We would like to thank the reviewers for careful consideration of our work and for providing thoughtful feedback that has improved the quality of the manuscript in many ways, including the documentation, the detail and value of the examples, and the clarity of the discussion about automatic differentiation. We have numbered the comments from each reviewer below. Responses are in blue with quoted text from the new tracked-changes manuscript in *italics* with associated line numbers.

Reviewer 2

This paper describes the JAX Circulation Model (JCM), a differentiable atmospheric model written in Python using JAX. The paper is well written. I downloaded the code, and it was easy to follow the documentation and run the code on a laptop. I would be happy to accept the paper subject to some minor suggestions.

1. Line 152: Could you explain a bit more what the orographic corrections are?

   Original Text: "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."

   Updated Text (lines 160–164): *"Speedy.f90 includes corrections to the temperature and humidity fields that are not described in the SPEEDY documentation. These corrections appear to adjust near-surface temperature and humidity profiles to account for subgrid-scale topography. Version 1.1 of the JCM does not yet implement these corrections, but their inclusion is a subject of future work. The absence of these corrections is one of the ways in which the JAX implementation deviates from the Fortran 90 version."*

2. Line 186: Is the RMS difference using the monthly data, or the average of the three-year simulation?

   We are using the RMS difference of the 3-year simulations and have clarified with the following statement.

   Updated Text (line 184): *"The root-mean-square (RMS) difference of the 3-year simulations is shown in the bottom row."*

3. Lines 189 - 198 and Figures 2 and 3: The differences in zonal wind between JCM and speedy look pretty large to me. Could you explain more about how the differences in horizontal diffusion and orographic corrections result in these changes?

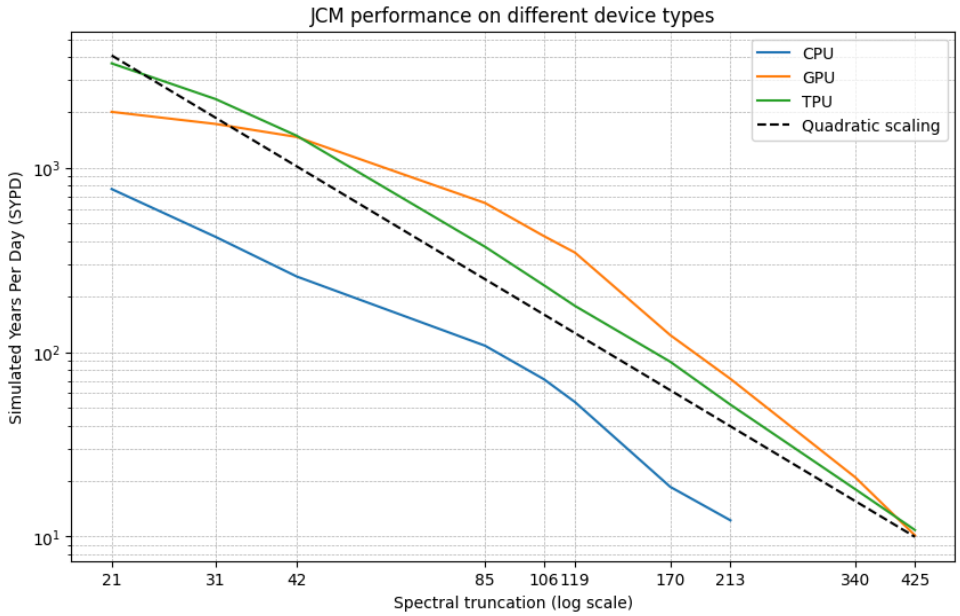
   To the best of our understanding, differences in zonal wind are primarily due to the fact that we have a different dynamical core (Dinosaur), integration method (Runge-Kutta), and horizontal diffusion (explicit instead of implicit and different order). We have done some tuning to attempt to bring the models closer together, but we have stabilized at the current version, which is similar but not exactly the same. While we do want to match the Fortran (and we have verified that the parameterizations give the same output), we are not prioritizing

reproducing SPEEDY exactly. Instead we are using SPEEDY as a starting point for an intermediate-complexity model. We have updated the text to clarify these points.

Updated Text (lines 190–198): *"The discrepancy between the zonal wind in speedy.f90 and JCM is likely due to differences in the implementation of the dynamical cores, time stepping, and horizontal diffusion. The JCM parameterizations have been tested against speedy.f90 to verify that they produce the same output. This does not include horizontal diffusion, which in speedy.f90 is implemented implicitly in coordination with the leap frog time stepping. Although the general formulation of the spectral dynamical cores is the same, they are not identical and the additional choices to use Runge-Kutta and explicit horizontal diffusion in the JCM lead to differing results. Speedy.f90 also implements additional linear drag on the mean vorticity and divergence in the stratosphere (this drag is undocumented and thus currently excluded from the JCM). JCM v1.1 has been tuned to reduce these differences where possible. While we aim to match the qualitative climate of the Fortran, we are not prioritizing reproducing speed.f90 exactly, and instead use it as a starting point for an intermediate-complexity model."*

4. Figures 5 and 6: I think it would be clearer to present performance metrics as SYPD (simulated years per day). It would also make it easier to compare with other models.

Figures 5 and 6 have been replaced with a single figure showing SYPD:



Original Text: "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 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.

[...]

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."

Updated Text (lines 223–240): *"Figure 6 shows the simulation speed, in units of simulated years per day (SYPD), of running JCM at various resolutions and on various device types. The behavior in the figure has been our typical experience: runtime on GPUs and TPUs is significantly faster than on CPUs, with parallelization (sharding the model state arrays so they can be distributed among multiple devices) giving an additional significant speedup to TPU runs on Google Cloud (parallelization was not used for the results in Figure 6). The dashed line in the figure represents the theoretical quadratic scaling of number of operations (and memory requirement) with spectral truncation.*

[...]

It should be noted that in practice, the runtime for the higher resolutions may pick up another factor of 2 or 3 relative to lower resolutions due to reductions to the model timestep that are eventually needed for stability. For the results shown in Figure 6, the timestep was left as the default 30 minutes for all resolutions."

5. Line 256: I don't think this is accurate. For example, ensemble methods that are gradient-free are commonly used in data assimilation in weather forecasting. They can be used for tuning parameters, too.

   Original Text: "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"

   Updated Text (lines 276–279): *"Without gradient information, this process involves ensemble methods and 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. Ensemble methods can be effective, but we often lack comparisons of these approximations against gradient-based approaches. Such comparisons are a focus for future work with the JCM."*

6. Section 6.2: I was hoping the authors could show a calibration of more parameters, especially as it is mentioned that "Calibration with gradient information allows for the assessment of

model behaviour across more of the parameter space". I understand if it is beyond the scope of this paper, though.

Although a truly complex parameter estimation example is outside the scope of this manuscript, we have updated the manuscript with a 2-parameter example. In this example we now optimize over both parameters at once and include a visualization of the 2-dimensional loss landscape.

Updated Figure 7:

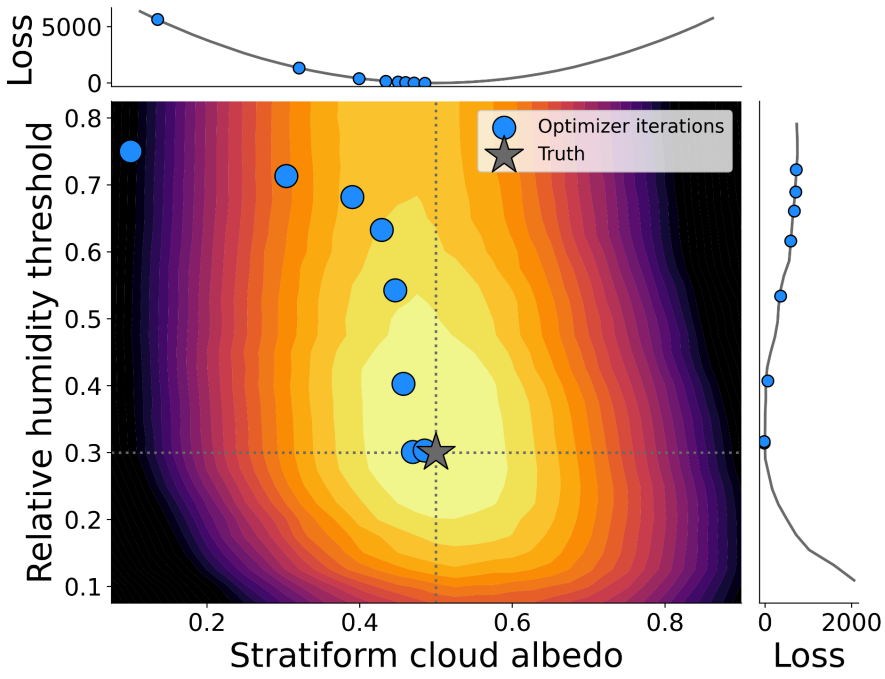


Figure 7 Caption: *Visualisation of the 2-D loss landscape for the described parameter estimation example. The optimization begins with values of 0.1 for stratiform cloud albedo and 0.75 for relative humidity threshold. The optimal value is successfully found near the true SPEEDY values of 0.5 and 0.3, respectively. Blue dots show the calculated loss at each iteration of gradient descent, and the grey star is the ``true'' value. Curves at the top and right sides of the contour plot show slices of the loss function obtained by varying the parameters independently.*

Updated Text (lines 282–291): *"A popular approach for ESM 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 that simultaneously estimates parameter values for stratiform cloud albedo and relative humidity threshold. We generate synthetic observed statistics by running the JCM in the default aquaplanet configuration, which has values of*

stratiform cloud albedo = 0.5 and relative humidity threshold = 0.3. The calculated statistics are the spatial and temporal averages of each model output variable over a 5-day model run. We initialize the starting values of stratiform cloud albedo and relative humidity threshold to be 0.1 and 0.75, respectively, and then optimize over the parameter space to find values that most closely reproduce the synthetic statistics. The observations denoted by y in Equation 2 correspond to the simulated statistics. This example can be adapted or extended to use any climate statistics and any set of parameters."

(cont. lines 300–304): *"…where x_k denotes the vector of parameter values at iteration k, $\boldsymbol{\alpha}$ denotes the step size, and the search direction is the negative gradient of the loss with respect to x_k . Figure 7 shows the calculated loss at each step of the optimization (blue dots) and the true parameter values (grey star). It can be seen that the optimizer succeeds at identifying the values that generated the simulated observations. We calculate the loss across the possible range for both parameters and include a visualization of the 2-dimensional loss landscape, where light colors signify lower cost."*

7. Line 269: Could you describe what is in y (the observation)?

   We have explained the "observations" and statistics in more detail (see response to question 6), and we have added the following statement to clarify further.

   Original Text: "As mentioned above, we are interested in optimizing over the parameter space to find the value that most closely reproduces "observed" statistics."

   Updated Text (line 290): *"The observations denoted by y in Equation (1) correspond to the simulated statistics."*