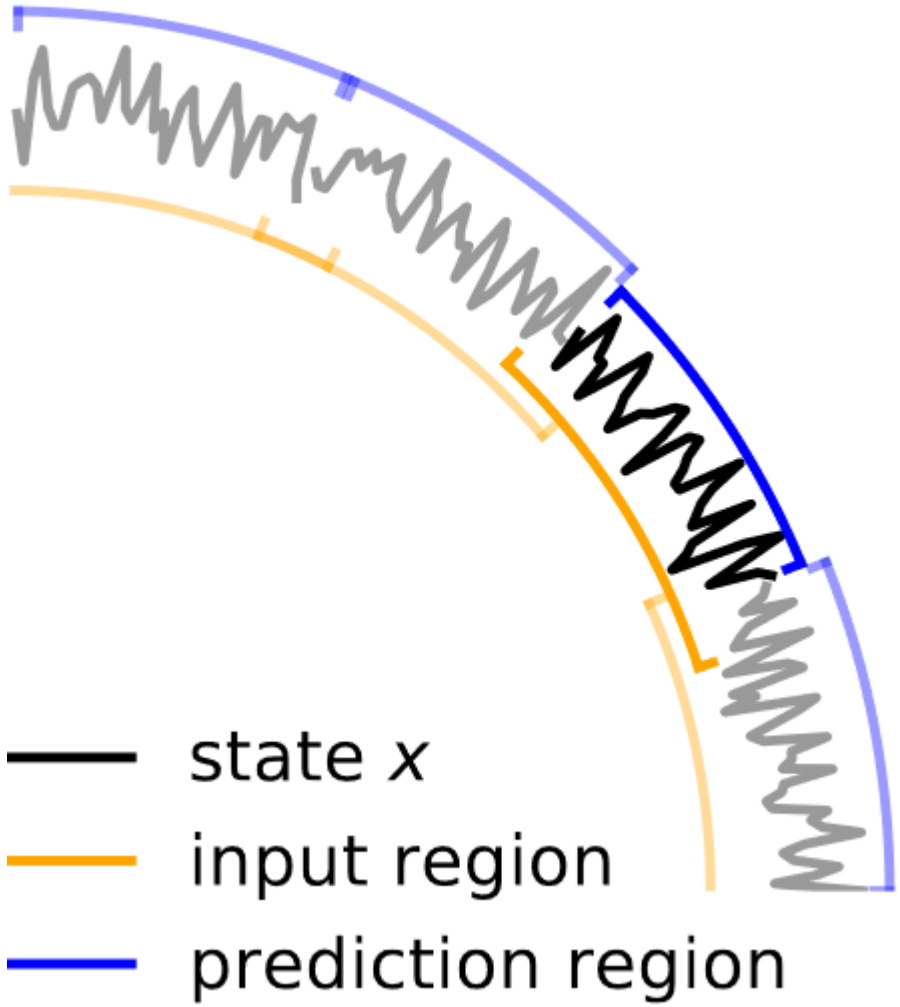
Outlook and Future Challenges

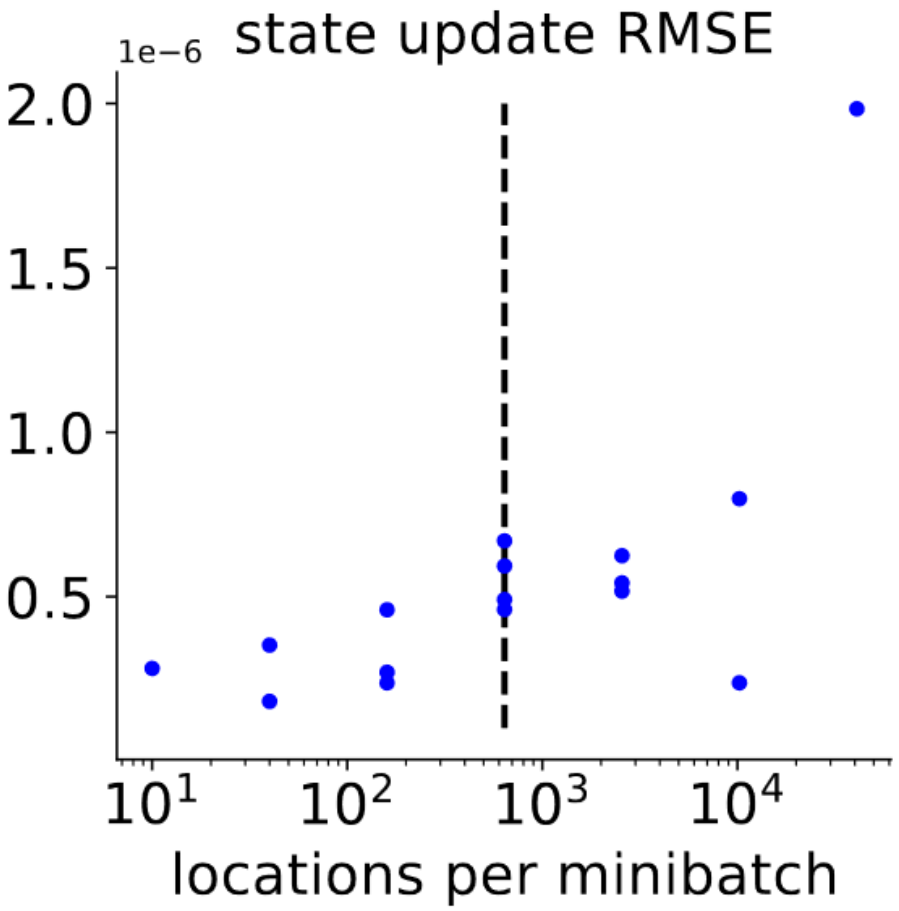
Differentiable emulators can provide $\frac{d\mathcal{M}}{dx}$ in order to optimize our predictions over x_0, θ or ψ.

But:

- physical knowledge is discarded
- too many potential interactions, especially for fluid dynamical integration steps.
- Difficult to scale to large and complex systems

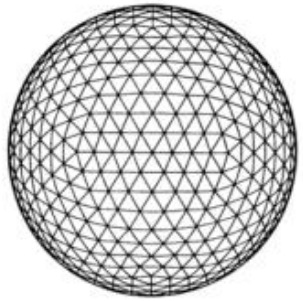
Training Emulators with Local Structure on Partial System States

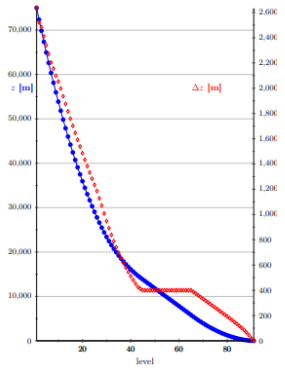


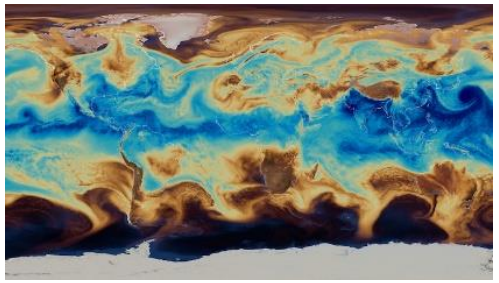


Forward pass compute for Earth science >> ML

Example: ICON model for numerical weather prediction (DWD, MPI-M)







2949120 grid cells

90 z-levels

10+ variables / location

Total values / time point: **2.6 billion**

Or with a time step of 120 seconds,
Total values / simulated day: **1.9 trillion**

Mistral, DKRZ (>10^5 processors, 3.6 PetaFLOPS)

Simulation code: FORTRAN, often > 10^5 lines

```
108   do i = 1,nx,1
109     im = max(1,i-1)
110     ip = min(nx,i+1)
111
112     if((d(i).gt.0.1).and.(d(ip).gt.0.1)) then
113       ce(i) = dt*dt*wimp*wimp*g*0.5*(h(i)+h(ip))/(dx*dx)
114     endif
115
116     if((d(i).gt.0.1).and.(d(im).gt.0.1)) then
117       cw(i) = dt*dt*wimp*wimp*g*0.5*(h(i)+h(im))/(dx*dx)
118     endif
119
120   enddo
```

Stevens et al., 2019, Reinert et al., 2021

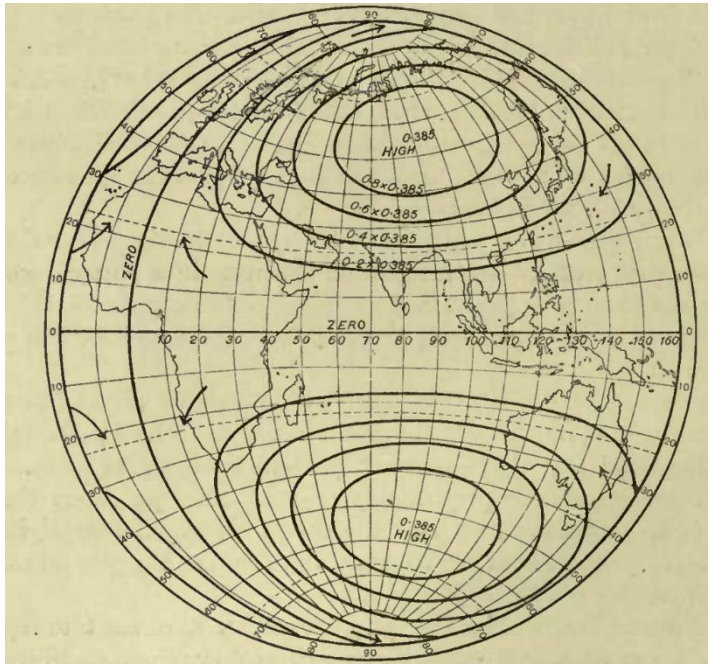
Outline

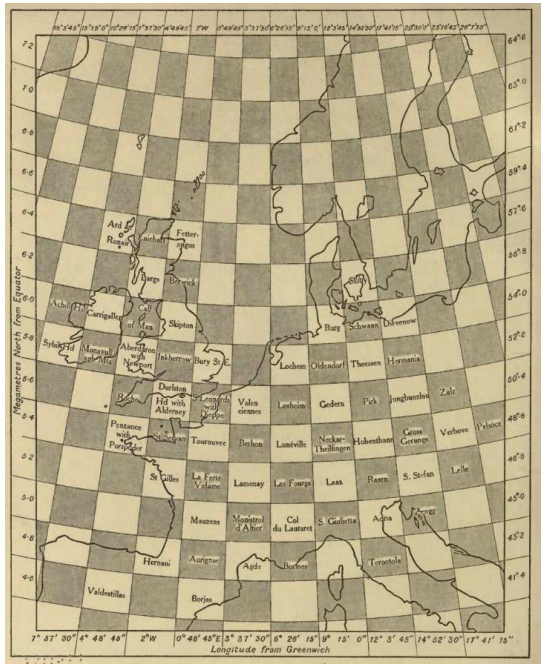
1 Deep Differentiable Emulators for Inverse Problems in the Geosciences

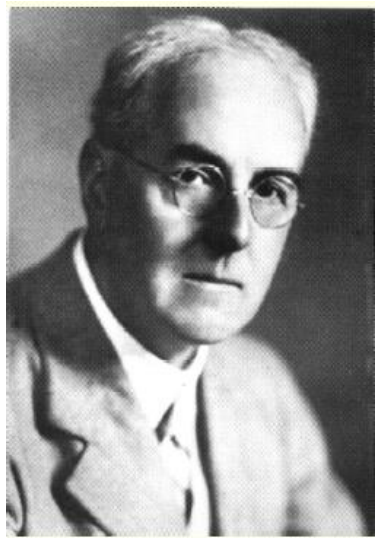
2 Hybrid Emulators

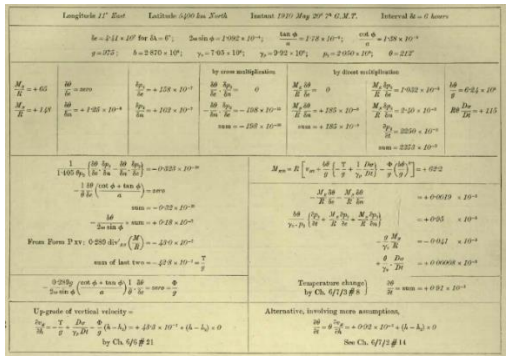
3 ML Models as Earth System Model Components

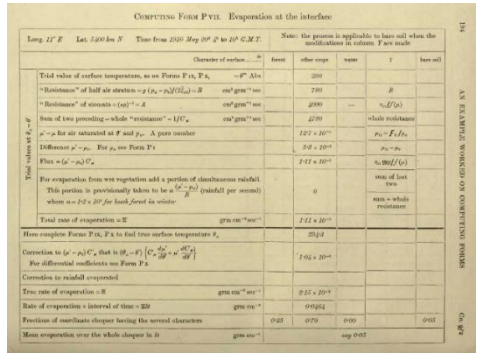
Weather Prediction by Numerical Process (Richardson, 1922)

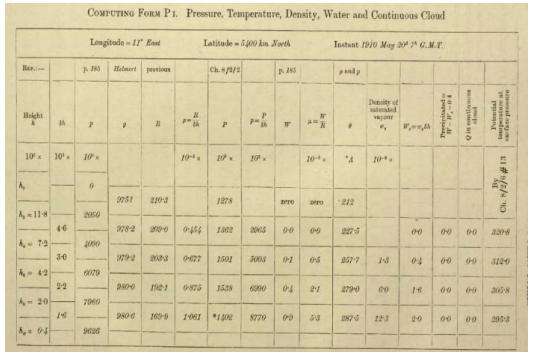












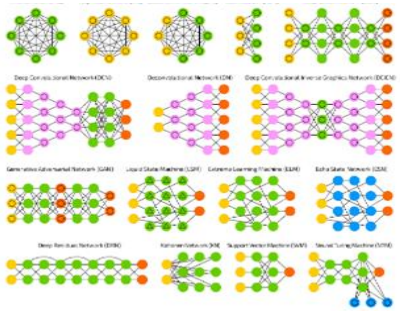
Complementary Advantages of ML and Simulation

Machine Learning

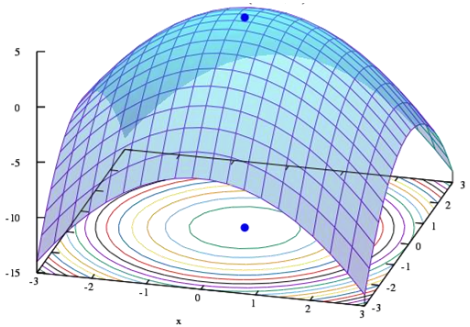
✓ Performant, efficient

✗ Non-interpretable, poor generalization

Architectures



Algorithms

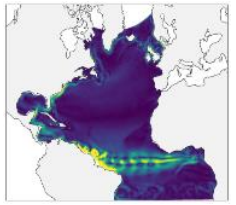


Numerical Simulation

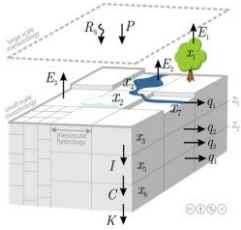
✓ Interpretable, physical, good generalization

✗ Hard to fit data, difficult inverse problems

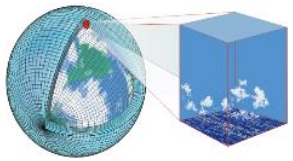
Ocean



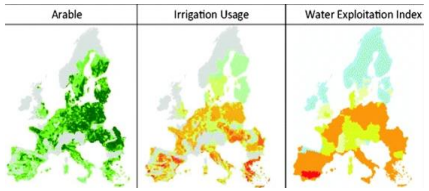
Hydrology



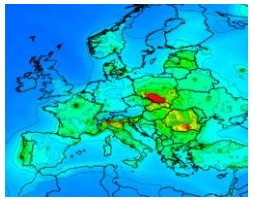
Atmosphere



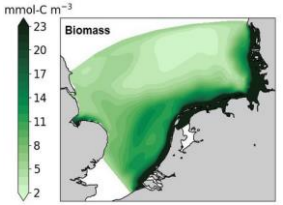
Impacts



Chemistry



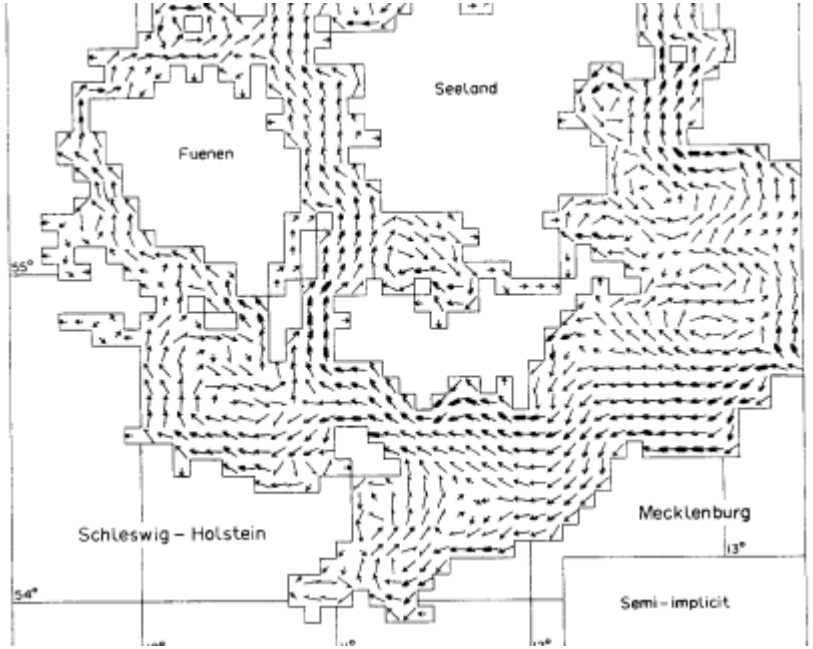
Ecosystems



Hybrid Methods

Shallow Water Equations – Semiimplicit Scheme

$$U_t + gH\zeta_x = X - \tau_B^x$$
$$V_t + gH\zeta_y = Y - \tau_B^y$$
$$\zeta_t + U_x + V_y = 0.$$



A semi-implicit scheme for the shallow water equations for application to shelf sea modelling

JAN O. BACKHAUS*

(*Received* 15 *February* 1983; *in revised form* 8 *September* 1983; *accepted* 10 *October* 1983)

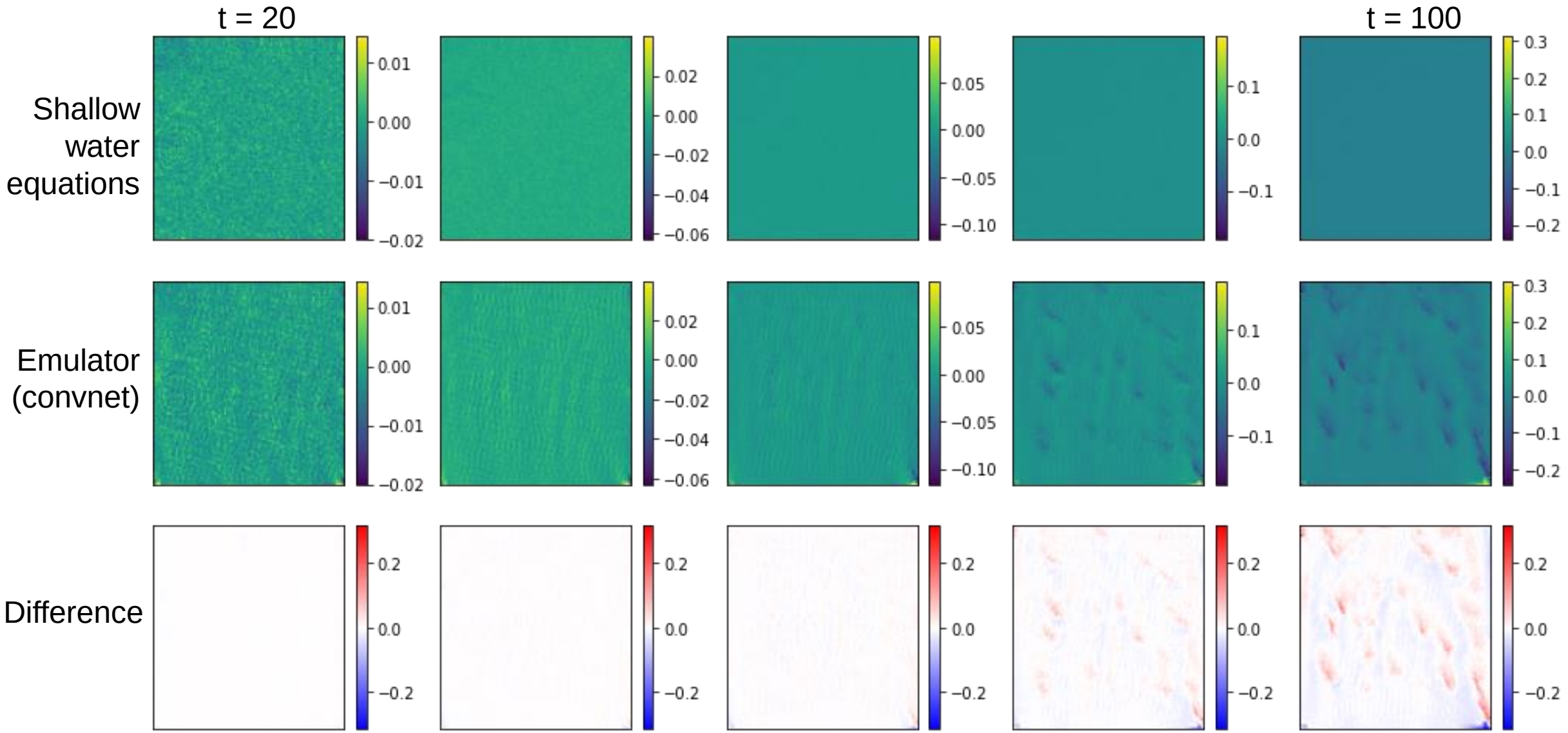
Abstract—A semi-implicit scheme for the numerical solution of the shallow water equations is proposed. The scheme is suitable for the simulation of shelf sea dynamics as is demonstrated by some examples of successful application covering a range of grid sizes typical for shelf sea models. The basic outlines of the method are presented. Some practical aspects of computation are discussed which illustrate that an explicit model can be modified easily to the semi-implicit version proposed here. Compared to explicit schemes the semi-implicit approach has two major advantages: (1) its economy (a saving of at least 50% in computing time can be achieved); (2) a closer match is obtained between the time-stepping procedure and the time scales of processes, the spatial scales of which are close to the lower limit of the resolution of the model grid.

With the index-notation described above this system is approximated as

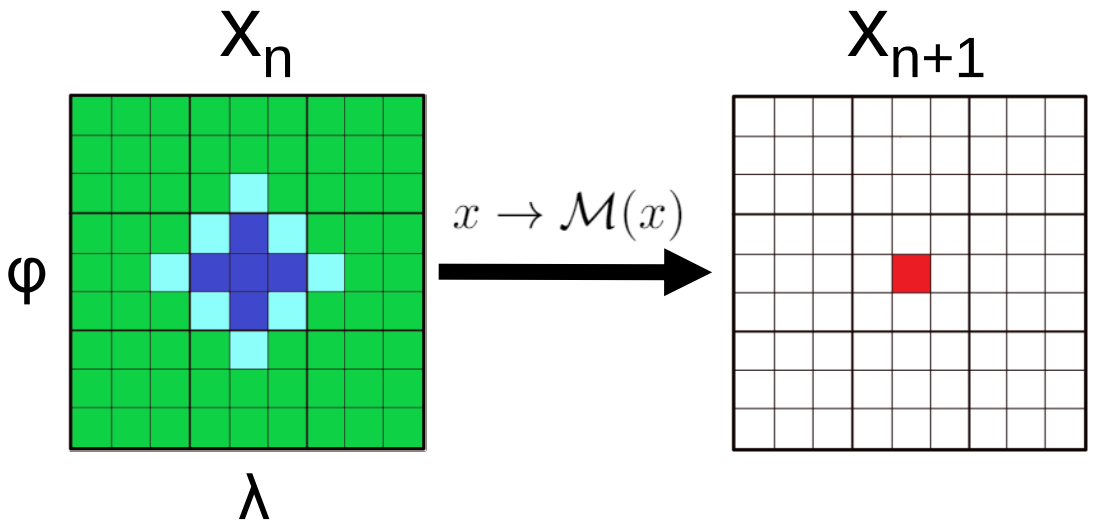
$$U^{n+1} = F^x(U^n + \Delta t(X^n - g\bar{H}^x(\zeta_E^{n+1} - \zeta_W^{n+1} + \zeta_E^n - \zeta_W^n)/2\ \Delta x)) \quad (2)$$
$$V^{n+1} = F^y(V^n + \Delta t(Y^n - g\bar{H}^y(\zeta_N^{n+1} - \zeta_S^{n+1} + \zeta_N^n - \zeta_S^n)/2\ \Delta y)) \quad (3)$$
$$\zeta^{n+1} = \zeta^n - \Delta t((U_E^{n+1} - U_W^{n+1} + U_E^n - U_W^n)/2\Delta x + (V_N^{n+1} - V_S^{n+1} + V_N^n - V_S^n)/2\Delta y), \quad (4)$$

where U, V denote east- and north-components of vertically integrated transport; ζ is the surface elevation; H is the total water depth ($H = D + \zeta$) at time-level n; D is the undisturbed depth; $F^{x,y}$ is the nondimensional friction function; $\tau_B^{x,y}$ is the bottom stress; Δx, Δy are space increments along east- and north-axis; Δt is time-step; n is the time-level index; g is acceleration due to gravity; and N, E, S, W are relative space indices.

Can we learn 2D SWE with standard convnets?



Spatial dependency of updates



PDE

$$\frac{\partial x(t,\lambda,\phi)}{\partial t} = \mathbf{f}\left(x(t,\lambda,\phi), \frac{\partial x(t,\lambda,\phi)}{\partial \lambda}, \frac{\partial x(t,\lambda,\phi)}{\partial \phi}, \ldots\right)$$

Discretization

$$\frac{\partial x(t,\lambda,\phi)}{\partial \lambda} \approx \frac{x(t,\lambda+\Delta\lambda,\phi) - x(t,\lambda,\phi)}{\Delta\lambda}$$

Explicit Euler Integration

$$x_{n+1} \leftarrow x_n + \Delta t \mathbf{f}(x_n, \ldots)$$

2nd order explicit

$$x_{n+1} \leftarrow x_n + \Delta t \mathbf{f}(x_n + (\Delta t/2)\mathbf{f}(x_n, \ldots), \ldots)$$

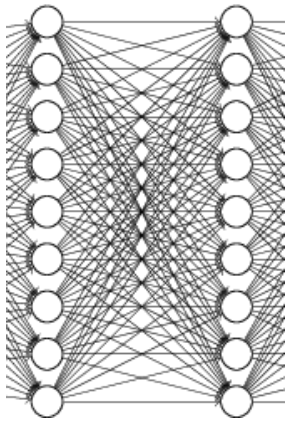
(Semi)implicit

$$x_{n+1} \leftarrow x_n + \Delta t \left(\alpha \mathbf{f}(x_n, \ldots) + (1-\alpha)\mathbf{f}(x_{n+1}, \ldots)\right)$$

← CNNs scale poorly with size of x

Putting a Linear Solver into a Neural network

input x output y

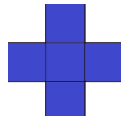


Fully connected layer

$$\underbrace{y}_{m \times 1} = \underbrace{A}_{m \times m} \cdot \underbrace{x}_{m \times 1}$$

Convolutional layer

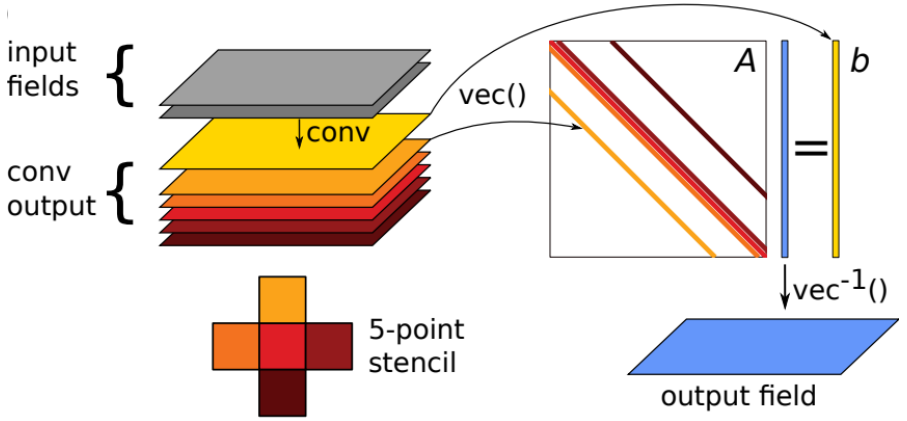
$$y = \underbrace{K}_{k \times k,\ k \ll m} * x = \underbrace{C}_{m \times m,\ k^2 m \text{ elements} \neq 0} \cdot x$$

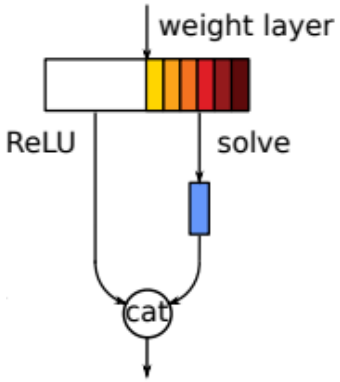


Implicit Linear Layer

$$\underbrace{A(x)}_{m \times m} \cdot \underbrace{y}_{m \times 1} = \underbrace{b(x)}_{m \times 1}$$

k^2m elements $\neq 0$



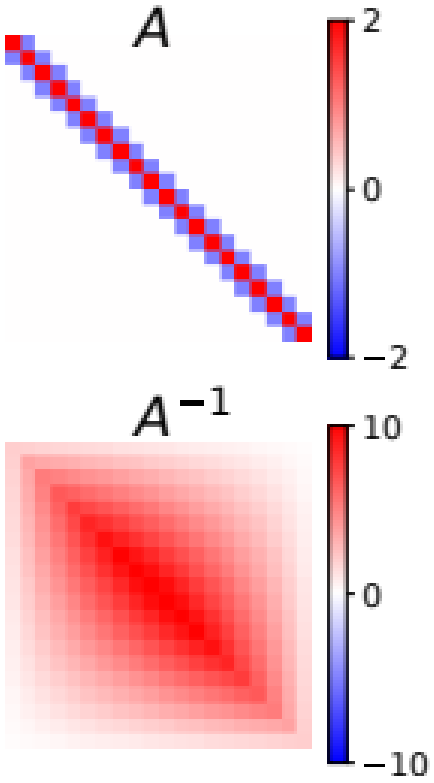


Nonnenmacher & Greenberg, 2021

Linear Implicit Layers: Forward Pass

Banded matrix:

$A_{ij} = 0$ when $|i - j| > B$



A is sparse, but its inverse is not!

Forward pass

Goal: compute y from x, θ

$$A(x) = \sum_{k \in K \subset \mathbb{Z}} A_k$$

$$A_k = \sum_i e_i e_{i+k}^T g_i^k(x, \theta_i)$$

$$b(x) = g_b(x, \theta_b)$$

We solve $Ay = b$ using an iterative solver

$$y_m \to y$$

e.g. conjugate gradient or red-black Gauss-Seidel.

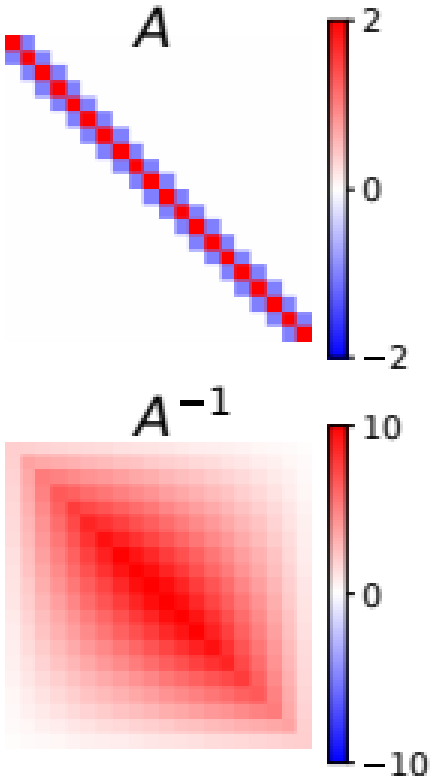
This doesn't require inverting or factorizing A, only calculating $Az = \sum_k A_k z = diag(g^k) \cdot S^{-k} z$, where S^{k} shifts the vector elements forward by k positions

This allows us to solve the system even for very large A, such as when x and y are images.

Linear Implicit Layers: Backward Pass

Banded matrix:

$A_{ij} = 0$ when $|i - j| > B$



A is sparse, but its inverse is not!

Backward pass

Goal: compute $\frac{d\mathcal{L}}{dx}, \frac{d\mathcal{L}}{d\theta}$ from $x, y, \theta, \frac{d\mathcal{L}}{dy}$

Implicit differentiation w.r.t b gives

$$\frac{d}{db}(Ay) = I$$

$$\frac{dy}{db} = A^{-1}$$

$$\frac{d\mathcal{L}}{db} = \frac{d\mathcal{L}}{dy}\frac{dy}{db} = \frac{d\mathcal{L}}{dy}A^{-1}$$

$$A^T\frac{d\mathcal{L}}{db}^T = \frac{d\mathcal{L}}{dy}^T$$

so we have to solve the transposed (adjoint) system in the backward pass.

A similar argument gives

$$\frac{d\mathcal{L}}{dA} = -\frac{d\mathcal{L}}{db}vec(z)^T$$

So we only need one solve.

LILs: How to Ensure the Learned System can be Solved

We want to ensure

A) Invertibility of A

B) Convergence of the iterative solver

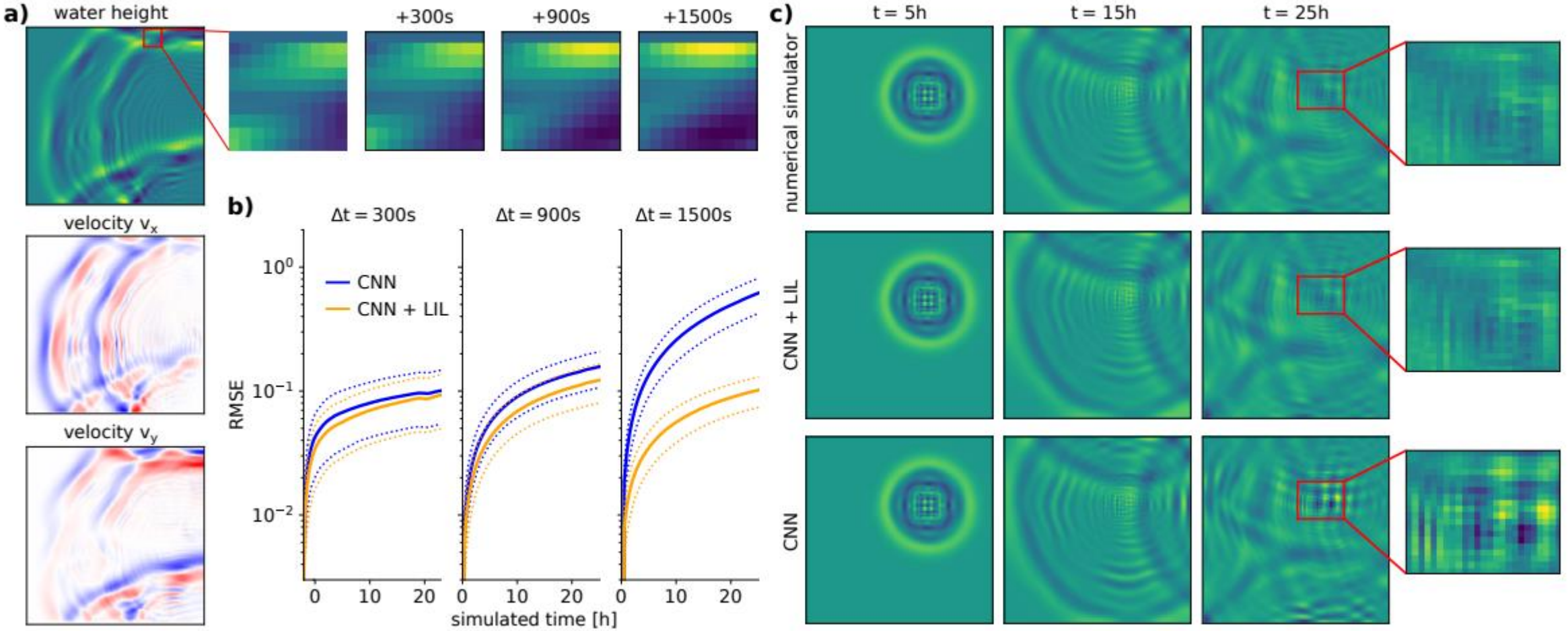
But A, b are data-dependent!

Solution: built-in diagonal dominance

$$\mathrm{diag}_0(A_\theta) = \sum_{\sigma \neq 0} |\mathrm{diag}_\sigma(A_\theta)| + \exp((\mathbf{M}_\theta)_k)$$

- Main diagonal defined as row-wise sum of absolute values of other entries, plus an exponential function
- Ensures convergence of Gauss-Seidel, and many other iterative solvers

LILs Are Accurate and Stable for Larger Time Steps



Exploiting Known PDE Symmetries

For our neural network $f: x \to y$

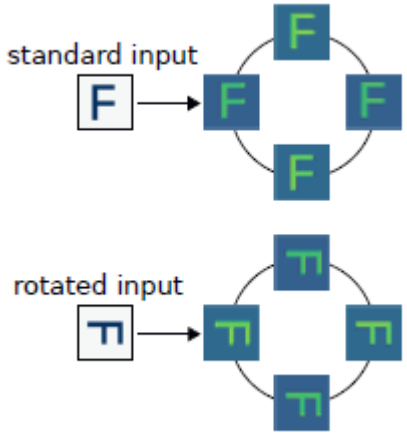
We want the equivariance relation

$$f(\mathcal{T}_g(x)) = \mathcal{T}_g(f(x))$$

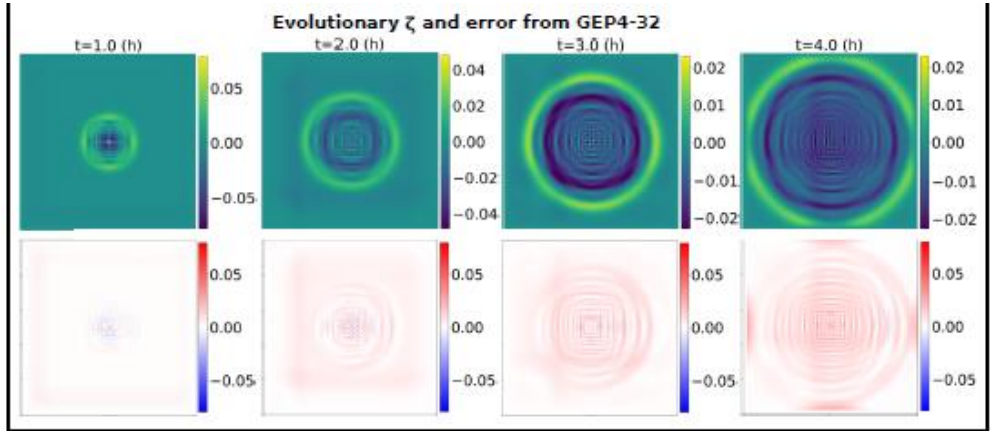
to hold for any input x and transformation g in a symmetry group.

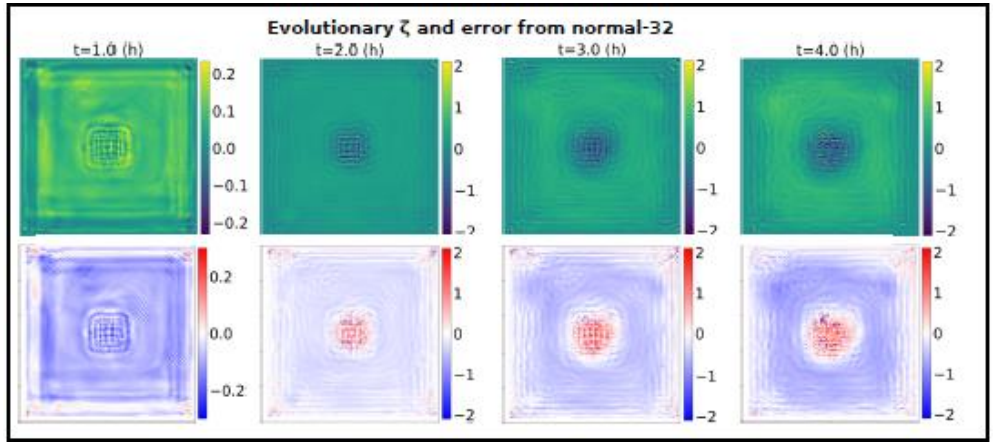
Example: for fully convolutional nets, translating the input translates that output.

But what about other symmetries?



Specialized Convolutional Layers (Cohen & Welling, 2016)
2D Shallow water equations





Yunfei Huang

Learn Process Rates/Fluxes, not Tendencies, to Enforce Conservation Laws

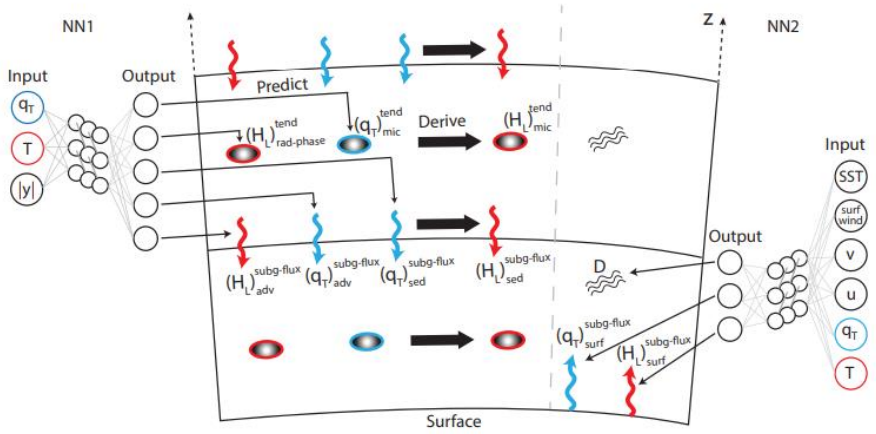
Use of neural networks for stable, accurate and physically consistent parameterization of subgrid atmospheric processes with good performance at reduced precision

Janni Yuval[1], Paul A. O'Gorman[1], and Chris N. Hill[1]

[1]Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, USA

Abstract

A promising approach to improve climate-model simulations is to replace traditional subgrid parameterizations based on simplified physical models by machine learning algorithms that are data-driven. However, neural networks (NNs) often lead to instabilities and climate drift when coupled to an atmospheric model. Here we learn an NN parameterization from a high-resolution atmospheric simulation in an idealized domain by coarse graining the model equations and output. The NN parameterization has a structure that ensures physical constraints are respected, and it leads to stable simulations that replicate the climate of the high-resolution simulation with similar accuracy to a successful random-forest parameterization while needing far less memory. We find that the simulations are stable for a variety of NN architectures and horizontal resolutions, and that an NN with substantially reduced numerical precision could decrease computational costs without affecting the quality of simulations.



Potential and Limitations of Machine Learning for Modeling Warm-Rain Cloud Microphysical Processes

Axel Seifert[1] and Stephan Rasp[2]

[1]Deutscher Wetterdienst, Offenbach, Germany, [2]TU München, Munich, Germany

Abstract The use of machine learning based on neural networks for cloud microphysical parameterizations is investigated. As an example, we use the warm-rain formation by collision-coalescence, that is, the parameterization of autoconversion, accretion, and self-collection of droplets in a two-moment framework. Benchmark solutions of the kinetic collection equations are performed using a Monte Carlo superdroplet algorithm. The superdroplet method provides reliable but noisy estimates of the warm-rain process rates. For each process rate, a neural network is trained using standard machine learning techniques. The resulting models make skillful predictions for the process rates when compared to the testing data. However, when solving the ordinary differential equations, the solutions are not as good as those of an established warm-rain parameterization. This deficiency can be seen as a limitation of the machine learning methods that are applied, but at the same time, it points toward a fundamental ill-posedness of the commonly used two-moment warm-rain schemes. More advanced machine learning methods that include a notion of time derivatives, therefore, have the potential to overcome these problems.

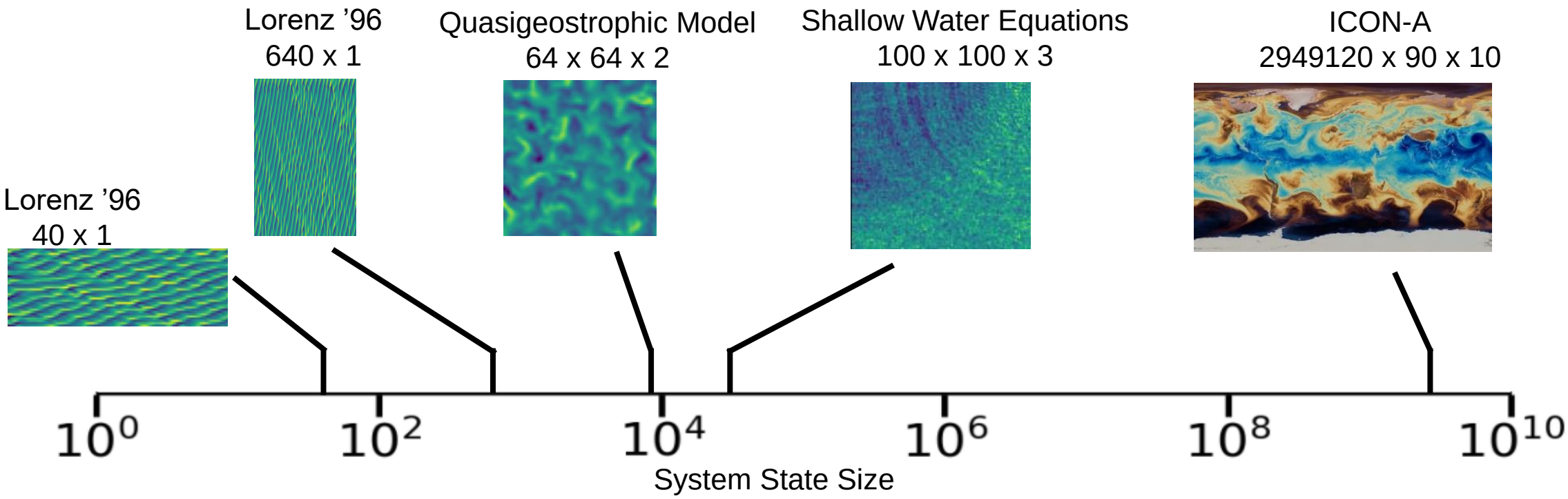
$$\frac{dL_c}{dt} = -AU - AC,$$

$$\frac{dL_r}{dt} = +AU + AC,$$

$$\frac{dN_c}{dt} = -2AU_N - AC_N - SC_c = -\frac{2}{x_*}AU - \frac{1}{\bar{x}_c}AC - SC_c,$$

$$\frac{dN_r}{dt} = +AU_N + AC_N - SC_r = +\frac{1}{x_*}AU - SC_r,$$

Scaling up further?



New technical problems: data flow, specialized distributed computations

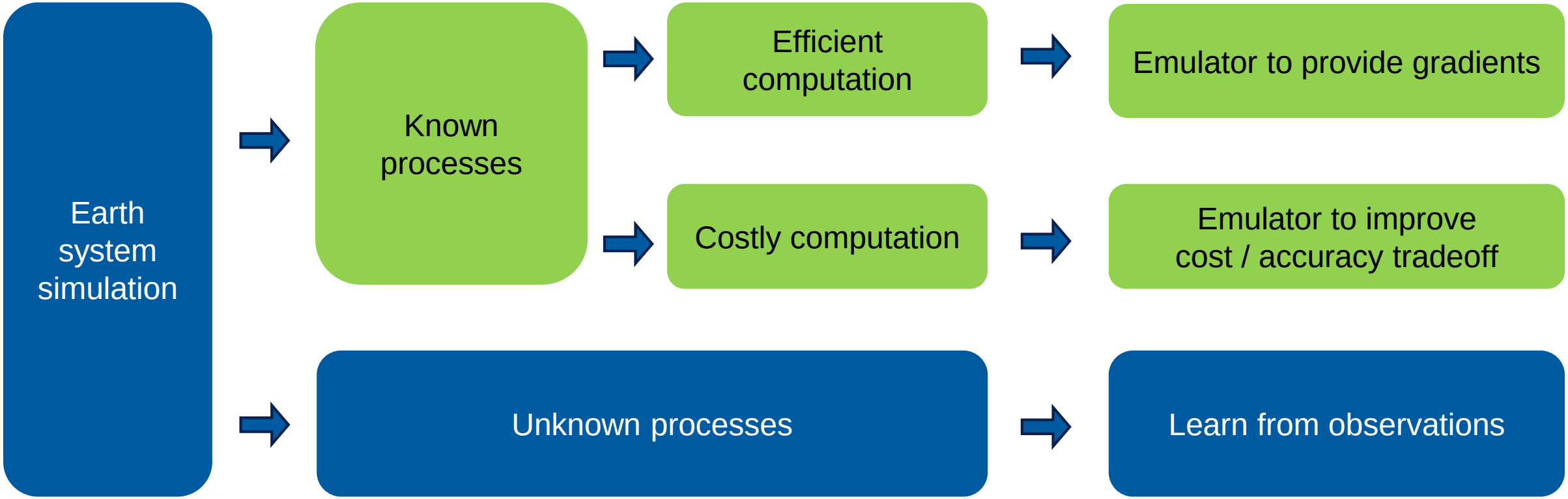
Outline

1 Deep Differentiable Emulators for Inverse Problems in the Geosciences

2 Hybrid Emulators

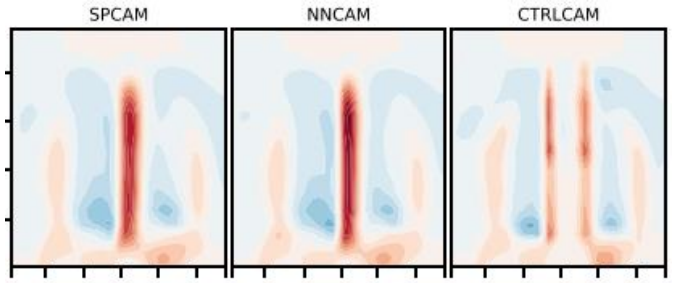
3 ML Models as Earth System Model Components

ML Models for Specific Processes in Earth System Simulations

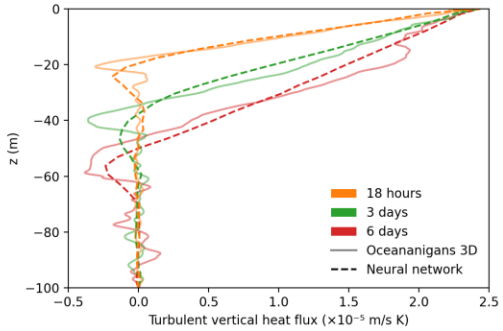


Parameterization Learning with ML: State of the Art

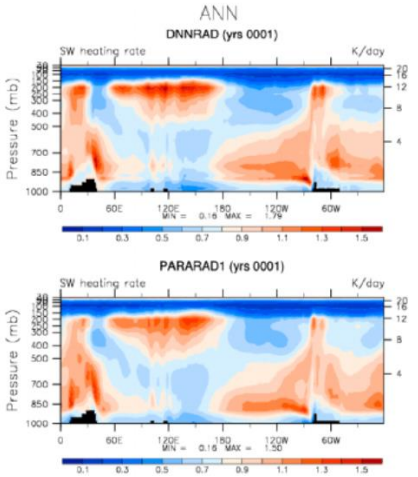
Superparameterized convection/radiation
Rasp et al., 2018



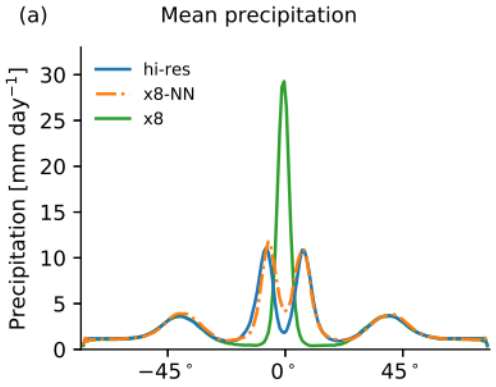
Turbuluent heat flux
Ramadhan et al., 2019



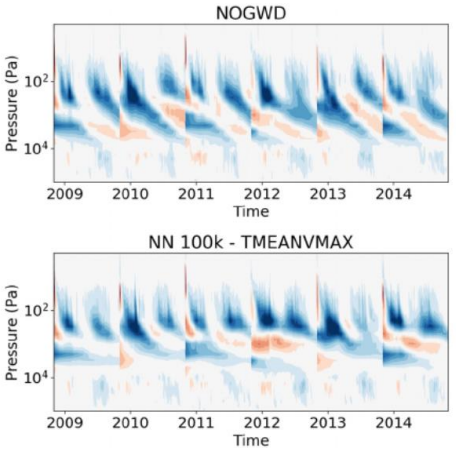
Radiation
Pal et al., 2019



Moist Convection
Yuval et al., 2021

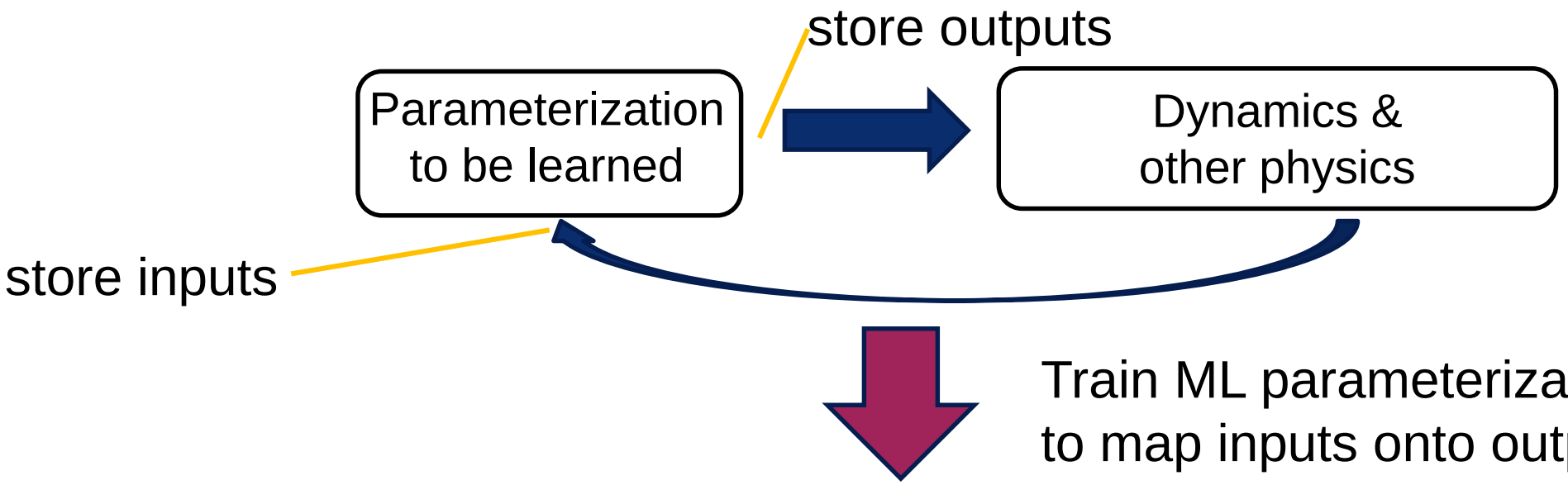


Nonorographic gravity wave drag
Chantry et al., 2021



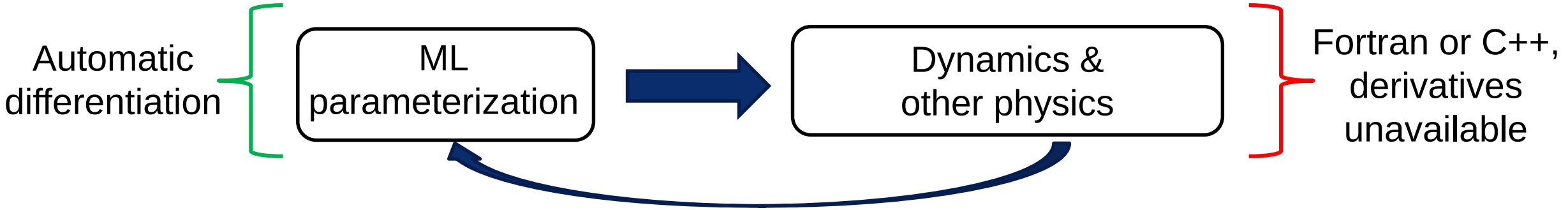
Common theme: good performance on 1-step updates doesn't guarantee long term accuracy or stability!

The “Coupling Problem”



Train ML parameterization **offline** to map inputs onto outputs

We can couple the trained ML parameterization to the full model…

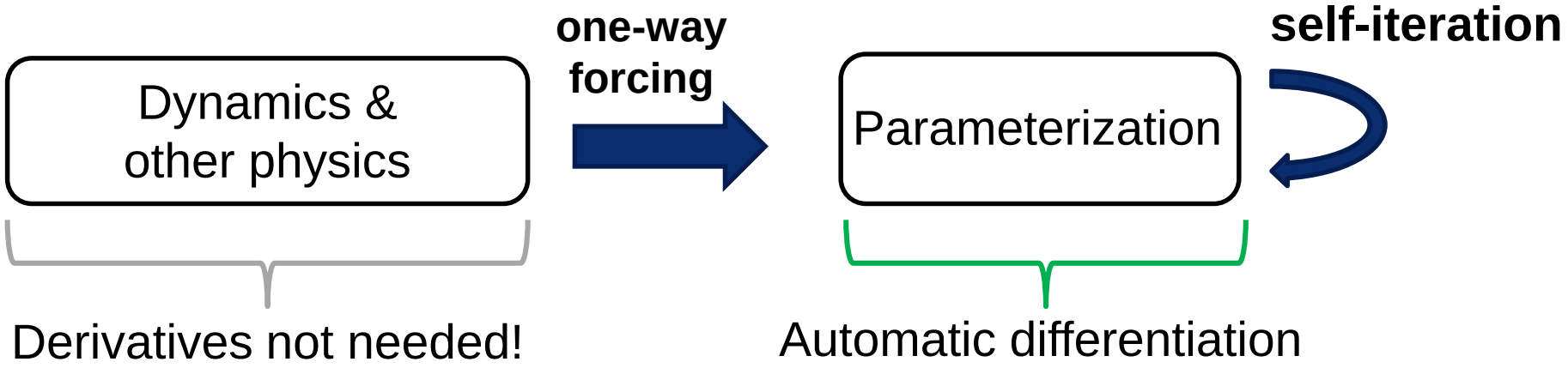


..but we can`t couple during training!

Single column & box models

Key idea: local data, no horizontal motion, fixed forcing / boundary conditions

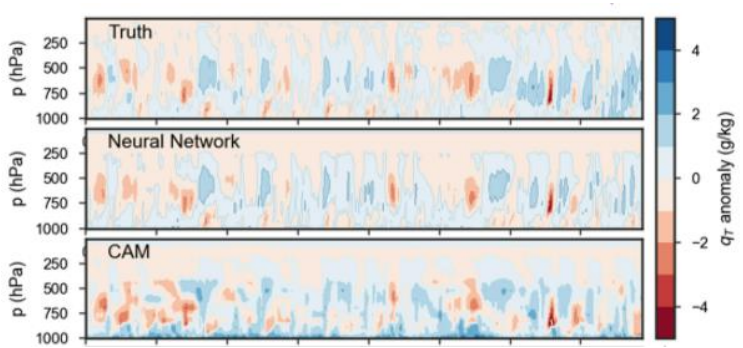
By avoiding two-way coupling, we can easily incorporate multiple time steps during training



Self-iteration during training increases accuracy and stability!

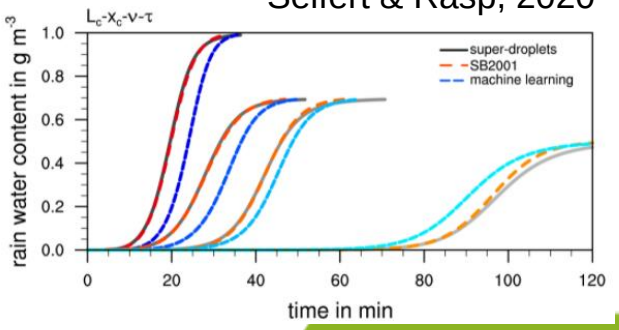
Convection / radiation

Brenowitz & Bretherton, 2018



Warm rain microphysics

Seifert & Rasp, 2020



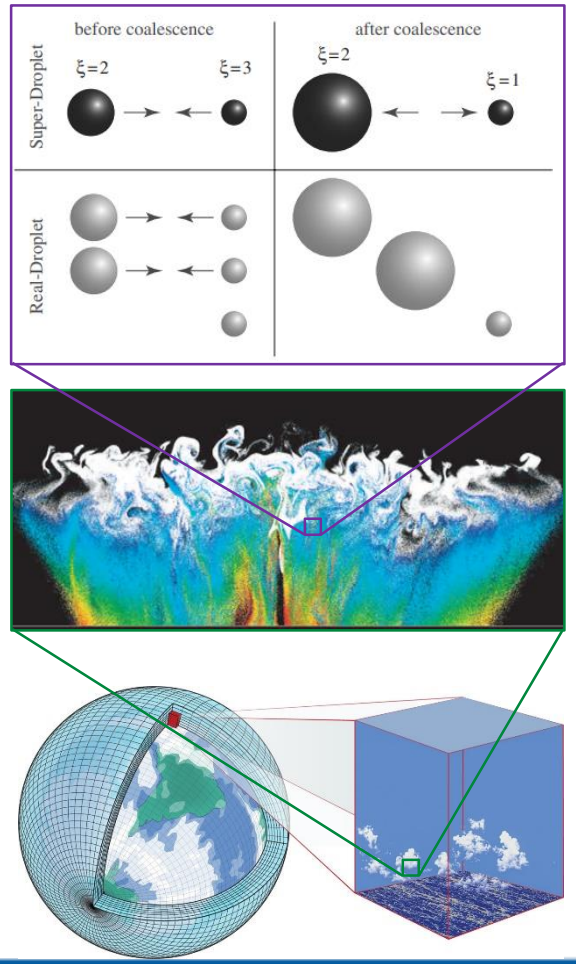
Learning Bulk-moment Microphysics Consistent with Droplet Simulations

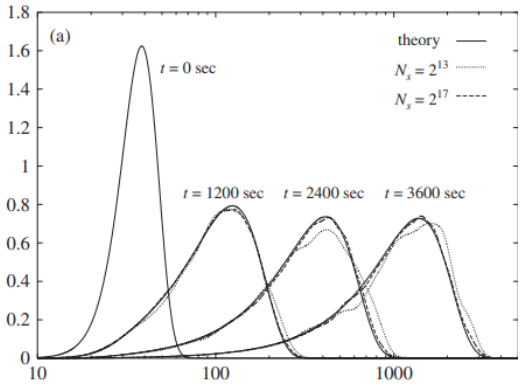
Goal: realistic cloud simulations for accurate climate and weather prediction

Challenge: droplet-based simulations are accurate, but inefficient

Solution: ML model trained to predict statistics of droplet size

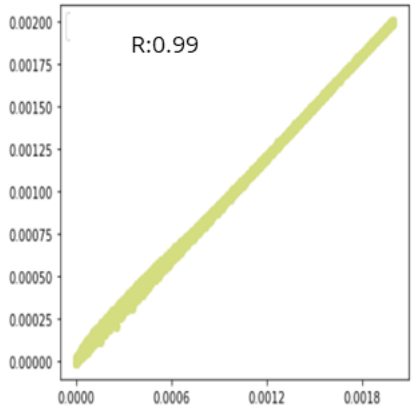
Hierarchy of spatial scales





Accurate prediction of rain water mass

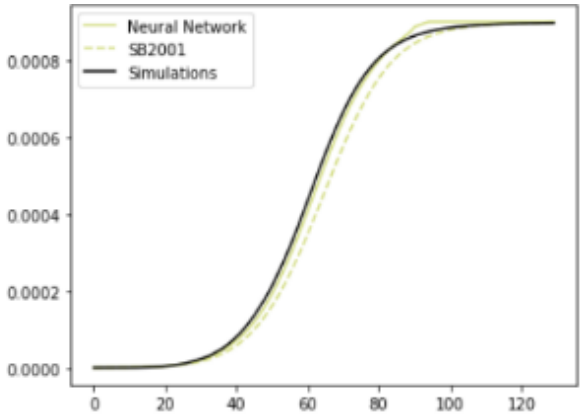
Prediction



True rain mass

Accurate cloud → rain transition

Rain mass



Time (min)

Next steps: coupling to ICON, spatial structure, new hydrometeors, aerosols

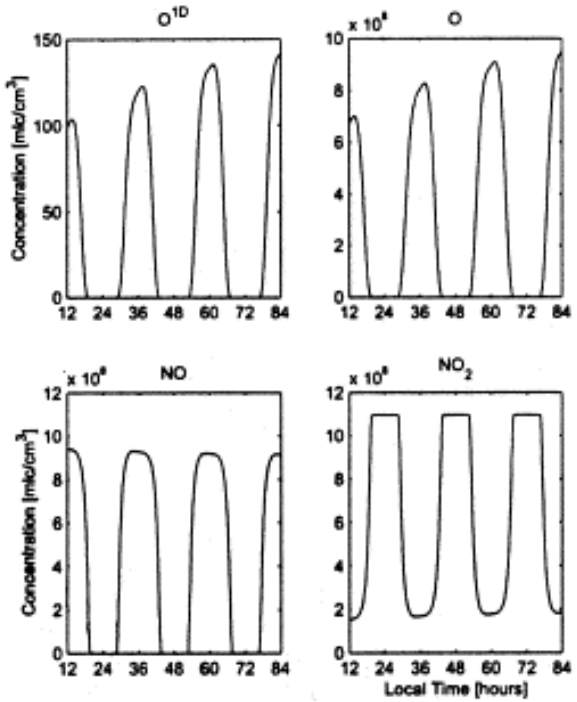
Shivani Sharma

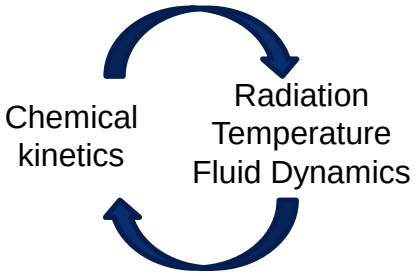
ML-accelerated Time Stepping for Stiff Chemical Systems

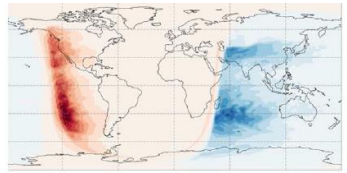
$O_2 + O^1D \rightarrow O^3P + O_2$
$O_2 + O^3P + M \rightarrow O_3 + M$
$O_3 + O^3P \rightarrow 2O_2$
$N_2 + O^1D \rightarrow O^3P + N_2$
$OH + O_3 \rightarrow HO_2 + O_2$
$HO_2 + O_3 \rightarrow OH + 2O_2$
$HO_2 + OH \rightarrow H_2O + O_2$
$H_2O + O^1D \rightarrow 2OH$
$N_2O + O^1D \rightarrow O^3P + N_2O$
$N_2O + O^1D \rightarrow 2NO$
$NO + O_3 \rightarrow NO_2 + O_2$
$NO_2 + O^3P \rightarrow NO + O_2$
$NO_2 + O_3 \rightarrow NO_3 + O_2$
$NO_3 + NO_2 \rightarrow N_2O_5$
$NO_3 + NO_2 + M \rightarrow N_2O_5 + M$
$HNO_3 + OH \rightarrow H_2O + NO_3$

Photolysis reactions

$O_2 + h\nu \rightarrow O^3P + O^3P$
$O_3 + h\nu \rightarrow O^1D + O_2$
$O_3 + h\nu \rightarrow O^3P + O_2$
$NO_2 + h\nu \rightarrow O^3P + NO$
$NO_3 + h\nu \rightarrow O^3P + NO_2$
$N_2O_5 + h\nu \rightarrow NO_3 + NO_2$
$HNO_3 + h\nu \rightarrow NO_2 + OH$







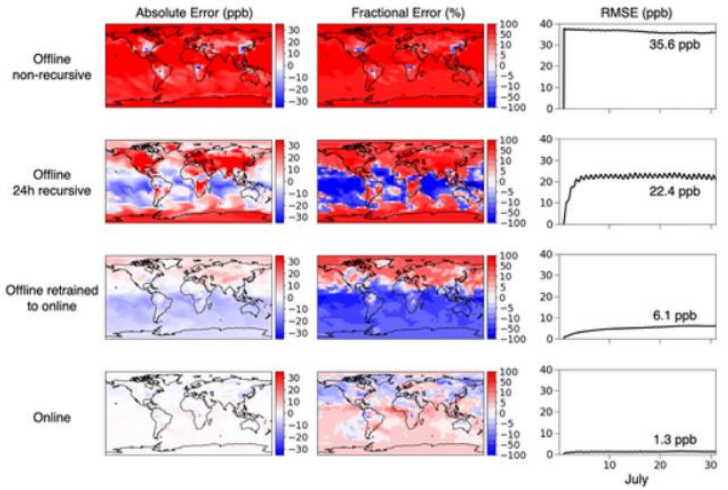
$d[HNO_3]/dt$

Rosenbrock Solver (s-stages)

$$y_{n+1} = y_n + \sum_{i=1}^{s} b_i k_i,$$

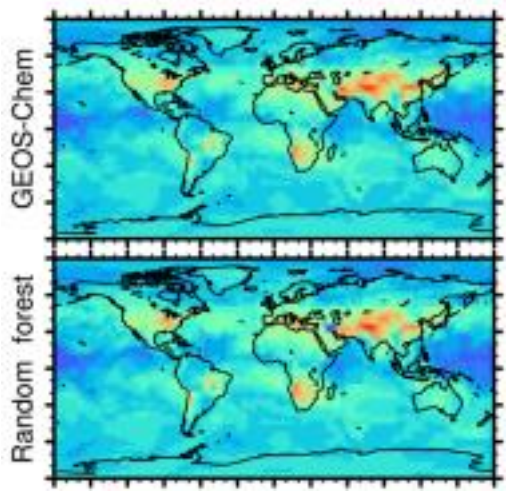
$$k_i = \tau f\left(y_n + \sum_{j=1}^{i-1} \alpha_{ij} k_j\right) + \tau J \sum_{j=1}^{i} \gamma_{ij} k_j.$$

s linear systems to be solved at each time step



Kelp et al., 2022

Keller & Evans, 2019



Summary

- Emulators can provide missing derivatives for solving inverse problems
- Accuracy can be improved by physical or symmetry constraints, or by mimicing numerical solver components
- For large Earth system simulations, individual processes can be emulated
- One-way coupling during training can (sometimes) improve accuracy in fully coupled mode
- We’re still lacking many important tools and components for building and exploiting fully differentiable simulations