Hybrid Deep Learning and Physics Models for Lake Water Quality Forecasting

Journées annuelles de la fédération Occimath 2026

Statistique et Optimisation

David Métivier[1] Céline Casenave[1] Brigitte Vinçon-Leite[2]

Interns: Ali Elhishi & Mathieu Le Séac'h

[1]UMR MISTEA, INRAE

[2]LEESU, École nationale des ponts et chaussées, Université Paris-Est Créteil

INRAE

science for people, life & earth

MISTEA

Mathematics, Informatics and Statistics for Environment and Agronomy

This talk

Goal: Time series forecasting
Application: Lake temperature and water quality forecasting

Physics based models

- Partial/Ordinary Differential Equations (PDEs/ODEs)
- Expert models

This talk

Goal: Time series forecasting
Application: Lake temperature and water quality forecasting

Physics based models

- Partial/Ordinary Differential Equations (PDEs/ODEs)
- Expert models

Data based methods

- Random Forest, MLP
- Auto-regressive models: ARIMA, SARIMA, etc.
- Deep learning: RNN, LSTM, Transformer, etc.

This talk

Goal: Time series forecasting
Application: Lake temperature and water quality forecasting

Physics based models

- Partial/Ordinary Differential Equations (PDEs/ODEs)
- Expert models

Data based methods

- Random Forest, MLP
- Auto-regressive models: ARIMA, SARIMA, etc.
- Deep learning: RNN, LSTM, Transformer, etc.

$\rightarrow$ Hybrid models: Physics-informed + data-driven
e.g. Physics-Informed Neural Networks (PINNs), Universal Differential Equations (UDEs)

This talk

Goal: Time series forecasting
Application: Lake temperature and water quality forecasting

Physics based models

- Partial/Ordinary Differential Equations (PDEs/ODEs)
- Expert models

Data based methods

- Random Forest, MLP
- Auto-regressive models: ARIMA, SARIMA, etc.
- Deep learning: RNN, LSTM, Transformer, etc.

$\rightarrow$ Hybrid models: Physics-informed + data-driven
e.g. Physics-Informed Neural Networks (PINNs), Universal Differential Equations (UDEs)
+ Discussion on differentiable programming and optimization of models

Objective

→ "Simple" (easy to deploy) hybrid IA-Physics model for lake temperature and variables related to water quality (chlorophyll, oxygen, etc.)

\+ predicting chlorophyll concentration

\++ anticipating algal blooms

→ Oxygen depletion in the lake

→ Toxic

Algal bloom in Lake Créteil.

Objective

→ "Simple" (easy to deploy) hybrid IA-Physics model for lake temperature and variables related to water quality (chlorophyll, oxygen, etc.)

\+ predicting chlorophyll concentration

\++ anticipating algal blooms

→ Oxygen depletion in the lake

→ Toxic

Algal bloom in Lake Créteil.

Factors promoting algal blooms

- Temperature increase
- Lake stratification → $\Delta_{T_\mathrm{w}} = (T_\mathrm{w}^{(\mathrm{surface})} - T_\mathrm{w}^{(\mathrm{bottom})})$

Objective

→ "Simple" (easy to deploy) hybrid IA-Physics model for lake temperature and variables related to water quality (chlorophyll, oxygen, etc.)

\+ predicting chlorophyll concentration

\++ anticipating algal blooms

→ Oxygen depletion in the lake

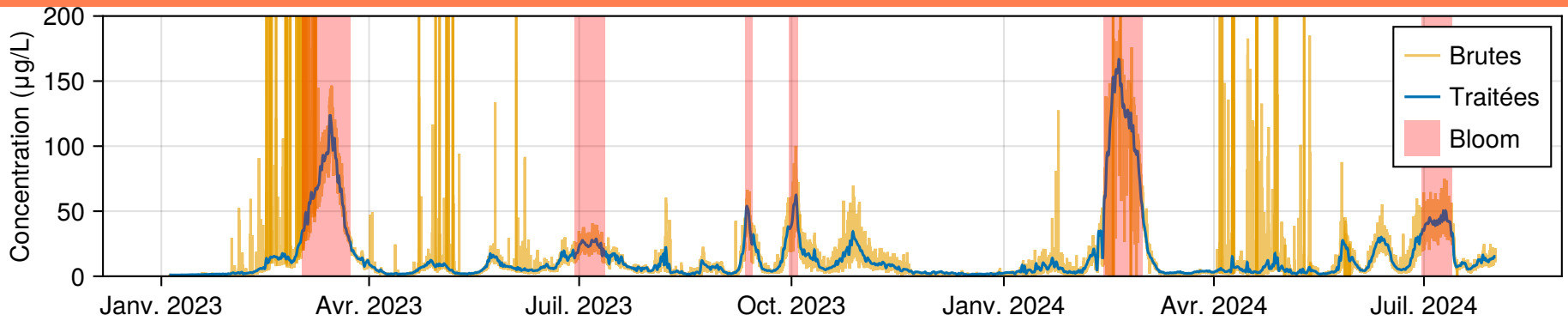
→ Toxic

Algal bloom in Lake Créteil.

Factors promoting algal blooms

- Temperature increase
- Lake stratification → $\Delta_{T_\mathrm{w}} = (T_\mathrm{w}^{(\mathrm{surface})} - T_\mathrm{w}^{(\mathrm{bottom})})$
- Increase in nutrients (nitrogen and phosphorus) in the lake → not modeled for now

Bloom in data

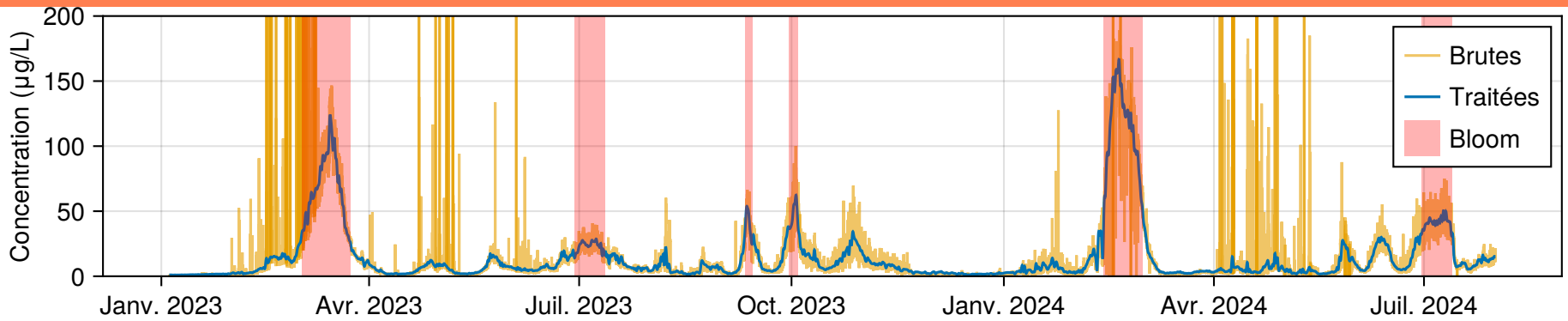


Chlorophyll concentration.

Variable	**Meaning**	**Depth** [m]	**Units**
$T^{(\mathrm{air})} = T_a$	Air temperature	-	°C
T_{w}	Water temperature	0.5, 1.5, 2.5, 3.5, 4.5	°C
Chl	Chlorophyll concentration	1.50	$\mu\mathrm{g\,L^{-1}}$
Cyano	Cyanobacteria concentration	1.50	$\mu\mathrm{g\,L^{-1}}$
O2	Oxygen saturation	1.50	%

Roughly 18 months of data every 15 minutes $\rightarrow$ aggregate to hourly

Bloom in data



Chlorophyll concentration.

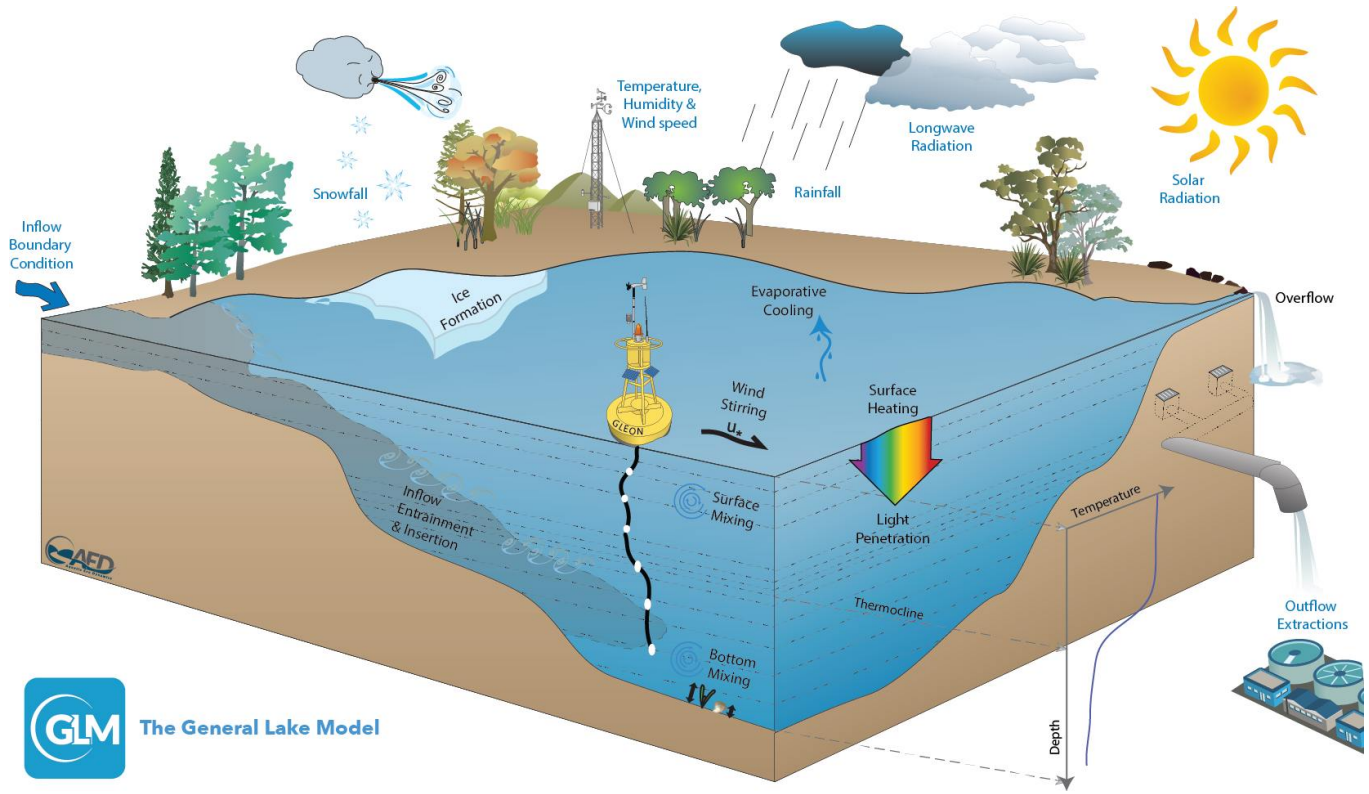
Variable	**Meaning**	**Depth** [m]	**Units**
$T^{(\mathrm{air})} = T_a$	Air temperature	-	°C
T_{w}	Water temperature	0.5, 1.5, 2.5, 3.5, 4.5	°C
Chl	Chlorophyll concentration	1.50	$\mu\mathrm{g\,L}^{-1}$
Cyano	Cyanobacteria concentration	1.50	$\mu\mathrm{g\,L}^{-1}$
O2	Oxygen saturation	1.50	%

Roughly 18 months of data every 15 minutes $\rightarrow$ aggregate to hourly

$\rightarrow$ **This talk:** Water temperature forecasting at different depths with minimal input (air temperature only).

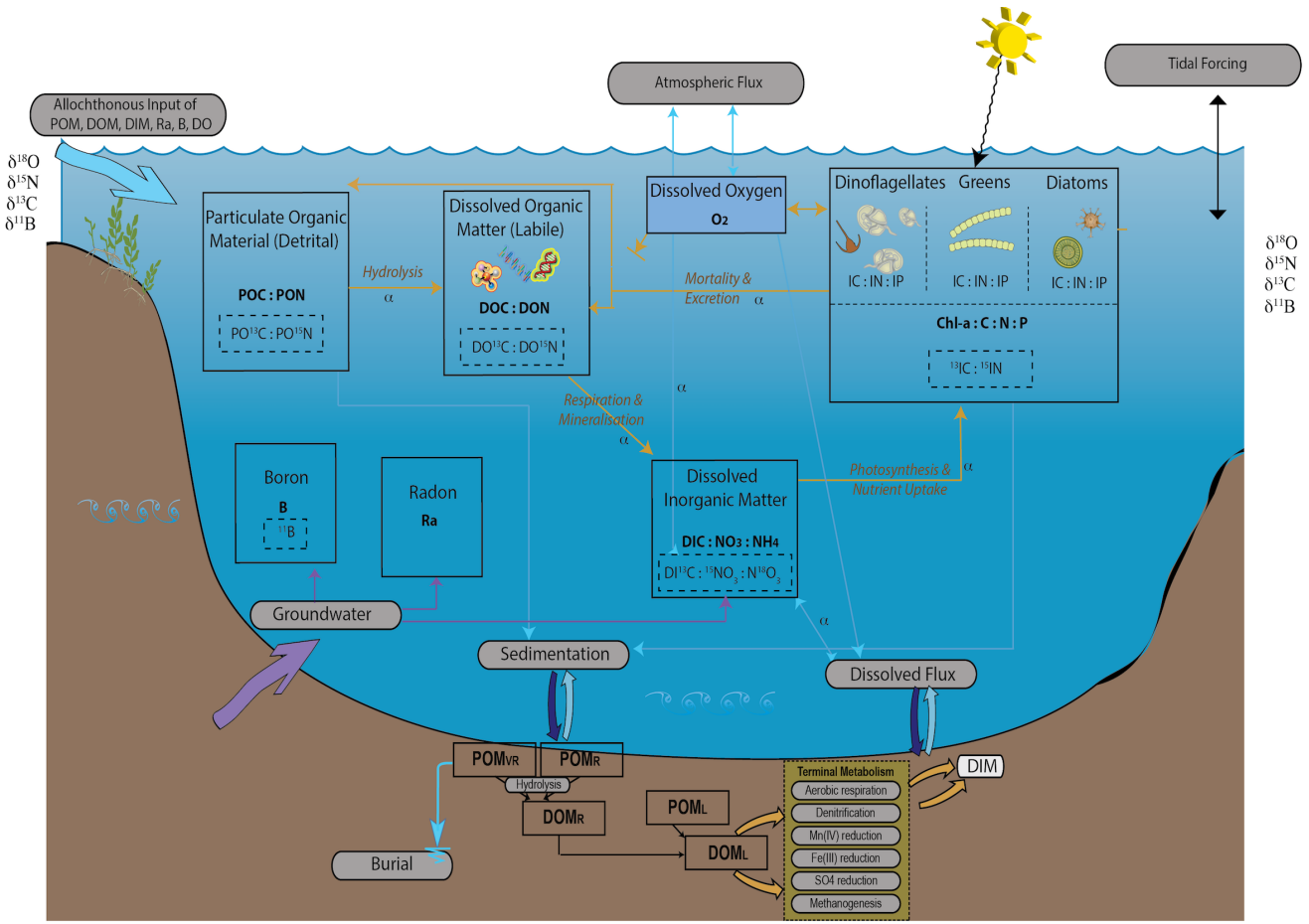
Process-based: Complete physics models

General Lake Model (GLM): 1D hydrodynamics model PDE based



Process-based: Complete physics models

General Lake Model (GLM): 1D hydrodynamics model PDE based
Aquatic Eco-Dynamics (AED): also PDE based



Isotope-enabled carbon & nutrient ecosystem model

Process-based: Complete physics models

General Lake Model (GLM): 1D hydrodynamics model PDE based
Aquatic Eco-Dynamics (AED): also PDE based

Pros & Cons

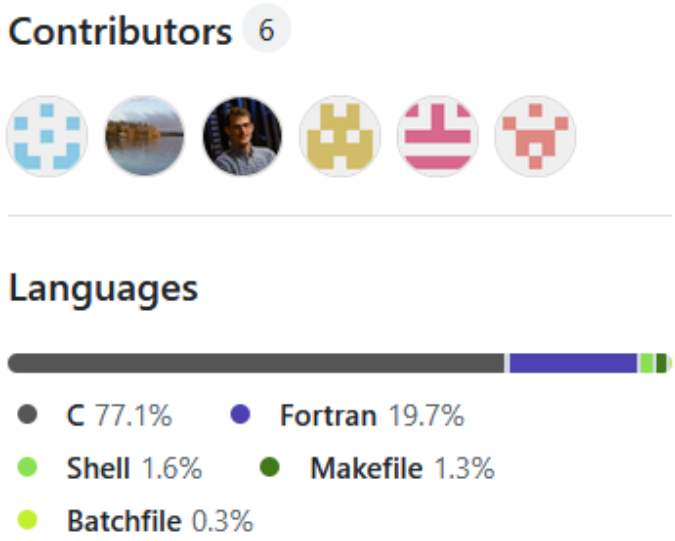
+ Interpretable and physics based
- Hard to calibrate (physical constant, lake-specific parameters, bathymetry)
- Require a lot of inputs hourly air temperature, shortwave and longwave radiation, relative humidity, wind speed, and precipitation!
- Hard to couple GLM and AED
- Not 3D and already complex

Process-based: Complete physics models

General Lake Model (GLM): 1D hydrodynamics model PDE based
Aquatic Eco-Dynamics (AED): also PDE based

Pros & Cons

+ Interpretable and physics based
- Hard to calibrate (physical constant, lake-specific parameters, bathymetry)
- Require a lot of inputs hourly air temperature, shortwave and longwave radiation, relative humidity, wind speed, and precipitation!
- Hard to couple GLM and AED
- Not 3D and already complex



From GLM GitHub

Process-based: Reduced models (MinPhy)

Surface $T_{\mathrm{w}}^{(0.5)}$ — Air2Temp (Piccolroaz et al., 2013):

$$\frac{\mathrm{d}T_{\mathrm{w}}^{(0.5)}}{\mathrm{d}t} = \frac{\varphi_1 + \varphi_2(T_a - T_{\mathrm{w}}^{(0.5)}) + \varphi_3 T_{\mathrm{w}}^{(0.5)}}{\exp(-\Delta_T/\varphi_4)} =: f_{\varphi}^{(0.5)}(T_{\mathrm{w}}^{(0.5)}, T_a)$$

Process-based: Reduced models (MinPhy)

Surface $T_{\mathrm{w}}^{(0.5)}$ — Air2Temp (Piccolroaz et al., 2013):

$$\frac{\mathrm{d}T_{\mathrm{w}}^{(0.5)}}{\mathrm{d}t} = \frac{\varphi_1 + \varphi_2(T_a - T_{\mathrm{w}}^{(0.5)}) + \varphi_3 T_{\mathrm{w}}^{(0.5)}}{\exp(-\Delta_T/\varphi_4)} =: f_{\varphi}^{(0.5)}(T_{\mathrm{w}}^{(0.5)}, T_a)$$

Depths $z \in \{1.5, 2.5, 3.5, 4.5\}\,\mathrm{m}$ — discretized advection–diffusion:

$$\frac{\mathrm{d}T_{\mathrm{w}}^{(z)}}{\mathrm{d}t} = \underbrace{A_z \frac{T_{\mathrm{w}}^{(z+1)} - 2T_{\mathrm{w}}^{(z)} + T_{\mathrm{w}}^{(z-1)}}{(\Delta z)^2}}_{\text{diffusion}} - \underbrace{w_z \frac{T_{\mathrm{w}}^{(z+1)} - T_{\mathrm{w}}^{(z-1)}}{2\Delta z}}_{\text{advection}}$$

\+ Only air temperature as input; 12 parameters total

\+ Trainable by gradient descent (differentiable programming)

± Qualitative but not quantitative

Data-driven: Machine Learning and Deep Learning

Different types of models (Random forest, MLP, RNN, etc.)

+ Learn with different source of training data
+ Can learn very complex dynamic
+ Beneficiate from AI boom

Data-driven: Machine Learning and Deep Learning

Different types of models (Random forest, MLP, RNN, etc.)

- \+ Learn with different source of training data
- \+ Can learn very complex dynamic
- \+ Beneficiate from AI boom
- \- Times series with seasonal patterns, rare events, changing environment
- \- Do not know physics
- \- Sensitive to quantity and quality of data
- \- Interpretability

Data-driven: Machine Learning and Deep Learning

Different types of models (Random forest, MLP, RNN, etc.)

+ Learn with different source of training data
+ Can learn very complex dynamic
+ Beneficiate from AI boom
- Times series with seasonal patterns, rare events, changing environment
- Do not know physics
- Sensitive to quantity and quality of data
- Interpretability
- Not state of the art for lake temperature forecast

Data-driven: Machine Learning and Deep Learning

Different types of models (Random forest, MLP, RNN, etc.)

+ Learn with different source of training data
+ Can learn very complex dynamic
+ Beneficiate from AI boom
- Times series with seasonal patterns, rare events, changing environment
- Do not know physics
- Sensitive to quantity and quality of data
- Interpretability
- Not state of the art for lake temperature forecast

Journal of Hydrology

journal homepage: www.elsevier.com/locate/jhydrol

Research papers

Forecasting surface water temperature in lakes: A comparison of approaches

Senlin Zhu[a,b,*], Mariusz Ptak[c], Zaher Mundher Yaseen[d], Jiangyu Dai[a,*], Bellie Sivakumar[e]

[a] State Key Laboratory of Hydrology-Water Resources and Hydraulic Engineering, Nanjing Hydraulic Research Institute, Nanjing 210029, China
[b] Department of Civil and Environmental Engineering, Cullen College of Engineering, University of Houston, Houston 77204, USA

Deep Learning for Time Series: RNN

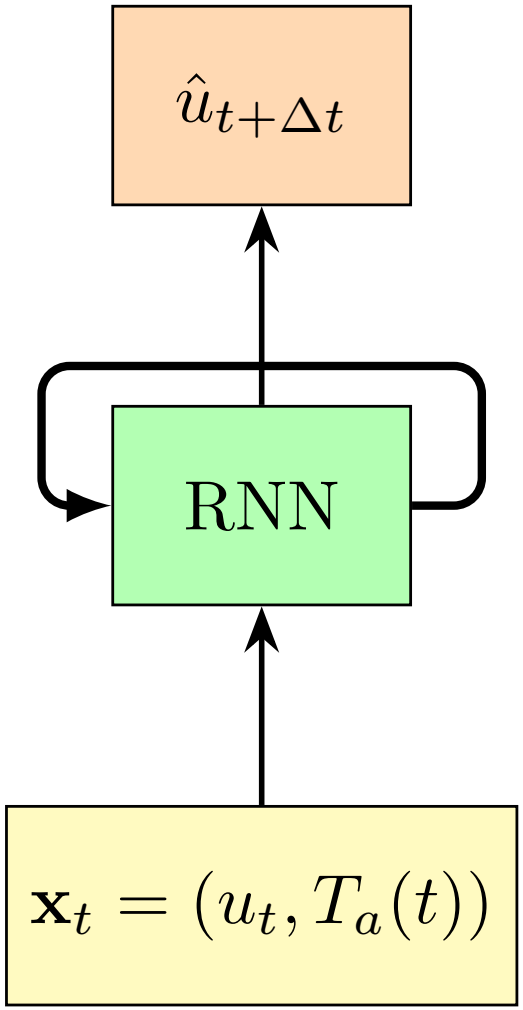
Hidden state $\mathbf{h}_t \in \mathbb{R}^m$ updated at each step:

$$\mathbf{h}_t = \Phi(\mathbf{W}_x \mathbf{x}_t + \mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{b}_h)$$

$$\hat{u}_{t+\Delta t} = \psi(\mathbf{W}_y \mathbf{h}_t + \mathbf{b}_y)$$

where W are matrices, $\mathbf{b}$ are bias vectors and Φ, ψ are activation functions.

+ Variable-length input, shared parameters
+ Autoregressive forecasting: feed output as next input
- Vanishing/exploding gradients
- Fix time step Δt; not adapted for irregular sampling



Deep Learning for Time Series: RNN

Hidden state $\mathbf{h}_t \in \mathbb{R}^m$ updated at each step:

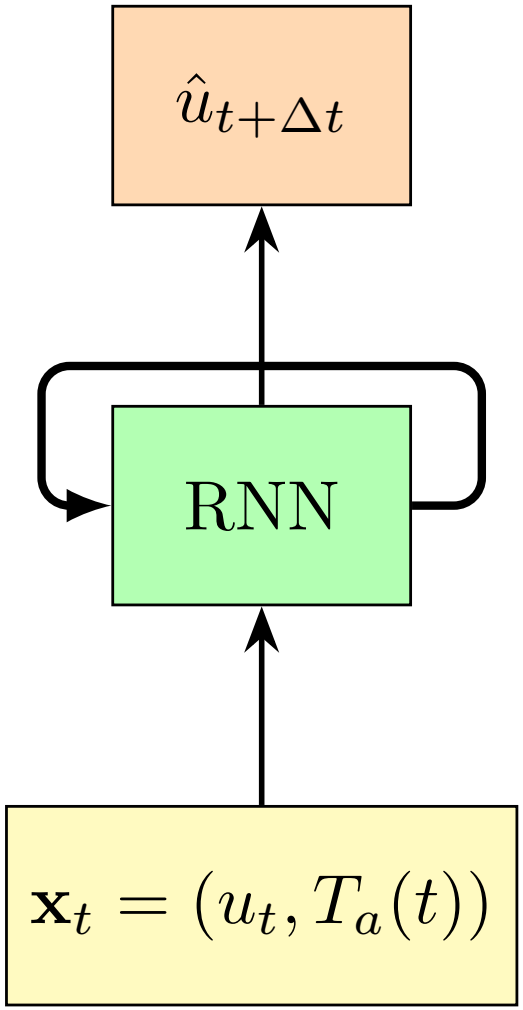
$$\mathbf{h}_t = \Phi(\mathbf{W}_x \mathbf{x}_t + \mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{b}_h)$$

$$\hat{u}_{t+\Delta t} = \psi(\mathbf{W}_y \mathbf{h}_t + \mathbf{b}_y)$$

where W are matrices, $\mathbf{b}$ are bias vectors and Φ, ψ are activation functions.

+ Variable-length input, shared parameters
+ Autoregressive forecasting: feed output as next input
- Vanishing/exploding gradients
- Fix time step Δt; not adapted for irregular sampling

LSTM: improved RNN



Deep Learning for Time Series: Transformers

Vaswani et al. (2017) "*Attention Is All You Need*".

Idea Replace recurrence with **self-attention**: every time step attends to all others simultaneously

+ Captures long-range dependencies without sequential computation
+ Highly parallelisable $\rightarrow$ faster training on GPU
+ State of the art in LLMs and increasingly in time series
- Requires large datasets + computational resources
- Less natural inductive bias for physical dynamics

Hybrid models

Neural Networks + physics models

- \+ Best of both worlds
- \+ More robust and need less data
- \+ Satisfy better physics
- ? Can improve reduced models to make them quantitative
- ? Times series with seasonal patterns, rare events, changing environment
- $\pm$ Interpretability
- \- **How?** $\rightarrow$ Under active research
- \- Harder to train

Physics Informed Neural Networks (PINNs) — PDE form

$u_\theta(x,t) = \mathrm{NN}_\theta(x,t)$ approximates the solution of a PDE on $\Omega \times [t_0, t_f]$:
Example — heat equation $\partial_t u = \alpha\, \partial_{xx} u$

Physics Informed Neural Networks (PINNs) — PDE form

$u_\theta(x,t) = \mathrm{NN}_\theta(x,t)$ approximates the solution of a PDE on $\Omega \times [t_0, t_f]$:
Example — heat equation $\partial_t u = \alpha\, \partial_{xx} u$

$$\mathcal{P}_u(x,t) = \frac{\partial u_\theta}{\partial t}(x,t) - \alpha\, \frac{\partial^2 u_\theta}{\partial x^2}(x,t) = 0$$

Physics Informed Neural Networks (PINNs) — PDE form

$u_\theta(x,t) = \mathrm{NN}_\theta(x,t)$ approximates the solution of a PDE on $\Omega \times [t_0, t_f]$:
Example — heat equation $\partial_t u = \alpha\, \partial_{xx} u$

$$\mathcal{P}_u(x,t) = \frac{\partial u_\theta}{\partial t}(x,t) - \alpha\, \frac{\partial^2 u_\theta}{\partial x^2}(x,t) = 0$$

$$\mathcal{L}(\theta) = \underbrace{\frac{1}{N_c}\sum_{i=1}^{N_c} \left|\mathcal{P}_u\big(x_i, t_i,\, u_\theta(x_i,t_i)\big)\right|^2}_{\text{PDE residual at collocation points}} + \underbrace{\frac{1}{N_d}\sum_{i=1}^{N_d} \left|u_\theta(x_i,t_i) - u_i^{\mathrm{obs}}\right|^2}_{\text{data fit}} + \text{BC/IC}$$

Derivatives of u_θ w.r.t. inputs computed by **automatic differentiation**.

- \+ No PDE solver; handles complex geometries and high dimensions
- \+ Forward & inverse problems (e.g. identify α from data)

Physics Informed Neural Networks (PINNs) — ODE

For a time-series ODE $\mathcal{P}_u(t) = \dot{u}(t) - f(u,t) = 0$, the NN learns $u_\theta(t)$ on $[t_0, t_f]$:

$$\mathcal{L}(\theta) = \underbrace{\frac{1}{N_c}\sum_{i=1}^{N_c}\left|\dot{u}_\theta(t_i) - f(u_\theta(t_i), t_i)\right|^2}_{\text{ODE residual}} + \underbrace{\frac{1}{N_d}\sum_{i=1}^{N_d}\left|u_\theta(t_i) - u_i^{\text{obs}}\right|^2}_{\text{data fit}} + \text{IC}$$

+ Still no ODE solver needed
- Learns the solution $u_\theta(t)$ — fixed to interval $[t_0, t_f]$
- **Not suited for forecasting**: $u_\theta(t)$ unreliable beyond t_f

Neural ODE: Pure Data-Driven Dynamics

For a variable of interest $u(t)$ and an external forcing $g(t)$, learn the full dynamics with a NN:

$$\frac{\mathrm{d}u}{\mathrm{d}t}(t) = f_\theta^{(\mathrm{NN})}(u(t), g(t)), \qquad L(\theta) = \frac{1}{N}\sum_{j=1}^{N}\left(u_\theta(t_j) - u_{\mathrm{obs},j}\right)^2$$

Neural ODE: Pure Data-Driven Dynamics

For a variable of interest $u(t)$ and an external forcing $g(t)$, learn the full dynamics with a NN:

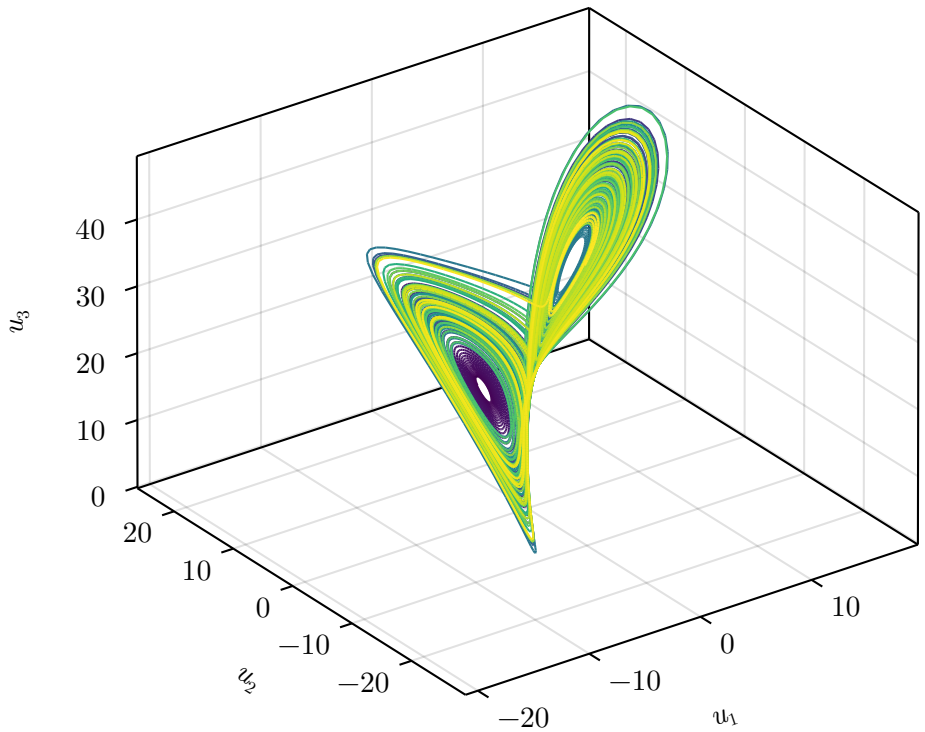
$$\frac{\mathrm{d}u}{\mathrm{d}t}(t) = f_\theta^{(\mathrm{NN})}(u(t), g(t)), \qquad L(\theta) = \frac{1}{N}\sum_{j=1}^{N}\left(u_\theta(t_j) - u_{\mathrm{obs},j}\right)^2$$

+ Flexible: can approximate complex dynamics from data
+ Continuous-time model, adapted for irregular sampling
- Does not exploit any available physical knowledge
- **No Universal approximation theorem for dynamics**
  - e.g. $\dot{u} = f_\theta^{(\mathrm{NN})}(u)$ cannot produce a periodic $u(t)$
    $\Rightarrow$ structural limitations on learnable dynamics

Example: Lorenz system

True dynamics (chaotic, 3 variables):

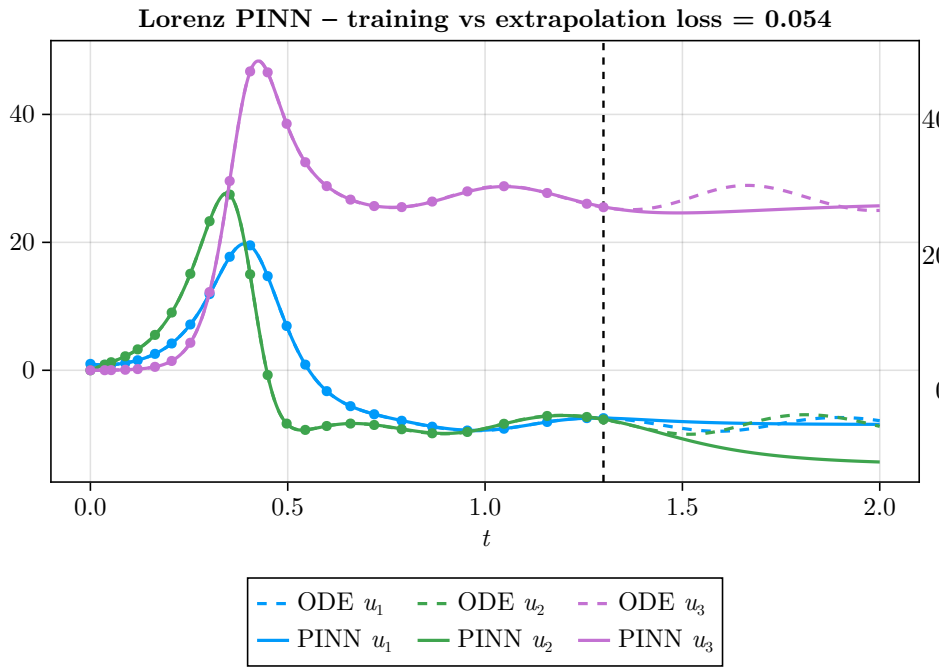
$$\begin{cases} \dot{u}_1 = \sigma\,(u_2 - u_1) \\ \dot{u}_2 = u_1\,(\rho - u_3) - u_2 \\ \dot{u}_3 = u_1\,u_2 - \beta\,u_3 \end{cases}$$

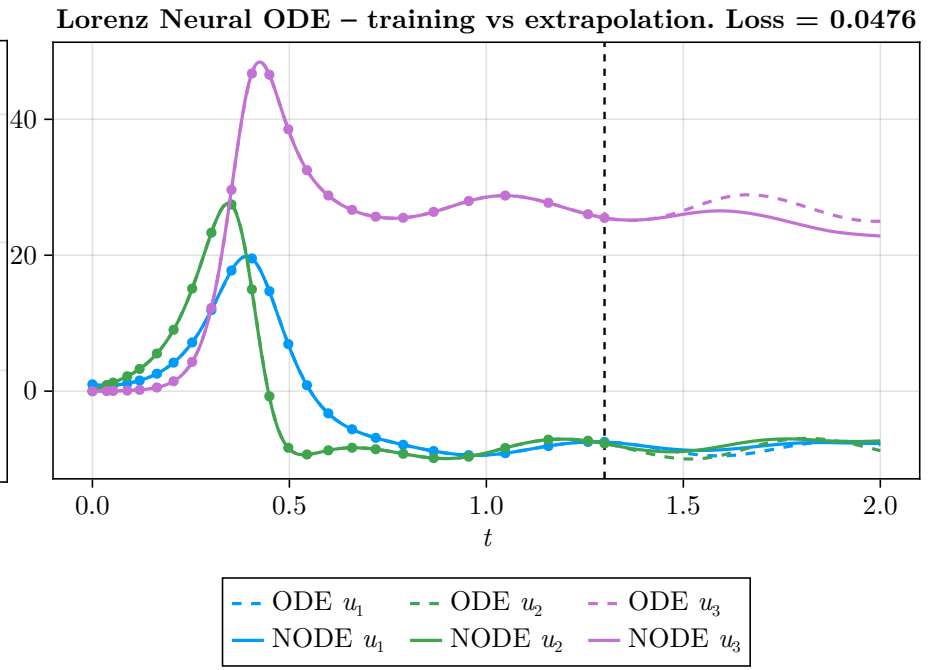


Example: Lorenz system

True dynamics (chaotic, 3 variables):

$$\begin{cases} \dot{u}_1 = \sigma\,(u_2 - u_1) \\ \dot{u}_2 = u_1\,(\rho - u_3) - u_2 \\ \dot{u}_3 = u_1\,u_2 - \beta\,u_3 \end{cases}$$





u_θ is very complex

f_θ is a simple polynomial

Universal Differential Equations (UDE)

Augment known physics with a Neural Network for the missing/uncertain part:

$$\frac{\mathrm{d}u}{\mathrm{d}t}(t) = \underbrace{f_\phi^{(\mathrm{Phy})}(u(t), g(t))}_{\text{known physics}} + \underbrace{f_\theta^{(\mathrm{NN})}(u(t), g(t))}_{\text{Neural Network}}$$

+ Physics structure guides the learning $\rightarrow$ works with less data
+ Already good results with a simple NN
+ More physics-consistent predictions (see density constraint later)
+ Adapted for forecasting: learns the dynamics, not the solution
- Training harder than plain NN (adjoint/sensitivity methods)
- No universal approximation theorem (inherited from NODE)

Different flavor of UDEs

Three coupling strategies between Phy and NN:

Different flavor of UDEs

Three coupling strategies between Phy and NN:

1. Additive UDE — independent contributions:

$$\frac{\mathrm{d}u}{\mathrm{d}t} = f_{\varphi}^{(\mathrm{Phy})}(u, T_a) + f_{\theta}^{(\mathrm{NN})}(u, T_a)$$

Different flavor of UDEs

Three coupling strategies between Phy and NN:

1. Additive UDE — independent contributions:

$$\frac{\mathrm{d}u}{\mathrm{d}t} = f_{\varphi}^{(\mathrm{Phy})}(u, T_a) + f_{\theta}^{(\mathrm{NN})}(u, T_a)$$

2. Additive Informed UDE — NN conditioned on physics output:

$$\frac{\mathrm{d}u}{\mathrm{d}t} = f_{\varphi}^{(\mathrm{Phy})}(u, T_a) + f_{\theta}^{(\mathrm{NN})}\left(u,\, T_a,\, f_{\varphi}^{(\mathrm{Phy})}(u, T_a)\right)$$

Different flavor of UDEs

Three coupling strategies between Phy and NN:

1. Additive UDE — independent contributions:

$$\frac{\mathrm{d}u}{\mathrm{d}t} = f_{\varphi}^{(\mathrm{Phy})}(u, T_a) + f_{\theta}^{(\mathrm{NN})}(u, T_a)$$

2. Additive Informed UDE — NN conditioned on physics output:

$$\frac{\mathrm{d}u}{\mathrm{d}t} = f_{\varphi}^{(\mathrm{Phy})}(u, T_a) + f_{\theta}^{(\mathrm{NN})}\left(u,\, T_a,\, f_{\varphi}^{(\mathrm{Phy})}(u, T_a)\right)$$

3. Free Informed UDE — NN learns full dynamics, physics as context:

$$\frac{\mathrm{d}u}{\mathrm{d}t} = f_{\theta}^{(\mathrm{NN})}\left(u,\, T_a,\, f_{\varphi}^{(\mathrm{Phy})}(u, T_a)\right)$$

Different flavor of UDEs

Three coupling strategies between Phy and NN:

1. Additive UDE — independent contributions:

$$\frac{\mathrm{d}u}{\mathrm{d}t} = f_\varphi^{(\mathrm{Phy})}(u, T_a) + f_\theta^{(\mathrm{NN})}(u, T_a)$$

2. Additive Informed UDE — NN conditioned on physics output:

$$\frac{\mathrm{d}u}{\mathrm{d}t} = f_\varphi^{(\mathrm{Phy})}(u, T_a) + f_\theta^{(\mathrm{NN})}\left(u,\, T_a,\, f_\varphi^{(\mathrm{Phy})}(u, T_a)\right)$$

3. Free Informed UDE — NN learns full dynamics, physics as context:

$$\frac{\mathrm{d}u}{\mathrm{d}t} = f_\theta^{(\mathrm{NN})}\left(u,\, T_a,\, f_\varphi^{(\mathrm{Phy})}(u, T_a)\right)$$

Best results: Free Informed UDE. Novel architecture w.r.t. the UDE literature.

Optimizing differential systems

$$\frac{\mathrm{d}u}{\mathrm{d}t} = f_\theta(u), \qquad L(\theta) = \mathrm{MSE}(u_{f_\theta}, u_{\mathrm{obs}}) = \frac{1}{N}\sum_{j=1}^{N}\left(u_{f_\theta}(t_j) - u_{\mathrm{obs},j}\right)^2$$

Differentiable programming: compute $\nabla_\theta L(\theta)$ by differentiating through the ODE solver.

Optimizing differential systems

$$\frac{\mathrm{d}u}{\mathrm{d}t} = f_\theta(u), \qquad L(\theta) = \mathrm{MSE}(u_{f_\theta}, u_{\mathrm{obs}}) = \frac{1}{N}\sum_{j=1}^{N}\left(u_{f_\theta}(t_j) - u_{\mathrm{obs},j}\right)^2$$

Differentiable programming: compute $\nabla_\theta L(\theta)$ by differentiating through the ODE solver.

Different approaches to get $\nabla_\theta L(\theta)$:

1. **Adjoint method**: backward ODE (can be unstable)
2. **Forward-mode AD**: differentiate solver steps

$\rightarrow$ Training of NeuralODE challenging!

Optimizing differential systems

$$\frac{\mathrm{d}u}{\mathrm{d}t} = f_\theta(u), \qquad L(\theta) = \mathrm{MSE}(u_{f_\theta}, u_{\mathrm{obs}}) = \frac{1}{N}\sum_{j=1}^{N}\left(u_{f_\theta}(t_j) - u_{\mathrm{obs},j}\right)^2$$

Differentiable programming: compute $\nabla_\theta L(\theta)$ by differentiating through the ODE solver.

Different approaches to get $\nabla_\theta L(\theta)$:

1. **Adjoint method**: backward ODE (can be unstable)
2. **Forward-mode AD**: differentiate solver steps

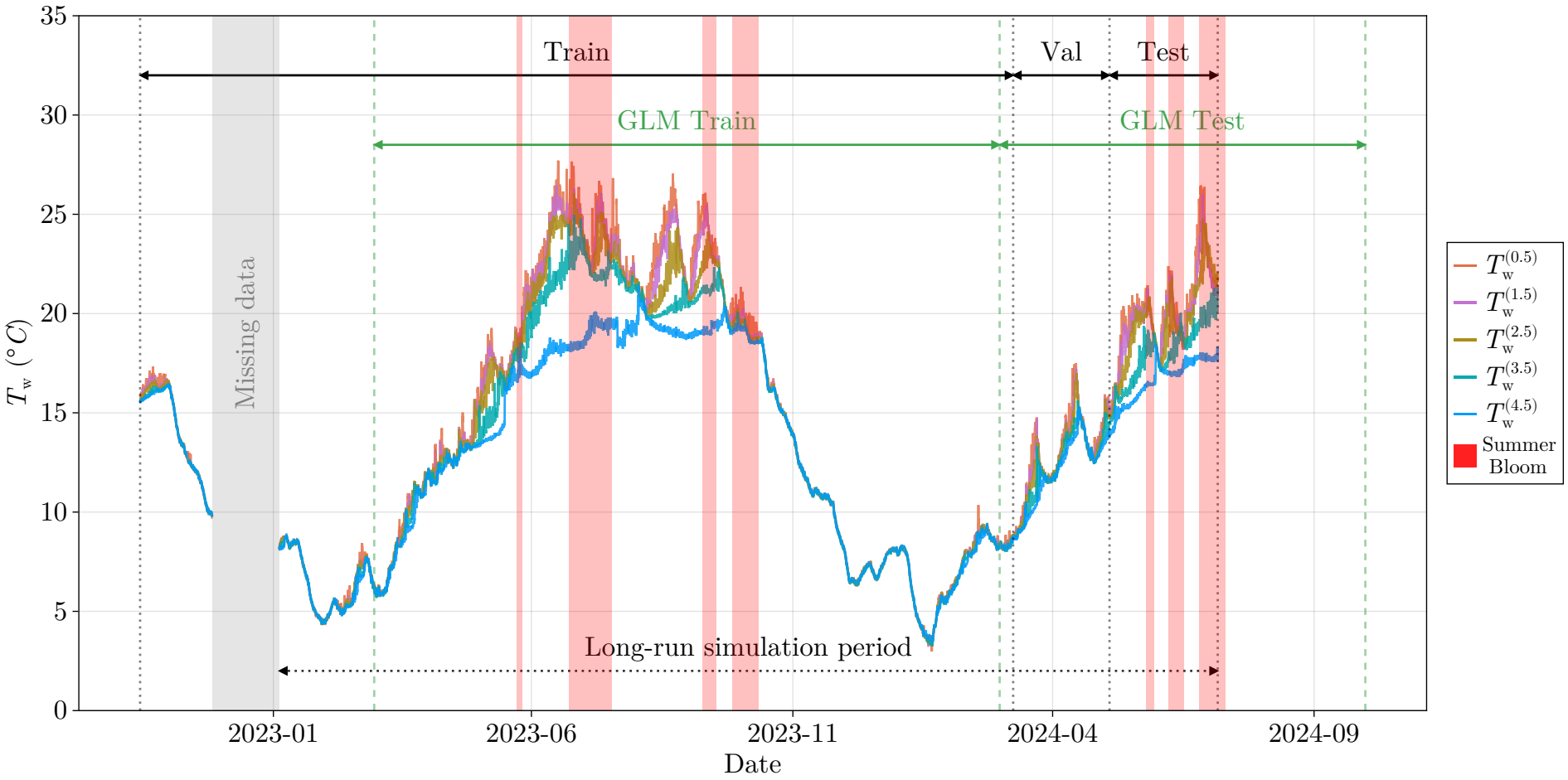
$\rightarrow$ Training of NeuralODE challenging!

GLM caveat

Written in `C/Fortran` $\rightarrow$ not easily differentiable.
Requires gradient-free (black-box) calibration instead.

$\rightarrow$ Julia language, JAX ecosystem, Pytorch ecosystem

Training/Validation/Test split



Water temperature at different depths.

Objective function

1. 7-day horizon: operationally relevant (reliable air-temperature forecast window)
2. Later evaluated: long-run (> 18 months) without re-training

Train on N_W non-overlapping **7-day windows**, all $d_u = 5$ depths jointly:

$$L(\theta) = \overline{\mathrm{RMSE}}(\theta) = \frac{1}{N_W} \sum_{i=1}^{N_W} \sqrt{\frac{1}{d_u\, n_i} \sum_{z,j} \left(\hat{U}_{z,j}^{(i)}(\theta) - U_{z,j}^{(i)}\right)^2}$$

Objective function

1. 7-day horizon: operationally relevant (reliable air-temperature forecast window)
2. Later evaluated: long-run (> 18 months) without re-training

Train on N_W non-overlapping **7-day windows**, all $d_u = 5$ depths jointly:

$$L(\theta) = \overline{\mathrm{RMSE}}(\theta) = \frac{1}{N_W} \sum_{i=1}^{N_W} \sqrt{\frac{1}{d_u\, n_i} \sum_{z,j} \left(\hat{U}_{z,j}^{(i)}(\theta) - U_{z,j}^{(i)}\right)^2}$$

Training details

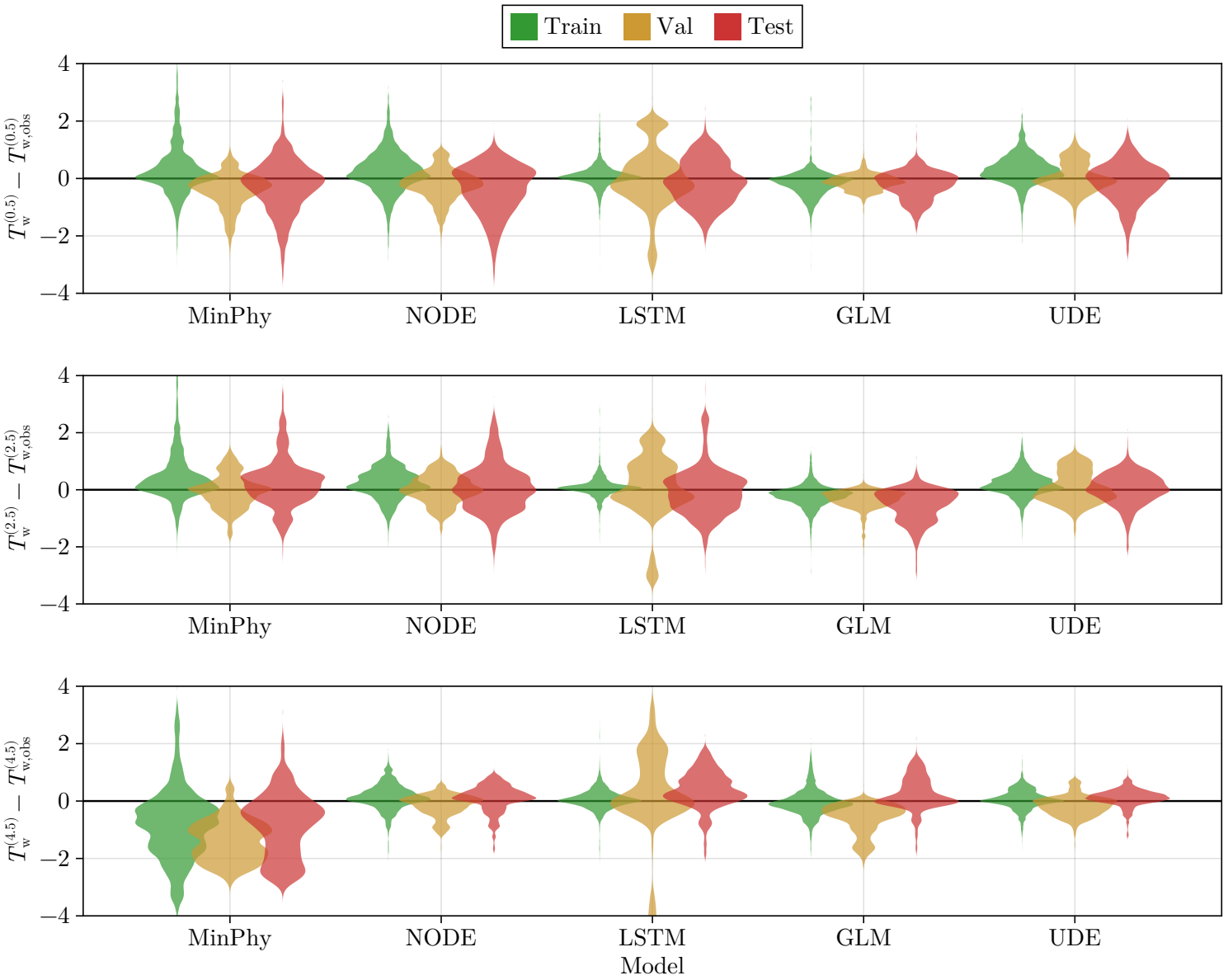
- Test different activation functions: **Snake**, ReLU, Tanh
- Growing intervals, learning Adam optimizer, early stopping,
- Brute force NN architectures: number of layers, hidden size

7-day forecast: model comparison

Model	N_θ	**Train RMSE** [°C]	**Val RMSE** [°C]	**Test RMSE** [°C]
UDE	586	0.42	**0.47**	**0.55**
GLM	∼22	0.42	0.49	0.61
NODE	595	0.55	0.49	0.71
LSTM	16 625	**0.34**	0.98	0.75
MinPhy	12	0.96	0.86	1.09

- UDE: best val & test RMSE with only 586 parameters (30× fewer than LSTM)
- LSTM overfits: best train RMSE but worst val RMSE (0.98 °C)
- UDE excels at depth: RMSE 0.29 °C at $z = 4.5$ m — best of all models

7-day forecast: per-depth errors



Model	Train	Val	Test
MinPhy	0.99	0.73	0.95
NODE	0.75	0.58	0.93
LSTM	0.43	1.06	0.76
GLM	0.5	0.31	0.54
UDE	0.59	0.52	0.72

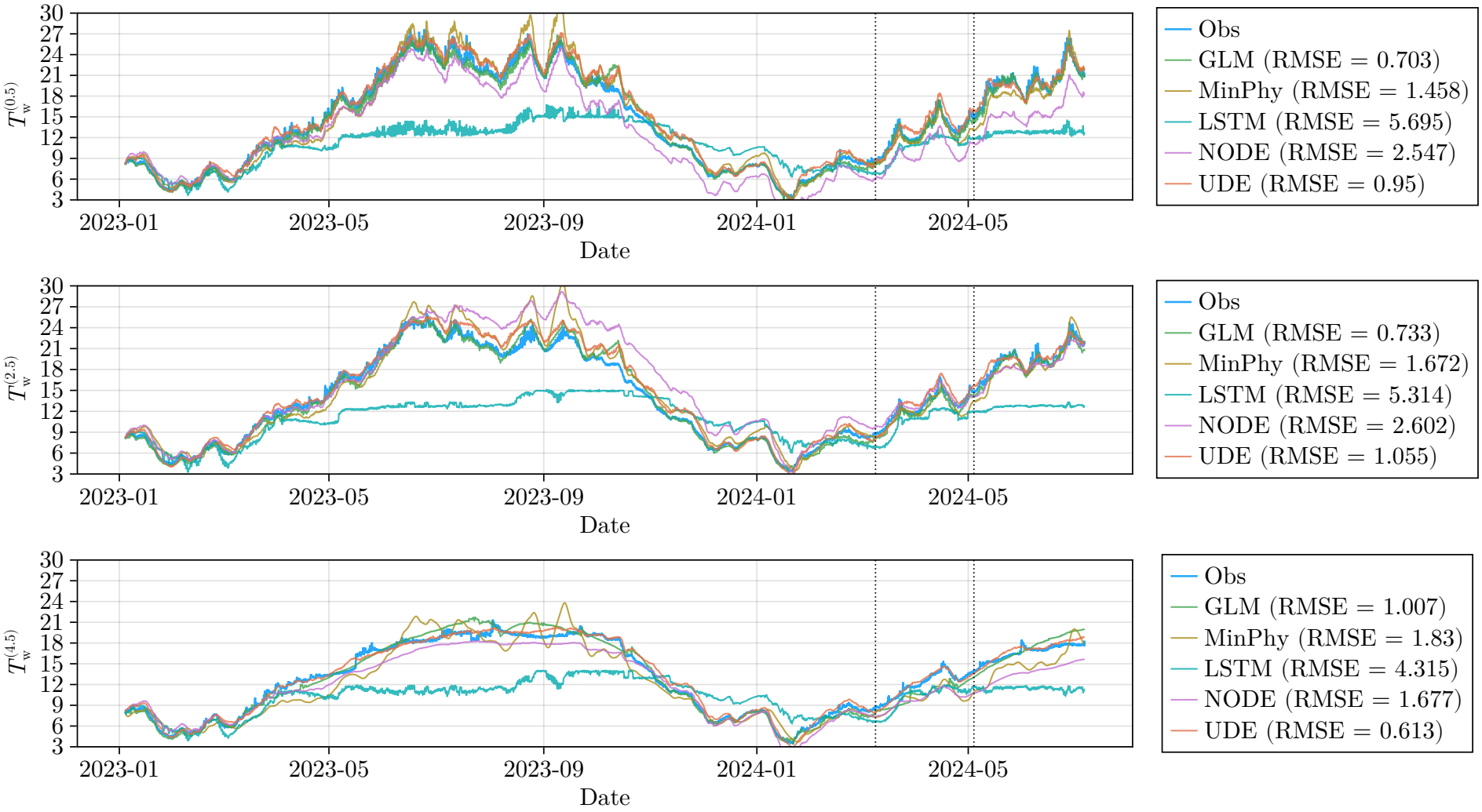
Model	Train	Val	Test
MinPhy	1.11	0.58	0.87
NODE	0.61	0.5	0.83
LSTM	0.37	1.1	0.81
GLM	0.45	0.49	0.78
UDE	0.46	0.56	0.54

Model	Train	Val	Test
MinPhy	1.46	1.55	1.37
NODE	0.45	0.41	0.46
LSTM	0.37	1.52	0.74
GLM	0.46	0.81	0.62
UDE	0.35	0.39	0.32

Long-term prediction (> 18 months)

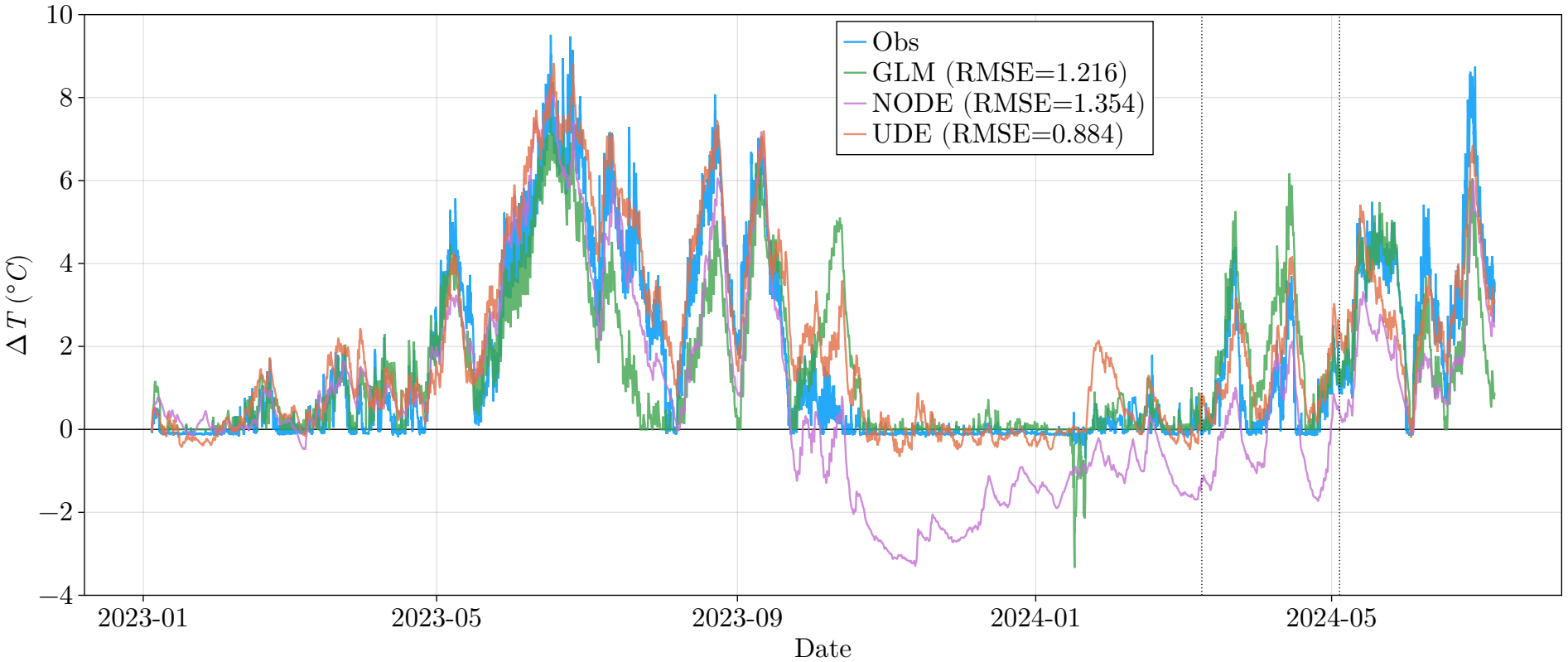
Single **long** simulation from Jan 2023 driven by air temperature only.

Long-run RMSE — GLM: 0.782 | MinPhy: 1.653 | LSTM: 5.212 | NODE: 2.636 | UDE: 0.909



Stratification

Stratification $\Delta T = T_{\mathrm{w}}^{(0.5)} - T_{\mathrm{w}}^{(4.5)}$: key indicator for bloom risk.



Thermal stratification ΔT over the full period. Dotted lines: train/val/test transitions.

Long-run performance summary

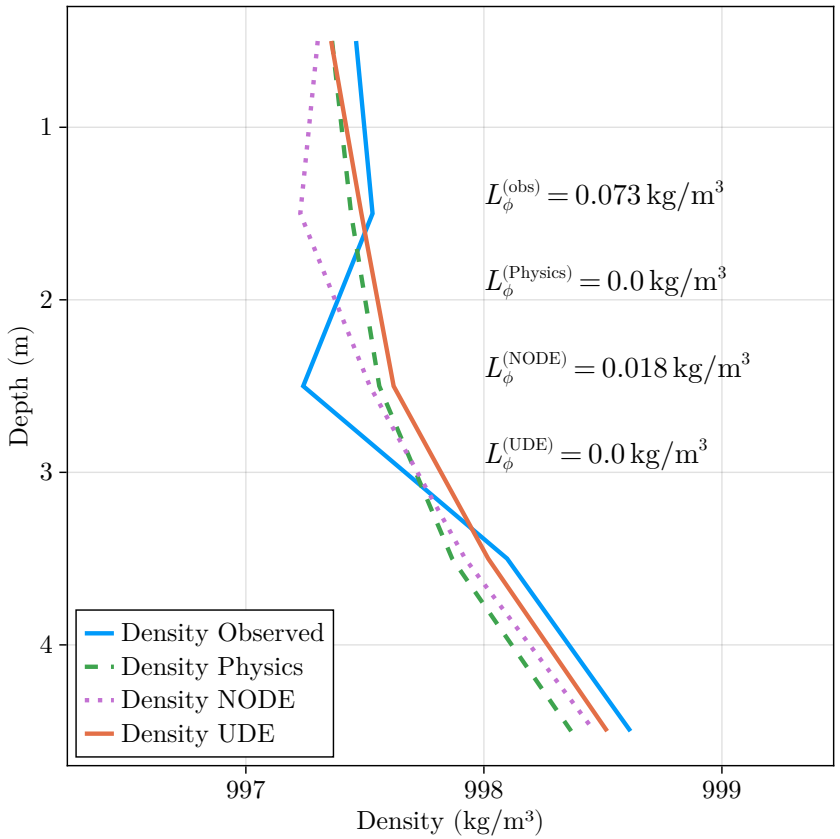
Model	Test 7d RMSE [°C]	Long RMSE [°C]	Strat ΔT RMSE [°C]
UDE	**0.55**	0.91	**0.88**
GLM	0.61	**0.78**	1.22
NODE	0.71	2.64	1.35
LSTM	0.75	5.21	1.78
MinPhy	1.09	1.65	1.59

- UDE: **best stratification** RMSE — despite higher surface error than GLM
- GLM: best long-run RMSE but requires full meteo forcings + "manual" calibration
- NODE & LSTM diverge on long horizons — no physics to stabilize extrapolation

Physics consistency of the models

Density of water $\rho(T)$ should **not decrease with depth**.

$$\text{Physical consistency loss } \mathcal{L}_{\text{phys}}(T) = \sum_{i=1}^{N_{\text{Layers}}-1} \max(-\Delta\rho_i, 0)$$



- Observations:
  $L_\phi^{\text{test}} \simeq 9.6 \times 10^{-3}\,\text{kg/m}^3$
  (sensor-induced inversions)
- UDE:
  $L_\phi^{\text{test}} \simeq 2.4 \times 10^{-3}\,\text{kg/m}^3$
  more consistent than raw data

Conclusions

- UDEs embed physics in the structure of the model and learn the missing physics with a Neural Network.
- UDEs can learn lake temperature dynamics and are more physics-consistent than physics-only or data-only models.
- Many extensions possible:
  - Augmented Neural ODEs (hidden variables representing nutrients)
  - Neural SDEs (uncertainty quantification)
- Hard to train!

→ (**Ongoing**) Encouraging results for chlorophyll or oxygen concentration but more work needed to get both at the same time.

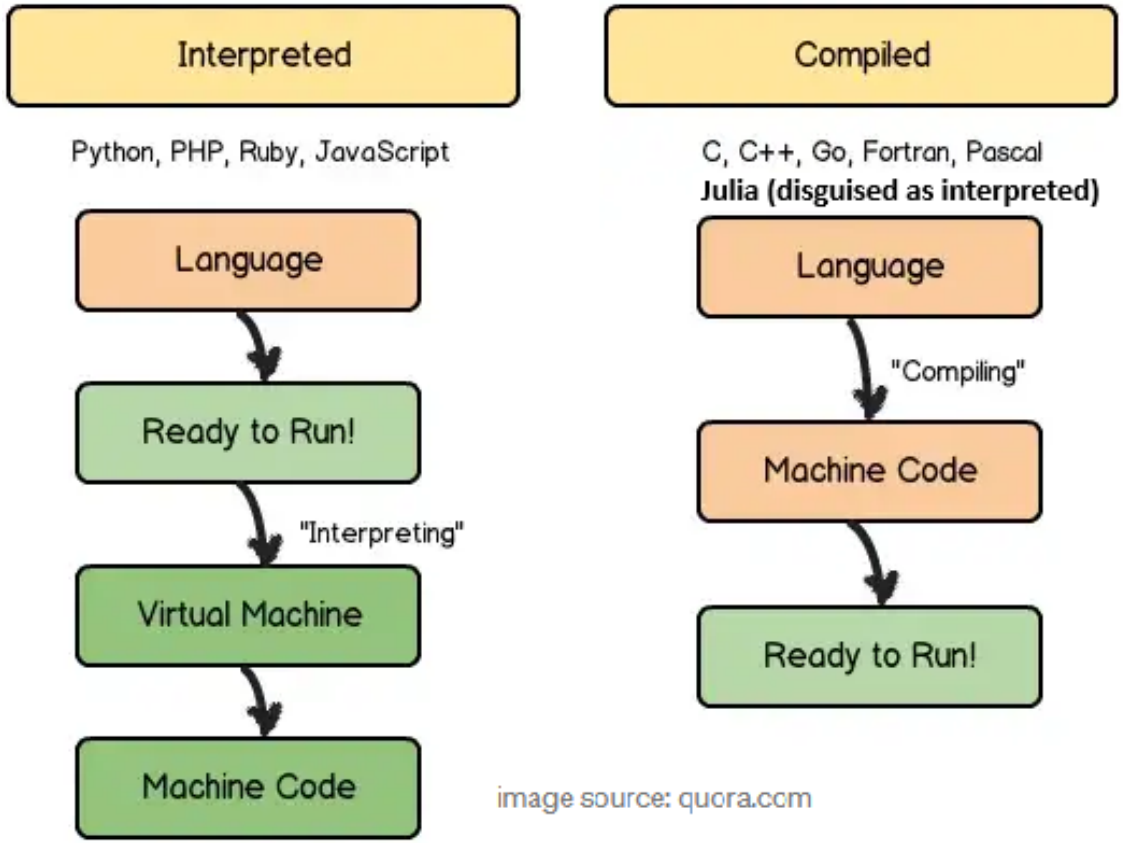
Conclusions

- UDEs embed physics in the structure of the model and learn the missing physics with a Neural Network.
- UDEs can learn lake temperature dynamics and are more physics-consistent than physics-only or data-only models.
- Many extensions possible:
  - Augmented Neural ODEs (hidden variables representing nutrients)
  - Neural SDEs (uncertainty quantification)
- Hard to train!

→ (**Ongoing**) Encouraging results for chlorophyll or oxygen concentration but more work needed to get both at the same time.

Thank You!

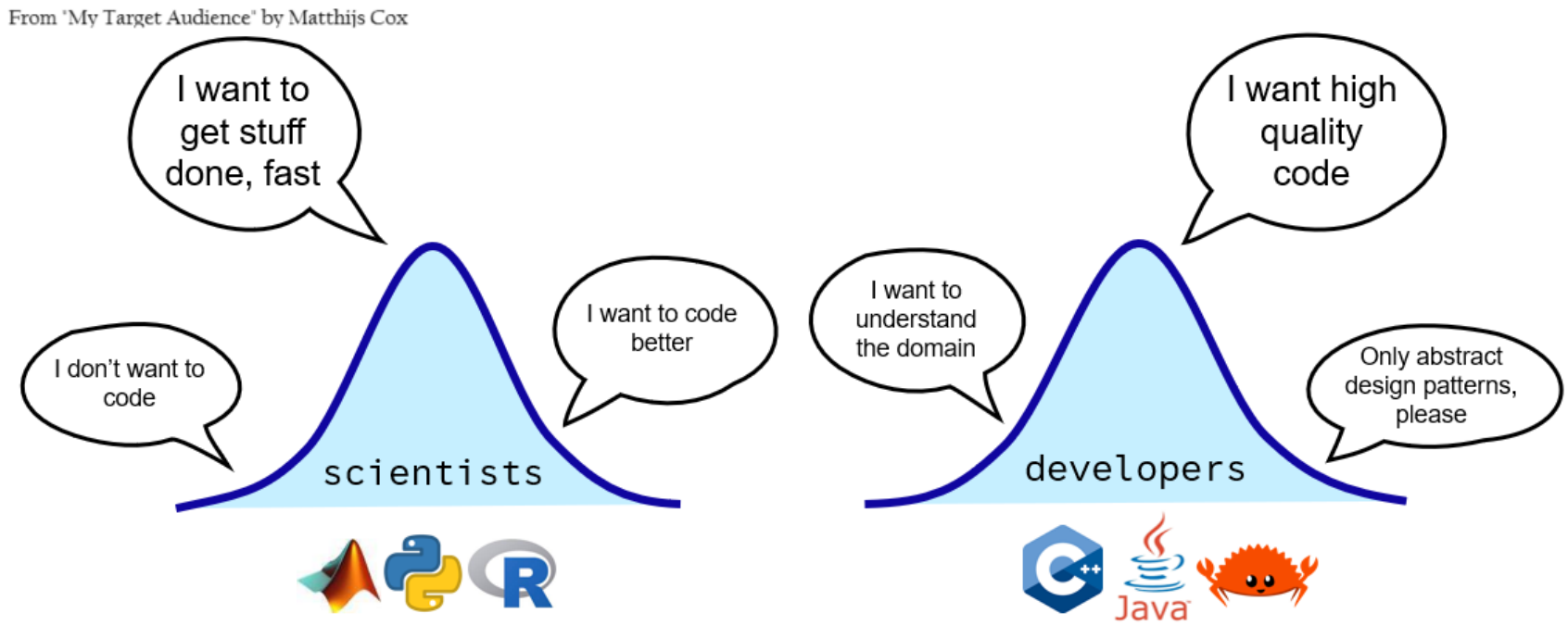
The Two-Language Problem: Compiled vs. Interpreted



Who knows C++? Who wants to learn it?

- **Compiled languages** are fast but hard to write
- **Interpreted languages** are easy to write but slow
- Interpreted languages need compiled languages under the hood

The Two-Language Problem: Scientists vs. Developers



Common workflow:

1. Prototype in R/Python
2. Rewrite some parts in C++/Fortran or Cython, Rcpp, Pytorch etc.

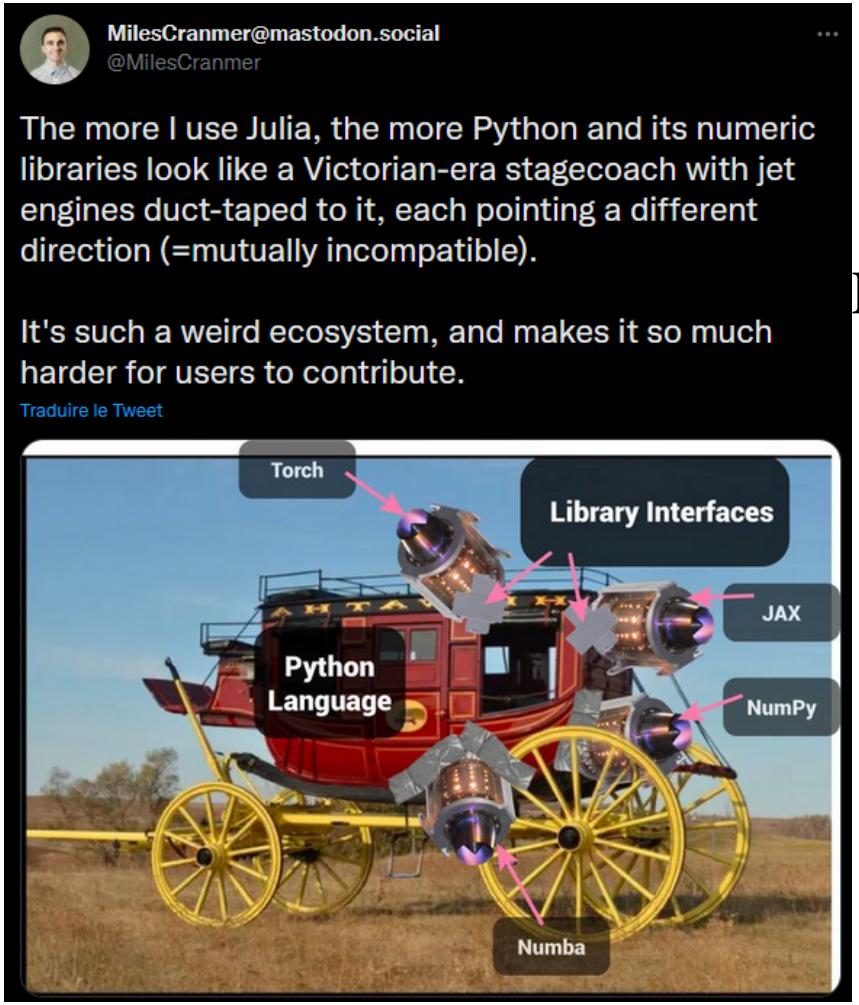
The Two-Language Problem: Scientists vs. Developers

Common workflow:

1. Prototype in R/Python
2. Rewrite some parts in C++/Fortran or Cython, Rcpp, Pytorch etc.

or use Julia "feels like Python/R but fast like C" + a lot more

Composability in Python



Python HMM packages

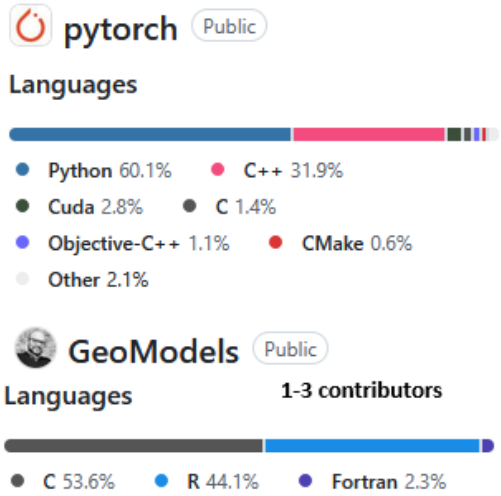
- `hmmlearn` (NumPy), `pomegranate` (PyTorch), `dynamax` (JAX)
- Each locked to its framework
- Built-in distributions only e.g.

```
jax.random.multivariate_normal
torch.randn
numpy.random.multivariate_normal
```

$\Rightarrow$ Each framework $\rightarrow$ isolated ecosystem = mutually incompatible

Two languages packages examples

glue code

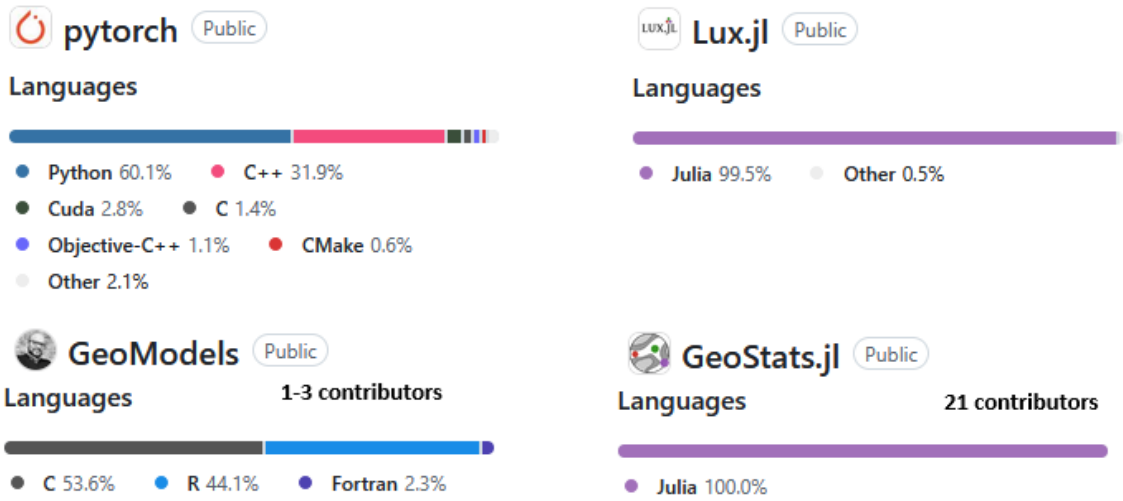


Costs of two languages:

- Few people know both languages well enough to contribute
- Installation/Development/Maintenance nightmare with dependencies

Two languages packages examples

glue code



Costs of two languages:

- Few people know both languages well enough to contribute
- Installation/Development/Maintenance nightmare with dependencies

In Julia most packages just work together!

- Each package = one domain (distributions, optim, diff equations, etc.)
- Improving one package can benefit all others and users
- **Easier to contribute!**

PINNs vs UDEs

	PINNs	**UDEs**
What is learned	Solution $u_\theta(t)$	Dynamics $\dot{u}(t)$
Physics	Via residual loss $\lvert\mathcal{P}_u\rvert^2$	In the ODE structure
Interval	Fixed $t \in [t_0, t_f]$	Any $t \geq t_0$
Forecasting	Not suited	Suited
Training data	Collocation points	Time series observations

+ Dynamics ($\sim$ derivative) is often simpler than the solution itself

$\rightarrow$ UDEs learn a model that can be used for forecasting

$\rightarrow$ PINNs are better suited to solving a PDE (with differential operators) on a complex domain, does not require any ODE/PDE solver! (e.g. fluid simulation)