JCM v1.0: A Differentiable, Intermediate-Complexity Atmospheric Model

Ellen H. Davenport[1,*], J. Varan Madan[1,*], Rebecca Gjini[1, 2], Jared Brzenski[1], Nick Ho[1], Tien-Yiao Hsu[1], Yueshan Liang[1], Zhixing Liu[1], Veeramakali Manivannan[1], Eric Pham[1], Rohith Vutukuru[1], Andrew I. L. Williams[1], Zhiqi Yang[1], Rose Yu[3], Nicholas J. Lutsko[1], Stephan Hoyer[5], and Duncan Watson-Parris[1,4]

[*]These authors contributed equally to this work.
[1]Scripps Institution of Oceanography, University of California San Diego, La Jolla, USA
[2]Cecil H. and Ida M. Green Institute of Geophysics and Planetary Physics, University of California San Diego, La Jolla, USA
[3]Department of Computer Science and Engineering, University of California San Diego, La Jolla, USA
[4]Halıcıoğlu Data Science Institute, University of California San Diego, La Jolla, USA
[5]Google Research, Mountain View, CA, USA

Correspondence: Duncan Watson-Parris (dwatsonparris@ucsd.edu)

Abstract. In this paper we present version 1.0 of the JAX Circulation Model (JCM). JCM is an open-source, differentiable atmospheric model built in Python using the JAX numerical library. Earth system modeling is rapidly evolving, particularly through hybrid approaches that combine known dynamics with data-driven components. However, the training and validation of hybrid methods in traditional models remain difficult due to the absence of gradients and the complexity of legacy code. Differentiable models written in modern frameworks offer a path forward. JCM couples physics parameterizations to the Dinosaur dynamical core through a flexible interface that makes substitution of other schemes easy. The default parameterization scheme uses the SPEEDY (Simplified Parameterizations, primitivE-Equation DYnamics) intermediate-complexity physics scheme. This modularity supports benchmarking across physical and machine-learned schemes, with direct access to gradients for sensitivity analysis, calibration, and online learning. We show validation of JCM against the original Fortran SPEEDY code at T31 resolution. We also highlight JCM's differentiability and efficiency and outline plans for extending the framework to a differentiable Earth system model. JCM provides a lightweight yet expressive platform for accelerating research in climate modeling.

1 Introduction

Earth system models (ESMs) are critical tools for understanding and forecasting weather and climate, but the execution of these tasks is plagued by computational and scientific challenges (Shaw and Stevens, 2025). Predictions of the climate system are complicated by uncertainties in model architecture, initial and boundary conditions, and model parameters (Hawkins and Sutton, 2009). Furthermore, high-resolution or long-running global simulations are prohibitively expensive. Many strategies have been, and continue to be, developed to address some of these issues. For example, large ensembles can address uncertainty quantification (Kay et al., 2015; Rodgers et al., 2021; Eidhammer et al., 2024), hybrid models can leverage both data-driven and deterministic methods (Arcomano et al., 2020; Slater et al., 2023; Arcomano et al., 2023; Yu et al., 2023), and Bayesian inference can improve estimates of model parameters and forcings (Watson-Parris et al., 2021; Watson-Parris, 2025; Regayre et al., 2023). However, the development and intercomparison of these approaches in traditional climate models is hindered by complex legacy code, a lack of derivative information, and the inability to use modern hardware accelerators. Continued improvement of ESMs requires an upgrade in modeling infrastructure that takes advantage of state-of-the-art computational tools, particularly modern software frameworks (Gelbrecht et al., 2023).

Differential equations describe the evolution of dynamical systems and are common across a variety of disciplines. Gradients of these functions are relevant for analyzing input-output relationships, quantifying the influence of interactions between parameters, and assessing the robustness of results (Sapienza et al., 2025). In the case of ESM components such as atmosphere and ocean dynamics, ice flow, ecology, etc., the set of relevant differential equations is large and nonlinear. The analytical gradients of these functions are unknown and intractable to derive or assess by hand. Automatic-differentiation (AD), also known as algorithmic-differentiation, uses software to compute the gradient of a function by systematically applying the chain rule. This technique has been in use for nearly 30 years but was historically tedious to implement (Giering and Kaminski, 1998). More recently, the explosion of machine learning (ML) applications has elevated the need for gradient information in order to train algorithms through back-propagation. This has led to the inclusion of AD in more accessible languages, such as Python, and has made differentiable programming a foundation of modern scientific computing (Blondel and Roulet, 2024). Differentiable software provides a common framework for Bayesian inference, inverse methods, optimization, and ML, leveraging decades of discovery from highly related fields of science and engineering (Sapienza et al., 2025).

One of the most promising uses of AD in geoscience is the online training of hybrid climate models (Rasp, 2020; Kochkov et al., 2024). Hybrid models combine ML components with traditional dynamical models. Data-driven components can support or replace existing physics parameterizations that are simplified and uncertain, with the hope that leveraging large amounts of data can improve model performance. The well-known equations that describe large-scale dynamics (i.e., a dynamical core) are left untouched. Popular candidates for data-driven parameterizations are atmospheric convection, cloud-aerosol interactions, and ocean turbulence. In theory, hybrid methods have the potential to benefit from the robustness of a deterministic dynamical core and the improved accuracy of ML-based parameterizations. In practice, the integration of ML components trained offline with dynamics is finicky and unstable (Rasp et al., 2018; Brenowitz and Bretherton, 2019). Differentiability allows for simultaneously tuning ML and physics-based parameterizations over multiple time steps, accounting for the online effects of coupling to a dynamical core. This can significantly improve the stability and accuracy of the resulting hybrid models, as exemplified by NeuralGCM, a hybrid atmospheric model that replaces the full physics-suite with a neural network (Kochkov et al., 2024).

To evaluate the potential of hybrid climate modeling, there is a need for computationally efficient physics-based climate models with AD capabilities. Simplified chaotic models such as Lorenz 96 (Lorenz, 1995) do not provide a sufficiently complex test bed to meaningfully evaluate the merits of hybrid modeling (Rasp, 2020). NeuralGCM includes a realistic atmospheric dynamical core, but lacks physical parameterizations required for process-based understanding. Even without considering the availability of AD tools, running traditional climate models is computationally expensive. Fortran climate models were originally designed to run on CPUs and are extremely costly, if not impossible, to port to state-of-the art processors. This is a fundamental limitation for their use in rapidly evolving climate modeling research with limited resources.

The JAX Circulation Model (JCM) is an intermediate-complexity atmospheric circulation model written in the Python programming language that takes advantage of the JAX library for hardware acceleration and differentiability (Bradbury et al., 2018). Python is the most commonly used programming language in the world, and its libraries are especially powerful for scientific analysis (e.g., Xarray and Dask) and ML (e.g., PyTorch and TensorFlow) (Hoyer et al., 2014; Dask Development Team, 2015; Paszke et al., 2016; Abadi et al., 2015). JAX is conveniently designed to use the familiar NumPy interface for array operations, while additionally providing efficiency and differentiability (NumPy Developers, 2006). Furthermore, programs written in Python and JAX can be deployed on central, graphics and tensor processing units (CPUs, GPUs and TPUs respectively) without modifications to the code. JCM takes advantage of these tools and thus is a user-friendly, differentiable model that is designed for hardware-acceleration and parallelism. Version 1.0 includes parameterizations for condensation, convection, radiation, and other atmospheric phenomena, yet is simplified compared to the most realistic models. JCM's modern software framework and representative physics schemes make it an ideal test bed for hybrid modeling and other AD applications.

This paper documents the version 1.0 release of the open-source JCM software and is organized as follows. Section 2 describes the model architecture and Section 3 shows examples of the model interface. Model validation and performance are discussed in Sections 4 and 5. Section 6 discusses the use of AD in JCM in detail, including example applications. Finally, a summary and description of future directions is included in Section 7.

2 Model Architecture

JCM consists of a spectral dynamical core (dycore) and a suite of default physics parameterizations. The physics parameterizations are designed to be modular and easily swappable for future additions (e.g., replaceable by other physical parameterizations, prescribed forcings, or ML models). This section describes the dycore (dynamics), default physics parameterizations (physics), and the architectural choices that define version 1.0.

2.1 Dynamical core

The dycore of JCM is called Dinosaur and was developed for use in NeuralGCM (Kochkov et al., 2024). The Dinosaur dycore is spectral and uses sigma levels (surfaces at fixed fractions of the surface pressure) for the vertical coordinate. It integrates the hydrostatic equations with a semi-implicit gravity wave scheme in addition to specified physics "terms" (see 2.2). JCM prognostic variables include horizontal wind (vorticity and divergence), temperature (anomaly from a reference temperature), geopotential, and log surface pressure. Additional prognostic variables can be specified as tracers. In the default configuration of JCM, the only additional tracer is specific humidity. Dinosaur supports a range of horizontal grid resolutions, labeled by their spectral truncation: the grid for spectral truncation N has a grid box size that can resolve fluctuations at spatial frequencies up to the Nth spherical harmonic. Dinosaur supports a wide range of grid resolutions between T21 and T425, with JCM's default being T31 (roughly $4^\circ \times 4^\circ$ grid cell size at the equator). Dinosaur is capable of operating with many vertical levels. In JCM we use the 8 sigma levels expected by the default physics suite (described below). Sigma layer boundaries are at σ values of 0, 0.05, 0.14, 0.26, 0.42, 0.60, 0.77, 0.90 and 1. Prognostic variables other than log surface pressure are specified at the center of the sigma levels (σ = 0.025, 0.095, 0.20, 0.34, 0.51, 0.685, 0.835, 0.95).

Time integration in JCM is performed with the IMEX (implicit-explicit) SIL3 (semi-implicit Lorenz three-cycle) Runge-Kutta scheme, which treats the linear terms of the primitive equations implicitly and the nonlinear advection and physics terms explicitly (Whitaker and Kar, 2013). IMEX SIL3 is second order accurate (discretization error scales with the square of the timestep) for the implicit terms and third order accurate for the explicit terms. The SPEEDY Fortran model uses a leapfrog scheme, but for simplicity we use IMEX SIL3, which does not require keeping track of state at multiple time-steps. The default time step is 30 minutes, although this can be easily varied at model initialization time. At each time step, several filters are applied to the model state. First, an exponential filter is applied to suppress computational modes. Horizontal diffusion of the prognostic variables is computed with another filter, and finally the surface pressure field is corrected to maintain constant global mean surface pressure (mass conservation).

2.2 Physics parameterizations

JCM uses the physical parameterizations from SPEEDY (Simplified Parameterizations, primitivE-Equation DYnamics) version 41, which have been translated from Fortran90 to JAX. SPEEDY is an intermediate-complexity general circulation model developed at the International Centre for Theoretical Physics (ICTP) (Molteni, 2003). SPEEDY represents physical processes with simplified schemes that operate on 8 vertical levels. The top two layers represent the stratosphere, the bottom layer represents the planetary boundary layer, and the intermediate layers represent the free troposphere. SPEEDY has been used in a large number of studies, both on its own as well as in coupled contexts (with many different ocean models). Its lightweight design makes it an efficient sandbox for prototyping numerical schemes (Amezcua et al., 2011) or data assimilation methods (Sluka et al., 2016; Cintra and de Campos Velho, 2014; Ruiz and Pulido, 2015), looking at long-term climate interactions (Kucharski et al., 2006), or looking at the large-scale impacts of a wide range of climate forcings (Kucharski et al., 2013).

The SPEEDY parameterizations were developed by simplifying more complex physics schemes while retaining underlying physical principles. These simplifying assumptions are specifically intended to work for a model with few (order 10) vertical levels. Here we provide a brief overview of each of the SPEEDY physics parameterizations, and full details are available in the original model description paper (Kucharski and Molteni, 2013). The primary difference between the SPEEDY paramterizations and JCM is the handling of horizontal diffusion and orographic corrections, which is described in detail below.

Convection: Convection is simulated with a simplified mass-flux scheme based on Tiedtke's 1989 paper (Tiedtke, 1989). Convection is activated in areas meeting two simultaneous criteria for conditional instability: humidity in the planetary boundary layer (the lowest layer) exceeding a prescribed threshold, and saturation moist static energy decreasing with height between the PBL and upper troposphere (the highest non-stratospheric layer). The cloud-base mass flux (at the top of the PBL) is calculated to relax the PBL humidity to its threshold value on a time scale of 6 hours. The air in the updrafts is assumed to be saturated. Entrainment occurs in the lower half of the troposphere, and detrainment occurs only at the cloud-top level (determined by the conditional instability criterion).

Large-Scale Condensation: Each sigma level has an associated threshold for relative humidity; when relative humidity exceeds this threshold, it is reduced back to the threshold value with a 4-hour relaxation timescale, and the equivalent amount of latent heat is added to the dry static energy.

Clouds: Non-stratocumulus cloud cover (and thickness) is diagnosed from the amount of precipitation and the free-tropospheric (excluding stratosphere and PBL layers) relative humidity profile in each air column. Stratocumulus clouds are diagnosed from the static stability in the PBL.

Shortwave Radiation: SPEEDY represents shortwave radiation with a two-band radiation scheme, with one band for the near-IR part of the spectrum. The shortwave transmissivity of each model layer is computed as a function of layer mass, specific humidity, and cloud cover. Shortwave reflection occurs at cloud tops and at the surface.

Longwave Radiation: SPEEDY's longwave radiation scheme has four spectral bands. One band represents the atmospheric 'window,' while the other three bands represent the spectral regions of absorption by water vapor and carbon dioxide. Layer transmissivities are computed from layer mass and humidity, with cloud cover being represented as a decrease in 'window' band transmissivity. At each layer, the downward longwave radiation flux is computed as a weighted function of air temperature at the full level and the interpolated air temperature at the half-level below. Upward flux is computed similarly, using the half-level above instead.

Surface Fluxes of Momentum and Energy: Fluxes of momentum and energy from the surface to the atmosphere are computed using bulk aerodynamic formulas; one set of exchange coefficients is used over land and another set is used over ocean. For momentum flux, the land coefficient is a function of topographic height. For sensible and latent heat fluxes, the coefficients depend on a simple near-surface stability index based on surface energy balance. Evaporation is estimated using a scheme similar to that outlined in Viterbo and Beljaars (1995).

Vertical Diffusion: SPEEDY's vertical diffusion module simulates three processes. First, shallow convection is simulated by mixing moisture and dry static energy between the lowest two model layers. Second, lower-troposphere diffusion of water vapor in stable conditions is estimated based on the column's humidity profile. Third, diffusion is computed for dry static energy in areas where air temperature approaches a dry-adiabatic lapse rate.

Horizontal Diffusion: SPEEDY incorporates implicit horizontal diffusion filtering into its leap-frog time integration, in addition to utilizing a Robert-Asselin-Williams filter (Williams, 2011). As mentioned above, JCM does not use leap-frog and instead uses Runge-Kutta. The exact same horizontal diffusion implementation is therefore not appropriate. In Version 1.0, JCM instead implements explicit Laplacian horizontal diffusion of the prognostic state. The timescales and order of the filter vary across the elements of the state. Divergence is diffused at a timescale of two hours with filter order 1. A second order filter is applied to vorticity and specific humidity with a timescale of 12 hours and to temperature with a timescale of 24 hours. The longer timescales associated with vorticity, temperature, and humidity are intended to preserve strong gradients that are important to drive realistic atmospheric circulation. Tuning and improvement of this filtering scheme is subject to future work.

Orographic Corrections: JCM does not yet implement orographic corrections in Version 1.0, but this is a subject of future work. The absence of these corrections is one of the primary differences between the JAX and Fortran implementations, in addition to the differences in horizontal diffusion.

3 Model Interface (API)

JCM's documentation is available at https://jax-gcm.readthedocs.io. A key feature of the model is the simple and intuitive user interface in Python. Users construct a `jcm.Model` object with their desired grid geometry (including orography), and desired suite of physics parameterizations (e.g. SPEEDY), then call `Model.run`, specifying run length, (optionally time-varying) forcing terms, and the frequency at which to save output. Users can also specify a custom initial state and land surface boundary conditions. `Model.run` returns a struct containing the values of the prognostic dynamics variables and diagnostic physics variables over the course of the run. The struct can effortlessly be converted to an Xarray dataset.

```python
from jcm.geometry import Geometry
from jcm.forcing import ForcingData
from jcm.model import Model
from jcm.physics.speedy.speedy_physics import SpeedyPhysics

geometry_T31 = Geometry.from_file('terrain_t31.nc') # includes orography
model = Model(geometry=geometry_T31, physics=SpeedyPhysics())

forcing_T31 = ForcingData.from_file('forcing_daily_t31.nc') # includes surface boundary conditions
predictions = model.run(forcing=forcing_T31)

preds_dataset = predictions.to_xarray()
```

4 Model Validation

In this section we provide validation of JCM against the version of the SPEEDY model implemented in speedy.f90 (Hatfield, 2022). We evaluate the prognostic state in addition to the global climatology of parameterized quantities such as cloud cover and precipitation.

4.1 JCM versus SPEEDY

Both JCM and speedy.f90 simulations begin at isothermal rest and run for three years. The models are forced by a monthly climatology of sea-surface temperature with realistic orography. This monthly climatology is obtained from ERA-Interim over the period 1979-2008 (Dee et al., 2011). In addition to sharing initial and boundary conditions, both simulations are run at T31 resolution with a time step of 40 minutes. The first three months of the model runs are discarded as model "spin up". Figures 1 through 4 validate the mean state of JCM against the mean state of speedy.f90. Figure 1 shows surface temperature and specific humidity in both models. The root-mean-square (RMS) difference between the two simulations is shown in the bottom row. As mentioned in Section 2, JCM does not yet have orographic corrections. This leads to slight differences between the two models in temperature and specific humidity over land. The average RMS difference at the surface is 1.8 degrees C and 0.68 g/kg.

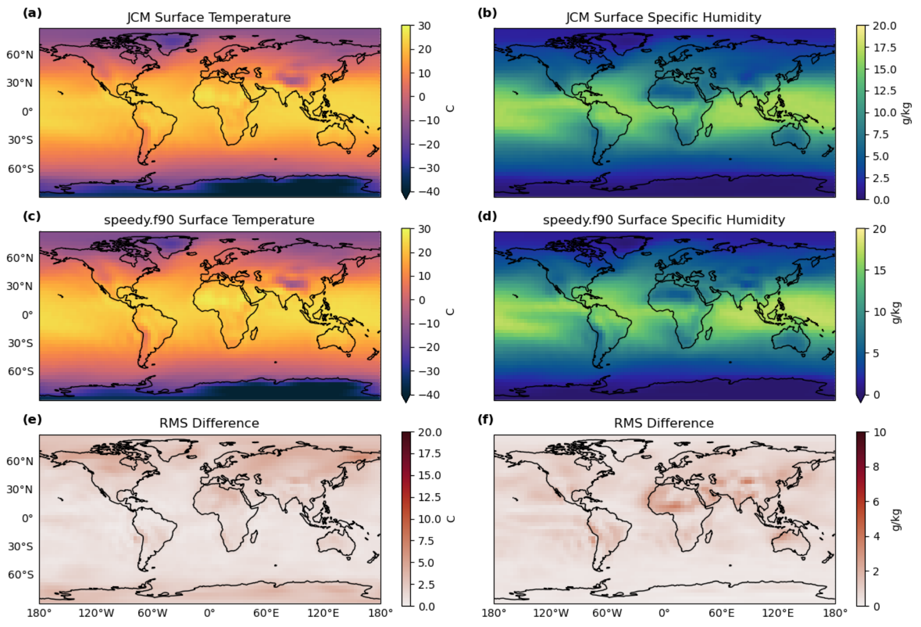
(a)
JCM Surface Temperature
(b)
JCM Surface Specific Humidity
(c)
speedy.f90 Surface Temperature
(d)
speedy.f90 Surface Specific Humidity
(e)
RMS Difference
(f)
RMS Difference
C
g/kg

Figure 1. Average surface temperature and surface specific humidity in the surface layer in JCM (a and b) and in speedy.f90 (c and d). The root-mean-square (RMS) difference between the two simulations is shown in panels (e) and (f).

Figure 2 shows the average meridional (left) and zonal (right) wind in the surface layer. Again the RMS difference is shown in the bottom row. Large-scale patterns in meridional wind are the same between the two models with slight differences in strength, with JCM winds tending to be stronger than in speedy.f90. The zonal winds are generally stronger everywhere in JCM than in speedy.f90. This difference is most notable in the mid-latitude zonal winds. This discrepancy between the model runs is likely due to the (1) differences in horizontal diffusion and (2) the application of additional wind drag, which is present in speedy.f90 and absent in JCM. These architectural choices are also described in Section 2. The average RMS difference of meridional and zonal winds at the surface are 2.1 m/s and 3.5 m/s respectively. Figure 3 shows a comparison of the winds at the top of the troposphere. Again, the large-scale patterns are qualitatively similar, but JCM winds are stronger than in speedy.f90. The average RMS difference of meridional winds at the top of the troposphere is similar to at the surface, but the average RMS difference in the zonal winds is roughly 9 m/s.

4.2 Cloud Cover and Precipitation

In addition to validating the prognostic JCM state against SPEEDY, we verify the spatial patterns of cloud cover and precipitation. Figure 4 shows average stratiform cloud cover fraction, precipitation due to convection, and precipitation due to large scale condensation. The panel (a) indicates that low clouds are most prevalent along the eastern boundaries of ocean basins in both JCM and speedy.f90, with average cloud cover near 40%. This pattern agrees well with the climatology described by (Hartmann, 2016). Panels (b), (c), (e) and (f) show precipitation patterns associated with different physical processes. Con-

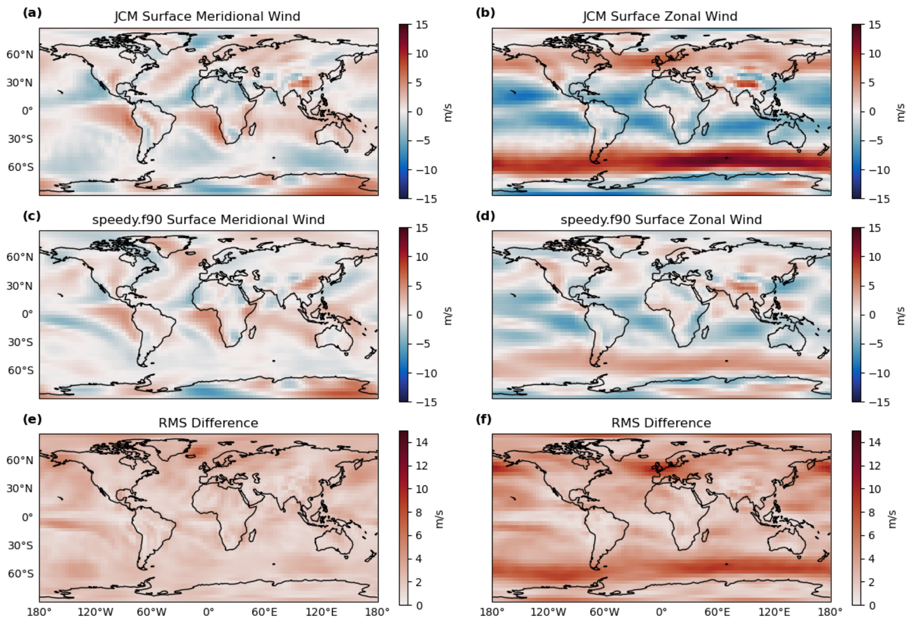

Figure 2. Average meridional and zonal winds in the surface layer in JCM (a and b) and in speedy.f90 (c and d). The RMS difference between the two simulations is shown in panels (e) and (f).

vective precipitation (b and e) is highest in the tropics, and is particularly strong near the inter-tropical convergence zone (ITCZ), the Pacific warm pool, the Indian Ocean, and the equatorial Atlantic. While JCM and speedy.f90 show the same spatial patterns, precipitation due to convection is more concentrated in speedy.f90 and more diffuse in JCM. Precipitation due to large-scale condensation (c and f) is highest in the mid-latitudes with hotspots over the North Pacific Ocean, the Gulf Stream, the Himalayas, and the Southern Ocean. Large-scale condensation appears to be slightly stronger in JCM than in speedy.f90.

5 Performance

JCM is lightweight enough to be practical for running on even small-scale systems such as a personal laptop CPU. For higher grid resolutions, model runs longer than a few months of simulation time, or intensive gradient computations, JCM is best run with some form of hardware acceleration (i.e. on a GPU or TPU), which is natively supported through JAX. Figure 5 shows the total time to run a default configuration of JCM at T85 (approx. 150km resolution at the Equator) using either a laptop CPU, laptop GPU (NVIDIA RTX 4050), or Google Cloud TPU. The behavior in the figure has been our typical experience: runtime on GPUs and TPUs is at least an order of magnitude faster than on CPUs (except for Apple Silicon chips), with parallelization (sharding the model state arrays so they can be distributed among multiple devices) giving a significant speedup to the TPU runs on Google Cloud. For this set of parameters, the laptop GPU was faster than the cloud TPUs, but initial testing indicates that the TPU (like any high-performance cloud platform) will scale better for higher resolutions than the laptop GPU with its

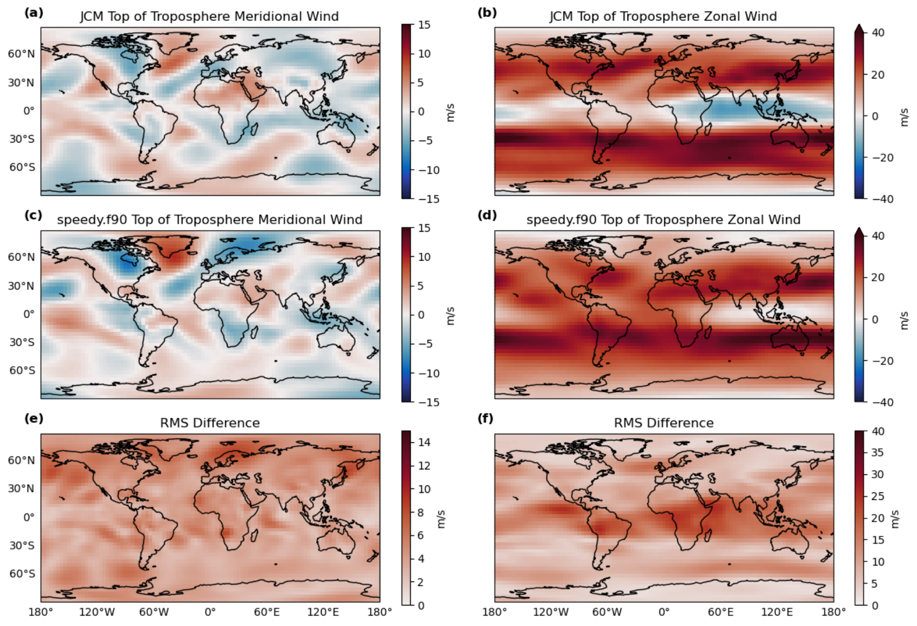
(a)
JCM Top of Troposphere Meridional Wind
(b)
JCM Top of Troposphere Zonal Wind
(c)
speedy.f90 Top of Troposphere Meridional Wind
(d)
speedy.f90 Top of Troposphere Zonal Wind
(e)
RMS Difference
(f)
RMS Difference
m/s
60°N
30°N
0°
30°S
60°S
180°
120°W
60°W
0°
60°E
120°E
180°

Figure 3. Average meridional and zonal winds at the top of the troposphere in JCM (a and b) and in speedy.f90 (c and d). The top of the troposphere, as defined by the SPEEDY model, is the third sigma layer from the top of the atmosphere. The RMS difference between the two simulations is shown in panels (e) and (f).

limited VRAM. The platforms used here do not all have the same number of cores or clock speeds; the goal is to illustrate typical runtime on common devices.

As another point of comparison, JCM on the RTX 4050 laptop GPU was about 40% faster per year of model time than speedy.f90 on the same laptop's Intel i9 13th Gen CPU.

Beyond basic vectorization of loops and conditionals in the code logic, no effort has yet been made to optimize JCM's performance on all platforms, especially for cloud computing at high resolution. As such, we expect there will be significant performance improvements in future.

With regards to the model run configuration itself, the number of operations for a model run typically increases with the square of the spectral truncation, with an additional factor for reductions to the model timestep (sometimes needed for stability at a higher resolution). However, runtime is observed to scale better than the number of operations, presumably due to parallelization. Figure 6 shows how model runtime scales with the spatial resolution of the model grid, using as a benchmark a one year simulation with default 30 minute time step, run on a NVIDIA RTX 4050 laptop GPU. At T31 resolution, one year of model time takes only a minute and a half, and even T119 can be run for a year in less than 10 minutes.

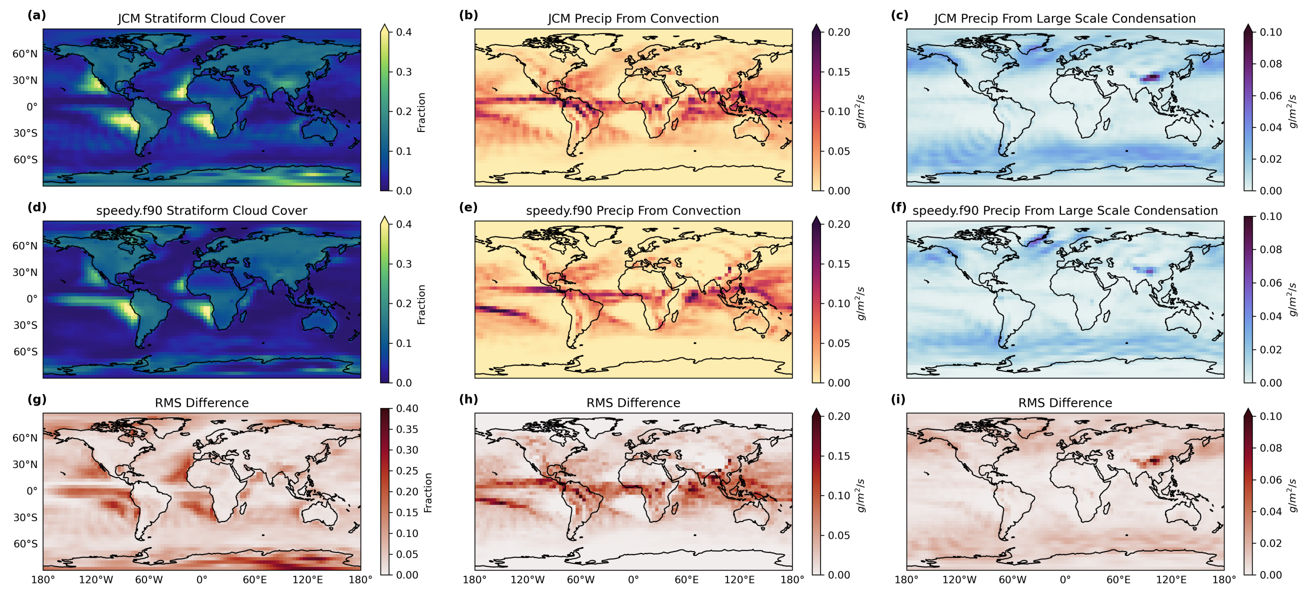
(a) JCM Stratiform Cloud Cover
(b) JCM Precip From Convection
(c) JCM Precip From Large Scale Condensation
(d) speedy.f90 Stratiform Cloud Cover
(e) speedy.f90 Precip From Convection
(f) speedy.f90 Precip From Large Scale Condensation
(g) RMS Difference
(h) RMS Difference
(i) RMS Difference
Fraction
g/m²/s
60°N
30°N
0°
30°S
60°S
180°
120°W
60°W
0°
60°E
120°E
180°

Figure 4. Average stratiform cloud cover (a and d), precipitation due to convection (b and e), and precipitation due to large scale condensation (c and f). RMS difference between JCM and speedy.f90 is shown in panels (g)-(i)

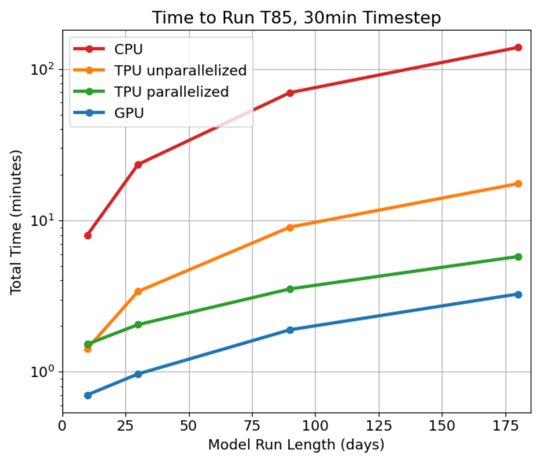

Figure 5. Model runtimes on different hardware.

6 Gradients

One of the benefits of JAX is the accessibility of gradient information. JCM gradients can be taken with respect to inputs (e.g., initial conditions and forcing), outputs (e.g., the future model state), and parameters (e.g., SPEEDY physics parameters). In this section we explain and highlight uses cases of these gradients including calibration and sensitivity analysis.

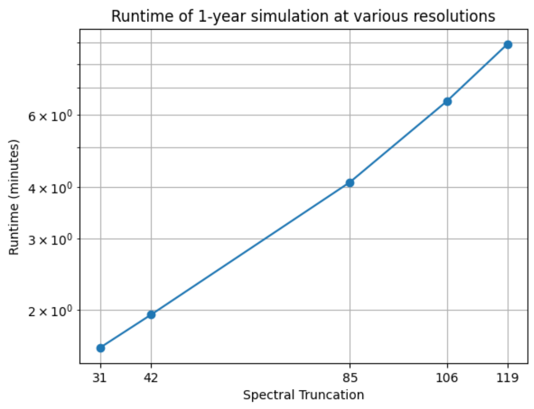

Figure 6. Model runtimes at different resolutions.

6.1 Automatic-differentiation (VJP and JVP)

Automatic differentiation (AD, also called algorithmic differentiation) computes derivatives of functions by recursively applying the chain rule to the computational graph. When AD is not available, tangent linear and adjoint models must be manually derived and implemented. Given the complexity of state-of-the-art Fortran models, this manual implementation of gradients is generally intractable and derivative information is typically inaccessible.

JAX provides JCM with its AD capabilities (Bradbury et al., 2018). In JAX, gradients can be computed in either forward (tangent linear) or reverse (adjoint) mode. Each mode has multiple use cases and can be applied to a wide variety of science problems. Reverse mode differentiation is most easily accessed through computation of the vector Jacobian product (VJP). VJP evaluates the multiplication of a user-specified co-tangent vector with the adjoint of the function. This function might be a loss function of an optimization problem that is wrapped around the forward model trajectory. VJP is computationally efficient when the output space of the function is much smaller than the input space. Forward mode differentiation is applied through the Jacobian vector product (JVP). This evaluates the multiplication of a user-specified tangent vector with the tangent linear model of the function. In contrast to VJP, JVP is computationally efficient when the output dimension is larger than or equal to the input dimension of a function. Tangent linear models are especially useful for sensitivity experiments. Both VJP and JVP avoid computing the entire Jacobian of the function to minimize computational cost, but they can be used to do so if necessary. Example applications are discussed in more detail in the following sections.

6.2 Calibration

Calibration of GCMs against observations is a problem of interest to the geoscience community and benefits from gradient information. The goal of GCM calibration is to estimate physics parameters that generate the best match between a model trajectory and observations. Without gradient information, this process often involves hand-tuning by domain experts. Hand-tuning is limited in the degree to which it utilizes observational information and leaves much of the parameter space unexplored.

With gradient information, we can take a more systematic approach to calibration using optimization techniques. Calibration with gradient information allows for the assessment of model behaviour across more of the parameter space.

A popular approach for GCM calibration is to estimate model parameters against climate statistics (Kennedy and O'Hagan, 2001; Schneider et al., 2017, 2024). The optimization problem is formulated to minimize a loss defined by the misfit between observed and modeled statistics. In order to illustrate how JCM can be calibrated using AD, we show an idealized example estimating the value of stratiform cloud albedo. For the sake of demonstration, we generate synthetic observations y by running the forward model with the "true" value of stratiform cloud albedo $= 0.5$. As mentioned above, we are interested in optimizing over the parameter space to find the value that most closely reproduces "observed" statistics. This simple example is not realistically complex but can be adapted or extended to use any climate statistics and any set of parameters. We define a function $\mathcal{G}(\cdot)$ that takes the input parameter x, runs a 5-day JCM simulation with stratiform cloud albedo = x, and outputs model statistics from the trajectory.

The loss function $\mathcal{L}(\cdot)$ is defined as

$$\mathcal{L}(x) = \frac{1}{2}\left\|R^{-\frac{1}{2}}y - \mathcal{G}(x)\right\|_2^2, \tag{1}$$

where $\|\cdot\|_2$ defines the L-2 norm, and R is the observational covariance matrix. R is assumed to be a diagonal matrix with variances = 1. We use a gradient descent optimizer to solve for the value of x that minimizes the loss. There are many other types of optimization algorithms, but given the low complexity and smooth loss landscape, we chose the simplest approach. The gradient descent update is as follows:

$$x_{k+1} = x_k - \alpha\left(\frac{\partial \mathcal{L}}{\partial x_k}\right), \tag{2}$$

where x_k denotes the parameter value at iteration k, α denotes the step size, and the search direction is the negative gradient of the loss with respect to x_k. We initialize the first estimate as $x_0 = 0.1$ and do 7 gradient descent iterations before the algorithm converges.

Figure 7 shows calculated loss for different values of x_k, and it can be seen that the lowest loss is at the "true" value, $x_k = 0.5$. The gradient descent steps are plotted as orange dots to show the loss at each iteration. We obtain $\partial\mathcal{L}/\partial x_k$ using reverse mode automatic differentiation, providing the direction for the next parameter update. The gradient descent optimizer successfully identifies the value of stratiform cloud albedo used to generate the "true" statistics.

6.3 Sensitivity analysis

JVP facilitates sensitivity experiments with JCM. When using JVP, the tangent vector is effectively a perturbation that is propagated through the tangent linear model. If the function we differentiate is a JCM trajectory, then JVP gives us the evolution of an input perturbation to the model. This could be a perturbed parameter value, model forcing, or initial conditions. We wrap the model run in a function $\mathcal{F}(\cdot)$, where argument x is the model input we will perturb and output f is the model prediction(s) of interest. Forward mode automatic differentiation will calculate the change in the outputs, Δf, given a change in input Δx,

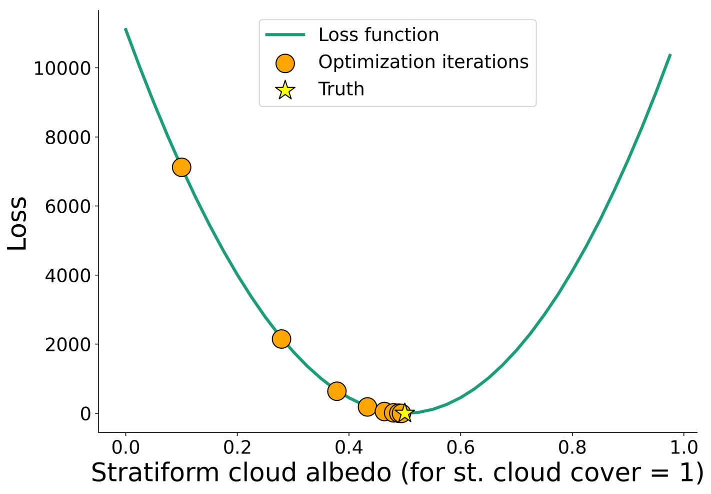

Figure 7. Loss versus stratiform cloud albedo. The orange dots show the calculated loss at each iteration of gradient descent, and the yellow star is the “true” value.

which is approximated by

$$\Delta f \approx \left.\frac{\partial \mathcal{F}}{\partial x}\right|_{x=x_0} \Delta x, \tag{3}$$

where x_0 is the unperturbed value of the input. JVP is used to evaluate the right-hand side of equations 3. The user only needs to provide the function $\mathcal{F}(\cdot)$ and the perturbation Δx (i.e., the tangent vector).

As an example, we show the influence of a sea surface temperature (SST) perturbation on the future state of a default JCM run. Here x is the SST forcing for the model run and f is the predicted meridional wind, zonal wind, and top-of-atmosphere outgoing longwave radiation. $\mathcal{F}(x)$ is a function that executes the forward run of JCM with the given SSTs and isolates the desired predictions. We apply an SST perturbation, Δx, to obtain corresponding changes in the predictions, Δf. JVP propagates the perturbation (tangent vector) through the tangent linear model of $\mathcal{F}(x)$. Figure 8(a) shows an example SST perturbation. Figure 8(b) shows the change in meridional wind, zonal wind, and TOA longwave radiation at 12, 24, and 36 hours in the future. JVP circumvents the need for large ensembles and finite differences to estimate model sensitivities and instead directly calculates those quantities.

As an alternative example, we analyze the sensitivity of the predicted top-of-atmosphere (TOA) outgoing longwave radiation with respect to the open-ocean surface albedo. Instead of perturbing a forcing field, we perturb a model parameter. In this scenario, our output f of the forward map $\mathcal{F}(\cdot)$ is the predicted TOA outgoing longwave radiation and our input x will be the JCM physics parameters. In order to extract the sensitivity of TOA outgoing longwave radiation to open-ocean surface albedo, we set all values of the tangent vector Δx to 0, except for the value corresponding to our parameter of interest which is set to 1. This method effectively extracts individual elements of the Jacobian. The gradient is always computed at the input value x_0, which in this case is the set of default parameter values from SPEEDY. The code block below implements the

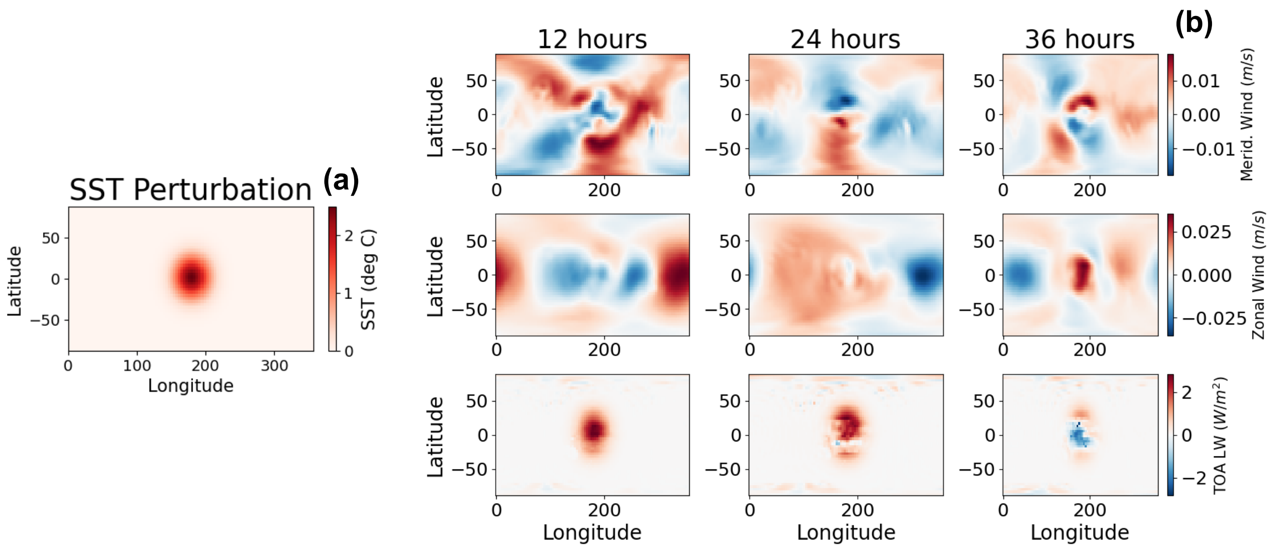

Figure 8. Influence of a sea surface temperature (SST) perturbation on the future model state. (a) Sea surface temperature (SST) perturbation in $^{\circ}C$. (b) Changes to meridional wind (row 1), zonal wind (row 2), and outgoing longwave radiation at the top of the atmosphere (row 3) at 12 hours (column 1), 24 hours (column 2), and 36 hours (column 3) in the future.

described experiment, calculating the gradient of f (`longwave_rad.ftop`) with respect to the open-ocean surface albedo (`Parameters.albsea`) using JVP (`jax.jvp`). Figure 9 shows the output of this calculation, which is the sensitivity of TOA outgoing longwave radiation after one and two days of simulation to a unit perturbation in open-ocean albedo.

```
13: # Differentiate model predictions with respect to the ocean albedo (Parameters.albsea)
14: import jax
15: from jcm.physics.speedy.params import Parameters
16: from jcm.utils import zeros_like_tangent
17:
18: # First, create a wrapper function to differentiate
19: def model_run_wrapper(params):
20:     model = Model(
21:         geometry=geometry_T31,
22:         physics=SpeedyPhysics(parameters=params) # inject parameters here
23:     )
24:     predictions = model.run(
25:         save_interval=1,
26:         total_time=2,
27:         forcing=forcing_T31,
28:     )
29:     return predictions
30:
31:
```

```
# Define the 'primal' (value of input at which to differentiate)
primal = Parameters.default()
# Define the tangent vector for directional derivative
tangent = zeros_like_tangent(primal)
tangent.mod_radcon.albsea = 1. # extract the derivative with respect to albsea

# Compute the jvp
predictions, d_predictions_d_albsea = jax.jvp(model_run_wrapper, (primal,), (tangent,))
# Convert output to xarray dataset:
# Use timestamps from predictions, because d(preds)/d(albsea).times = d(times)/d(albsea) = zeros
ds_dpda = d_predictions_d_albsea.replace(times=predictions.times).to_xarray()

# Plot sensitivity of ftop w.r.t albsea
ds_dpda['longwave_rad.ftop'].plot(x='lon', y='lat', col='time', col_wrap=1, vmin=0, aspect=2, size=3)
```

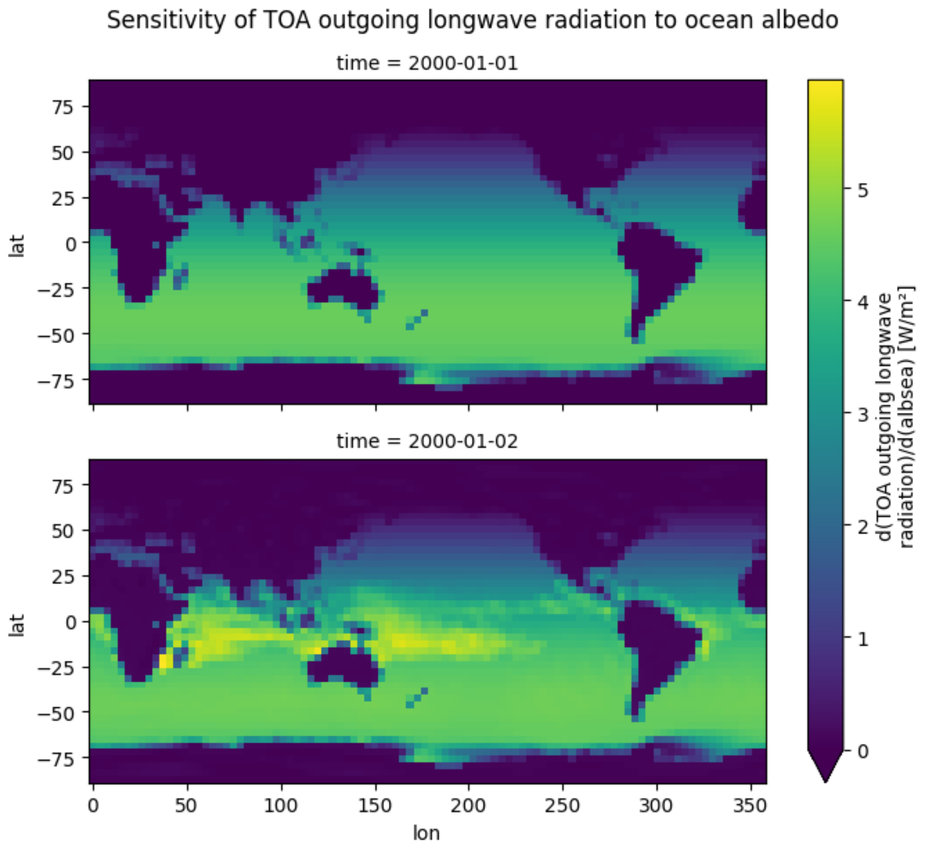
Sensitivity of TOA outgoing longwave radiation to ocean albedo
time = 2000-01-01
time = 2000-01-02
lat
lon
d(TOA outgoing longwave radiation)/d(albsea) [W/m²]

Figure 9. Derivative of predicted top-of-atmosphere outgoing longwave radiation with respect to changes in the sea-surface albedo parameter.

7 Conclusions

The JAX Circulation Model (JCM), is an open-source, differentiable atmospheric model written in Python using the JAX library. The design of JCM is user friendly and the software is hardware agnostic, making the model easy to run on personal machines while including capability for long, high-resolution simulations on GPUs and TPUs. Because it is written in Python, JCM is accessible to a wide range of developers, scientists, and educators. JCM v1.0 uses the SPEEDY model intermediate-complexity parameterizations which provide a valuable test bed for hybrid methodologies. Furthermore, model gradients obtained through automatic differentiation provide access to a broad set of optimization and estimation algorithms. These methods can be used for parameter estimation and sensitivity analysis, supporting experiments that serve to increase understanding of Earth system predictability.

Tools like JCM will help tackle unanswered climate science questions that are computationally challenging or require gradient information. Future JCM studies will test how gradient information changes the ability to calibrate atmospheric models against Earth observations. This will increase our understanding of climate model parameters and their effect on climate projections. We will continue investigating the JCM physics parameters through more realistic sensitivity experiments, providing insight on the dynamics and processes that drive atmospheric variability. These gradients can also be used to explore parameter uncertainty, where the efficiency of JAX will be advantageous to generate large ensembles.

JCM parameterizations are also flexible, meaning that physics schemes can be easily interchanged. "Plug-and-play" parameterizations allows for testing and intercomparison of new models and methods. ML architectures (i.e., neural networks) are currently under consideration to represent subgrid-scale processes that are poorly understood. In JCM, ML and traditional physics schemes can be tested side-by-side. ML components can be trained either offline or online within a calibration framework. Future versions of JCM will also include options with more comprehensive physics schemes such as parameterizations from the icosahedral nonhydrostatic (ICON) atmospheric general circulation model (Giorgetta et al., 2018). In parallel with JCM development, we are working towards a fully coupled ESM written in pure Python. This includes ongoing work to couple JCM to the Versatile Ocean Simulator (Veros, Häfner et al. (2021); Häfner et al. (2018)), along with efforts to integrate the Python and JAX land model, DifferLand (Fang et al., 2024). These next steps will extend the JCM framework, creating a lightweight, fully differentiable ESM framework that can aid in the acceleration of climate modeling research.

Code and data availability. All JCM code and associated data is available in the public repository, linked on JCM's doi page at https://doi.org/10.5281/zenodo.18189579 (Davenport et al., 2026b). The JCM and SPEEDY outputs and modified SPEEDY.f90 code used to generate figures 1, 2, 3 and 4 is available at https://doi.org/10.5281/zenodo.18334906 (Davenport et al., 2026a).

Acknowledgements. DWP and JVM acknowledge funding from U.S. National Science Foundation award 2441832. This research was supported with Cloud TPUs from Google's TPU Research Cloud (TRC).

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Józefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, https://www.tensorflow.org, software library for numerical computation using data flow graphs, 2015.

Amezcua, J., Kalnay, E., and Williams, P. D.: The effects of the RAW filter on the climatology and forecast skill of the SPEEDY model, Monthly Weather Review, 139, 608–619, https://doi.org/10.1175/2010MWR3530.1, 2011.

Arcomano, T., Szunyogh, I., Wikner, A., Hunt, B. R., and Ott, E.: A Hybrid Atmospheric Model Incorporating Machine Learning Can Capture Dynamical Processes Not Captured by Its Physics-Based Component, Journal of the Atmospheric Sciences, 77, https://doi.org/10.1175/JAS-D-20-0082.1, 2020.

Arcomano, T., Szunyogh, I., Wikner, A., Hunt, B. R., and Ott, E.: A Hybrid Atmospheric Model Incorporating Machine Learning Can Capture Dynamical Processes Not Captured by Its Physics-Based Component, Geophysical Research Letters, 50, https://doi.org/10.1029/2022GL102649, 2023.

Blondel, M. and Roulet, V.: The Elements of Differentiable Programming, ArXiv, abs/2403.14606, https://api.semanticscholar.org/CorpusID:268553523, 2024.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q.: JAX: composable transformations of Python+NumPy programs, http://github.com/jax-ml/jax, 2018.

Brenowitz, N. D. and Bretherton, C. S.: Spatially Extended Tests of a Neural Network Parametrization Trained by Coarse-graining, J. Adv. Model. Earth Sy., 11, 2728–2744, https://doi.org/10.1029/2019MS001711, 2019.

Cintra, R. S. and de Campos Velho, H. F.: Data Assimilation by Artificial Neural Networks for an Atmospheric General Circulation Model: Conventional Observation, CoRR, abs/1407.4360, https://arxiv.org/abs/1407.4360, arXiv:1407.4360, 2014.

Dask Development Team: Dask: Library for dynamic task scheduling and parallel computing in Python, https://www.dask.org, parallel computing library for analytics, 2015.

Davenport, E., Madan, J. V., and Watson-Parris, D.: JCM v1.0 Supplementary Data, https://doi.org/10.5281/zenodo.18334906, 2026a.

Davenport, E. H., Madan, J. V., Gjini, R., Brzenski, J., Ho, N., Hsu, T.-Y., Liang, Y., Liu, Z., Manivannan, V., Pham, E., Vutukuru, R., Williams, A. I., Yang, Z., Yu, R., Lutsko, N., Hoyer, S., and Watson-Parris, D.: JAX Circulation Model, https://doi.org/10.5281/zenodo.18189579, 2026b.

Dee, D. P., Uppala, S. M., Simmons, A. J., Berrisford, P., Poli, P., Kobayashi, S., Andrae, U., Balmaseda, M. A., Balsamo, G., Bauer, P., Bechtold, P., Beljaars, A. C. M., van de Berg, L., Bidlot, J., Bormann, N., Delsol, C., Dragani, R., Fuentes, M., Geer, A. J., Haimberger, L., Healy, S. B., Hersbach, H., Hólm, E. V., Isaksen, L., Kållberg, P., Köhler, M., Matricardi, M., McNally, A. P., Monge-Sanz, B. M., Morcrette, J.-J., Park, B.-K., Peubey, C., de Rosnay, P., Tavolato, C., Thépaut, J.-N., and Vitart, F.: The ERA-Interim reanalysis: Configuration and performance of the data assimilation system, Quarterly Journal of the Royal Meteorological Society, 137, 553–597, https://doi.org/10.1002/qj.828, 2011.

Eidhammer, T., Gettelman, A., Thayer-Calder, K., Watson-Parris, D., Elsaesser, G., Morrison, H., van Lier-Walqui, M., Song, C., and McCoy, D.: An extensible perturbed parameter ensemble for the Community Atmosphere Model version 6, Geoscientific Model Development, 17, 7835–7853, https://doi.org/10.5194/gmd-17-7835-2024, 2024.

Fang, J., Bowman, K., Zhao, W., Lian, X., and Gentine, P.: Differentiable Land Model Reveals Global Environmental Controls on Ecological Parameters, https://arxiv.org/abs/2411.09654, 2024.

Gelbrecht, M., White, A., Bathiany, S., and Boers, N.: Differentiable programming for Earth system modeling, Geosci. Model Dev., 16, 3123–3135, https://doi.org/10.5194/gmd-16-3123-2023, 2023.

Giering, R. and Kaminski, T.: Recipes for adjoint code construction, Trans. Math. Software, 24, 437–474, https://doi.org/10.1145/293686.2936, 1998.

Giorgetta, M. A., Brokopf, R., Crueger, T., Esch, M., Fiedler, S., Helmert, J., Hohenegger, C., Kornblueh, L., Köhler, M., Manzini, E., Mauritsen, T., Nam, C., Raddatz, T., Rast, S., Reinert, D., Sakradzija, M., Schmidt, H., Schneck, R., Schnur, R., Silvers, L., Wan, H., Zängl, G., and Stevens, B.: ICON-A, the Atmosphere Component of the ICON Earth System Model: I. Model Description, Journal of Advances in Modeling Earth Systems, 10, 1613–1637, https://doi.org/https://doi.org/10.1029/2017MS001242, 2018.

Häfner, D., Jacobsen, R. L., Eden, C., Kristensen, M. R. B., Jochum, M., Nuterman, R., and Vinter, B.: Veros v0.1 – a fast and versatile ocean simulator in pure Python, Geoscientific Model Development, 11, 3299–3312, https://doi.org/10.5194/gmd-11-3299-2018, 2018.

Hartmann, D. L.: Global Physical Climatology, Elsevier, 2 edn., https://doi.org/10.1016/C2009-0-00030-0, 2016.

Hatfield, S.: speedy.f90 (version 1.0.0), https://doi.org/10.5281/zenodo.5816982, accessed: 2024-08-10, 2022.

Hawkins, E. and Sutton, R.: The Potential to Narrow Uncertainty in Regional Climate Predictions, Bulletin of the American Meteorological Society, 90, 1095–1107, https://doi.org/10.1175/2009BAMS2607.1, 2009.

Hoyer, S., Hamman, J., and xarray Developers: xarray: N-D labeled arrays and datasets in Python, https://xarray.dev, open-source data analysis library for labeled multidimensional arrays, 2014.

Häfner, D., Nuterman, R., and Jochum, M.: Fast, Cheap, and Turbulent—Global Ocean Modeling With GPU Acceleration in Python, Journal of Advances in Modeling Earth Systems, 13, e2021MS002 717, https://doi.org/https://doi.org/10.1029/2021MS002717, e2021MS002717 2021MS002717, 2021.

Kay, J. E., Deser, C., Phillips, A., Mai, A., Hannay, C., Strand, G., Arblaster, J. M., Bates, S. C., Danabasoglu, G., and Edwards, J.: The Community Earth System Model (CESM) Large Ensemble Project: A Community Resource for Studying Climate Change in the Presence of Internal Climate Variability, Bulletin of the American Meteorological Society, 96, 1333–1349, https://doi.org/10.1175/bams-d-13-00255.1, 2015.

Kennedy, M. C. and O'Hagan, A.: Bayesian calibration of computer models, Journal of the Royal Statistical Society: Series B (Statistical Methodology), 63, 425–464, https://doi.org/10.1111/1467-9868.00294, 2001.

Kochkov, D., Yuval, J., Langmore, I., Norgaard, P., Smith, J., Mooers, G., Klöwer, M., Lottes, J., Rasp, S., Düben, P., Hatfield, S., Battaglia, P., Sanchez-Gonzalez, A., Willson, M., Brenner, M. P., and Hoyer, S.: Neural general circulation models for weather and climate, Nature, 632, 1060–1066, https://doi.org/10.1038/s41586-024-07744-y, 2024.

Kucharski, F. and Molteni, F.: Description of the ICTP AGCM (SPEEDY) Version 41, Tech. rep., Abdus Salam International Centre for Theoretical Physics (ICTP), https://users.ictp.it/~kucharsk/speedy_description/km_ver41_appendixA.pdf, 2013.

Kucharski, F., Molteni, F., and Bracco, A.: Decadal interactions between the western tropical Pacific and the North Atlantic Oscillation, Climate Dynamics, 26, 79–91, https://doi.org/10.1007/s00382-005-0085-5, 2006.

Kucharski, F., Molteni, F., King, M. P., Farneti, R., Kang, I.-S., and Feudale, L.: On the Need of Intermediate Complexity General Circulation Models: A "SPEEDY" Example, Bulletin of the American Meteorological Society, 94, 25–30, https://doi.org/10.1175/BAMS-D-11-00238.1, 2013.

Lorenz, E. N.: Seminar on Predictability, 4–8 September 1995, Vol. 1, https://www.ecmwf.int/node/10829, 1995.

Molteni, F.: Atmospheric simulations using a GCM with simplified physical parametrizations. I: model climatology and variability in multi-decadal experiments, Climate Dynamics, 20, 175–191, https://doi.org/10.1007/s00382-002-0268-2, 2003.

NumPy Developers: NumPy: Fundamental package for array computing in Python, https://numpy.org, open-source numerical computing library for Python, 2006.

Paszke, A., Gross, S., Massa, F., Lerer, A., Chanan, G., Bradbury, J., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E. Z., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S.: PyTorch, https://pytorch.org, deep learning framework, 2016.

Rasp, S.: Coupled online learning as a way to tackle instabilities and biases in neural network parameterizations: general algorithms and Lorenz 96 case study (v1.0), Geosci. Model Dev., 13, 2185–2196, https://doi.org/10.5194/gmd-13-2185-2020, 2020.

Rasp, S., Pritchard, M. S., and Gentine, P.: Deep learning to represent subgrid processes in climate models, P. Natl. Acad. Sci., 115, 9684–9689, https://doi.org/10.1073/pnas.1810286115, 2018.

Regayre, L. A., Deaconu, L., Grosvenor, D. P., Sexton, D. M. H., Symonds, C. C., Langton, T., Watson-Parris, D., Mulcahy, J. P., Pringle, K. J., Richardson, M., Johnson, J. S., Rostron, J. W., Gordon, H., Lister, G., Stier, P., and Carslaw, K. S.: Identifying climate model structural inconsistencies allows for tight constraint of aerosol radiative forcing, Atmospheric Chemistry and Physics, 23, 8749–8768, https://doi.org/10.5194/acp-23-8749-2023, 2023.

Rodgers, K. B., Lee, S.-S., Rosenbloom, N., Timmermann, A., Danabasoglu, G., Deser, C., Edwards, J., Kim, J.-E., Simpson, I. R., Stein, K., Stuecker, M. F., Yamaguchi, R., Bódai, T., Chung, E.-S., Huang, L., Kim, W. M., Lamarque, J.-F., Lombardozzi, D. L., Wieder, W. R., and Yeager, S. G.: Ubiquity of human-induced changes in climate variability, Earth System Dynamics, 12, 1393–1411, https://doi.org/10.5194/esd-12-1393-2021, 2021.

Ruiz, J. J. and Pulido, M.: Parameter Estimation Using Ensemble-Based Data Assimilation in the Presence of Model Error, Monthly Weather Review, 143, 1568–1582, https://doi.org/10.1175/MWR-D-14-00017.1, 2015.

Sapienza, F., Bolibar, J., Schäfer, F., Groenke, B., Pal, A., Boussange, V., Heimbach, P., Hooker, G., Pérez, F., Persson, P.-O., and Rackauckas, C.: Differentiable Programming for Differential Equations: A Review, https://arxiv.org/abs/2406.09699, 2025.

Schneider, T., Lan, S., Stuart, A., and Teixeira, J.: Earth System Modeling 2.0: A Blueprint for Models That Learn From Observations and Targeted High-Resolution Simulations, Geophysical Research Letters, 44, 12,396–12,417, https://doi.org/https://doi.org/10.1002/2017GL076101, 2017.

Schneider, T., Leung, L. R., and Wills, R. C. J.: Opinion: Optimizing climate models with process-knowledge, resolution, and AI, EGUsphere, 2024, 1–26, https://doi.org/10.5194/egusphere-2024-20, 2024.

Shaw, T. A. and Stevens, B.: The other climate crisis, Nature, 639, 877–887, https://doi.org/10.1038/s41586-025-08680-1, 2025.

Slater, L. J., Arnal, L., Boucher, M.-A., Chang, A. Y.-Y., Moulds, S., Murphy, C., Nearing, G., Shalev, G., Shen, C., Speight, L., Villarini, G., Wilby, R. L., Wood, A., and Zappa, M.: Hybrid forecasting: blending climate predictions with AI models, Hydrology and Earth System Sciences, 27, 1865–1889, https://doi.org/10.5194/hess-27-1865-2023, 2023.

Sluka, T. C., Penny, S. G., Kalnay, E., and Miyoshi, T.: Assimilating atmospheric observations into the ocean using strongly coupled ensemble data assimilation, Geophysical Research Letters, 43, https://doi.org/10.1002/2015GL067238, 2016.

Tiedtke, M.: A Comprehensive Mass Flux Scheme for Cumulus Parameterization in Large-Scale Models, Monthly Weather Review, 117, 1779–1800, https://doi.org/10.1175/1520-0493(1989)117<1779:ACMFSF>2.0.CO;2, 1989.

Viterbo, P. and Beljaars, A. C. M.: An improved land surface parametrization scheme in the ECMWF model and its validation, Journal of Climate, 8, 2716–2748, https://doi.org/10.1175/1520-0442(1995)008<2716:AILSPS>2.0.CO;2, 1995.

Watson-Parris, D.: Integrating top-down energetic constraints with bottom-up process-based constraints for more accurate projections of future warming, Geophysical Research Letters, 52, e2024GL114 269, https://doi.org/10.1029/2024GL114269, 2025.

Watson-Parris, D., Williams, A., Deaconu, L., and Stier, P.: Model calibration using ESEm v1.1.0 – an open, scalable Earth system emulator, Geoscientific Model Development, 14, 7659–7672, https://doi.org/10.5194/gmd-14-7659-2021, 2021.

Whitaker, J. S. and Kar, S. K.: Implicit–Explicit Runge–Kutta Methods for Fast–Slow Wave Problems, Monthly Weather Review, 141, 3426–3434, http://dx.doi.org/10.1175/mwr-d-13-00132.1, 2013.

Williams, P. D.: The RAW Filter: An Improvement to the Robert–Asselin Filter in Semi-Implicit Integrations, Monthly Weather Review, 139, 1996—2007, https://doi.org/10.1175/2010MWR3601.1, 2011.

Yu, S., Hu, Z., Subramaniam, A., Anandkumar, A., Brenowitz, N., Eyring, V., Geneva, N., Gentine, P., Mandt, S., Pathak, J., et al.: ClimSim: A Large Multi-scale Dataset for Hybrid Physics-ML Climate Emulation, in: Advances in Neural Information Processing Systems 36 (NeurIPS 2023), https://proceedings.neurips.cc/paper/2023/hash/45fbcc01349292f5e059a0b8b02c8c3f-Abstract-Datasets_and_Benchmarks.html, 2023.