Implementing machine learned parameterizations in climate models

Alistair Adcroft, Princeton University- GFDL

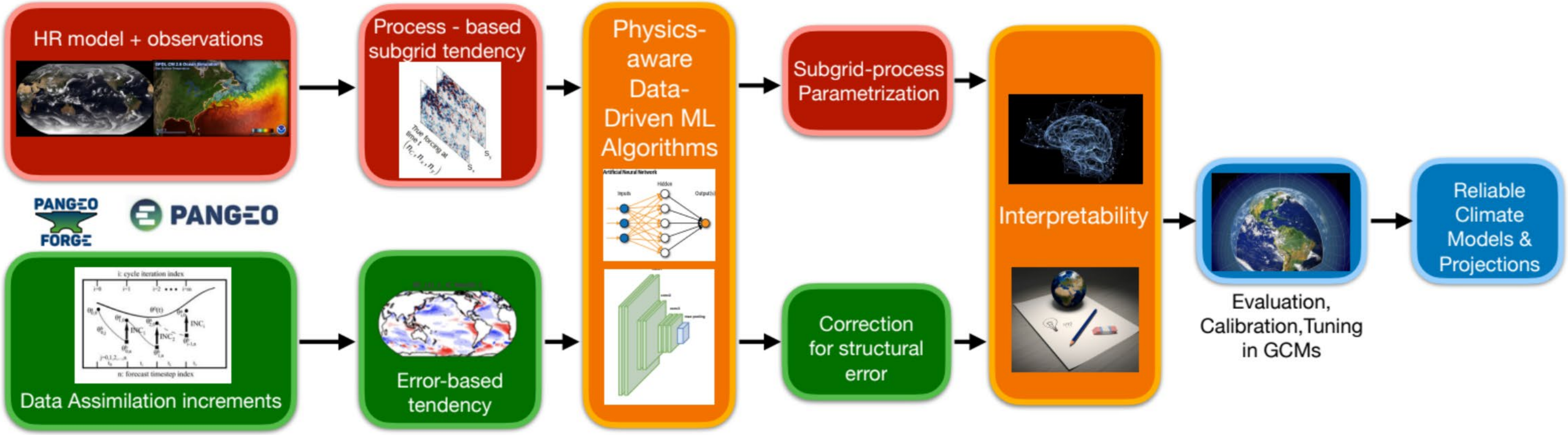
KITP November 2021

https://m2lines.github.io/

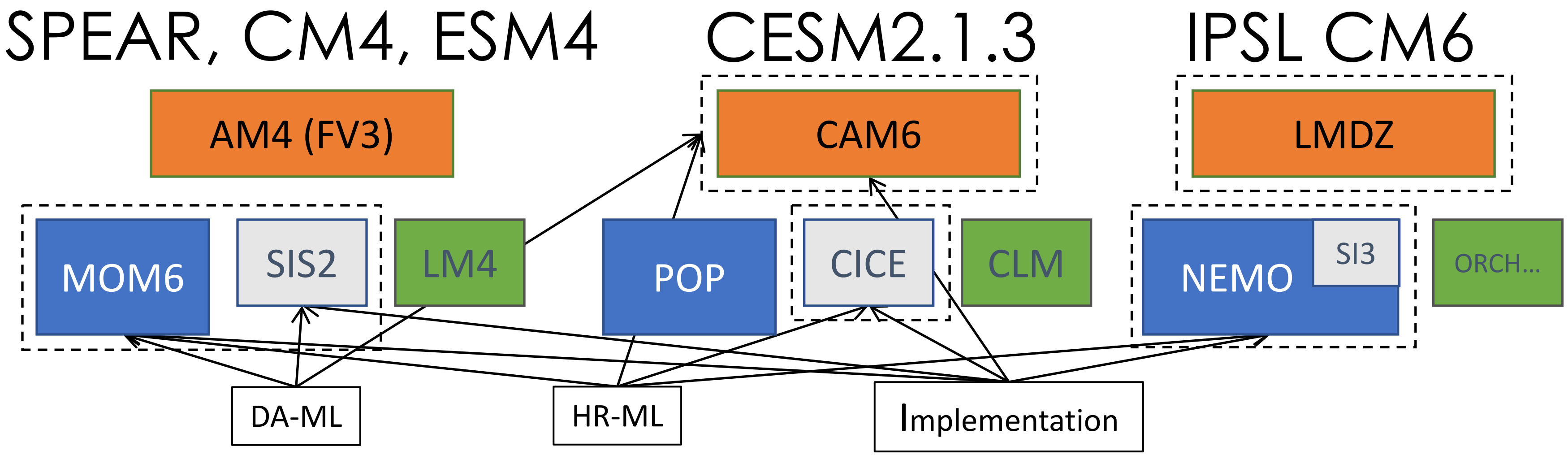
M^{2}LInES: Multiscale Machine Learning In Coupled Earth System Modeling

Goal: Improve the skill in modelled surface ocean, ice, atmospheric fields on timescales of hours to centuries in global climate models

- ***Development of new data-driven, physics-aware parameterizations*** of subgrid ocean, ice & atmosphere processes
- ***Reduction of structural model biases*** (numerics, missing physics & poor subgrid
- parameterizations) **in existing climate models** at NCAR, GFDL & IPSL



M2LInES: The climate models



- 11 institutions, US + Europe, climate modeling centers
- These are AR6 climate models, with well established component models
- Learning from data-assimilation (DA-ML)
- Learning from High-resolution simulations (HR-ML)
- Implementation across component models
- The diversity of components models, resolutions, and climates, requires ML parameterizations to be:
  - Transferable
  - Scale-aware
  - Generalizable

M^{2}LInES Team


Laure Zanna


Johanna Goldman


Alistair Adcroft


Carlos Fernandez-Granda


Ryan Abernathey


Joan Bruna


Pierre Gentine


Judith Berner


Mitch Bushuk


Feiyu Lu


Paul O’Gorman


Julie Deshayes


Brandon Reichl


Marika Holland


Julien Le Sommer


Arthur Guillaumin


Janni Yuval


Andrew Ross


Lorenzo Zampieri


Ziwei Li


Tarun Verma


Aakash Sane


Cem Gultekin


Dhruv Balwada


Alex Connolly


Cheng Zhang


Pavel Perezhogin

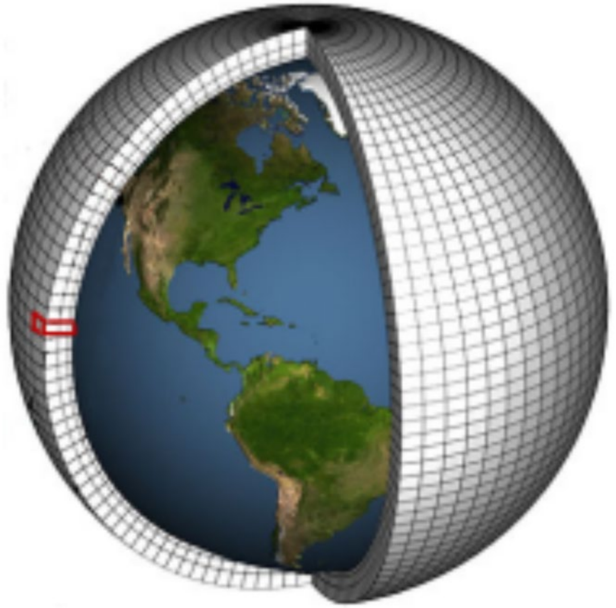

Anastasia Gorbunova


Will Chapman


Will Gregory

Challenge: to use ML in Climate Models

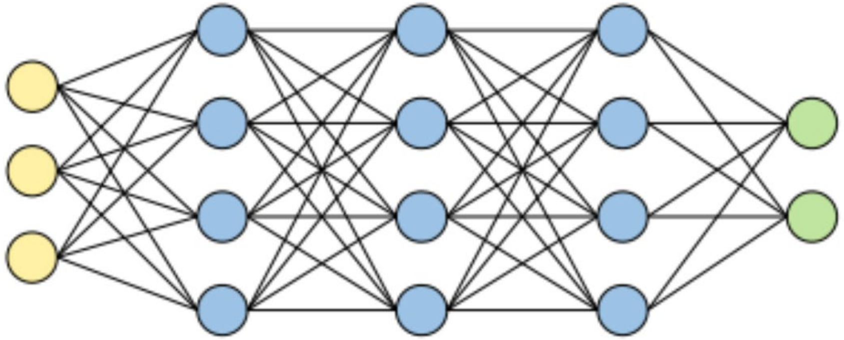
Climate models



NOAA's Gaea (ORNL)

- 1,000,000+ lines of Fortran/c
  - Decades of experience
  - Almost exclusively use CPUs

Machine learned models



web image

- 50+ lines of python/Julia
  - Leveraging widely adopted packages under the hood
  - Utilizing GPUs

There is an apparent (computer) language barrier

Using a trained ML model from climate code

Options to tackle inter-language barrier:

- Code final ML model in Fortran
  - e.g. Fortran-Keras bridge (Ott, Pritchard, et al., 2020)
- Build/use a Fortran-python interface using C-API
  - e.g. python-embedding (Python manual), `call_py_fort` (Noah Brenowitz)
- Use turn-key package (CPU-GPU)
  - HPE's SmartSim (Partee et al., 2021, subm.)

There are solutions – but there are more subtle challenges with code

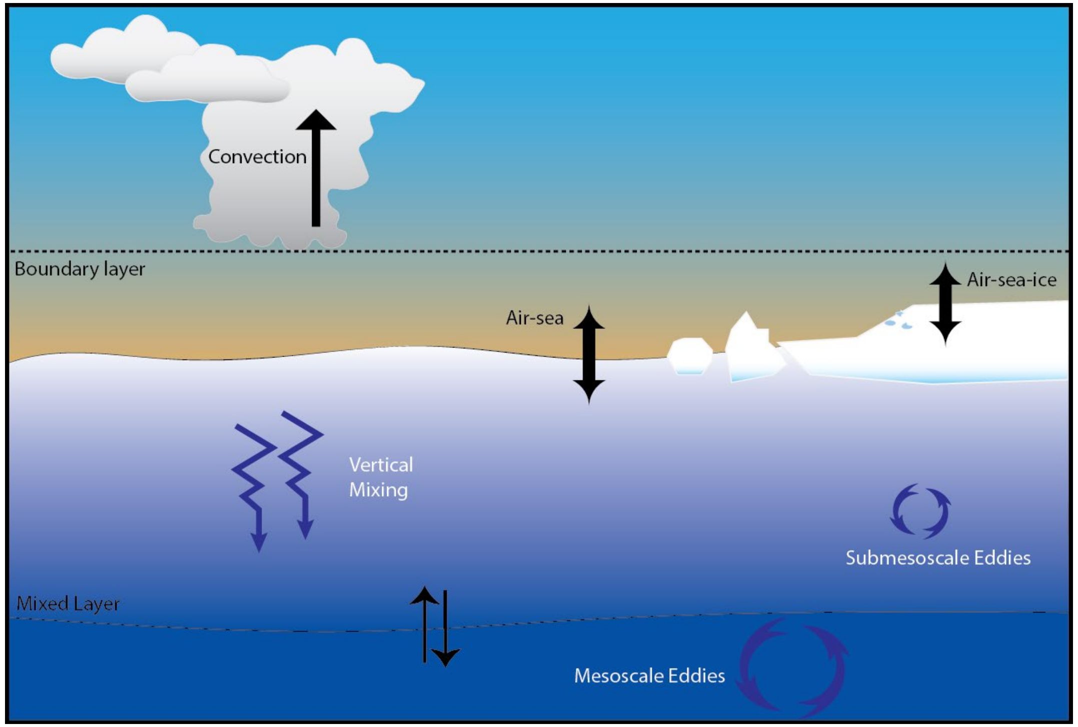
Other challenges with climate codes


- Climate model data structures unique to each model/component/process
  - Definitions, approximations, and units of variables vary between models, e.g.
    - Ocean: practical salinity vs absolute salinity
    - Atmosphere: CLUBB uses (anelastic) liquid water potential, rather than enthalpy, CAM expects enthalpy+KE conservation
    - Vertical coordinates differ i) between models, ii) between analysis and models, iii) between columns, iv) between dynamics and physics s/r
  - Parallelism: decomposed data might not support wide stencils (for CNN)
  - Data availability: parameterizations might only have access to single column data but ML model need multiple columns

Interactions between processes

- Information interaction
  - Use of intermediate/secondary variables, e.g. sub-grid parameters (cloud fraction, mixing length scale, …)
- Physical interaction

  e.g. mesoscale+sub-mesoscale eddy re-stratification balancing mechanical/buoyant mixing of ocean boundary layer
- Compensating errors
  - ML models will improve representation of model physics … but can make model look worse due to prior tuning



Online stability

- Offline training not always stable when used in a forward model
- Depends on effective eigenvalues
  - Lyapunov exponents
- Calculating/modelling increments vs tendencies vs fluxes makes a difference
- Conservation properties, and generalization, should be beneficial
  - **Recent successes, e.g. Yuval et al. 2021**
- Can ML model tendencies, N^n, simply be added to model tendencies M^n ?

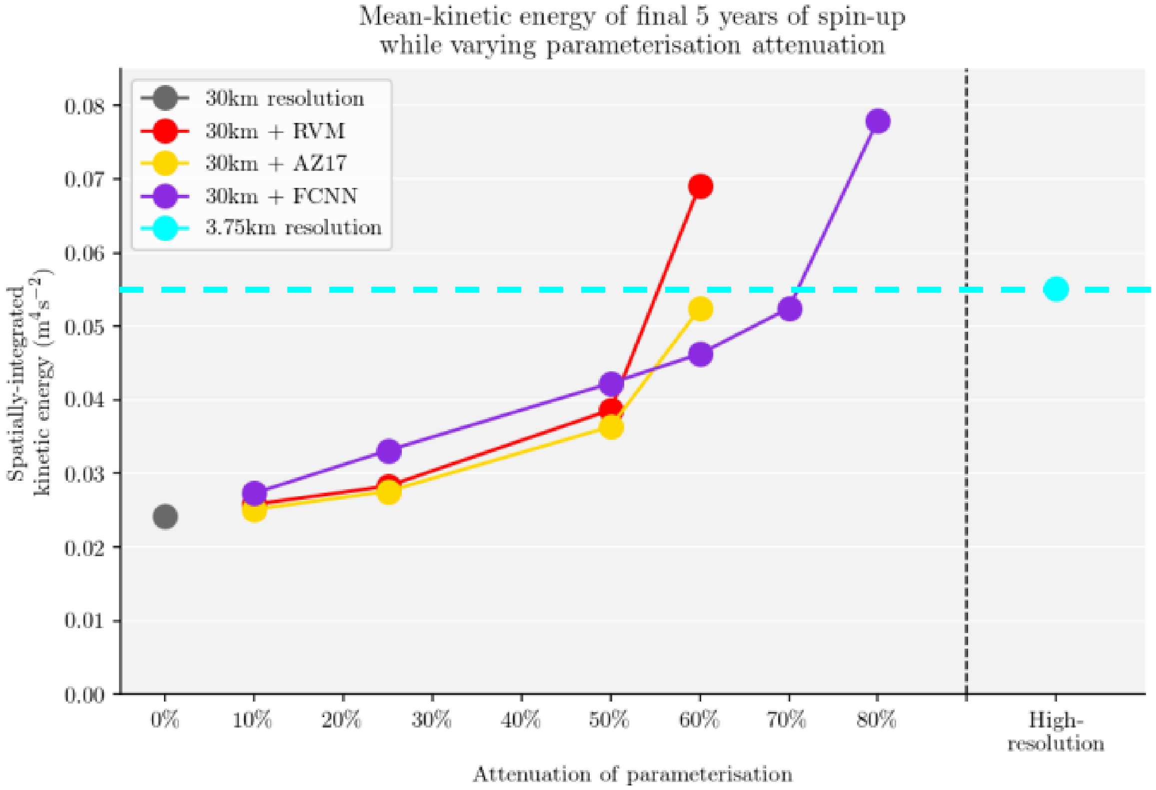
- GCMs each use different algorithms
  - e.g. (with $a = \frac{3}{2} + \epsilon$)
    $$u^{n+1} = u^n + \Delta t(aM^n - (1-a)M^{n-1})$$
- Damping modes
  - e.g. Euler forward
    $$u^{n+1} = u^n + \Delta t\big(M^n + N(u^n)\big)$$
- Oscillatory modes
  - Multi-level, multi-stage, e.g.
    $$u^* = u^n + \frac{1}{2}\Delta t\big(M^n + N(u^n)\big)$$
    $$u^{n+1} = u^n + \Delta t\big(M^n + N(u^*)\big)$$
- Fast modes
  - Implicit, e.g.
    $$u^{n+1} = u^n + \Delta t\big(M^n + N(u^{n+1})\big)$$
- Implications for training?

Stability and efficacy

- Effect of online parameterization often needs scaling
  - Feedback between basic state and parameterization missing when training offline
  - Resolved gradients imperfect
- e.g. ZB20 parameterization energizes flow
  - FCNN needed the least attenuation, but none could be used at full effect
- Parameterizations are always tuned

Coarse-resolution model with sub-grid forcing

$$\frac{\partial}{\partial t}\bar{u} + (\bar{u}.\bar{\nabla})\bar{u} = \bar{F} + \bar{D} + \bar{S}$$

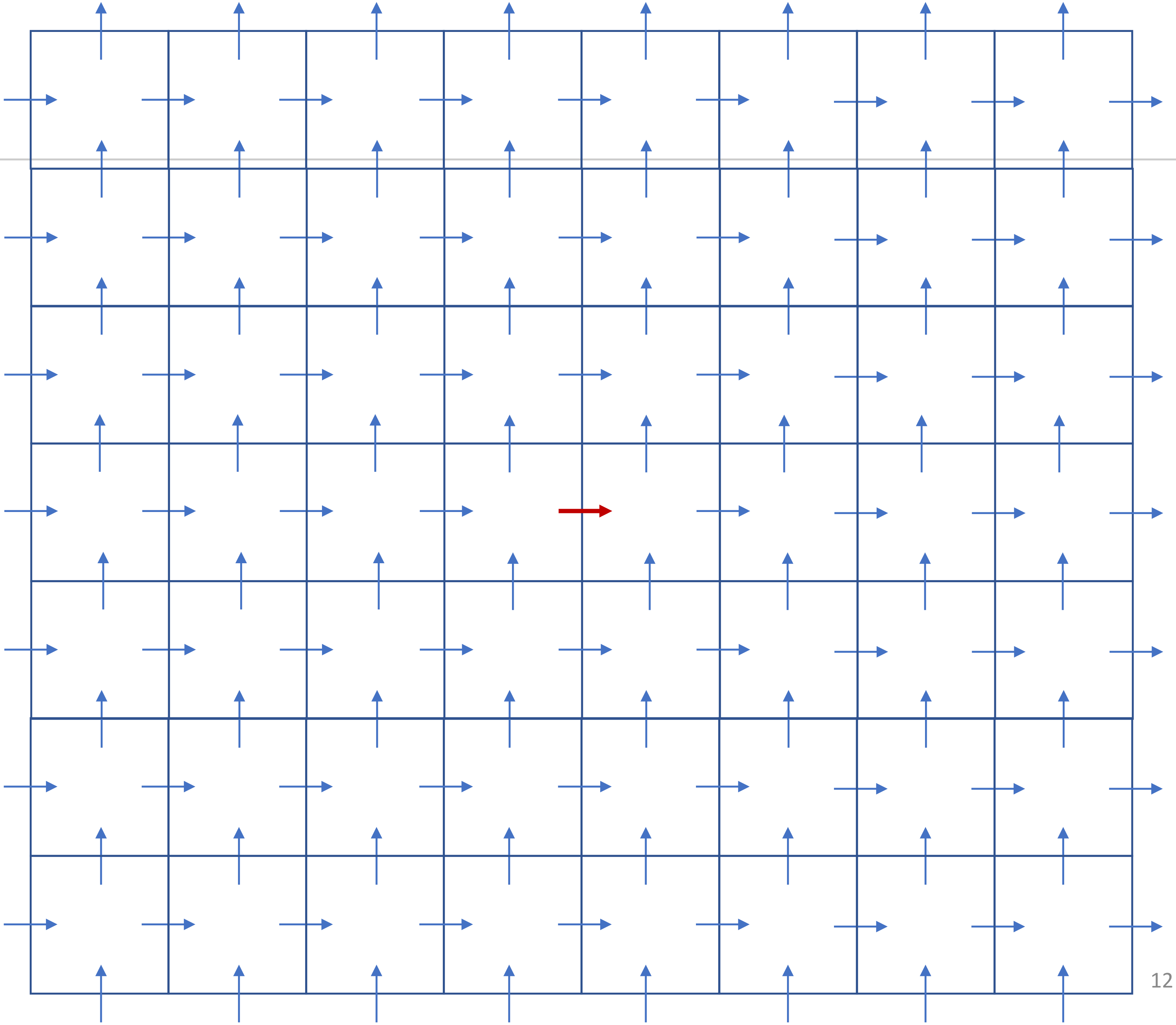


Zanna & Bolton, 2020 (Fig. S9)

Stencils

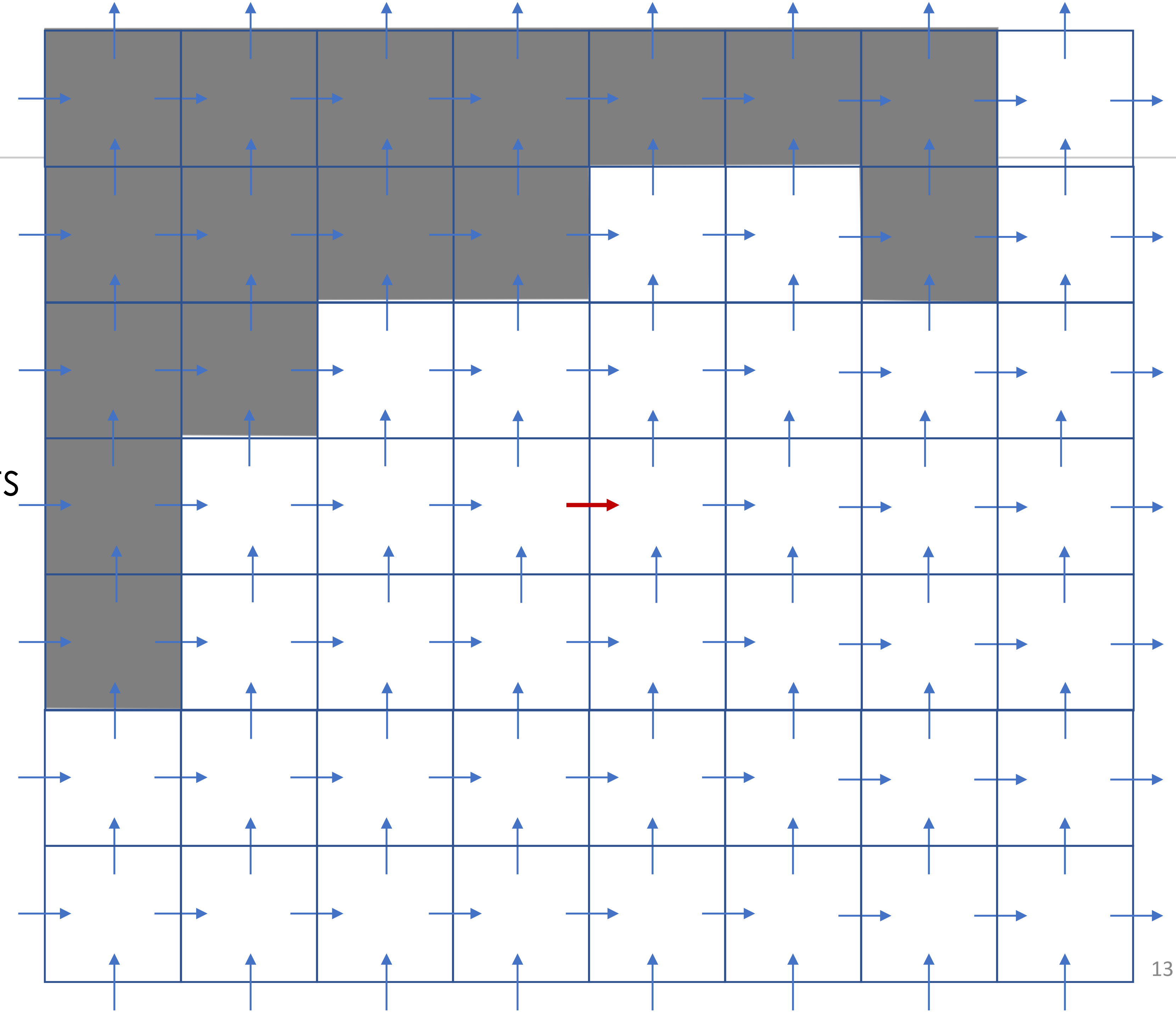
- ZB20 FCNN
  - 4 layers x 3x3
  - No padding

- For contrast, GZ21 CNN
  - 2 layers x 5x5
  - 5 layers x 3x3



Stencils

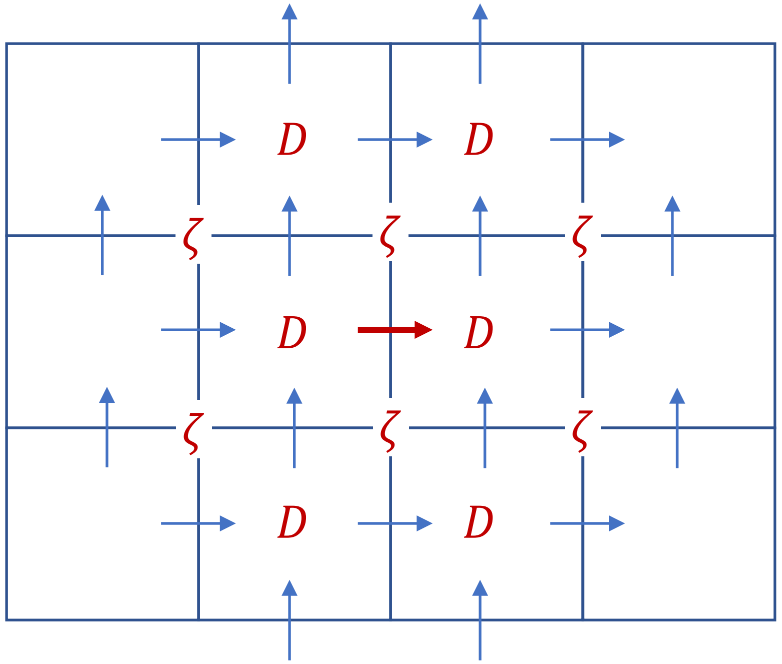
- Irregular coasts absent from training
- ZB20 FCNN
  - Disabled 4 points from boundary



Stencils

- Discovered equations in ZB20 (and AZ17) have small stencil (for 2nd order FV/FD)
- Expressions are using physical quantities likely already available in models
- ZB20 9-term does not extend stencil

$$\kappa\overline{\nabla}\cdot\begin{pmatrix}\zeta^2-\zeta D & \zeta D\\ \zeta D & \zeta^2+\zeta D\end{pmatrix}$$

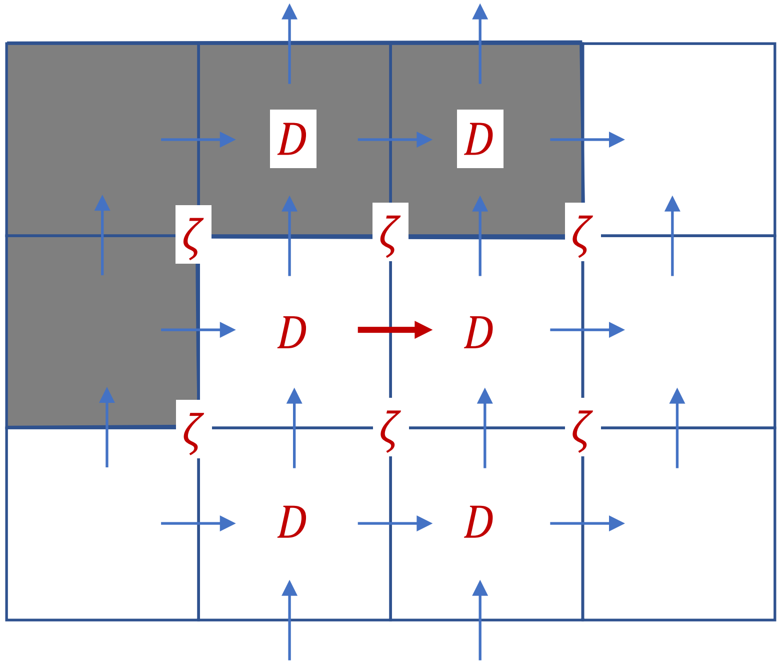


Stencils



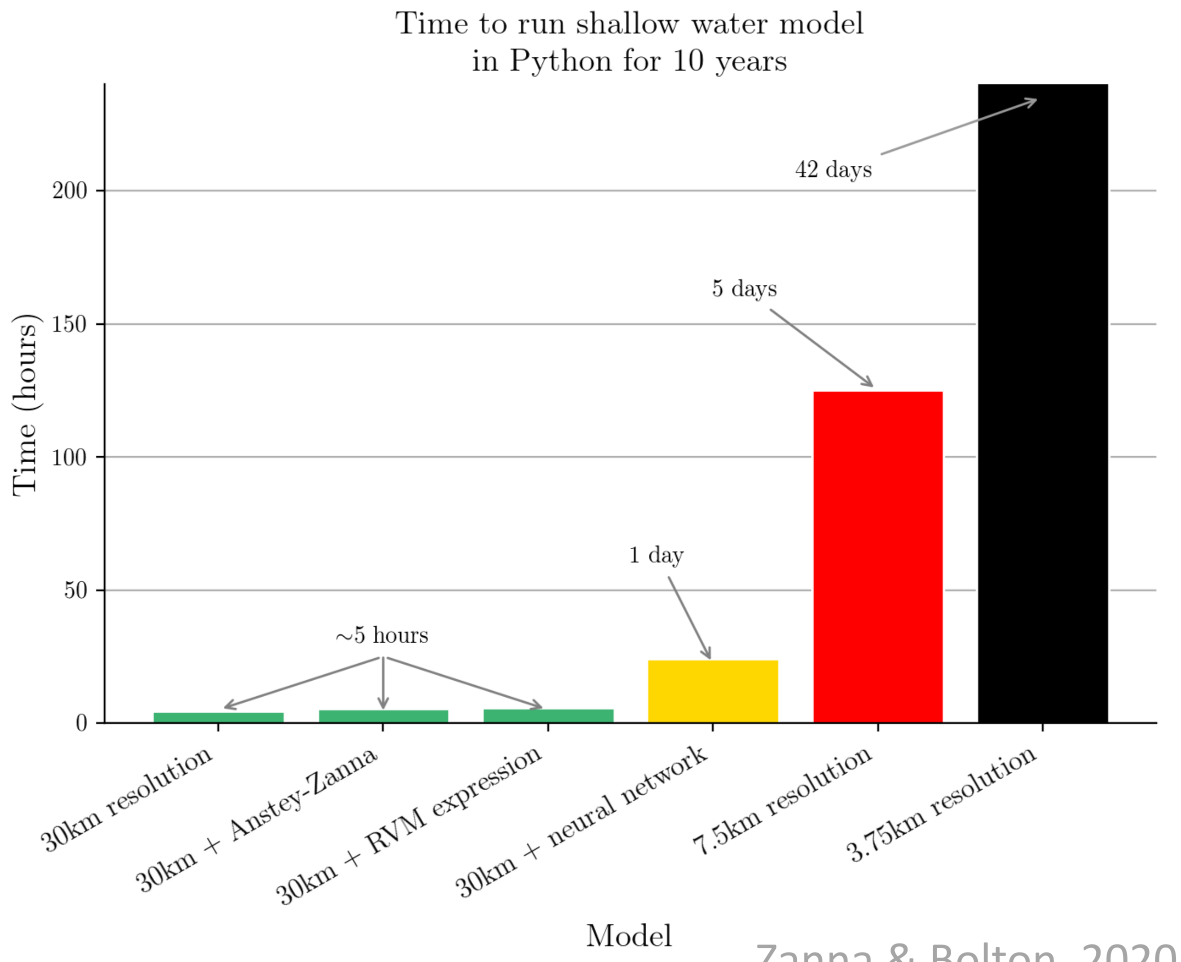
- We know how to implement physical boundary conditions when calculating the quantities
- Caveat: may not be the right boundary condition for the way the ML parameterization “apparently” used the quantities
  - If in testing we find near-boundary artifacts, we should train with boundaries

$$\kappa \overline{\nabla} \cdot \begin{pmatrix} \zeta^2 - \zeta D & \zeta D \\ \zeta D & \zeta^2 + \zeta D \end{pmatrix}$$



Performance

- Deep Learning is ideal on GPUs
  - High arithmetic intensity
- ML has been proposed as a mean to accelerate models
  - Balance between volume of inputs and cost of transferring inputs to GPU
- **Equation discovery**
  - More efficient (less arithmetic) implementation than NN
  - Scientific interpretation and understanding



Zanna & Bolton, 2020 (Fig. S12)

Active implementations

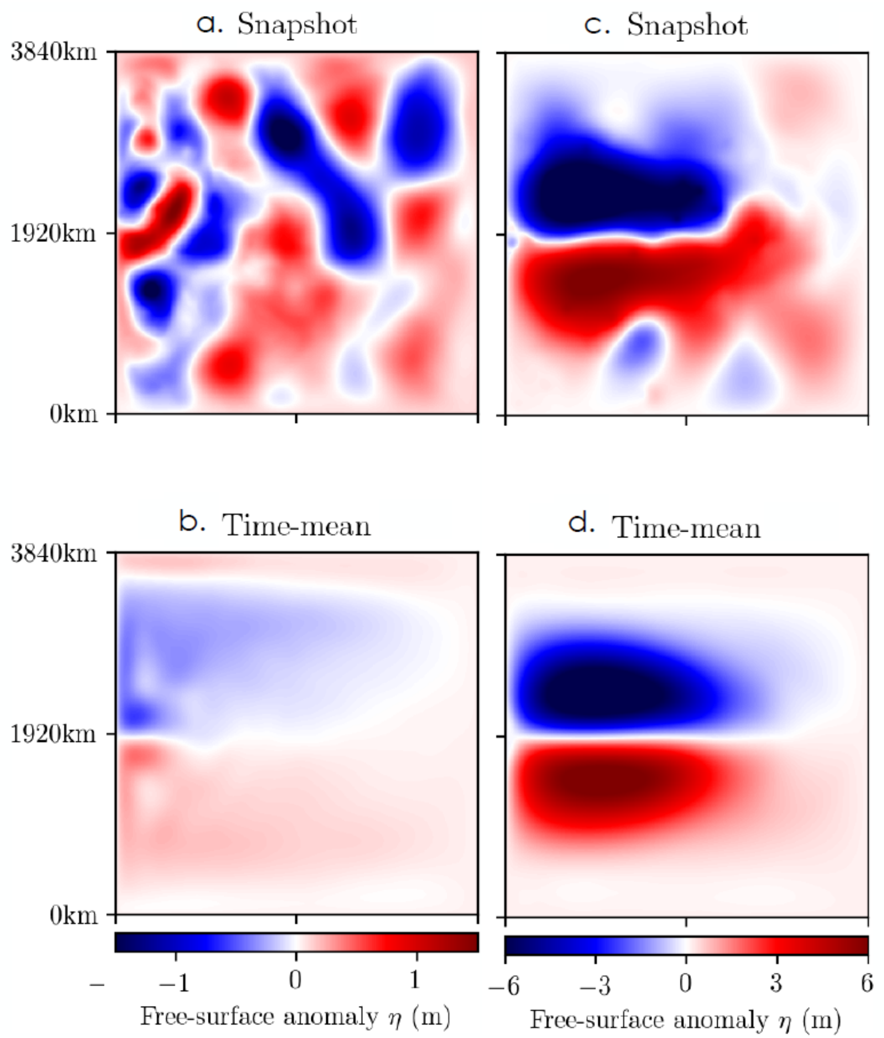
- Ocean horizontal momentum closure
  - Guillaumin & Zanna, 2021
  - Zanna & Bolton, 2021

- Atmospheric convection
  - Yuval & O'Gorman, 2020, Yuval et al., 2021, Yuval & O'Gorman, in prep.

- Ocean surface boundary layer parameter profiles
  - Ongoing work of Sane et al.

Momentum closures

- Fine-resolution model evolves as
$$\partial_t u + (u \cdot \nabla)u = F + D + S$$
- Sub-grid scale momentum transfer diagnosed by filtering and coarsening ($\overline{}$)
$$S = \begin{pmatrix} S_x \\ S_y \end{pmatrix} = (\overline{u} \cdot \overline{\nabla})\overline{u} - \overline{(u \cdot \nabla)u}$$
- So that coarse model evolves according to
$$\partial_t \overline{u} + (\overline{u} \cdot \overline{\nabla})\overline{u} = \overline{F} + \overline{D} + S$$



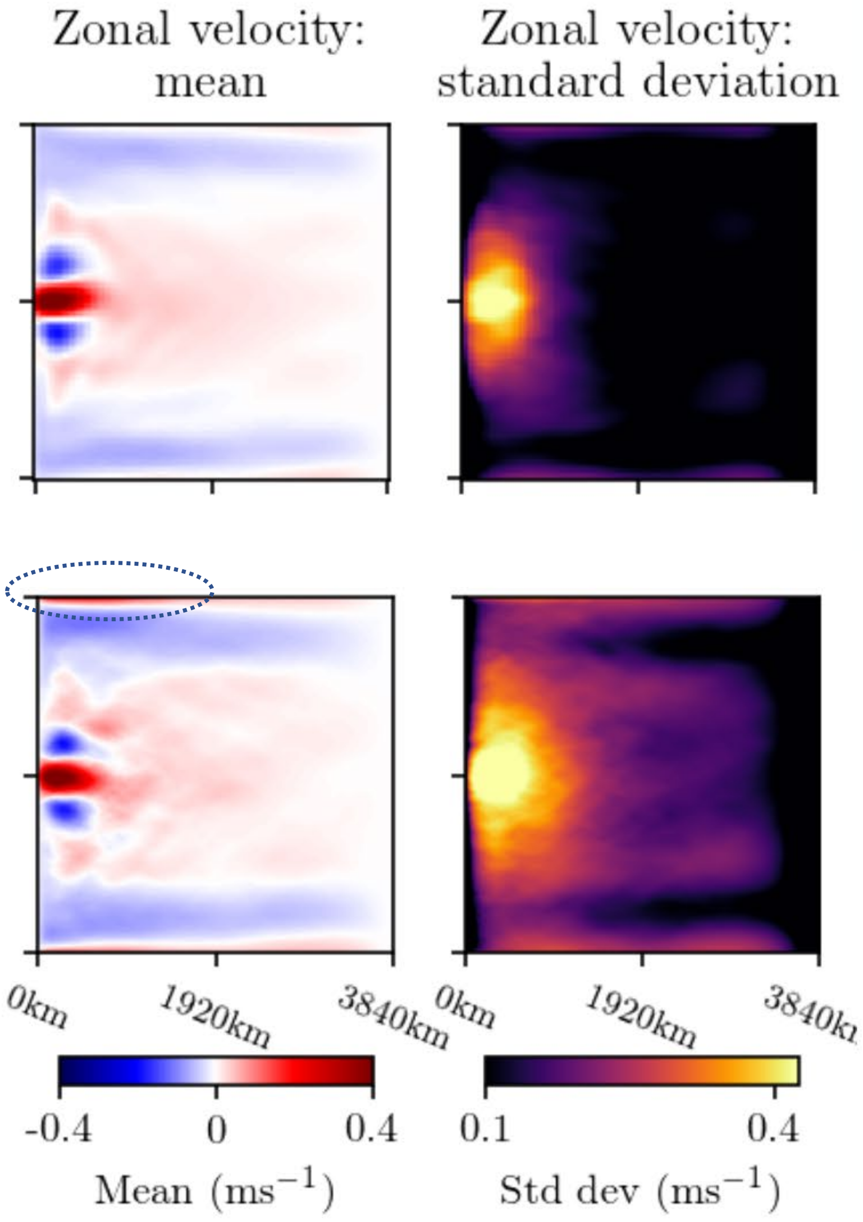
Zanna & Bolton, 2020 (Fig. S1)

Coarse resolution

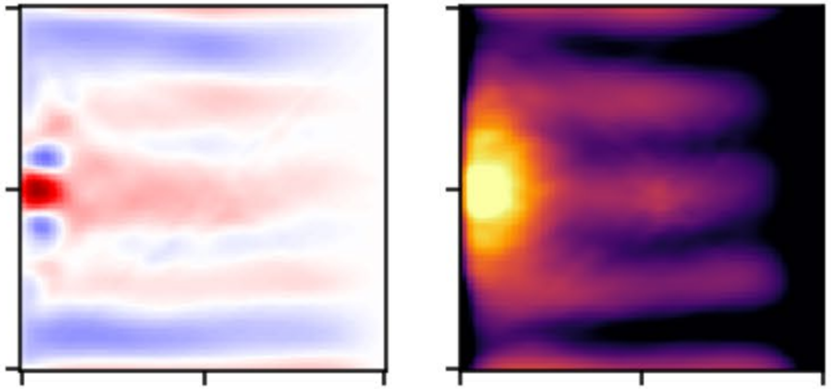
30km

Fine resolution

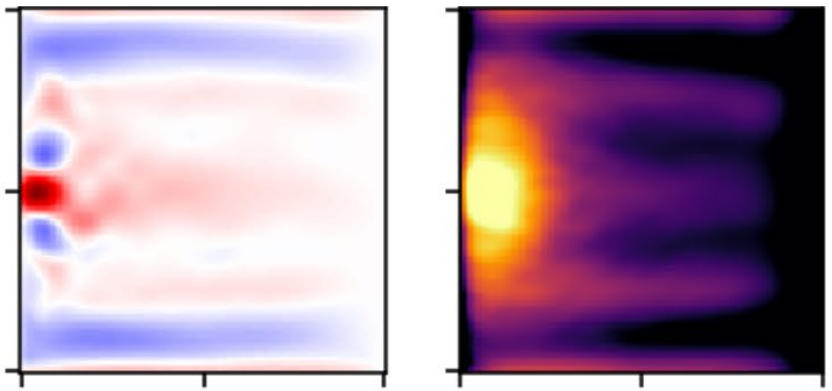
3.75km

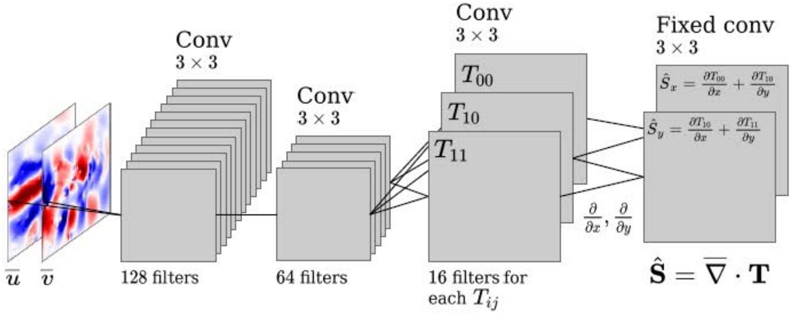


Coarse resolution + eqn discovery



Coarse resolution + NN + conservation





Zanna & Bolton, 2020 (Fig. S10)

High-resolution

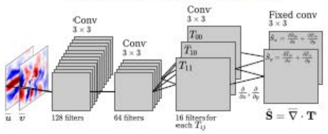
Physics-driven

$$\overline{\nabla} \cdot \begin{pmatrix} -\zeta D & \zeta D \\ \zeta D & \zeta D \end{pmatrix}$$

Equation-discovery

$$\kappa \overline{\nabla} \cdot \begin{pmatrix} \zeta^2 - \zeta D & \zeta D \\ \zeta D & \zeta^2 + \zeta D \end{pmatrix}$$

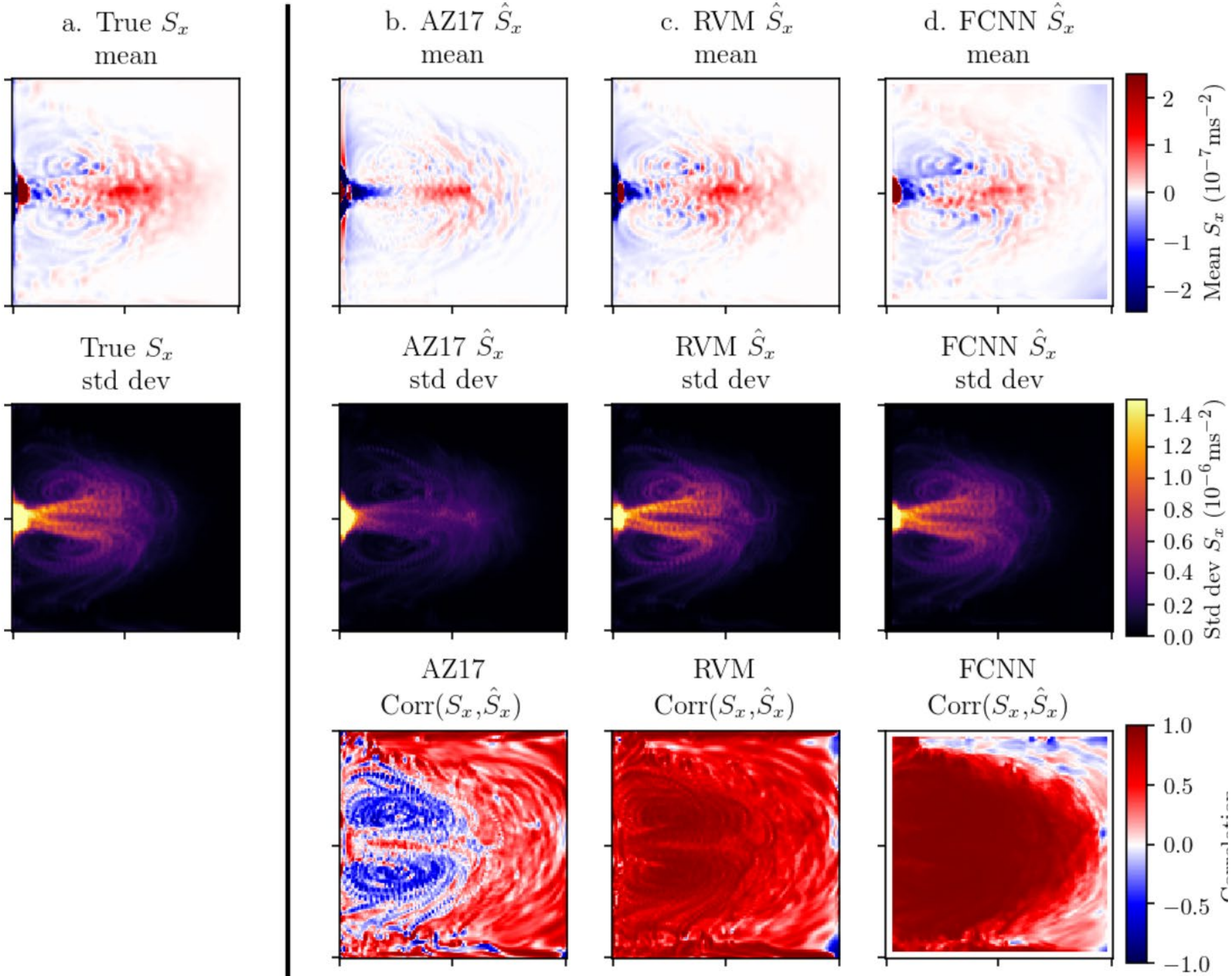
Neural Nets



Mean

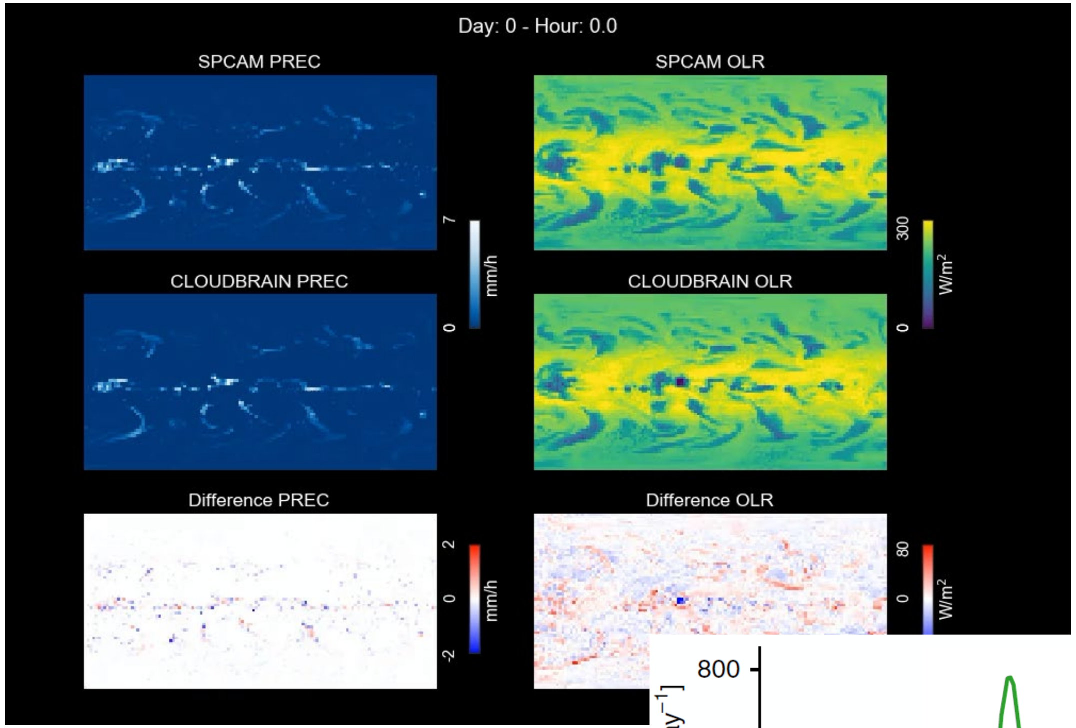
Standard Deviation

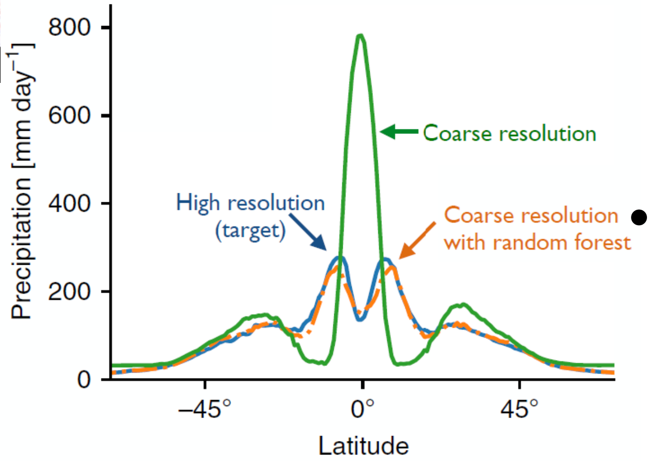
Correlation



Zanna & Bolton, 2020 (Fig. S2)

Atmospheric Convection





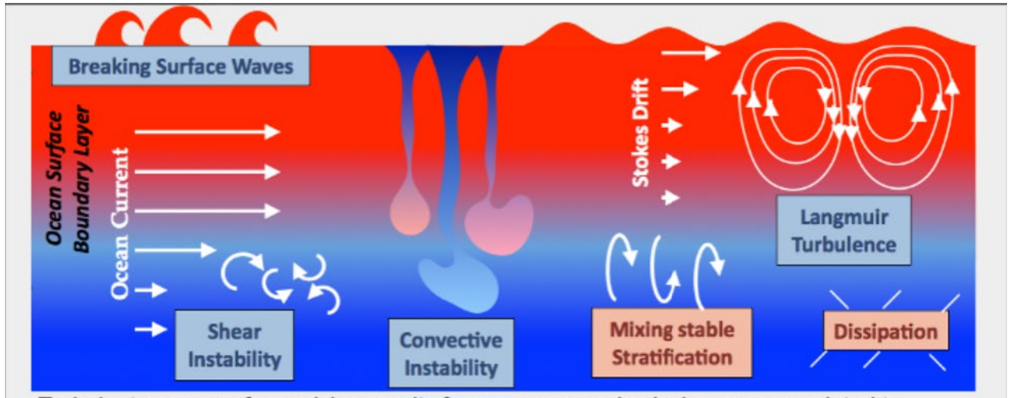
- Train a parameterization by coarse-graining a high-resolution simulation, or learning from SPCAM

1. Accuracy: calculate the subgrid terms exactly by coarse-graining the equations process by process
2. Ensure conservation of energy and water:
   a) Neural network: predict fluxes and sources/sinks (rather than net tendencies) or enforce in architecture/training
   b) Random forest: predictions are averages over training samples

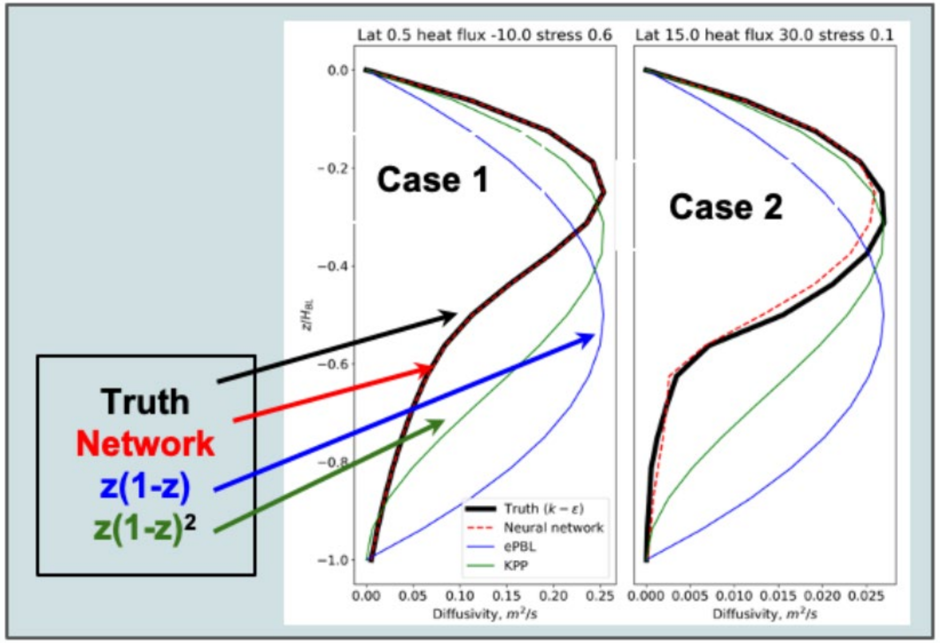
- Stable and accurate simulations using random forests *and* neural nets

Yuval and O'Gorman, Nat. Comm. '20
Yuval, O'Gorman, Hill, GRL, 2021
Gentine et al. GRL 2018
Rasp et al. PNAS 2018
Beucler et al. Phys Rev 2021

Ocean surface boundary layer



Turbulent ocean surface mixing results from numerous physical processes related to winds, buoyancy loss, and waves.



- ePBL uses an energetically constrained approach for turbulent fluxes in the surface boundary layer
- The energetic constraint provides numerical and physical stability benefits for climate simulation
- Neural networks to parameterize variation in the turbulent mixing profile that **maintains previously established energetic constraints**
- Embedding the network in the existing approach improves the turbulent flux profiles while maintaining the stability benefits
  - Improves on traditional profiles

Reichl & Hallberg, 2018 Reichl & Li, 2019

Summary

- M²LInES
- To date: implementation of ML parameterizations mostly evaluated in idealized models … we're working on getting them into GCMs
- Quite a few issues arise, some mundane, other's tricky
  - Stability not guaranteed
  - Online model data stencil needed for ML parameterizations can be wider than we are used to
  - Boundary conditions require more work
- Equation discovery
  - Addresses many of the above problems
  - And aids in interpretation