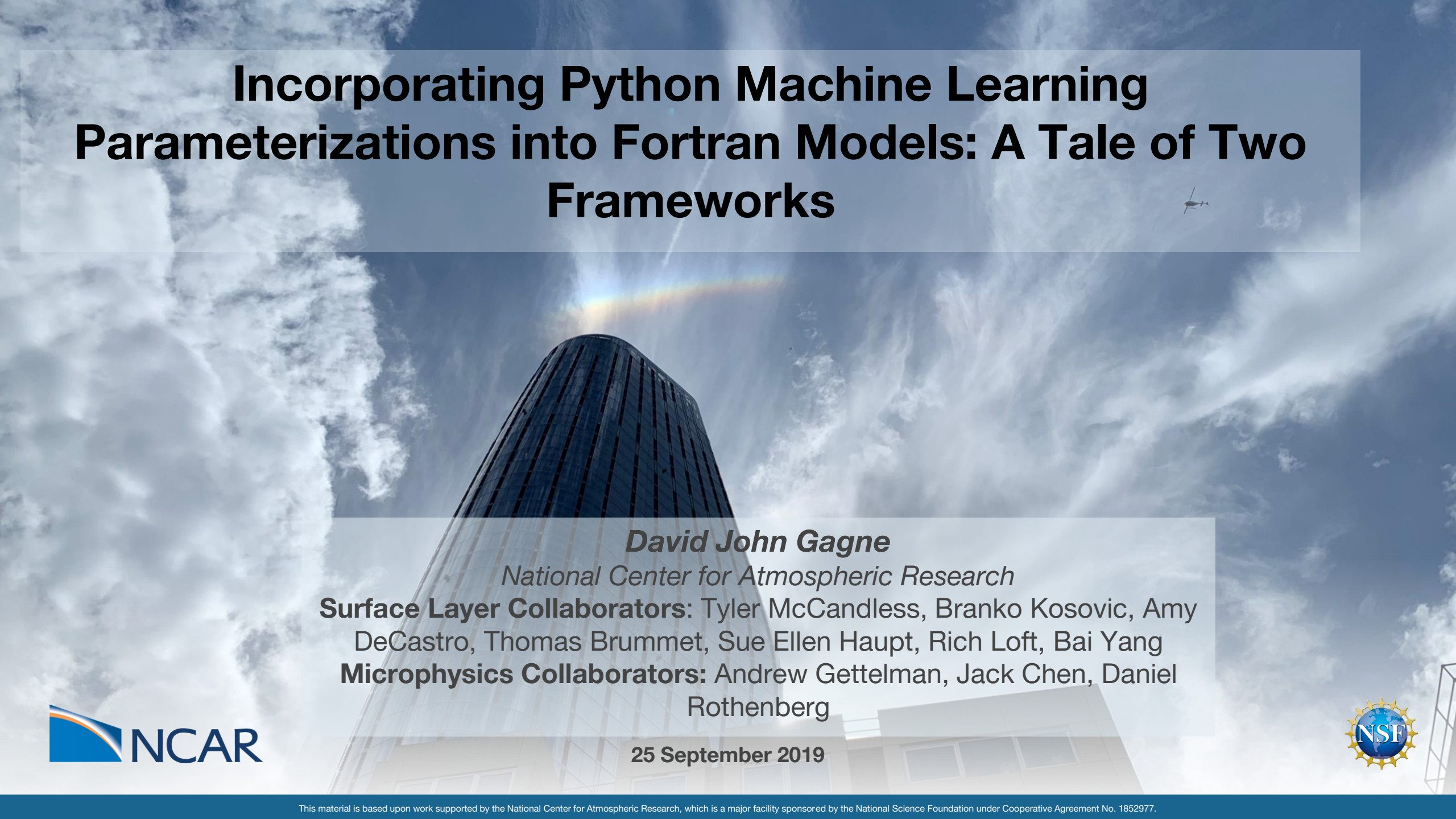
Incorporating Python Machine Learning Parameterizations into Fortran Models: A Tale of Two Frameworks



David John Gagne

National Center for Atmospheric Research

Surface Layer Collaborators: Tyler McCandless, Branko Kosovic, Amy DeCastro, Thomas Brummet, Sue Ellen Haupt, Rich Loft, Bai Yang

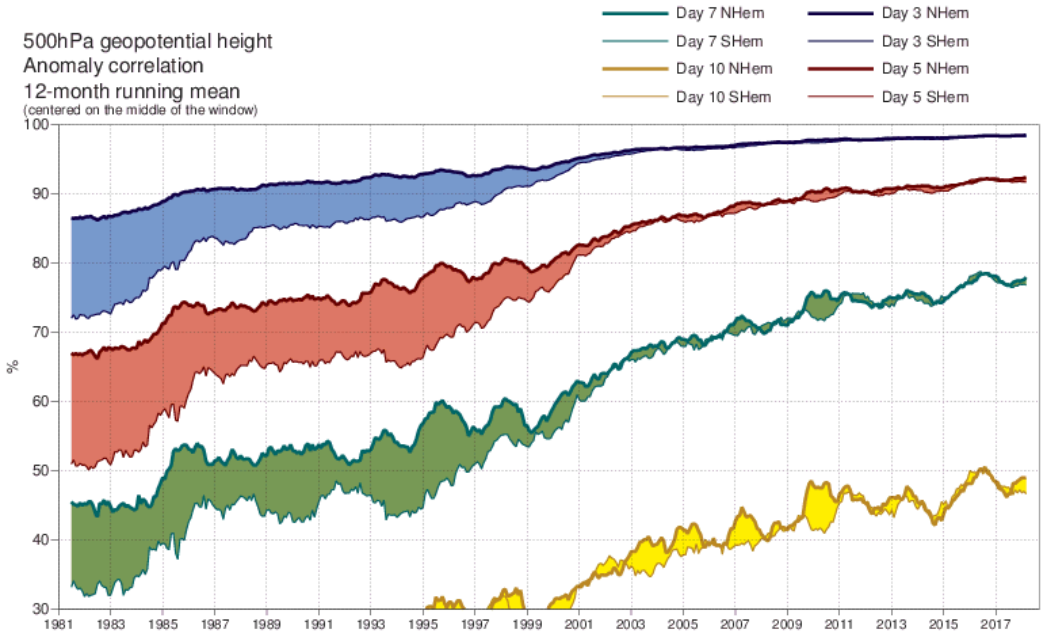
Microphysics Collaborators: Andrew Gettelman, Jack Chen, Daniel Rothenberg

25 September 2019

This material is based upon work supported by the National Center for Atmospheric Research, which is a major facility sponsored by the National Science Foundation under Cooperative Agreement No. 1852977.

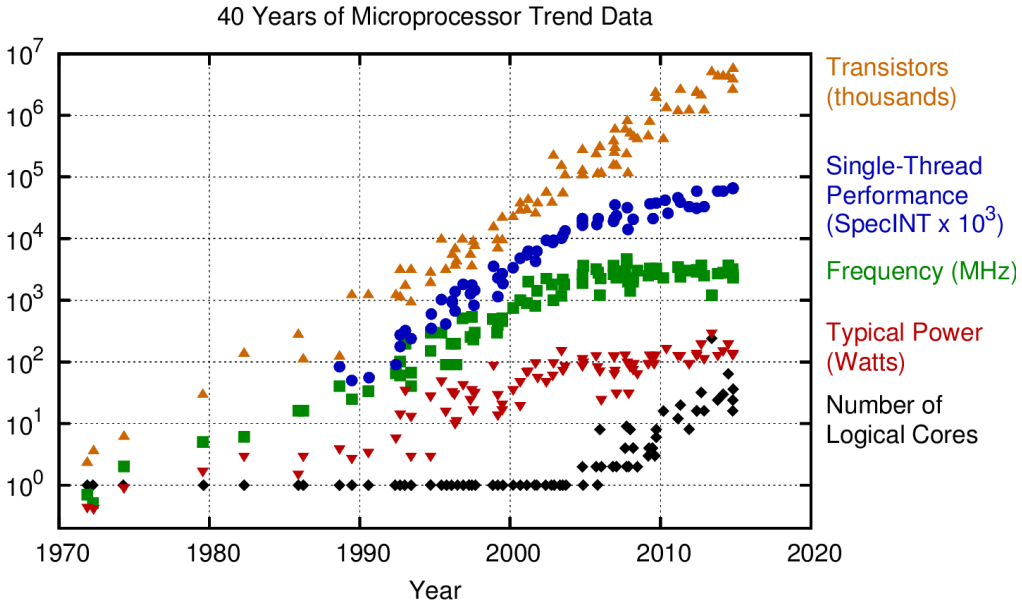
Motivation

It was the best of times.



- Numerical Weather Prediction model skill continues to increase
- Decision makers trust meteorologists more than ever

It was the worst of times.



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

- Both serial and parallel processing limits are limiting the further scalability of existing model codes
- Weather and climate models will need a new design paradigm to realize higher resolution and complexity

Contact: dgagne@ucar.edu, @DJGagneDos

Goals

- Machine learning offers a computationally efficient, expressive, and scalable framework for representing complex physical processes in numerical models
- **Problem**: machine learning libraries are written in Python or C++, but numerical models are generally written in Fortran
- **Goal:** Evaluate how machine learning models perform physically and computationally at representing subgrid physical processes with two frameworks
- **Surface Layer**: machine learning parameterization trained from observations to minimize assumptions required by Monin-Obukhov similarity theory
- **Microphysics**: machine learning emulator trained on simulation data from a bin microphysics process is inserted into bulk microphysics scheme

Esteemed parameterization with a complex past

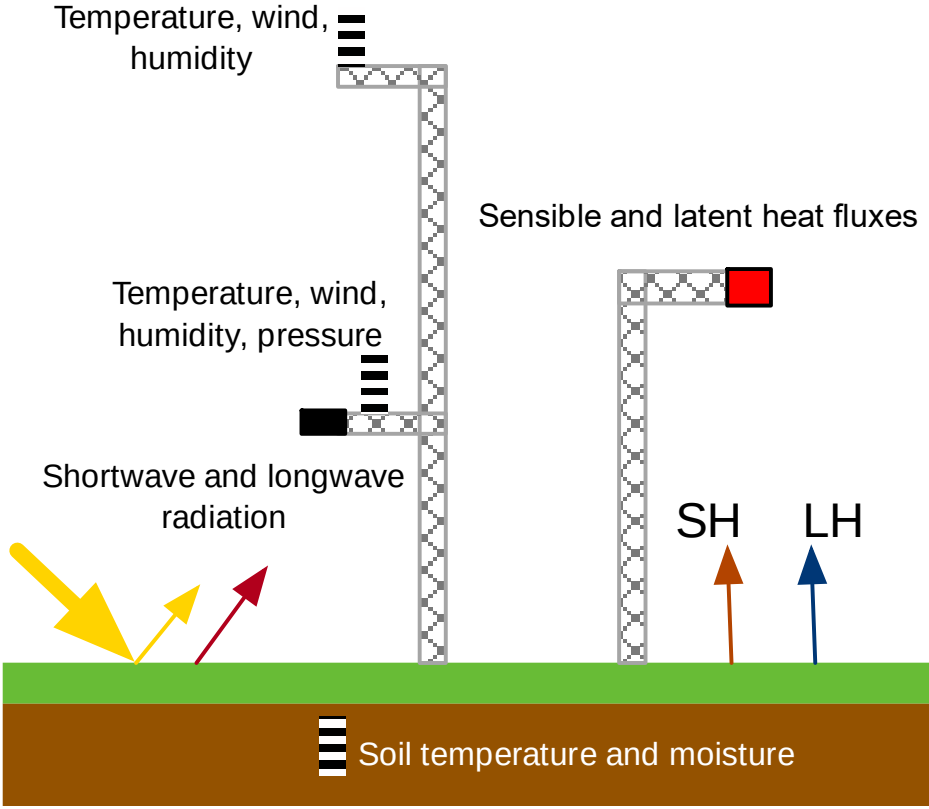
https://www.pinterest.com/pin/260012578456645879/?lp=true

Neural network emulator good enough to fool the guards?

Contact: dgagne@ucar.edu, @DJGagneDos

Motivation: Observed and Modeled Surface Layer

Observed Surface Layer



Model Surface Layer

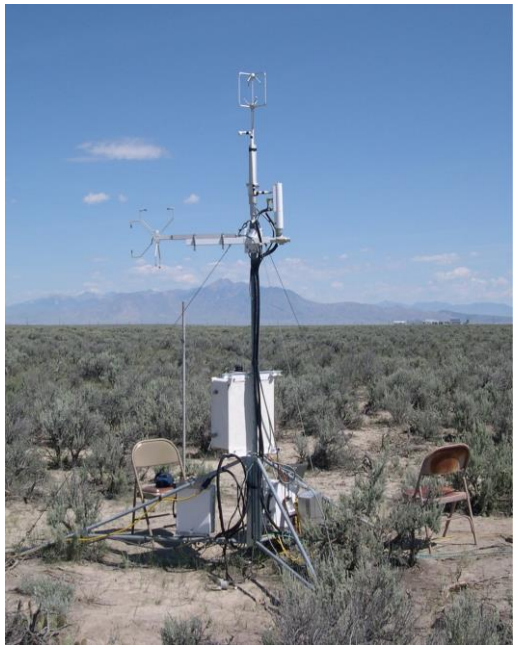
- Transfer of energy between the land surface and atmosphere is driven by radiation and sensible and latent heat fluxes
- Sensible and latent heat fluxes occur through unresolved turbulent eddies
- Processes currently represented in all numerical models through surface layer parameterization and land surface model
- Parameterization use assumptions of Monin-Obukhov similarity theory

Contact: dgagne@ucar.edu, @DJGagneDos

Motivation: Surface Layer Methods

- MO similarity theory depends on empirical “stability functions” fit to data from short field campaigns
- Field campaign data likely does not capture the full range of possible flux-gradient relationships that can occur
- Therefore, we use two sites with multiyear observational records for both weather and flux data to train machine learning models
- Fit random forests and neural networks to each site to predict friction velocity and scale terms to calculate sensible heat flux and latent heat flux
- Avoiding explicit calculation of stability functions

Cabauw, Netherlands
KNMI Mast
213 m tower
Data from 2003-2017



Scoville, Idaho, USA
FDR Tower
Flux tower
Data from 2015-2017

Contact: dgagne@ucar.edu, @DJGagneDos

Input and Output Variables

Input Variables	Heights (Idaho/Cabauw)
Potential Temperature Gradient (K)	Skin to 10 m, 15 m/20 m
Mixing Ratio Gradient (g kg^{-1})	Skin to 10 m, 20 m
Wind Speed (m s^{-1})	10 m, 15 m/20 m
Bulk Richardson number	10 m- 0 m
Moisture Availability (%)	5 cm/3 cm
Solar Zenith Angle (degrees)	0 m

Output equations

$$\tau = \rho u_*^2$$

$$H = -\rho c_p u_* \theta *$$

$$LH = L_e \rho u_* q_*$$

Predictands

u*=Friction velocity

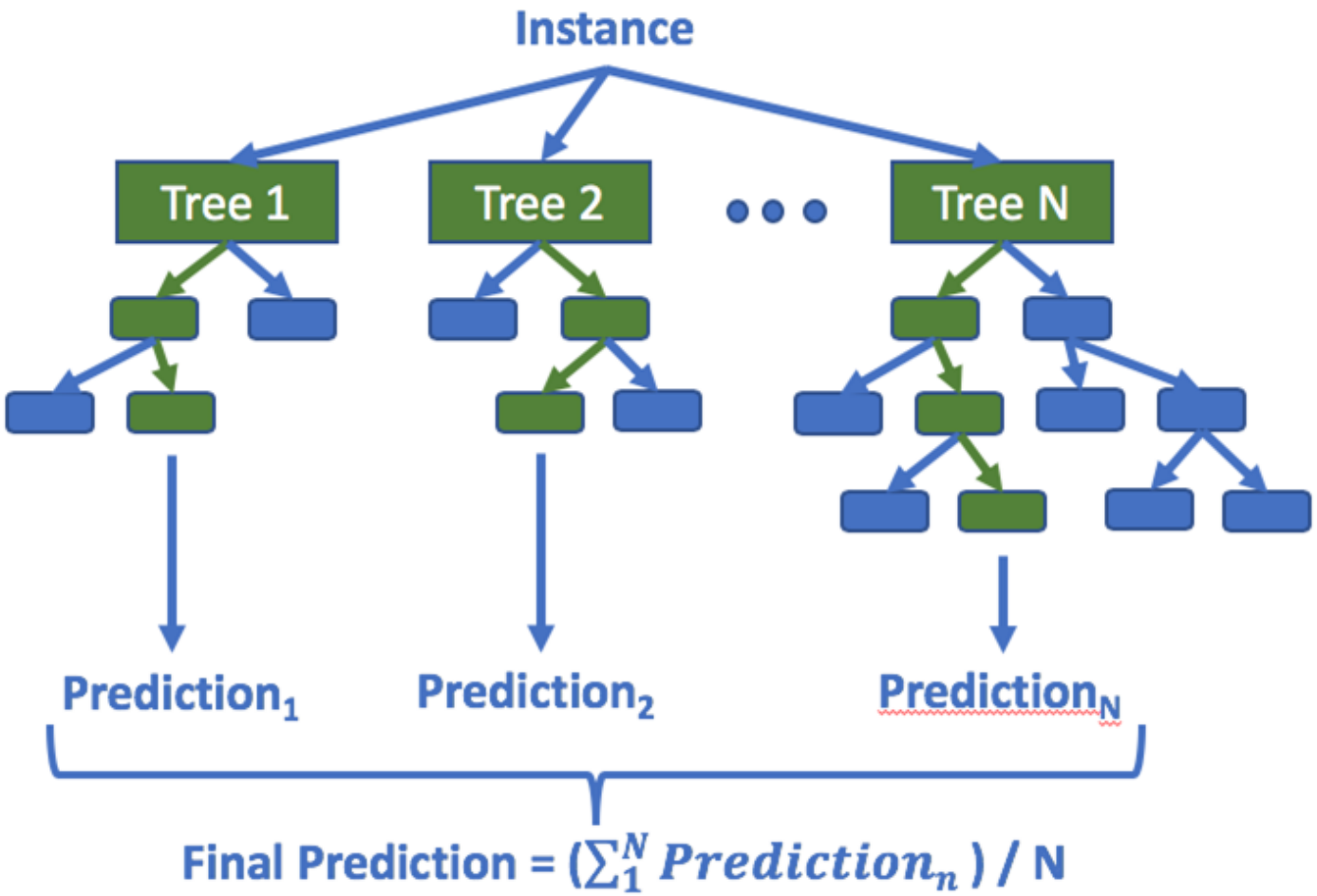
θ*=Temperature scale

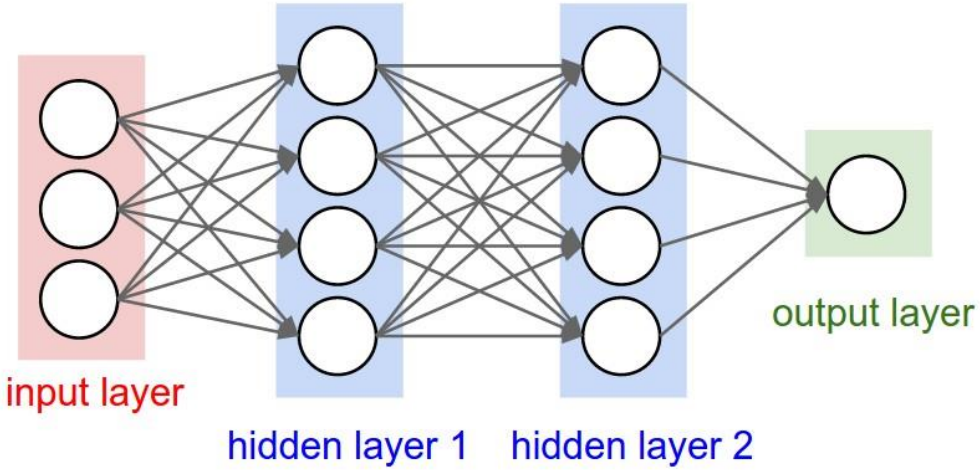
q*=Moisture scale

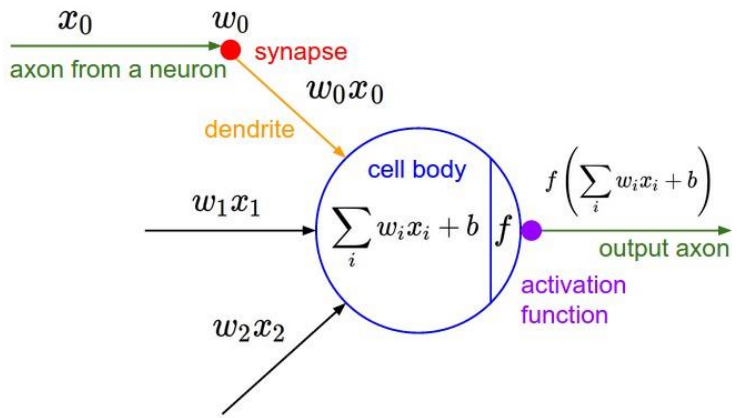
ML Procedure

1. Train ML models on observations
2. Plug in ML models to WRF in surface layer parameterization
3. Surface layer parameterization derives necessary outputs from ML predictions

Random Forest and Neural Network





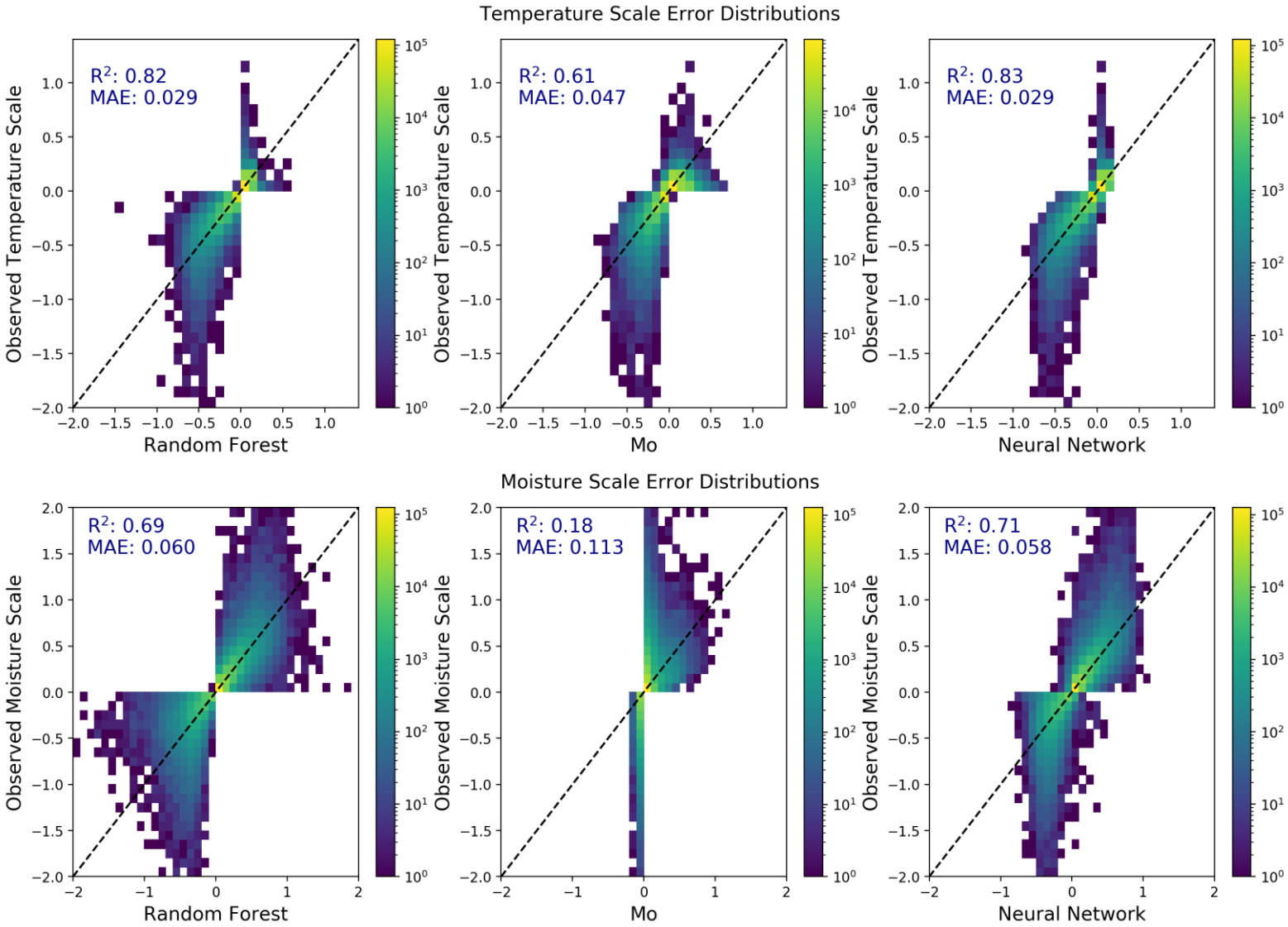


Key hyperparameter: max_leaf_nodes=1024

Images from http://cs231n.github.io/convolutional-networks/

Contact: dgagne@ucar.edu, @DJGagneDos

Offline Results: Temperature and Moisture Scale



Contact: dgagne@ucar.edu, @DJGagneDos

Cross-Testing ML Models

	R^2			MAE		
Idaho Test Dataset	**Friction Velocity**	**Temperature Scale**	**Moisture Scale**	**Friction Velocity**	**Temperature Scale**	**Moisture Scale**
MO Similarity	0.85	0.42		0.077	0.203	
RF Trained on Idaho	0.91	0.80	0.41	0.047	0.079	0.023
RF Trained on Cabauw	0.88	0.76	0.22	0.094	0.139	0.284

	R^2			MAE		
Cabauw Test Dataset	**Friction Velocity**	**Temperature Scale**	**Moisture Scale**	**Friction Velocity**	**Temperature Scale**	**Moisture Scale**
MO Similarity	0.90	0.61	0.18	0.115	0.062	0.135
RF Trained on Cabauw	0.93	0.82	0.73	0.031	0.030	0.055
RF Trained on Idaho	0.90	0.77	0.49	0.074	0.049	0.112

Results Courtesy Tyler McCandless

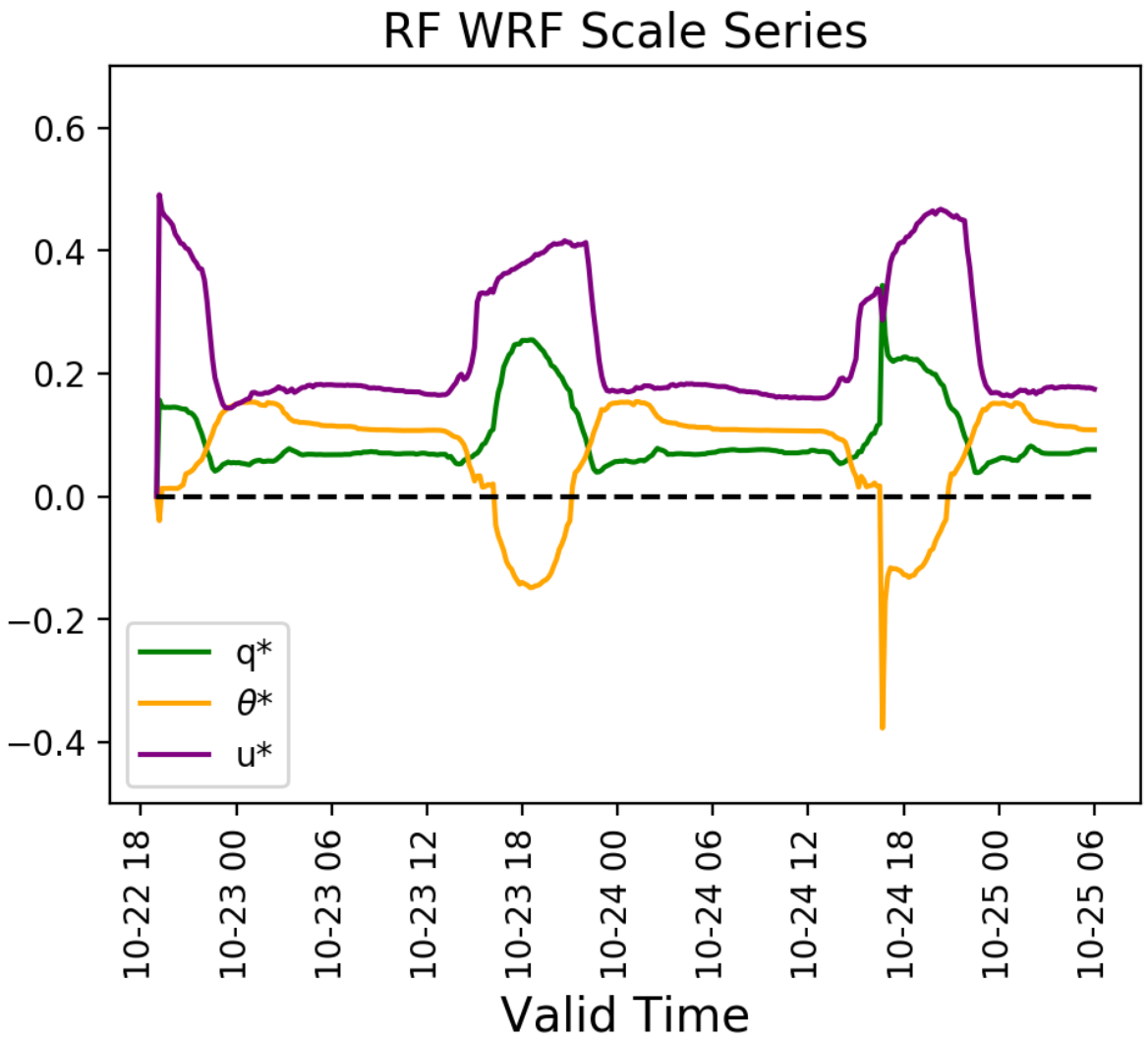
Random Forest Incorporation into WRF

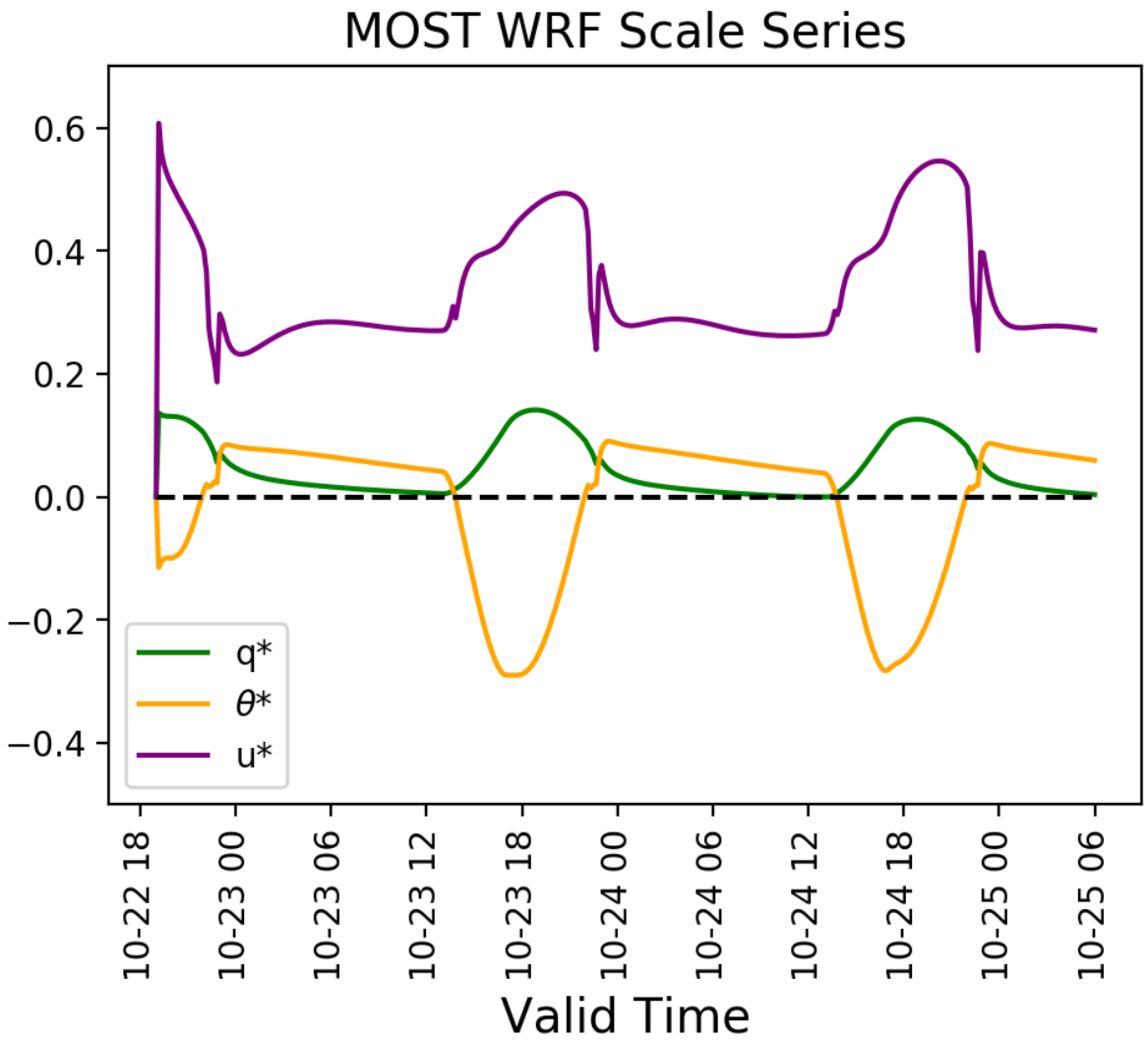
- Save scikit-learn decision trees from random forest to csv files
- Read csv files into Fortran array of decision tree derived types
- Random forest surface layer parameterization
  - Calculate derived input variables for ML models
  - Feed vectors of inputs to random forests for friction velocity, temperature scale, moisture scale
  - Calculate fluxes, exchange coefficients and surface variables
- Test with WRF Single Column Model on idealized case study
  - Using GABLS II constant forcing
  - YSU Boundary Layer
  - Slab Land Surface Model

```
type decision_tree
   integer :: nodes
   integer, allocatable :: node(:)
   integer, allocatable :: feature(:)
   real(kind=8), allocatable :: threshold(:)
   real(kind=8), allocatable :: tvalue(:)
   integer, allocatable :: children_left(:)
   integer, allocatable :: children_right(:)
   real(kind=8), allocatable :: impurity(:)
end type decision_tree

function decision_tree_predict(input_data_tree, tree) result(tree_prediction)
    real(kind=8), intent(in) :: input_data_tree(:)
    type(decision_tree), intent(in) :: tree
    integer :: node
    real(kind=8) :: tree_prediction
    logical :: not_leaf
    logical :: exceeds
    node = 1
    tree_prediction = -999
    not_leaf = .TRUE.
    do while (not_leaf)
       if (tree%feature(node) == -2) then
          tree_prediction = tree%tvalue(node)
          not_leaf = .FALSE.
       else
          exceeds = input_data_tree(tree%feature(node) + 1) > tree%threshold(node)
          if (exceeds) then
             node = tree%children_right(node) + 1
          else
             node = tree%children_left(node) + 1
          end if
       end if
    end do
end function decision_tree_predict
```

CASES-II WRF Idealized Single Column Model Comparison

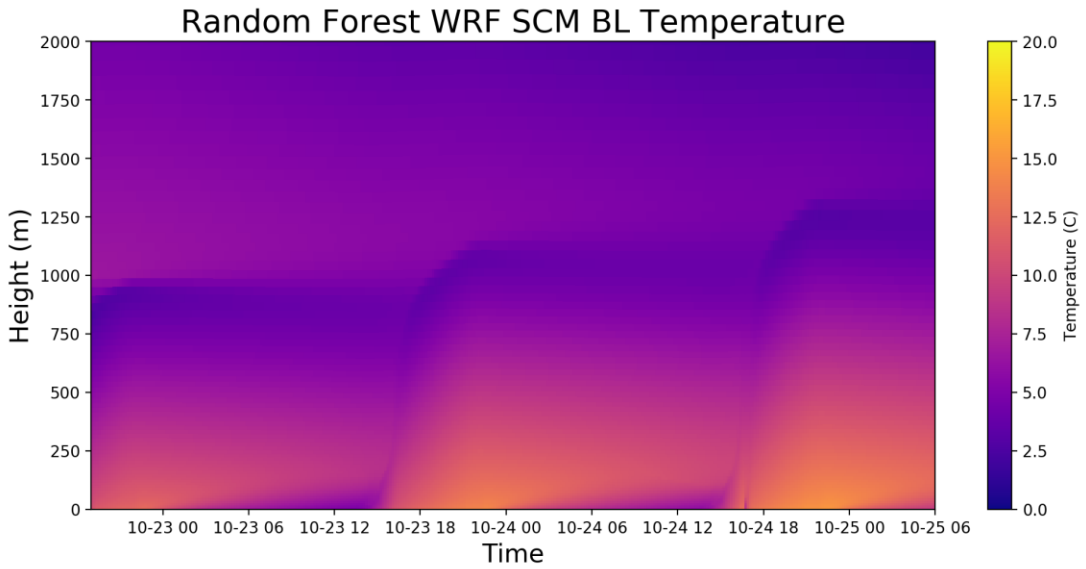


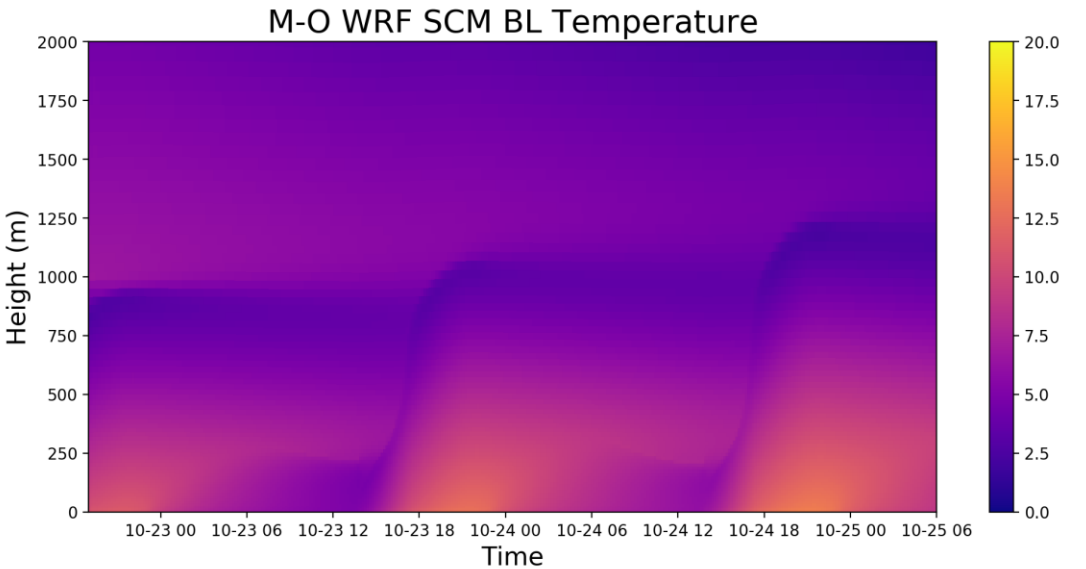


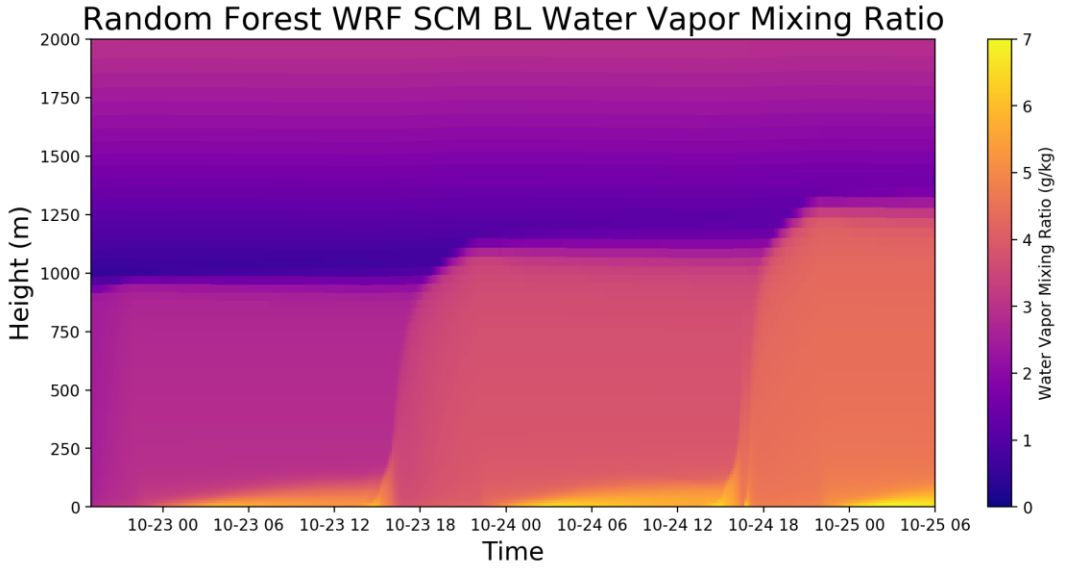
NCAR | UCAR

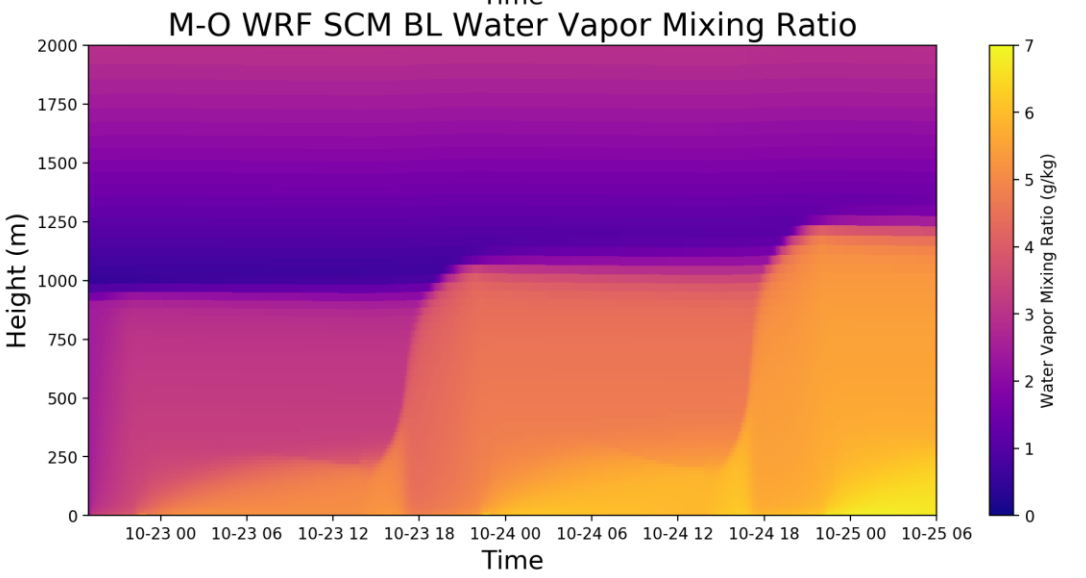
Contact: dgagne@ucar.edu, @DJGagneDos

WRF Idealized Single Column Model Comparison



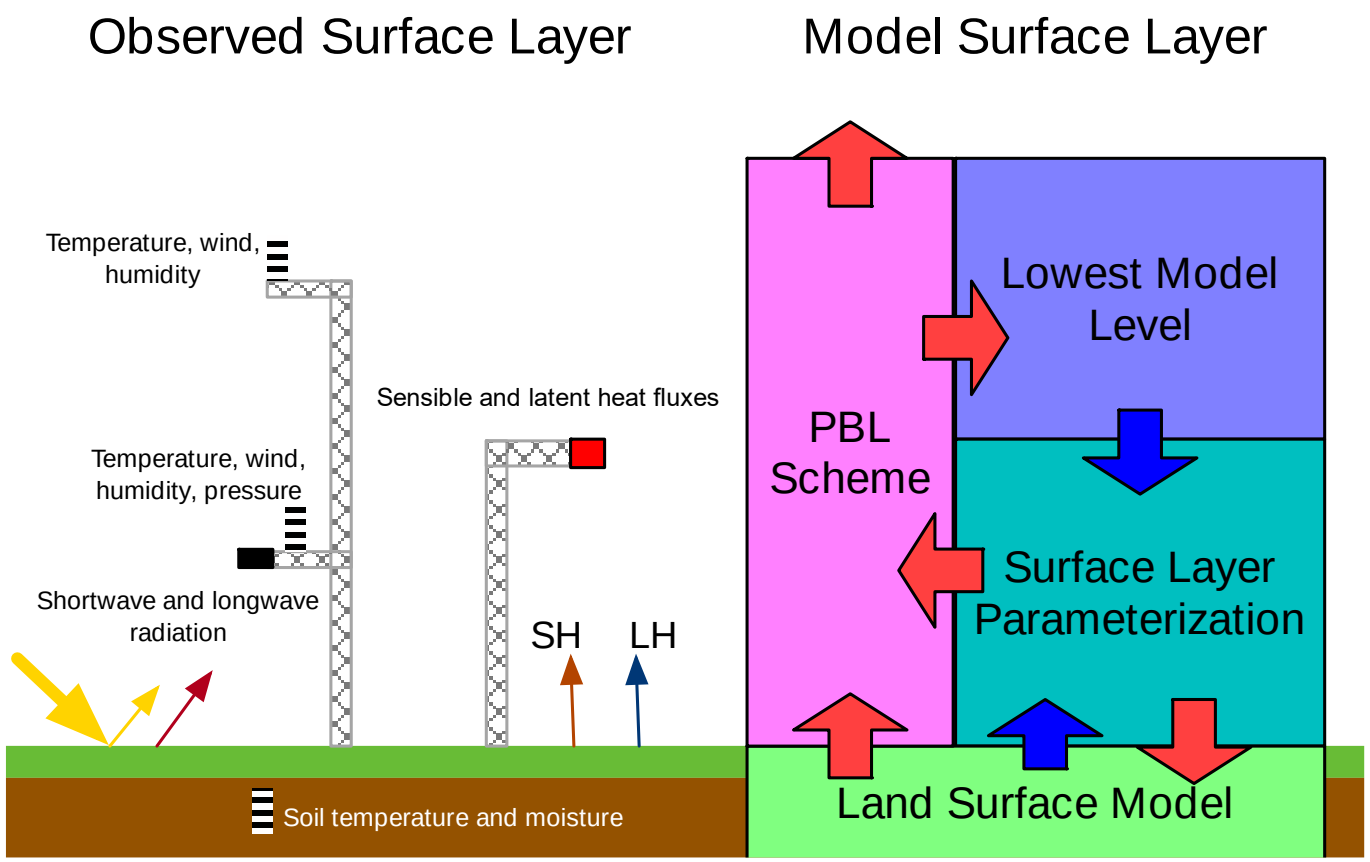






NCAR UCAR | Contact: dgagne@ucar.edu, @DJGagneDos

Surface Layer Takeaways



- Initial results appear promising but require further tuning and retraining to fix inconsistencies
- May need to ensure consistencies among friction velocity, temperature scale, and moisture scale
- We may need to modify land surface model and PBL scheme because of their dependencies on MO

Contact: dgagne@ucar.edu, @DJGagneDos

Motivation

Precipitation formation is a critical uncertainty for weather and climate models.

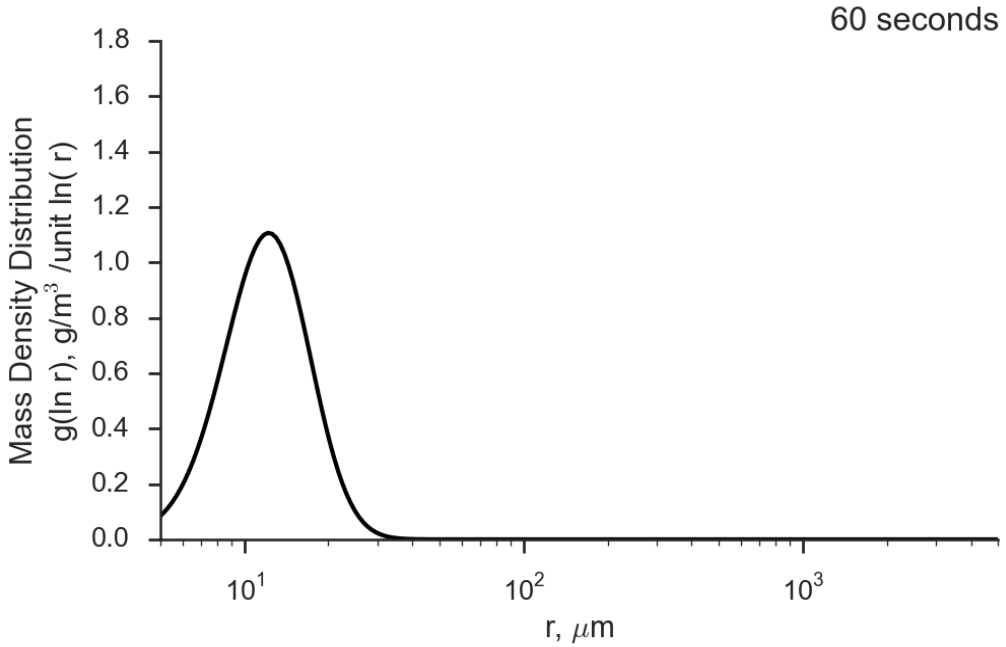
Different sizes of drops interact to evolve from small cloud drops to large precipitation drops.

Detailed codes (right) are too expensive for large scale models, so empirical approaches are used.

Let's emulate one (or more)

Goal: put a detailed treatment into a global model and emulate it using ML techniques.

Good test of ML approaches: can they reproduce a complex process, but with simple inputs/outputs?



Superdroplet model output animation
Credit: Daniel Rothenberg

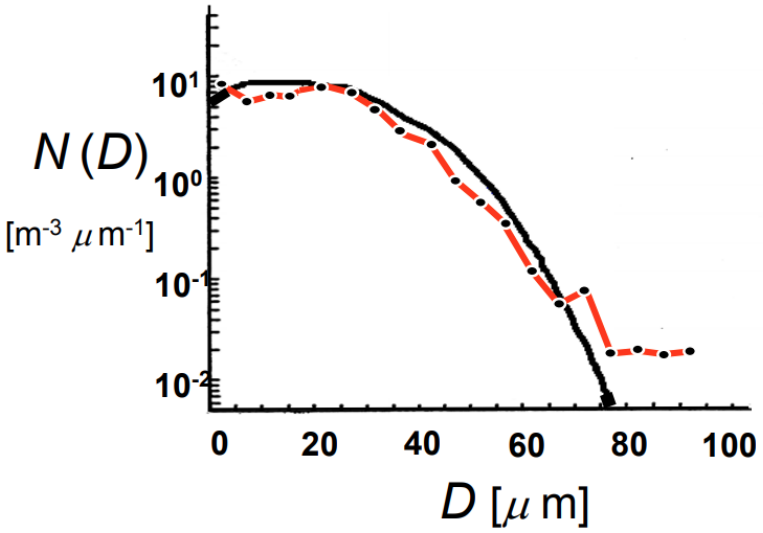
Contact: dgagne@ucar.edu, @DJGagneDos

Bulk vs. Bin Microphysics

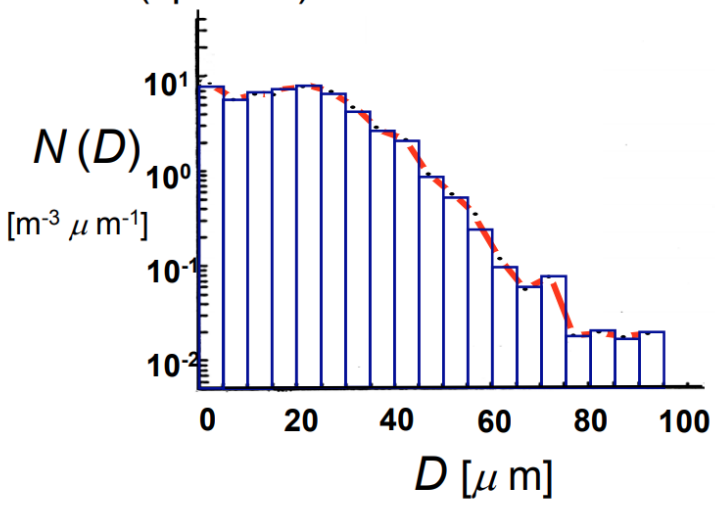
Bulk scheme (MG2 in CAM6):
Calculate warm rain formation processes with a semi-empirical particle size distribution (PSD) based on exponential fit to LES microphysics runs.

Bin Scheme (Tel Aviv University (TAU) in CAM6):
Divide particle sizes into bins and calculate evolution of each bin separately. Better representation of interactions but much more computationally expensive.

Bulk: $N(D) = N_0 D^{\alpha} e^{-\lambda D}$

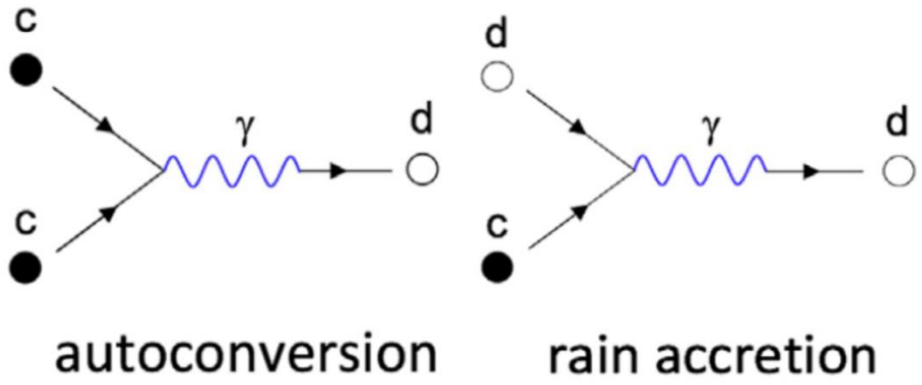


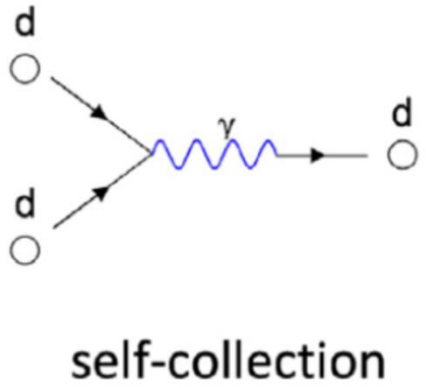
Bin-resolving: (spectral) $N(D) = \sum_{i=1}^{I} N_i$



Contact: dgagne@ucar.edu, @DJGagneDos

Cloud to Rain Processes





d: rain drop
c: cloud droplet
CCN: cloud condensation nuclei

Cloud droplets grow into rain droplets through 3 processes:

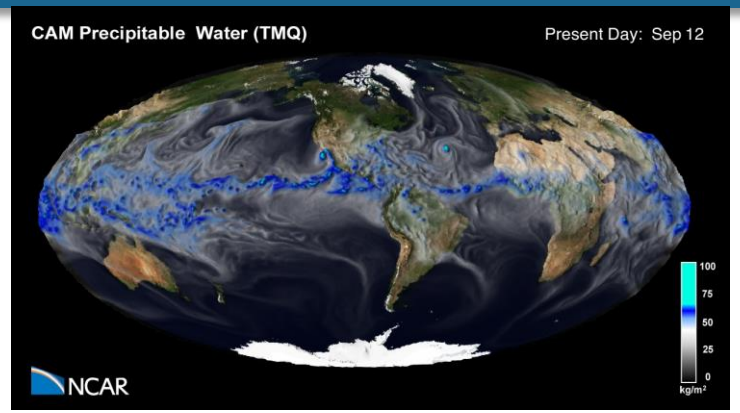
Autoconversion: cloud droplets collide in a chain reaction to form rain drops
dqc/dt < 0, dqr/dt > 0
dNc/dt < 0, dNr/dt > 0

Rain Accretion: rain drops collide with cloud droplets
dqc/dt < 0, dqr/dt > 0
dNc/dt < 0, dNr/dt = 0

Self-Collection: rain drops collide with other raindrops
dqc/dt = 0, dqr/dt = 0
dNc/dt = 0, dNr/dt < 0

Contact: dgagne@ucar.edu, @DJGagneDos

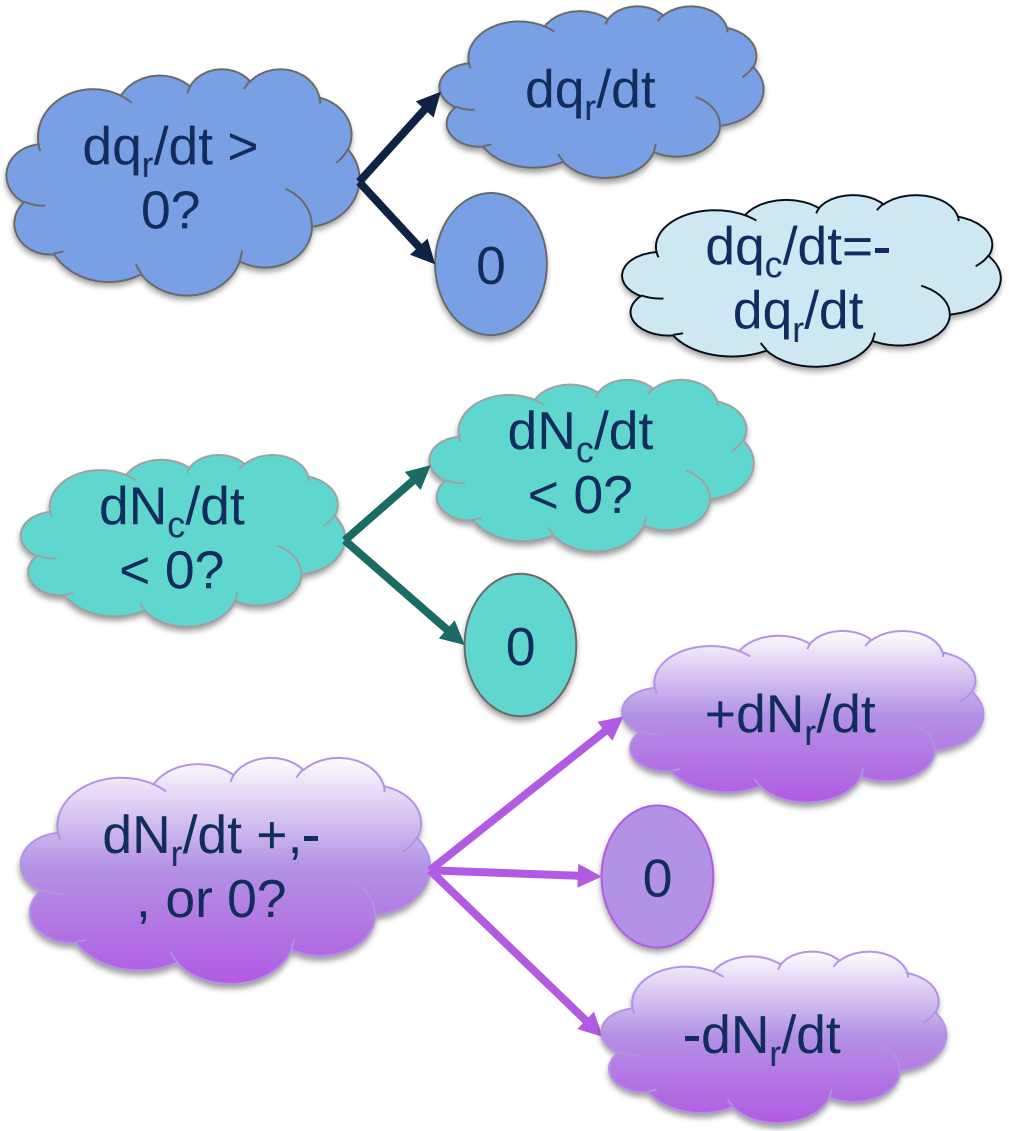
Microphysics Emulator Procedure



1. Run CAM6 for 2 years with fixed forcing from other CESM components
2. Output global microphysics input and output fields every 25 hours
3. Filter and subsample data to find grid points with realistic amounts of cloud water

Inputs

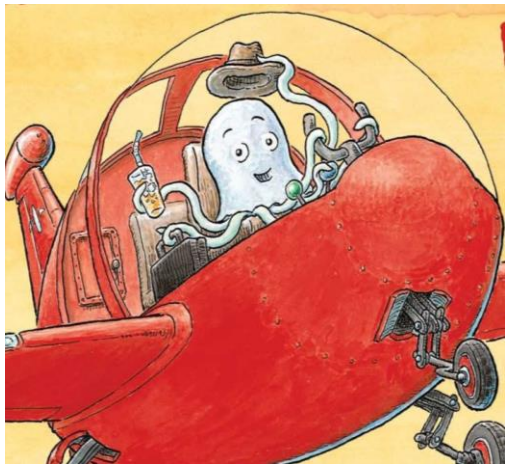
Cloud water mixing ratio (q_c)
Cloud water number concentration (N_c)
Rain water mixing ratio (q_r)
Rain water number concentration (N_r)
Air density
Cloud Liquid Slope Parameter
Rain Water Slope Parameter
Rain Water Intercept Parameter
Cloud Fraction
Precipitation Fraction
Spectra shape for cloud liquid water



NCAR UCAR | Contact: dgagne@ucar.edu, @DJGagneDos

Neural Network Settings

- 3 classifier networks, 4 regression networks
- 82,327 weights total (~16 jellyfish brains)
- Dense Neural Network Hyperparameters
  - 4 layers
  - 60 neurons per hidden layer
  - 11,761 total weights
  - Rectified Linear Unit (ReLU) activation functions
  - Batch Size: 4096 examples
  - Learning Rate: 1.0e-3
  - L2 Norm Weight: 1.0e-4
  - Training Period: 10 epochs
  - Loss function: Mean squared error (regression), cross-entropy (class)



Artistic rendering of neural network interface

Image from J. Fardell, 2001: *Jeremiah Jellyfish Flies High*

Contact: dgagne@ucar.edu, @DJGagneDos

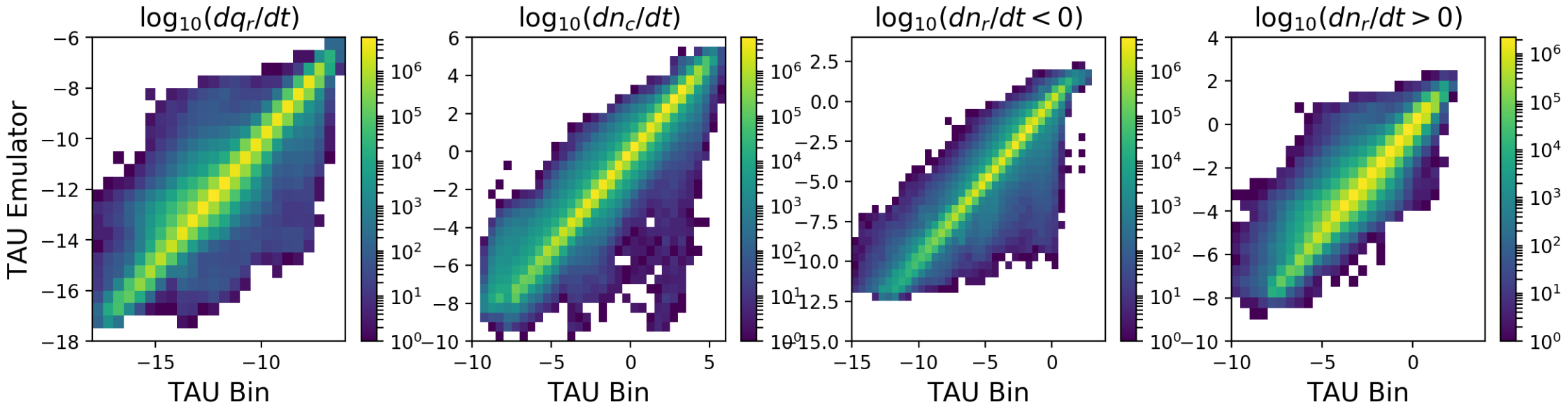
Classifier Results

	TAU QR 1	TAU QR 0	Total
NN QR 1	**41.7%**	0.7%	42.4%
NN QR 0	0.8%	**56.8%**	57.6%
Total	42.5%	57.5%	**98.4%**

	TAU NC 1	TAU NC 0	Total
NN NC 1	**52.9%**	0.5%	53.4%
NN NC 0	0.2%	**46.3%**	46.5%
Total	53.1%	46.8%	**99.3%**

	TAU NR -1	TAU NC 0	TAU NR 1	Total
NN NR -1	**35%**	0.0%	0.4%	35.4%
NN NR 0	0.1%	**43.1%**	0.3%	43.5%
NN NR 1	0.2%	0.5%	**20.4%**	21.1%
Total	35.3%	43.6%	21.1%	**98.5%**

Microphysics 2D Histogram Results



Output	R^2	MAE	Hellinger
dq_r/dt	0.991	0.095	4.53e-4
dn_c/dt	0.995	0.112	1.49e-3
$dn_r/dt < 0$	0.995	0.081	6.04e-4
$dn_r/dt > 0$	0.978	0.178	1.18e-3

Contact: dgagne@ucar.edu, @DJGagneDos

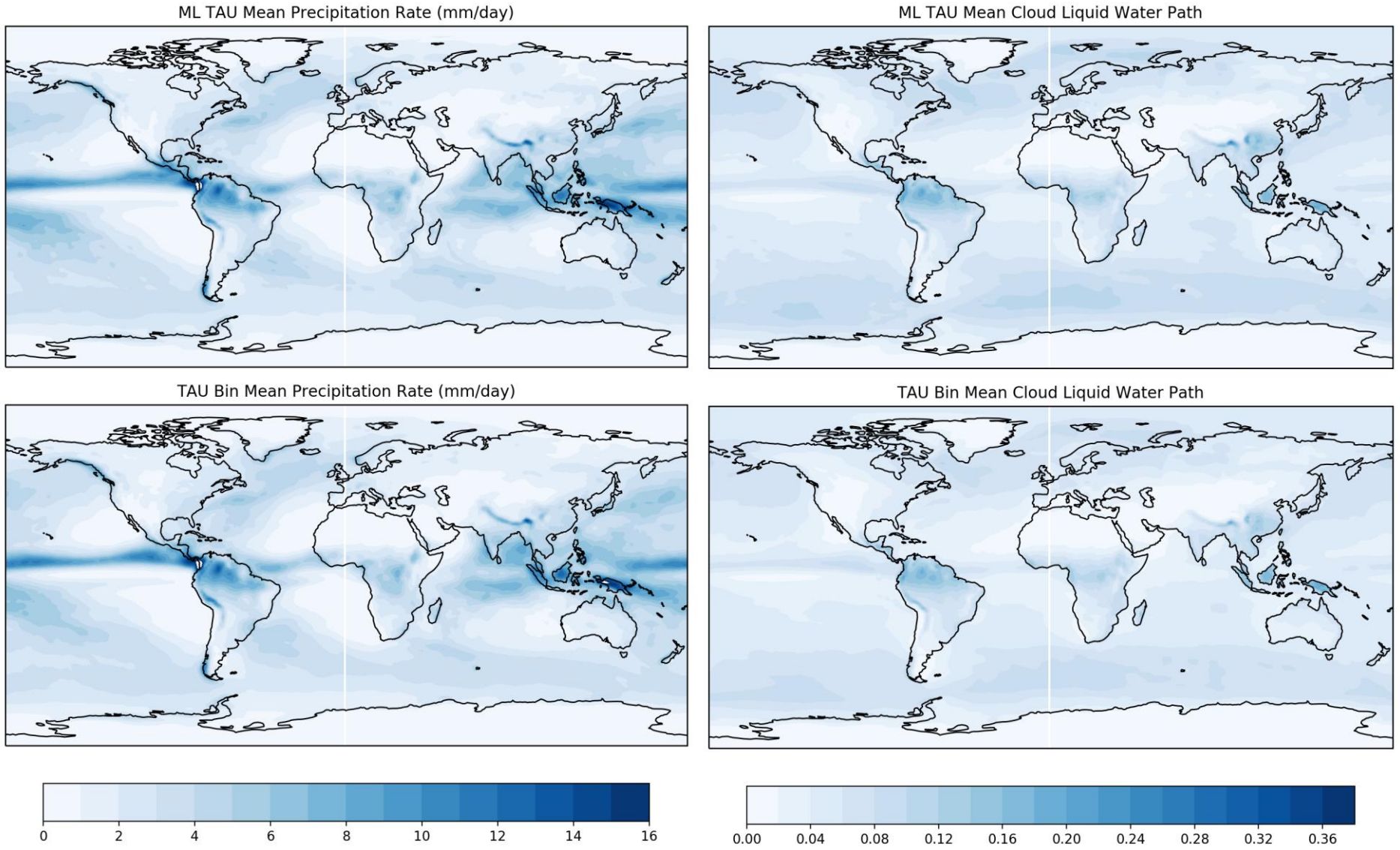
ML Integration with Numerical Models

- Problem: Atmospheric models are written in Fortran, but the ML model codes are written in Python
- Solution: Fortran neural network inference subroutine
- Subroutine Contents
  - Calculate derived input variables
  - Feed inputs into ML models
  - Calculate diagnostics from ML output
- Advantages
  - No outside library dependencies
  - ML models can be switched out easily when more data are available
- Disadvantage
  - More limited ML functionality/optimization compared with community ML models

```
type Dense
    integer :: input_size
    integer :: output_size
    real(kind=8), allocatable :: weights(:, :)
    real(kind=8), allocatable :: bias(:)
    character(len=10) :: activation
end type Dense
```


Contact: dgagne@ucar.edu, @DJGagneDos

CAM Run with Machine Learning Emulator

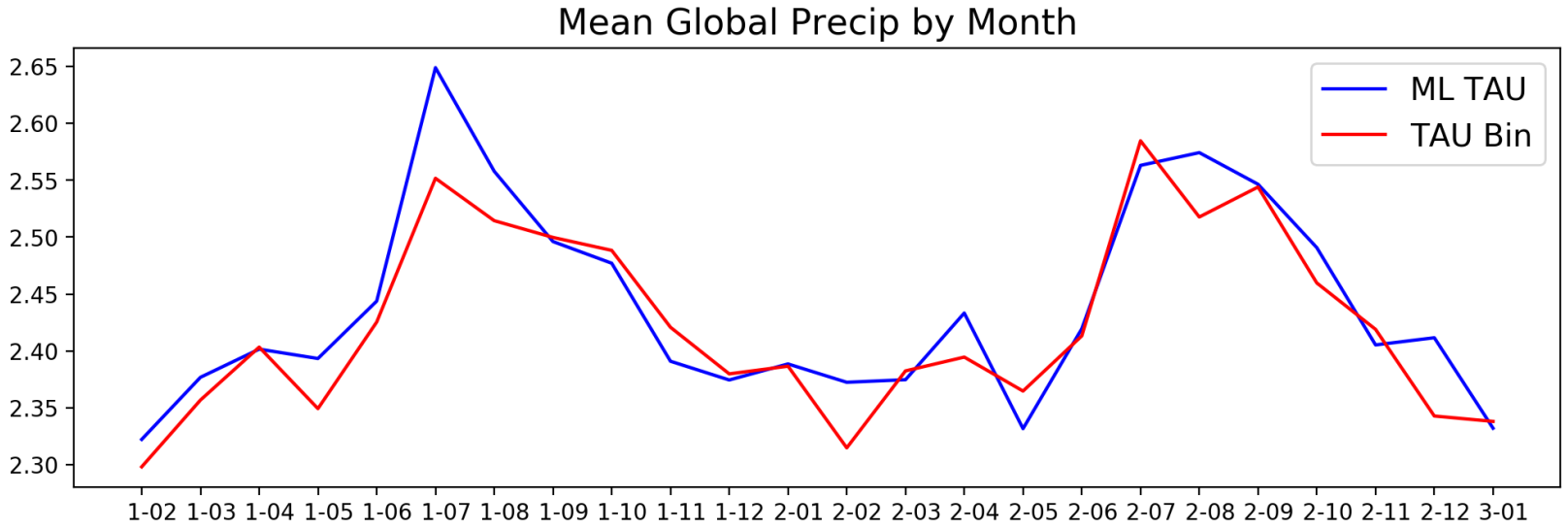


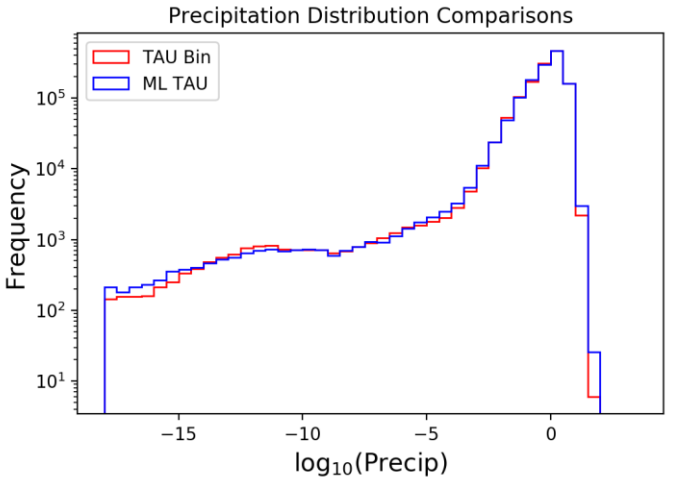
- ML runs at roughly same speed as CAM with MG2
- CAM with TAU is 3x slower than CAM with MG2
- CAM with ML emulator and training climate runs for 9 years
- ML emulator and +4C SST: 4.5 years before blowup
- ML emulator and preindustrial aerosols: 18 months before blowup

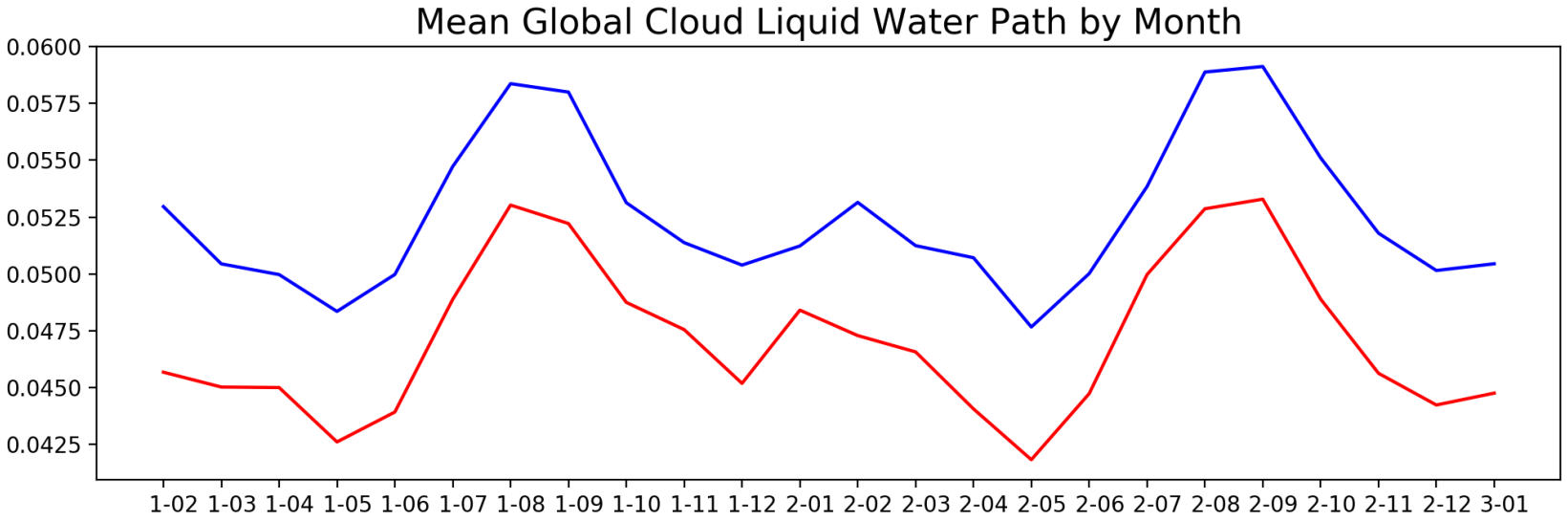
NCAR | UCAR

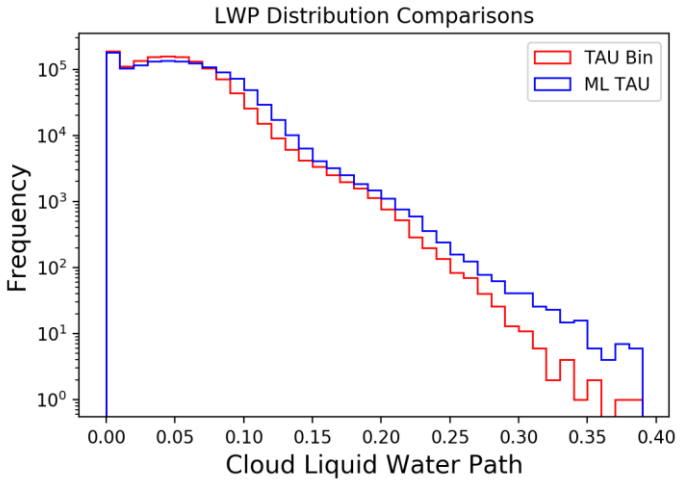
Contact: dgagne@ucar.edu, @DJGagneDos

CAM Run Distribution Comparisons









Contact: dgagne@ucar.edu, @DJGagneDos

Partial Dependence Plots

Goal: understand average sensitivities of input fields while accounting for nonlinear interactions within model

1. Set all instances for one variable in a dataset to a single value

Temperature	Dewpoint	Pressure
280	10	986
280	14	1014
280	2	992
280	25	1025
280	6	950

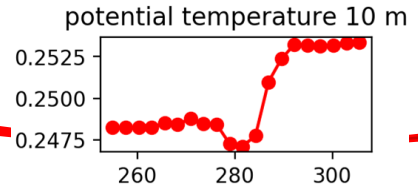
2. Feed fixed data through model

Machine Learning or Physical Model

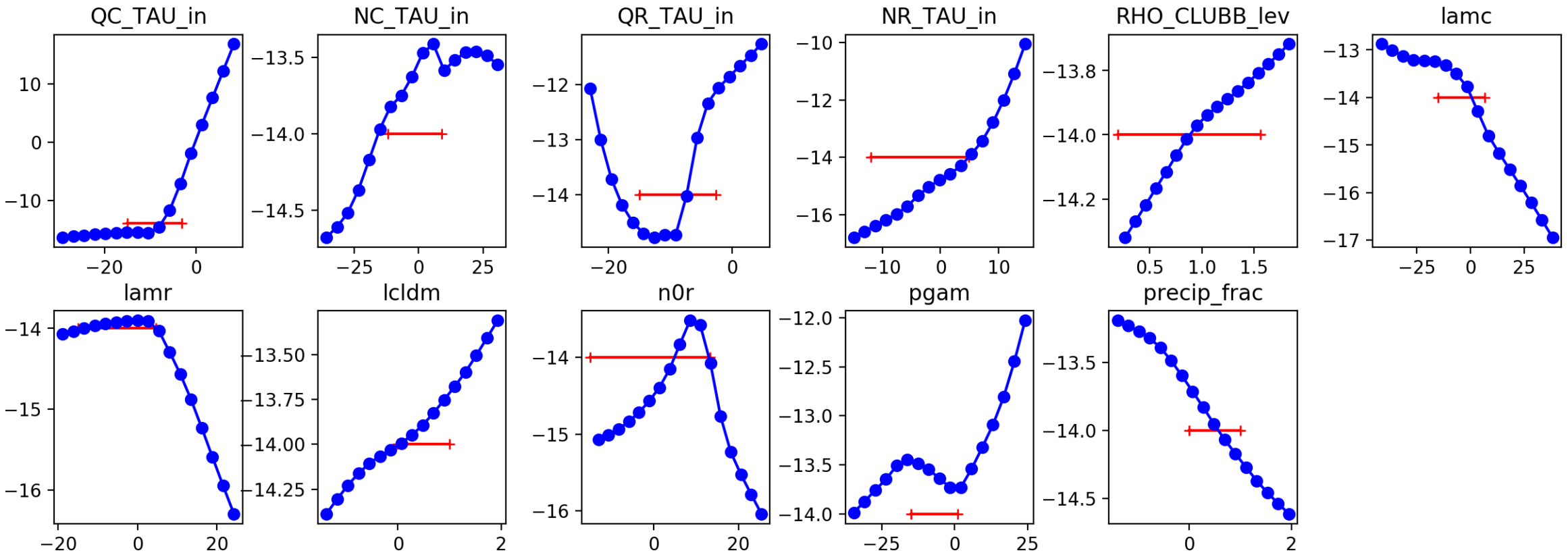
3. Calculate mean prediction for fixed value

Mean Prediction

4. Repeat process for range of input values



Microphysics Emulator Partial Dependence (ExpandedRange)

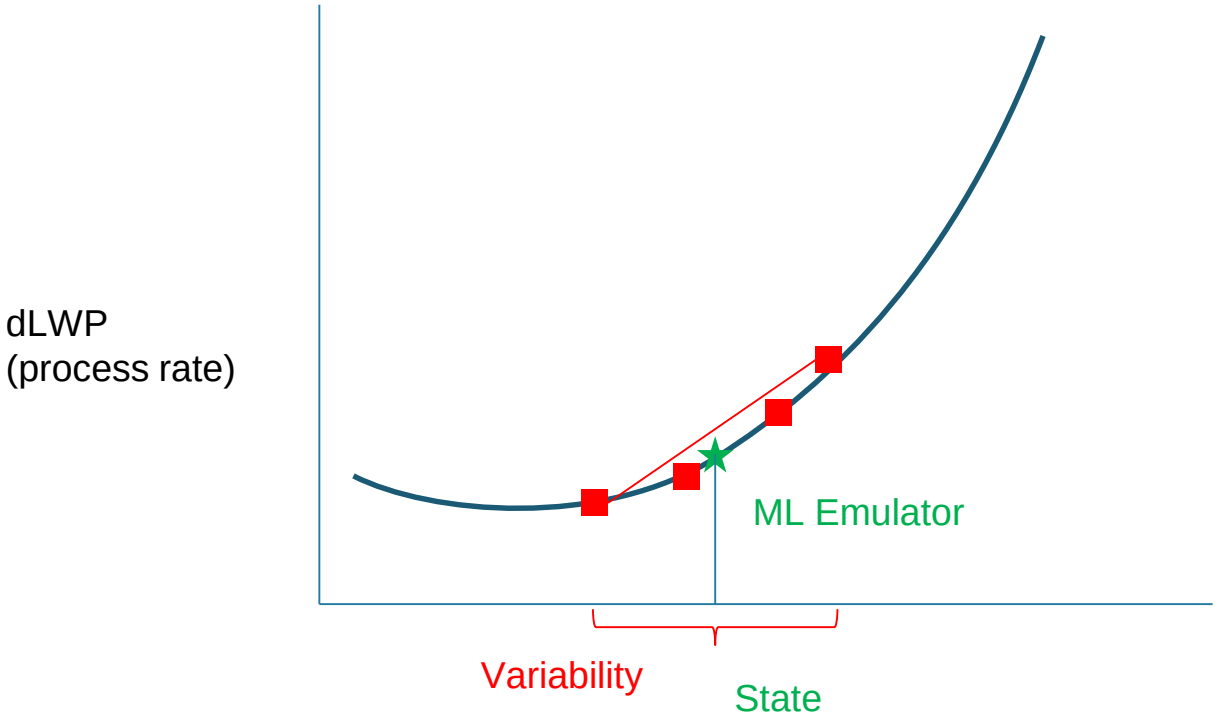


Outside the range of the training data (red) the neural network extrapolates mostly linearly

Contact: dgagne@ucar.edu, @DJGagneDos

Systematic Biases with Emulators

Possible that ML emulator has less variance than the input data.
General issue with many ML applications.
For a non-linear process, variability will yield a different average process rate than the mean…



Average of higher variability (Perfect Model) has larger process rate (dLWP/dT) than smoother Emulator, hence more LWP in the mean.

Python to Fortran Discussion

CFFI (Noah Brenowitz* approach)	Fortran Inference Modules (My Approach)	Fortran API to C++ Deep Learning Library
Pros: Call Python from Fortran Passes data quickly Don't have to run ML models on Fortran side Potential for online training	**Pros:** Train in Python; run in Fortran Runs really fast Supports dense neural networks of arbitrary depth	**Pros:** Can access networks of arbitrary complexity Potential for online training Bypass Python bottleneck
Cons: Requires running separate Python runtime along with Fortran model Python side may be a bottleneck for running at scale	**Cons:** Does not support convolutional neural networks or more complex architectures No online training	**Cons:** Does not exist yet Would require writing and maintaining API to make all library features accessible from Fortran

*https://www.noahbrenowitz.com/post/calling-fortran-from-python/

Summary

- The machine learning surface layer parameterization improves on Monin-Obukhov similarity theory in the calculation of sensible and latent heat fluxes based on comparisons with observations.
- The ML surface layer parameterization can recreate the diurnal cycle of the boundary layer but currently struggles with the transition from stable to unstable conditions.
- The ML bin microphysics emulator accurately captures when the autoconversion process is triggered and provides an unbiased estimate of the magnitudes of the tendencies.
- The ML bin emulator CAM run reproduces a similar climate to the original run with the bin scheme in place

Contact: dgagne@ucar.edu, @DJGagneDos