Neural network emulation of precipitation and condensation processes in FV3GFS

Noah Brenowitz, Andre Perkins, Jacqueline Nugent, Chris Bretherton,
Oliver Watt-Meyer, Anna Kwa, Jeremy McGibbon, Brian Henn, Spencer Clark
noahb@allenai.org

Why emulate microphysics?

- The emulation literature focuses on slow processes like radiation
- Cloud microphysics is perhaps the fastest process in the climate model
  - Bigger tendencies than any other
  - Especially condensation and latent heating
- Our other methods at AI2 mostly neglect explicit treatment of clouds
- Why Zhao-Carr microphysics?
  - We thought it was simple when we started the project
  - Is an option in the FV3GFS code base
  - “Diagnostic” treatment of clouds means simpler input/output structure (or so we thought)

Literature Review

- Gettelman, A. et al. Machine Learning the Warm Rain Process.
- Pros:
  - Emulate expense “bin” microphysics
  - Offline/online tests
- Limitations:
  - Warm rain only handles autoconversion of cloud into rain and accretion processes within the rain.
  - No latent heating…and debatably not a “fast” process (i.e. source term has small magnitude)

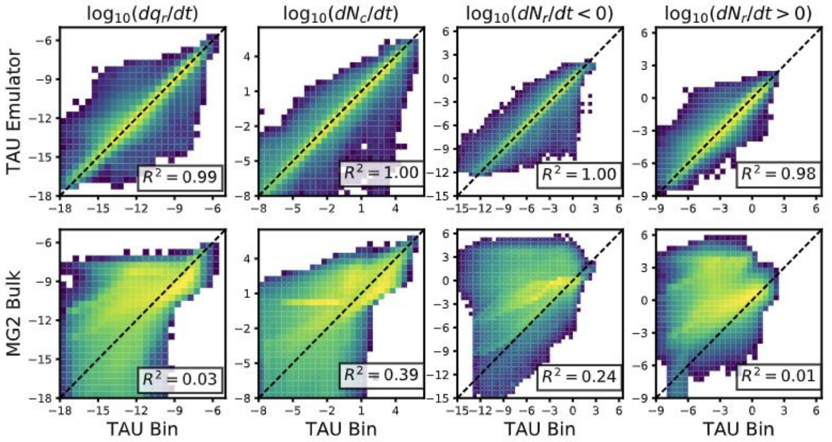


Figure 1. Frequency plots of the logarithm of TAU bin rates (horizontal) versus Neural Network Emulator (Top row) and MG2 Bulk Scheme (Bottom Row). Shown (left to right) are the rain mass tendency (dq_r/dt), condensate number tendency (dN_c/dt), negative rain number tendency ($dN_r/dt < 0$) and positive rain number tendency ($dN_r/dt > 0$). Correlation coefficient shown on the plots.

The Zhao-Carr Microphysics

Zhao, Q. & Carr, F. H. A prognostic cloud scheme for operational NWP models. *Mon. Weather Rev.* **125**, 1931–1953 (1997)

This is the Zhao-Carr...not so bad. Right?

The paper

a. Large-scale condensation (C_g)

If we let

$$A_q = q_{\text{non}} + E_r - C_b \tag{4}$$

$$A_t = T_{\text{non}} + \frac{L}{C_p} C_b - \frac{L}{C_p} E_r - \frac{L_f}{C_p} P_{sm}, \tag{5}$$

then (1) and (2) become

$$\frac{\partial q}{\partial t} = A_q + E_c - C_g \tag{6}$$

$$\frac{\partial T}{\partial t} = A_t - \frac{L}{C_p} E_c + \frac{L}{C_p} C_g. \tag{7}$$

Following Sundqvist (1988), an expression for large-scale condensation rate C_g can be obtained by combining (6) and (7) with equations $q = fq_s$, $q_s = \epsilon e_s/p$, and the Clausius–Clapeyron equation $de_s/dT = \epsilon L e_s/RT^2$, where q_s is the saturation specific humidity, e_s is the saturation vapor pressure, R is the specific gas constant for dry air, p is the pressure, f is the relative humidity, and $\epsilon = 0.622$. The expression for C_g has the form

$$C_g = \frac{M - q_s f_t}{1 + (f \epsilon L^2 q_s / R C_p T^2)} + E_c, \tag{8}$$

The code

```
call gscond (im, ix, levs, dtp, dtf, Statein%prsl, Statein%pgr,    &
             Stateout%gq0(1,1,1), Stateout%gq0(1,1,ntcw),           &
             Stateout%gt0, Tbd%phy f3d(1,1,1), Tbd%phy f3d(1,1,2),  &
             Tbd%phy_f2d(1,1), Tbd%phy_f3d(1,1,3),                  &
             Tbd%phy_f3d(1,1,4), Tbd%phy_f2d(1,2), rhc,lprnt, ipr)

  call precpd (im, ix, levs, dtp, del, Statein%prsl,                 &
             Stateout%gq0(1,1,1), Stateout%gq0(1,1,ntcw),           &
             Stateout%gt0, rain1, Diag%sr, rainp, rhc, psautco_l,   &
              prautco_l, Model%evpco, Model%wminco, lprnt, ipr)
```

Finding meaningful inputs/outputs of ZC scheme is hard

Hidden prognostic variables

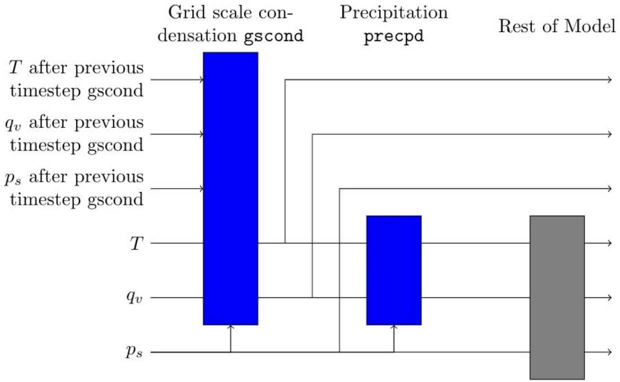
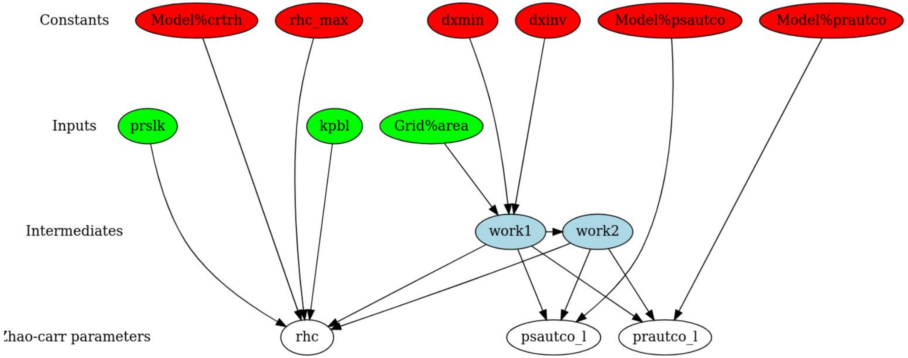


Figure 1: Stateful variables of the Zhao-Carr scheme shown over a single model timestep. Although the scheme is considered "diagnostic", it does track the state after the grid-scale condensation to infer the relative humidity tendency in subsequent timesteps. Components of the Zhao-Carr microphysics are highlighted in blue.

"Scale-dependent" parameters with poor names in driver routine



Analyzed from GFS_Physics_driver.F90

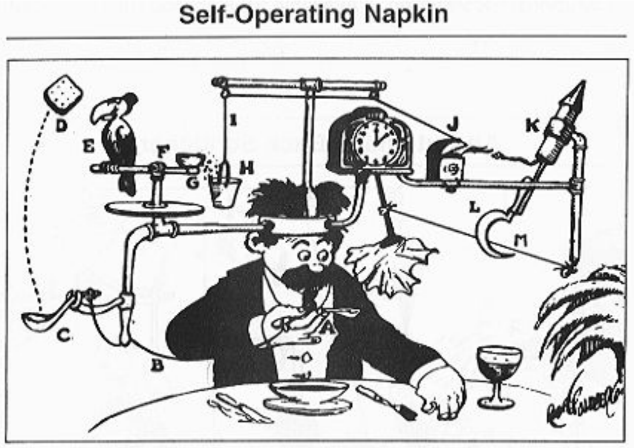
What priors should we use?

Many

Few

Brittle/hard to implement

Easy to implement, but low skill



Black box

Wikipedia

A minimal set of useful priors:

Microphysics least common denominators:

- Total column water (including soil) is conserved
- Hydrometeors fall down
- (possibly) condensation vs precipitation split
  - GFDL Microphysics has fast sat adj
  - ZC MP has “gscond” subroutine

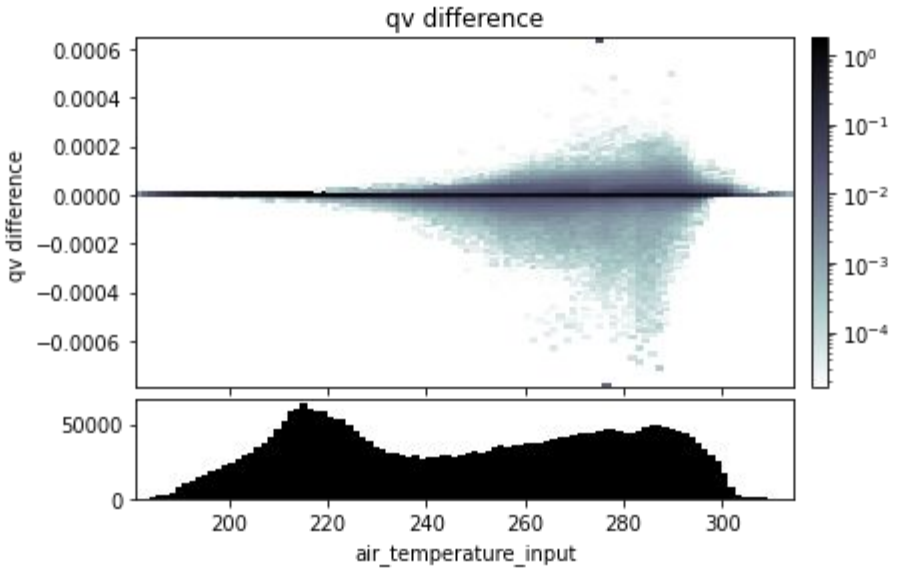
Would like to avoid adding more structure for generalizability to other schemes

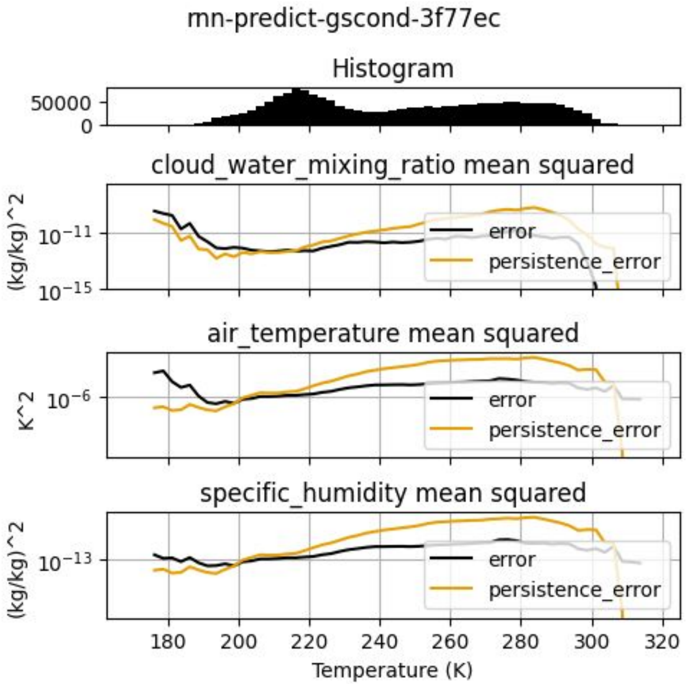
ML Methods

Training set

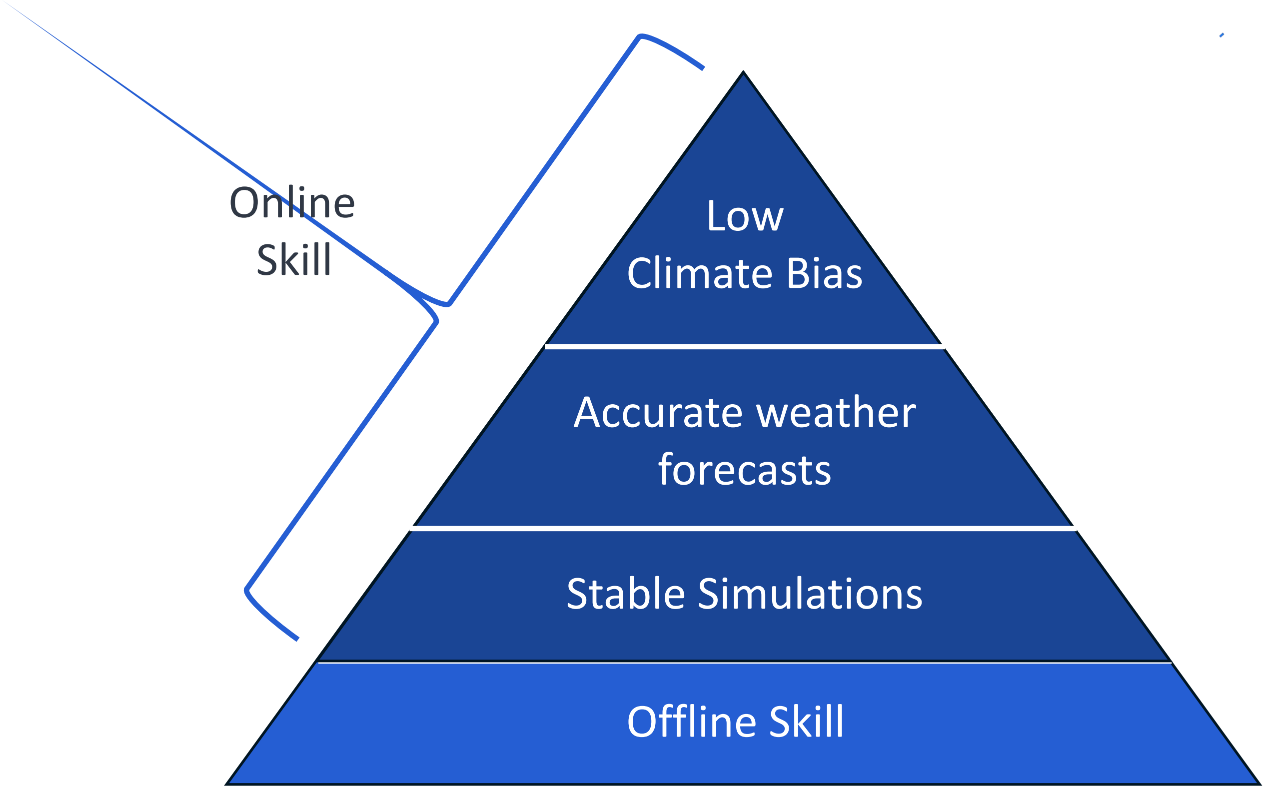
- 10 day runs initialized at beginning of every month in 2016 from GFS analysis
  - C48 resolution
  - Zhao-carr MP rather than GFDL
- 5 hourly outputs to sample all times of day
- Validation: Feb, June, and Sept
- Train set size: 7,464,960
  - 540 snapshots * 6 tiles * 48^2 cells per tile
- Only 1.5 % of training data for |lat| > 80 degrees
  - 1- sin(80 deg)

ZC Tendency amplitudes scale with temperature





Offline skill does not imply online skill



Measuring online skill by "Piggy-backing"

Offline mode: fortran drives
ML tendencies also saved

Offline mode: ML drives
Fortran tendencies also saved

ML Architectures

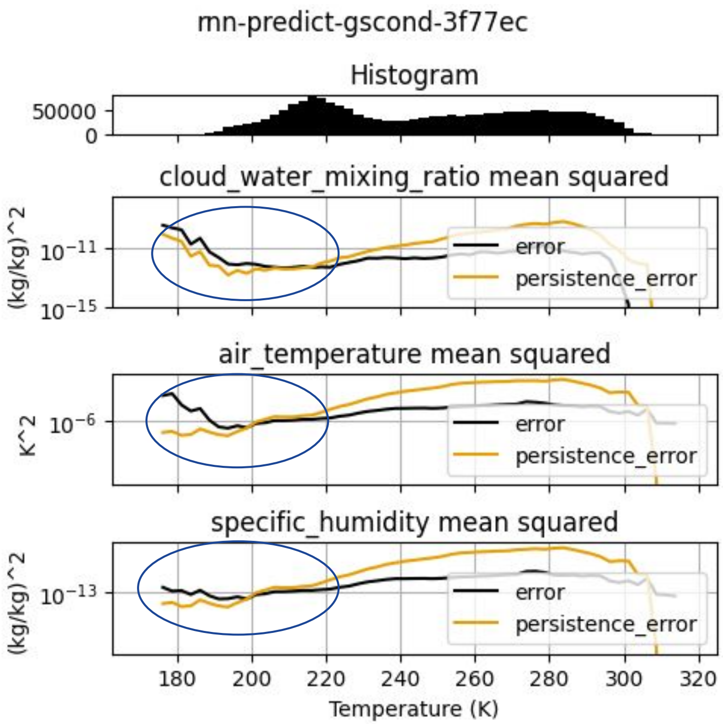
- RNN which integrates downwards (rain falls down)
  - **Dynamics:** h[n+1] = W h[n] + b + Ax[n]
  - **Read-out:** y[n] = M h[n] + b
  - Can stack multiple layers
  - Default architecture settings
    - Width of W is 256
    - 2 RNN layers + 1 output layer ~ 150k parameters
- “Dense” connects all levels to each other
  - Multilayer perceptron
  - 2 layers, 256 width + output layer ~ 150k parameters
  - Uses different input set (unstable when adding hidden ZC inputs)

Other important methods

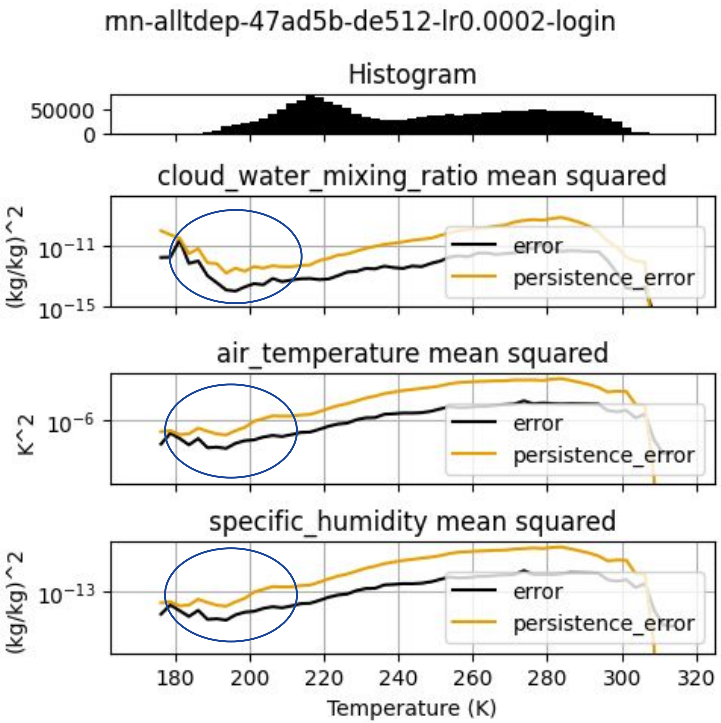
- Calling python from fortran: https://github.com/nbren12/call_py_fort
- Temperature scaling
  - Scale each output tendency by a temperature dependent factor
  - Loss includes MSE both scaled and unscaled spaces
- Finding the right inputs. We started with a minimal set and saw big performance gains from adding:
  - The hidden ZC “state” variables (helps RNN, hurts dense model)
  - log(cloud) and log(specific humidity) in addition to cloud, specific humidity
  - Air pressure in addition to pressure thickness
- Limiting output cloud and humidity > 0
  - Enforced during training (NNs can backprop through constraints no problem!)

Temperature Scaling improves cold temperature offline skill

Unscaled



All-scaled



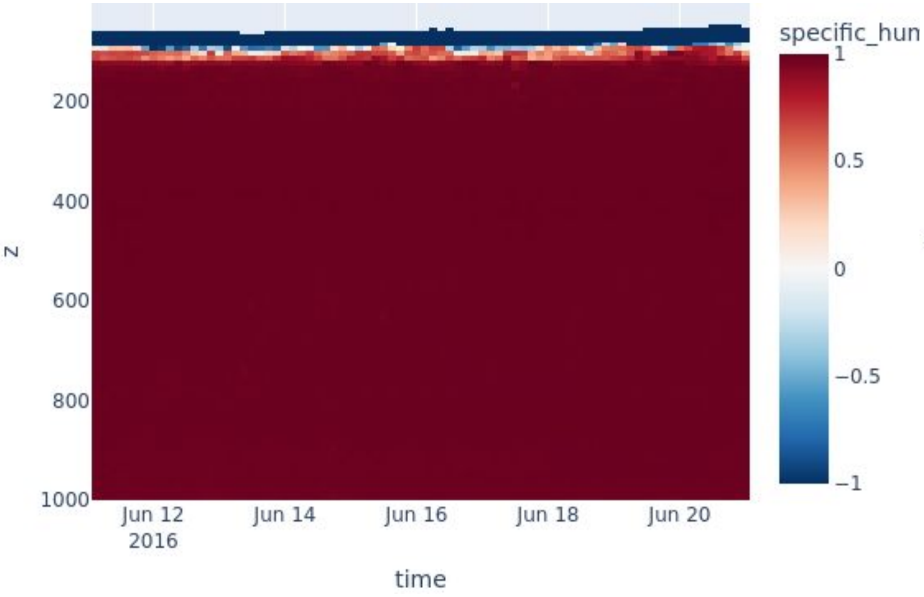
Skill = 1 - error / persistence_error

Results

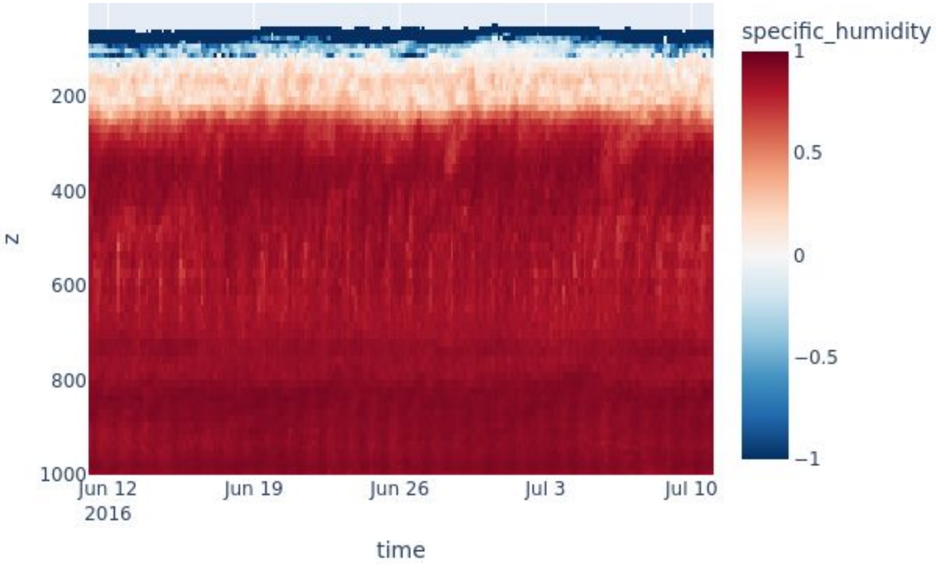
RNN has very impressive skill offline (almost “100%” accurate)

Skill = 1 - (MSE of ML model) / (MSE of predicting out=in)

RNN



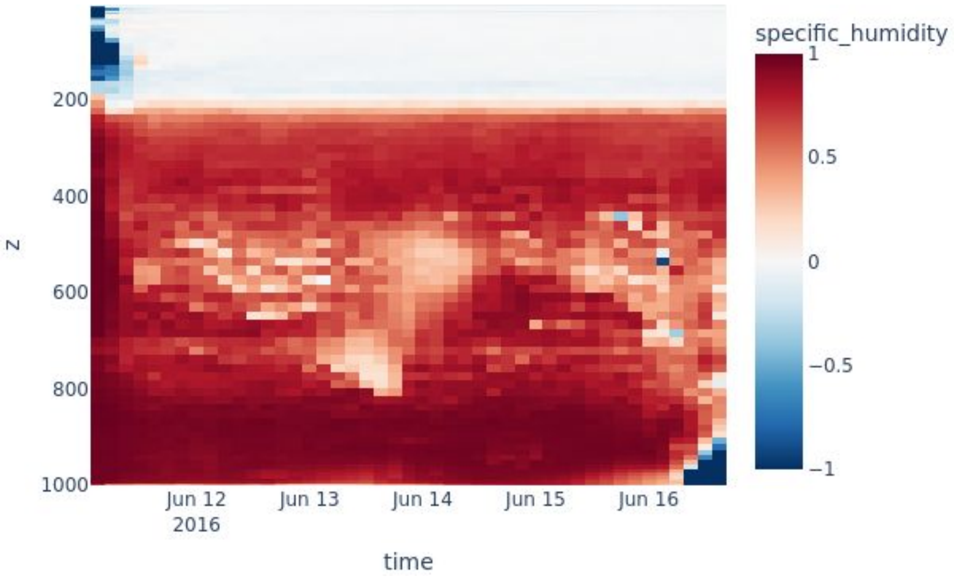
Dense



Note training data comes from June 1 - 10...so this is not overfitting

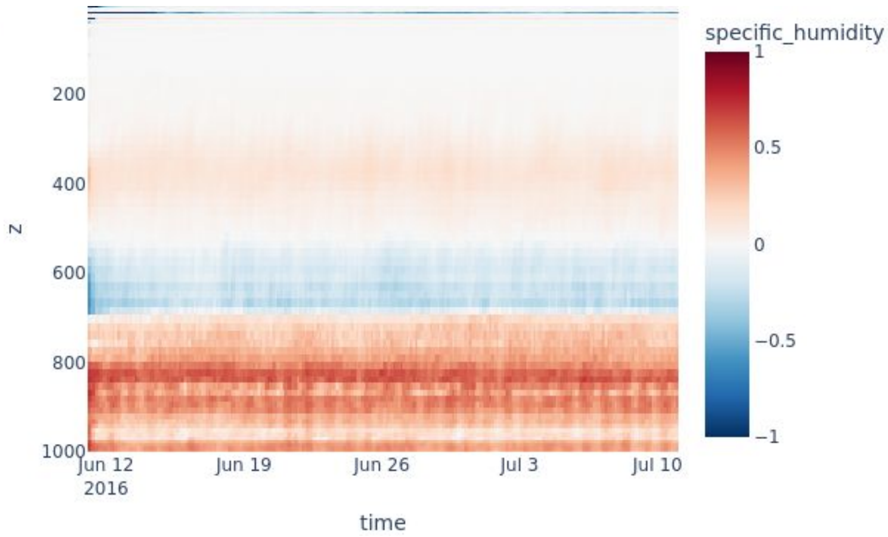
But the story is different online

RNN (cloud >= 0)



Crashes after 6 days

Dense



Survives 30 d

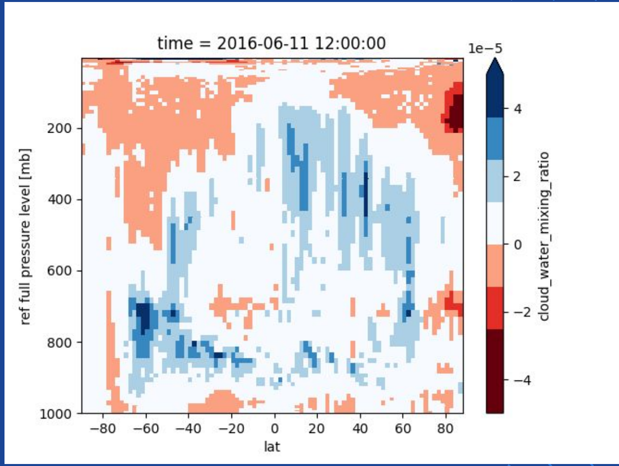
Leaderboard

Model	Offline RMSE Cloud Water (non-dimensional)	Run Duration (max 30 days)
Dense	0.139	**30**
RNN	0.056	23.8125
RNN (cloud >= 0)	**0.051**	5.75

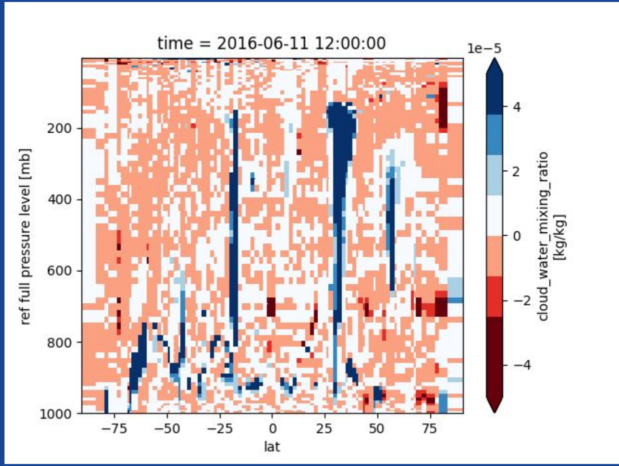
Non-limited RNN produces negative cloud

- All plots from 12 hours into run (2016-06-11T12)
- Model: [limit-tests-all-loss-rnn-7ef273]
- Physics precipitation removes negative cloud, but the ML does not.
- Condensation predictions more reasonable, but missing some evaporation of cloud at upper levels.

Cloud water Zonal average



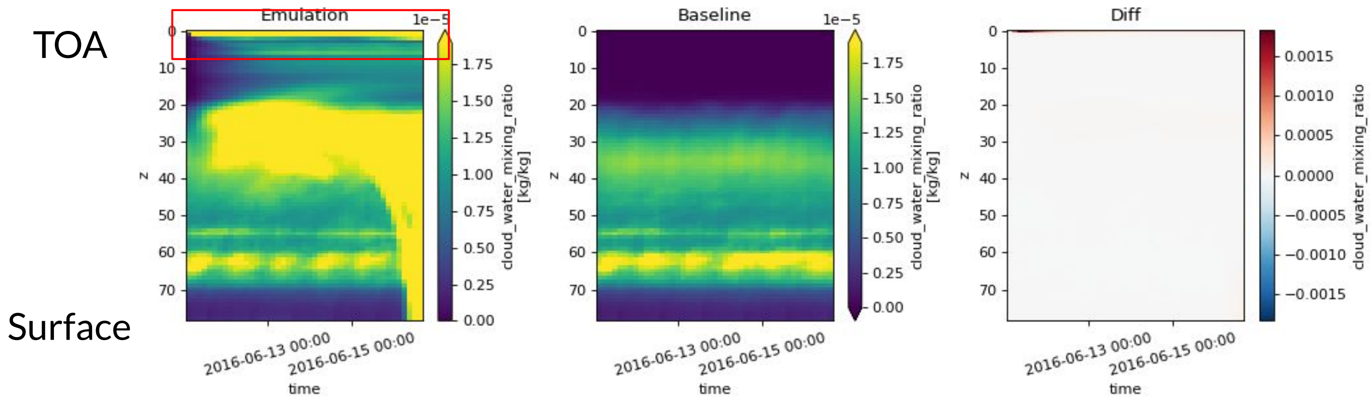
Cloud water lat = 0



By Noah

But including the limiter leads to rapid online drift

Global average cloud water mixing ratio: RNN (cloud > 0)



Conclusions

- Microphysics emulators have **stellar** offline skill
  - RNN helps significantly
- Still struggling with usual online issues: stability, climate drift, etc
  - But getting close and still iterating on offline biases/basic data modeling
  - Constraints that are very rarely active offline suddenly appear online (e.g. cloud limiters)
  - Input collinearity causes spurious correlations
- Other challenges:
  - FV3GFS integration with the “diagnostic” ZC scheme is hacky and has bugs
    - Hard to map conceptual inputs from paper onto the code
    - What in the world is ‘Tbd’?
    - Water phase partitioning copy-pasted 3 places…which one if any to use?
    - Maybe easier for the current operational GFDL microphysics scheme
  - Emulator development cycle is slow…lots of data, slow ML training, trial and error

Thanks!

Calling python from fortran

- Embed a python interpreter in Fortran using `call_py_fort`*
- Allows calling python ML libraries from any fortran code without
  - Refactoring
  - Reimplementing driver logic in Python
- Minimal performance penalty because ML libraries are fast

```fortran
        call set_state("air_temperature_at_previous_time_step", tp1_cpf)
        call set_state("specific_humidity_at_previous_time_step", qvp1_cpf)
        call set_state("surface_air_pressure_at_previous_time_step", psp1_cpf)


      call gscond (im, ix, levs, dtp, dtf, Statein%prsl, Statein%pgr,    &
                   Stateout%gq0(1,1,1), Stateout%gq0(1,1,ntcw),          &
                   Stateout%gt0, Tbd%phy_f3d(1,1,1), Tbd%phy_f3d(1,1,2), &
                   Tbd%phy_f2d(1,1), Tbd%phy_f3d(1,1,3),                 &
                   Tbd%phy_f3d(1,1,4), Tbd%phy_f2d(1,2), rhc,lprnt, ipr)

      call set_state("air_temperature_output", Stateout%gt0)
      call set_state("specific_humidity_output", qv_post_precpd)
      call set_state("cloud_water_mixing_ratio_output", qc_post_precpd)

      call set_state("total_precipitation", rain1)
      call set_state("ratio_of_snowfall_to_rainfall", Diag%sr)
      call set_state("tendency_of_rain_water_mixing_ratio_due_to_microphysics", rainp)

      if (Model%emulate_zc_microphysics) then
        ! apply microphysics emulator
        call call_function("emulation", "microphysics")
      endif

      if (Model%save_zc_microphysics) then
        call call_function("emulation", "store")
      endif

      call get_state("air_temperature_output", Stateout%gt0)
      call get_state("specific_humidity_output", qv_post_precpd)
      call get_state("cloud_water_mixing_ratio_output", qc_post_precpd)
      call get_state("total_precipitation", rain1)
```

Subroutine to emulate

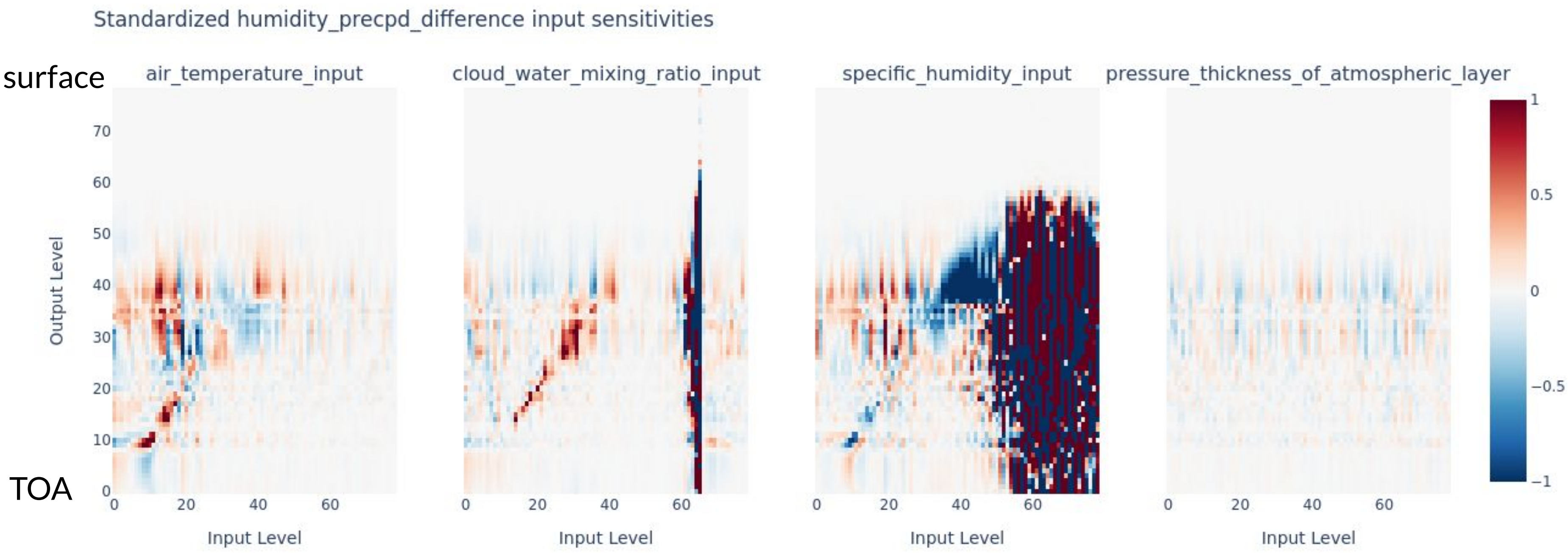
Calls to python

Communication between python and fortran

*https://github.com/nbren12/call_py_fort

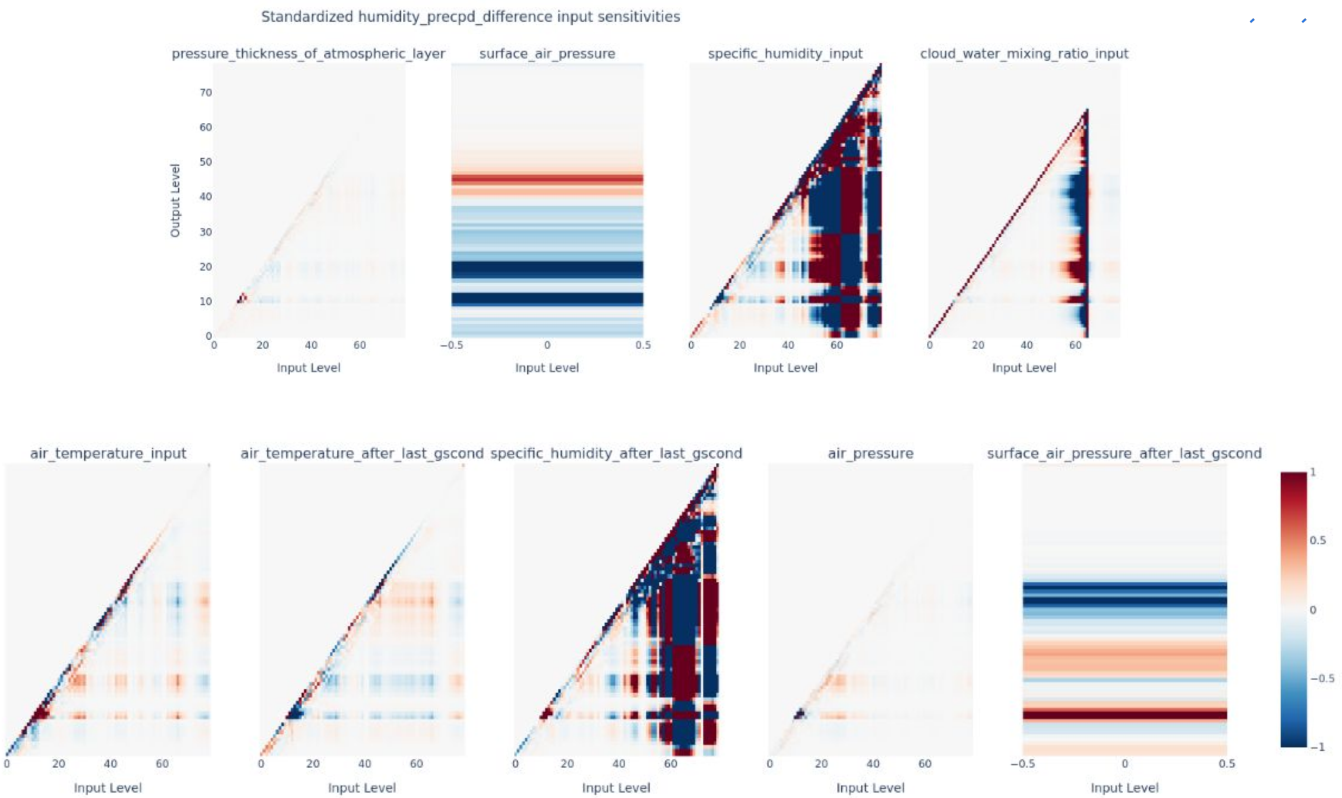
https://github.com/ai2cm/fv3gfs-fortran/tree/af0b98987480c31e009ea8d096eebf6d12e05c21/tests/emulation

Dense models learn that rain falls up



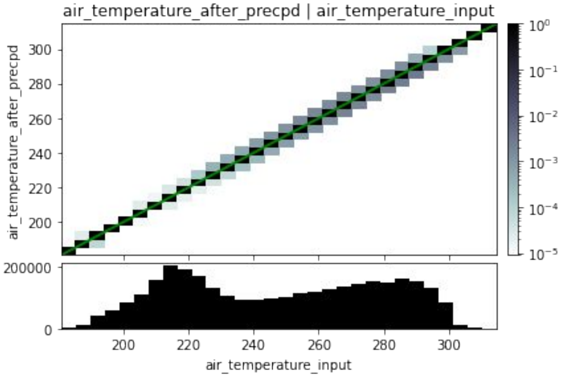
Should be lower triangular since “rain falls down”. Input collinearities introduce spurious correlations.

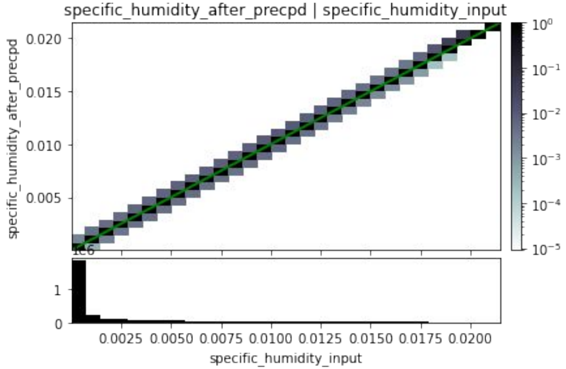
RNN enforces downward dependence

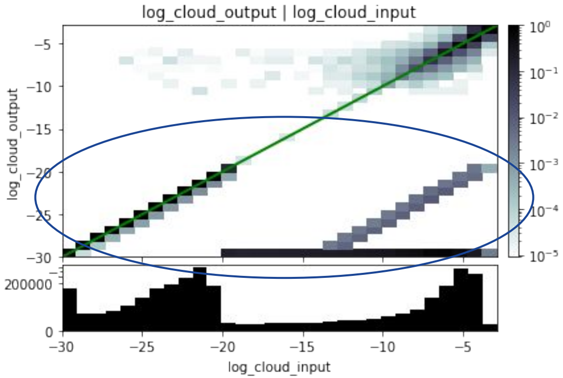


Clearly this data comes from a numerical model

log10(cloud +1e-30)







Floating point artifacts from enforcing cloud >= 0

Residual approximation: After = before + eps

Works better for humidity and temperature than cloud