Emulating Fast Processes in Climate Models

Noah D. Brenowitz[*]
Allen Institute for Artificial Intelligence
nbren12@gmail.com

W. Andre Perkins
Allen Institute for Artificial Intelligence
andrep@allenai.org

Jacqueline M. Nugent
University of Washington
jnug@atmos.washington.edu

Oliver Watt-Meyer
Allen Institute for Artificial Intelligence
oliverwm@allenai.org

Spencer K. Clark
Allen Institute for Artificial Intelligence,
Geophysical Fluid Dynamics Laboratory, NOAA
spencerc@allenai.org

Anna Kwa
Allen Institute for Artificial Intelligence
annak@allenai.org

Brian Henn
Allen Institute for Artificial Intelligence
brianhenn@allenai.org

Jeremy McGibbon
Allen Institute for Artificial Intelligence
jeremym@allenai.org

Christopher S. Bretherton
Allen Institute for Artificial Intelligence
christopherb@allenai.org

Abstract

Cloud microphysical parameterizations in atmospheric models describe the formation and evolution of clouds and precipitation, a central weather and climate process. Cloud-associated latent heating is a primary driver of large and small-scale circulations throughout the global atmosphere, and clouds have important interactions with atmospheric radiation. Clouds are ubiquitous, diverse, and can change rapidly. In this work, we build the first emulator of an entire cloud microphysical parameterization, including fast phase changes. The emulator performs well in offline and online (i.e. when coupled to the rest of the atmospheric model) tests, but shows some developing biases in Antarctica. Sensitivity tests demonstrate that these successes require careful modeling of the mixed discrete-continuous output as well as the input-output structure of the underlying code and physical process.

1   Introduction

Weather forecasts and climate simulations are predominantly made by evolving atmospheric and oceanic fluid dynamics forward in time on a discrete global grid. Most atmospheric models are organized into a dynamical core, which evolves the Navier-Stokes equations on the rotating sphere, and the "physics" which handle everything else such as cloud formation, rain, and boundary layer turbulence.

---

[*]Now at NVIDIA Research: https://research.nvidia.com/person/noah-brenowitz

Replacing climate model physical parameterizations with machine learning models is known as emulation. Emulation provides a principled path towards accelerating Fortran model codes, especially for use on computing architectures reliant on accelerators such as GPUs which can efficiently run many machine learning (ML) applications. Because of this, most emulation studies have focused on radiative transfer [4, 9], which is the slowest sub-component in the typical atmospheric physics suite. However, recent studies have emulated deep convection [10], gravity wave drag [3], atmospheric chemistry [7, 8, 12], and the warm rain process [5]. In spite of these successes, no general purpose approach to emulation has emerged and consistently high accuracy over all timescales and spatial locations has been elusive.

Emulation is also an excellent test-bed for more complex ML approaches to improve physical parameterizations since it is very clearly posed as a supervised learning task. It can be argued that successfully emulating a given climate model component is a necessary first step before trying to improve that component e.g by training models with fine resolution data [1, 11, 13, 2]. This work often focuses on moist atmospheric physics, a notoriously difficult process to parameterize. One key moist process is the cloud microphysics scheme, which handles fast phase changes like condensation, evaporation and precipitation. Microphysics are tightly coupled to the dynamical core through the corresponding latent heat release. Since cloud microphysics are computationally cheaper than radiation, no study has yet tried to emulate an entire microphysics scheme including phase changes. However, in view of its importance and its contrasting character to radiation, this is a worthy and educational ML challenge.

In this study, we train ML models to emulate an simple microphysical scheme written in Fortran, including the phase changes. We will show that our best scheme has excellent offline skill and works well, but not perfectly, online when coupled to the rest of the atmospheric solver.

2   Methods

The training data are generated by running the FV3GFS weather model [15, 6] with the simplified Zhao-Carr (ZC) microphysics (Appendix A), for simplicity. The ZC microphysics predict changes in cloud liquid and ice condensate, precipitation, and associated heating and moistening rates at each grid point in a grid column given a corresponding column of thermodynamic inputs. The change of temperature T, cloud c, or humidity q by the ZC microphysics is defined as $\Delta = \Delta_g + \Delta_p$. Δ_g is the change to due the grid-scale condensation subroutine (`gscond`). Δ_p is the change due the precipitation subroutine (`precpd`). We added hooks before and after these subroutines that are used both for saving the data and applying the ML emulators online.

FV3GFS is a compressible atmospheric model used for operational weather forecasts by the US National Weather Service. For that purpose, it is run with a cubed-sphere grid of approximately $13\,\mathrm{km}$ horizontal spacing and 64 vertical levels. For the results shown here, the grid spacing of the model is approximately $140\,\mathrm{km}$ resolution (C48) and there are 79 vertical levels. This coarser horizontal resolution greatly speeds up online testing of the emulated parameterization.

The training simulations are initialized from the GFS analysis from the first day of every month of 2016. The ZC inputs/outputs are saved every 5 hours in order to sample all times of day as the simulation progresses. The simulations of Feb, June, and Sept are reserved for validation. Overall the training set contains 1296 snapshots each consisting of $48^2 \cdot 6 = 13\,824$ atmospheric columns.

The net condensation $\Delta_g c$ at a given grid point (x, y, z) in 3D space depends only on the thermodynamic inputs there, a property we refer to as grid-point locality. To exploit this locality, the `gscond` subroutine is emulated with a single multi-layer perception which is applied to each grid-point separately. The inputs and hyper-parameters are listed in Appendix C.1. Notably, the input set includes T, q, and pressure p. In all the emulator takes in 14 scalar input channels and predicts a single scalar output: the net change $\Delta_g c$ in cloud condensate. For online applications, we use the Fortran code instead of the emulator in the top 5 vertical levels since $\Delta_g c = 0$ there for all training samples. The results are similar when we directly enforce that $\Delta_g c = 0$. Since Δc depends exponentially on temperature we normalize it by the standard deviations computed within bins of temperature. The corresponding change in specific humidity over the condensation step is $-\Delta_g c$, satisfying total water conservation. The temperature change is $(L_v/c_p)\Delta_g c$ where L_v is the latent heat of vaporization and c_p is the specific heat of air at constant pressure. This is an approximation since some ZC phase changes occur between ice and vapor which releases additional latent heat.

2

For approximately 80% of samples, either the condensate change $\Delta_g c$ or the cloud after `gscond` vanish. The regression model above makes small, but nonzero errors in these cases which accumulate in time and cause poor online performance. To avoid this, we train a classifier to identify such points (see Appendix C.2.2) with the same inputs as the grid-point model described above. This classifier is used online to either deactivate $\Delta_g c$, remove all the cloud, or return the grid-point model's output.

The `precpd` subroutine is vertically non-local so we use a different architecture to predict Δ_p using the same input variables as the `gscond` emulator. Because rain falls down rather than up, the predicted change at a given height of T, q, and c depend only on data from overlying grid points. We encode this causal assumption using an RNN which takes in a top-of-atmosphere to surface sequence of thermodynamic fields $\mathbb{R}^{79} \times \mathbb{R}^{14}$ and maps to $\Delta_p \in \mathbb{R}^{79} \times \mathbb{R}^3$. The surface precipitation is also output as a trainable affine transformation of the final hidden RNN state (at the surface). A two-layer vanilla RNN is used with ReLU activation, and 256 hidden nodes in each layer. As a sensitivity test, we also train a two-layer column-dense model for this task with 256 hidden nodes in each layer. Because heat and water conservation are not trivially enforced like they are for `gscond` we predict the change of all three thermodynamic variables separately at the risk of violating appropriate conservation laws.

All the models above are trained separately on the full training set with the Adam optimizer and a batch size of 512 columns. Mean-squared-error losses are used for regression models and cross-entropy is used for the `gscond` classifier. Hyper-parameters were not tuned exhaustively in this work apart from ensuring that the initial loss is $O(1)$ and stochastic gradient descent is stable. Both the precipitation and condensation regression models are trained by stochastic gradient descent over 25 epochs, with the Adam optimizer, a learning rate of 0.0001, a batch size of 512 columns ($40\,448$ grid points). The classifier is trained with an increased. learning rate of 0.001.

All computations were performed on Google Cloud Platform. Training a model for 25 epochs takes 8 hours on a NVIDIA Tesla P4 GPU. On an 8 processor node, a 30 day FV3GFS simulation takes 5 hours to complete without ML and 10 hours with ML emulators.

3   Results

For each ML configuration, we run two simulations, one where the ML is active and another where the Fortran is active. The inactive scheme still saves data, so that we can compute skill metrics. Metrics are computed over 30-day simulations initialized at the end of the June simulation from the training data. Even though this overlaps in time with the July training data, it represents a unique testing dataset since it was initialized with the final state of the June training simulation rather than the July 1 GFS analysis.

We consider five metrics—the offline and online skill at predicting the T and c tendencies as well as the bias of the global mass of cloud water condensate per unit surface area (i.e. cloud water path) on July 15. The Fortran-simulated cloud water path at this time was $102\,\mathrm{g\,m}^{-2}$. The tendency metrics measure the skill compared to the saved Fortran tendencies as given by a modified R^2 score $1 - \sum(y - \tilde{y})^2 / \sum y^2$, where y is the truth and $\tilde{y}$ is the prediction.

We first examine the offline and online metrics for runs where the subroutines `gscond` and `precpd` are replaced by ML one-at-a-time (Table 1). Neural network architecture matters when replacing `precpd`. Using the RNN instead of a dense model improves the offline skill score for condensate tendency significantly, but the improvement in online skill is more dramatic: the cloud bias is 20x worse with the dense model. This large increase in cloud pushes the online states far from the envelope of the training data so the offline skill scores degrade. This is a typical failure mode we have witnessed throughout this project. Typically, the ML-predicted tendencies are fairly insensitive to state drift, but the Fortran physics produces very large tendencies as they attempt to correct small physical inconsistencies introduced by the ML such as super-saturated air or negative cloud concentrations. The RNN avoids these deficiencies.

Using the classifier helps when replacing the just `gscond` scheme (Table 1, but the the impact is far more dramatic in an online simulation where *both* `precpd` and `gscond` are replaced by ML emulators (Table 2). In this run, `precpd` is replaced by the RNN and `gscond` is replaced with a dense-local model both with and without classifier-based masking. Classifier-based masking is essential for good online skill.

Table 1: Table of skill scores for the `precpd` or `gscond` emulators. The online skill scores are computed in an online run where only the indicated subroutine is emulated.

| | $\alpha =$ `precpd` | | $\alpha =$ `gscond` | |
	Dense	RNN	No classifier	Classifier
Offline $\Delta_\alpha c$ Skill	94%	98%	99%	99.2%
Offline $\Delta_\alpha T$ Skill	92%	98%	98.50%	98.7%
Online $\Delta_\alpha c$ Skill	14%	95%	92.60%	93.9%
Online $\Delta_\alpha T$ Skill	68%	98%	92.50%	94.1%
July 15 cloud water bias ($\mathrm{g\,m^{-2}}$)	23.6	1.3	25.0	21.2

Table 2: Online performance metrics for a simulation with emulators active for both `gscond` and `precpd`. The classifier predicts when cloud after `gscond` vanishes (Zero-cloud) and also when $\Delta_g = 0$. The columns show the impact of actually enforcing the predicted classes.

Metric	No Classifier	Enforce zero-cloud	Also enforce $\Delta_g = 0$
`gscond` skill ($\Delta_g c$)	-81%	-53%	94%
`precpd` skill ($\Delta_p c$)	36%	36%	77%
July 15 cloud water bias ($\mathrm{g\,m^{-2}}$)	64.2	34.2	7.6

The combined ML simulation performs well by the skill metrics we have considered thus far, but it has excessive cloud over the Antarctic Plateau (Figure 1). The globally trained emulator also has poor offline skill there, so we tried using a separately trained condensation model for the high antarctic plateau. This improved the offline skill, but not the online cloud biases The interested reader can refer to Appendix D for more diagnostics of the online performance.

The combined `gscond`-`precpd` model presented above required months of iterative development so we hope a brief description of our development trajectory will illuminate sensitivities we have not rigorously proven. We began by attempting to emulate the combined action of `gscond` and `precpd` with a single dense neural network and then with RNNs. The RNN was far better offline but worse online. We then found that the `gscond` subroutine used additional inputs: the humidity, surface pressure, and temperature after the last call to `gscond` (Appendix B). Accounting for these improved the online performance of the RNN. We temperature-scaled the condensation rate to improve offline skill in cold regions of Antarctica and the upper troposphere. The largest improvements in online skill came from developing separate models for condensation and precipitation. These schemes represent different physical processes and thus have different input-output structures and scales.

Figure 1: (a) Zonal average of cloud water mixing ratio (mg/kg) from the truth and (b) bias of the emulation run. The average is computed from July 20 to July 31, inclusive.

4
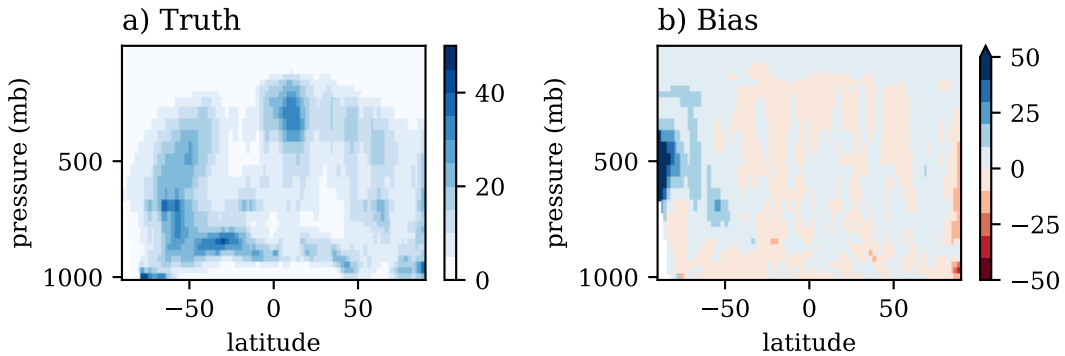
4   Conclusions

Our results demonstrate that feasibility of emulating cloud processes of a climate model with a machine learning model. Large-scale condensation is perhaps the fastest physical process in a climate model and we have developed the first emulator we know of it. Our final model respected conservation laws, enforced the correct notion of grid-point locality, and employed classifiers to represent the mixed discrete-continuous structure of the predictand. We also applied RNNs for the precipitation process in a novel way to enforce the fact that rain and snow fall down rather than up, leading to large improvements in offline and especially online skill. This architecture could be useful for emulating other microphysical parameterizations, or more broadly parameterizations of other atmospheric processes that have a directional structure (e.g. orographic wave drag, entraining plumes).

Despite these successes, the detailed care required and the remaining issues over Antarctica temper our expectations for machine learning parameterization of processes with strong time-scale separation. This manifests as a fat-tailed tendency targets which probably require high capacity models and vast training sets to learn. For emulators, we can generate infinite training data but cannot use very high capacity models since emulators should be faster than the Fortran code they are emulating. Machine learning parameterizations trained from observations or fine-resolution simulations face the further challenge of finite data and noise.

Broader Impacts

Climate change and extreme weather impact us all. This work helps illuminate a path for machine learning to improve weather and climate forecasts through ML emulation of physical parameterizations, which can in principle speed up the model and aid in the assimilation of new observations into the model. Such forecasts help underpin strategies to adapt to and mitigate future climate change and extreme weather. They also provide key scientific evidence for climate change and build the case for climate action. While we cannot envision in immediate negative impacts of this work, our method could be applied to simulate physical systems we have not considered.

References

[1] Noah D Brenowitz and Christopher S Bretherton. Spatially extended tests of a neural network parametrization trained by coarse-graining. *J. Adv. Model. Earth Syst.*, July 2019. ISSN 1942-2466, 1942-2466. doi: 10.1029/2019MS001711.

[2] C S Bretherton, B Henn, A Kwa, and others. Correcting Coarse-Grid weather and climate models by machine learning from global Storm-Resolving simulations. *Journal of Advances*, 2022.

[3] Matthew Chantry, Sam Hatfield, Peter Dueben, Inna Polichtchouk, and Tim Palmer. Machine learning emulation of gravity wave drag in numerical weather forecasting. *J Adv Model Earth Syst*, 13(7):e2021MS002477, July 2021. ISSN 1942-2466. doi: 10.1029/2021MS002477.

[4] F Chevallier, F Chéruy, N A Scott, and A Chédin. A neural network approach for a fast and accurate computation of a longwave radiative budget. *J. Appl. Meteorol.*, 37(11):1385–1397, November 1998. ISSN 0894-8763. doi: 10.1175/1520-0450(1998)037<1385:ANNAFA>2.0. CO;2.

[5] A Gettelman, D J Gagne, C-C Chen, M W Christensen, Z J Lebo, H Morrison, and G Gantos. Machine learning the warm rain process. *J. Adv. Model. Earth Syst.*, 13(2), February 2021. ISSN 1942-2466. doi: 10.1029/2020ms002268.

[6] Lucas Harris, Xi Chen, William Putman, Linjiong Zhou, and Jan-Huey Chen. A Scientific Description of the GFDL Finite-Volume Cubed-Sphere Dynamical Core. 2021. doi: 10.25923/6NHS-5897.

[7] Christoph A Keller and Mat J Evans. Application of random forest regression to the calculation of gas-phase chemistry within the GEOS-Chem chemistry model v10. *Geosci. Model Dev.*, 12 (3):1209–1225, March 2019. ISSN 1991-959X, 1991-9603. doi: 10.5194/gmd-12-1209-2019.

[8] Makoto M Kelp, Daniel J Jacob, Haipeng Lin, and Melissa P Sulprizio. An online-learned neural network chemical solver for stable long-term global simulations of atmospheric chemistry. *J. Adv. Model. Earth Syst.*, 14(6), June 2022. ISSN 1942-2466. doi: 10.1029/2021ms002926.

[9] Vladimir M Krasnopolsky, Michael S Fox-Rabinovitz, and Dmitry V Chalikov. New approach to calculation of atmospheric model physics: Accurate and fast neural network emulation of longwave radiation in a climate model. *Mon. Weather Rev.*, 133(5):1370–1383, May 2005. ISSN 0027-0644. doi: 10.1175/MWR2923.1.

[10] Paul A O'Gorman and John G Dwyer. Using machine learning to parameterize moist convection: Potential for modeling of climate, climate change, and extreme events. *J. Adv. Model. Earth Syst.*, 10(10):2548–2563, October 2018. ISSN 1942-2466. doi: 10.1029/2018MS001351.

[11] Stephan Rasp, Michael S Pritchard, and Pierre Gentine. Deep learning to represent subgrid processes in climate models. *Proc. Natl. Acad. Sci. U. S. A.*, 115(39):9684–9689, September 2018. ISSN 0027-8424, 1091-6490. doi: 10.1073/pnas.1810286115.

[12] John S Schreck, Charles Becker, David John Gagne, Keely Lawrence, Siyuan Wang, Camille Mouchel-Vallon, Jinkyul Choi, and Alma Hodzic. Neural network emulation of the formation of organic aerosols based on the explicit GECKO-A chemistry model. *J. Adv. Model. Earth Syst.*, August 2022. ISSN 1942-2466. doi: 10.1029/2021ms002974.

[13] Janni Yuval and Paul A O'Gorman. Stable machine-learning parameterization of subgrid processes for climate modeling at a range of resolutions. *Nat. Commun.*, 11(1):3295, July 2020. ISSN 2041-1723. doi: 10.1038/s41467-020-17142-3.

[14] Qingyun Zhao and Frederick H Carr. A prognostic cloud scheme for operational NWP models. *Mon. Weather Rev.*, 125(8):1931–1953, August 1997. ISSN 0027-0644, 1520-0493. doi: 10.1175/1520-0493(1997)125<1931:apcsfo>2.0.co;2.

[15] Linjiong Zhou, Shian-Jiann Lin, Jan-Huey Chen, Lucas M Harris, Xi Chen, and Shannon L Rees. Toward Convective-Scale prediction within the next generation global prediction system. *Bull. Am. Meteorol. Soc.*, 100(7):1225–1243, July 2019. ISSN 0003-0007, 1520-0477. doi: 10.1175/BAMS-D-17-0246.1.

Checklist

1. For all authors...

    (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]

    (b) Did you describe the limitations of your work? [Yes] See the last paragraphs of Sections 3 and 4

    (c) Did you discuss any potential negative societal impacts of your work? [Yes] See "Broader Impacts".

    (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...

    (a) Did you state the full set of assumptions of all theoretical results? [N/A]

    (b) Did you include complete proofs of all theoretical results? [N/A]

3. If you ran experiments...

    (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See the acknowledgements. Currently hidden to preserve anonymity.

    (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]

    (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] This is preliminary work and the sensitivies we discuss have large magnitudes.

    (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See last paragraph of Section 2

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

    (a) If your work uses existing assets, did you cite the creators? [Yes] We cited FV3GFS and Zhao Carr in Section 2.

    (b) Did you mention the license of the assets? [N/A]

    (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] See acknowledgements.

    (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] See acknowledgements.

    (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

    (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

    (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

    (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A   The Zhao-Carr Microphysics

This scheme handles both phase changes—condensation and evaporation—and precipitation processes. The former is typically 10x larger in magnitude. The prognostic variables used by the scheme are the temperature T, specific humidity q, and a combined cloud water/ice mixing ratio c.

The `gscond` scheme handles evaporation of cloud and condensation. Evaporation of cloud is given by $E_c = \frac{1}{\Delta t} \max[\min[q_s(f_0 - f), c], 0]$. f is relative humidity. f_0 is a critical relative humidity threshold which [14] describe as "was empirically set to 0.75 over land and 0.90 over ocean". q_s is the saturation specific humidity.

Condensation C_g on the other hand is given by a more complex formula involving a relative humidity tendency. See Eq. (8) of Zhao and Carr [14]. Both formulas depend only on the thermodynamic state of a single (x, y, z) location, but there is some non-local dependence on the assumed phase of the cloud and the corresponding latent heating rate.

The `precpd` scheme handles the conversion of cloud into rain/snow and the evaporation of the latter as it falls through the atmosphere. Broadly speaking, it can be written as the following

$$E_{rr} = E_r(T, f, P_r)$$
$$E_{rs} = E_r(T, f, P_s)$$
$$P = P(T, f, c, P_r, P_s)$$
$$P_{sm} = P_{sm}(T, f, c, P_r, P_s)$$
$$P_r = \int_{p_t}^{p} (P - E_{rr}) dp/g$$
$$P_s = \int_{p_t}^{p} (P_{sm} - E_{rs}) dp/g.$$

Most of the formulas are proportional to rainfall P_r and snowfall P_r rates at a given level, though are some rate constants that depend exponentially on temperature. p_t is the pressure at the top of the atmosphere.

B Implementation of the Zhao-carr Microphysics in FV3GFS

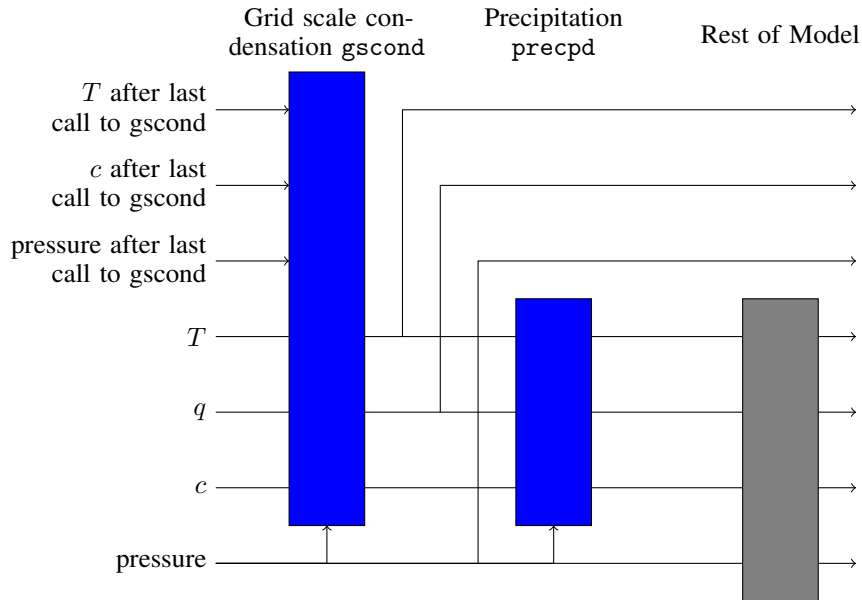Figure 2: Information flow of the Zhao-carr microphysics within FV3GFS. Inputs (outputs) of a given scheme are represented as inward (outward) arrows. The "after last call to gscond" inputs are used to compute a relative humidity tendency that encompasses the rest of the model and `prepcd`. This approach to computing the tendency effectively adds three new state variables to the model.

C Architectures and Hyper-parameters

C.1 Input data and normalization

The condensation emulator has 12 3d input variables: air pressure, air temperature, air temperature from the last time gscond was called (after-last gscond), cloud water mixing ratio, log of cloud, log of humidity, log of humidity after-last-gscond, pressure thickness of each atmospheric layer, specific humidity, specific humidity after-last-gscond. It has the following 2 2D inputs: surface air pressure and surface air pressure after-last-gscond. The Fortran implementation uses the after-last-gscond inputs to infer a relative humidity tendency. Before being passed to the ML models the 2D inputs are replicated in the vertical direction, and all the variables are stacked along a channel dimension. The final ML input $x \in \mathbb{R}^{79} \times \mathbb{R}^{14}$. This input vector is then centered by removing the per-level and per-channel mean $\mu \in \mathbb{R}^{79} \times \mathbb{R}^{14}$ computed over a subset of samples. Then it is normalized by the standard deviation $\sigma \in \mathbb{R}^{14}$ computed over all vertical levels (79) and samples.

C.2 Condensation

C.2.1 Regression Model

The output of the condensation model is the change in cloud water $\Delta_g c$ over a single step of the `gscond` subroutine. The condensation rate depends exponentially on temperature due to the Clausius-Clapeyron condition so we use a temperature-dependent normalization for the target. We divide the range of temperatures into 50 equally spaced bins and compute the mean $\mu(T)$ and standard deviation $\sigma(T)$ of the target within each bin from a subset of the training data. In initial tests, this improved the schemes performance in cold regions like the upper troposphere and Antarctica, though we have yet to show this with formal ablations of our finalized configuration. Finally, the temperature scaled data are normalized once more with a per-level mean and all-level standard deviation.

The same neural network $f : \mathbb{R}^{12} \to \mathbb{R}$ is used for each level independently. It is a multi-layer perceptron, with 2 hidden layers of 256 nodes each and ReLU activation. It outputs the temperature-scaled target $\Delta_g c / \sigma(T)$. To ensure the model has skill both in temperature-scaled and the final output cloud $c_o = c_i + \Delta_g c$ we include both terms in the mean-squared error loss function given by

$$\left| \frac{\Delta_g c - \mu(T)}{\sigma(T)} - f(x) \right|^2 + \lambda |\tilde{c}_o - c_o|^2.$$

The predicted output cloud is $\tilde{c}_o = f(x)\sigma(T) + \mu(T)$. The weight $\lambda = 50\,000/Std[c_o]$ where Std is the standard deviation computed from a subset of the training data. This is chosen to so that both the predicted tendency and output cloud have $O(1)$ contributions to the loss.

C.2.2 Classifier

For online application, we use a classifier to handle the mixed discrete-continuous nature of the target variable $\Delta_g c$. It has the same architecture as the the regression model above, except with four target variables to identify the following classes:

- $\Delta_g c = 0$
- cloud after gscond vanishes but none of the above,
- $\Delta_g c > 0$ but none of the above, and
- $\Delta_g c < 0$ but none of the above.

In online/offline evaluations, if the classifier identifies the first two cases then we enforce the corresponding constraint on $\Delta_g c$. Otherwise we use the regression model. It is trained with a categorical cross entropy loss with the same hyperparameters as the regression models except for an increased learning rate of 0.001. After training the classifier is 98% accurate over all classes and levels.

C.3 Precipitation

The precipitation emulator (precpd) has the same input set and input normalization strategy as gscond, the states at the beginning of the time-step. However, it predicts these 3D outputs: the

change in air temperature, humidity, and cloud due to the precipitation subroutine. It also predicts surface precipitation rate which will moisten with the FV3GFS land-surface. All the outputs are scaled by the standard deviation $\sigma \in \mathbb{R}$ computed over all vertical levels. The loss function includes MSE terms for these direct predictands but also MSEs for the absolute values of cloud water, humidity, and air temperature at the end of `precpd`. These MSEs are scaled empirically to have $O(1)$ contribution to the overall loss.

The precipitation emulator is vertically non-local unlike `gscond`. To enforce the prior that rain falls down we use an recurrent neural network architecture (RNN). Let $x[i] \in \mathbb{R}^c$ be a vector with c channels from level $i \in [0, 79)$, where $i = 0$ is the top of atmosphere and $i = 78$ is the surface. Similarly let $h[i]$ be a hidden state vector such that $h[i] = 0$. Then a single layer of the RNN is given by $h[i+1] = (W_h h[i] + W_x x[i] + b)^+$, where $(\cdot)^+$ is the ReLU activation function. We stack two such layers, with a hidden states of 256 channels. Affine transformations map the final hidden state to the output fields without mixing vertical levels like this $y[i] = Ah[i] + b$. This architecture therefore ensures that $\nabla_{x[j]} y[i] = 0$ if $j > i$.
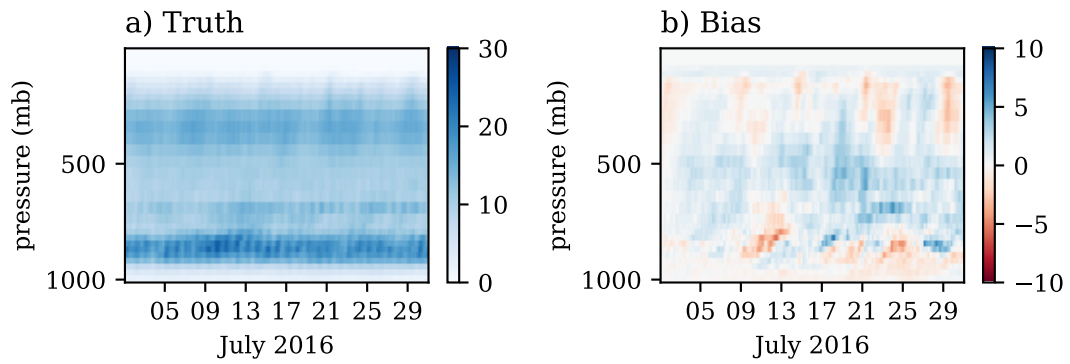
D Other aspects of online performance

Figure 3: The global average cloud water mixing ratio $(\mathrm{mg\,kg}^{-1})$ as a function of time and pressure. It shows (a) the truth and (b) bias of the emulated run.