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.

Maximilian Gelbrecht:

1. However, it is impossible to ignore that the authors of JCM seem to make no mention of other parallel efforts to develop modern, differentiable, and GPU-accelerated variants of SPEEDY, most notably the Julia package, SpeedyWeather.jl (Klöwer et al., 2024) which has recently been made differentiable using the Enzyme.jl automatic differentiation framework (Moses et al., 2025). Other similar efforts that might be mentioned include Oceananigans.jl (Wagner et al., 2025) for ocean modeling and the HOPE shallow water model in PyTorch (Zhou, Xue and Shen, 2025).

   Thank you for your comments. The citation of SpeedyWeather.jl, which is an important precursor to this work, is added to the introduction (and addressed in response to the comment by Reviewer 1 and by Milan Klöwer).

   We have also added a citation for (Wagner et al., 2025), which is relevant for the open-source and GPU-accelerated nature of our work. We have not added a citation for (Moses et al., 2025) as it is a concurrent preprint which was submitted very shortly before this manuscript. In addition, we have included citations for Veros and DifferLand (Python/JAX components) in the introduction (previously only mentioned in the conclusion).

   Updated Text (lines 67–69): *"This work builds upon a growing movement to generate modular, differentiable Earth system components in modern programming languages with GPU acceleration (Klöwer et al., 2024; Häfner et al., 2018; Fang et al., 2024; Wagner et al., 2025)."*

2. Another possible issue is visible in Figure 4, JCM seems to exhibit what could be spectral ringing artefacts in some of its parametrizations presented. Those are not present in the Fortran Speedy figures just below. It is hard to tell just from looking at these coarse figures, but it would be a good idea to investigate this further as they could point to problems with the parametrizations.

   (This is a duplicate of the response to Milan Klöwer, question 2). We agree this warrants investigation and have been looking into it. Based on our testing, we believe that it stems from two things: (1) ringing in low-resolution spectral models is a known phenomenon (there is some ringing seen in the results from speedy.f90 in Fig.5), and (2) additional ringing induced by the dynamical core. We do not believe it is an issue with the parametrizations.

   We will be looking into the role of the dynamical core in this phenomenon more in future

work. As of now we have run several “stress” tests of the current configuration (e.g., running for 100 years) in order to verify that the signal is not growing/leading to instability. In these tests we do not see signs that the ringing grows over time, nor do we see any induced instability. We are able to take the gradients in a variety of applications, including over these long runs. We have added a statement about the expected stability of the model to the text, and we do not believe that these artifacts lead to issues when actually using the model.

Updated Text (lines 215–219): *“Spectral ringing in low-resolution models is a common phenomenon (Durran, 1999). Artifacts of this type of ringing can be seen in Figure 5 in the results from both models. However, the ringing in JCM is stronger than in speedy.f90. We have investigated whether this difference affects model stability, and we have found no indication that it grows over time in these low-resolution configurations (T31-T85). This includes running the model for 100-year runs and taking gradients through those long integrations.”*

3. Furthermore, since JCM and NeuralGCM share an identical dynamical core, it would be really interesting to see a comparison between the learned parametrizations of NeuralGCM and the SPEEDY parametrizations of JCM, either in the present or future work.

   While we think this could be an interesting future comparison, we feel that it is out of the scope of this manuscript, which is intended to document general purpose use of the JCM. The JCM should also be more intentionally calibrated for such a comparison.

4. A third point where we feel the authors could be clearer is in their discussion regarding the benefits of JAX as the underlying numerical framework. The authors note that “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)...”. However, we find this statement to be potentially misleading due to the fact that none of the aforementioned python libraries are directly compatible with the JAX domain-specific language. Due to its reliance on JAX, JCM would not be able to directly integrate existing python code based on Xarray/Dask, PyTorch, or TensorFlow; rather, this code would need to be partially or wholly ported to JAX, which may range of mildly inconvenient to enormously difficult depending on the complexity of the software. These tools could of course still be used for preprocessing and postprocessing of input and output data; however, this is already possible with models written in other programming languages. The authors should therefore clarify these limitations and briefly outline a potential strategy for the integration of JCM into the broader Python ecosystem.

   We respectfully suggest that the concern raised here overstates the degree of incompatibility between JAX and the broader Python scientific ecosystem. JAX is designed around a NumPy-like array API (i.e., it does serve as a drop-in numerical backend in many contexts), and therefore does not preclude the use of higher-level Python libraries such as Xarray or Dask.

The comments here state: “Due to its reliance on JAX, JCM would not be able to directly integrate existing python code based on Xarray/Dask, PyTorch, or TensorFlow;” which we view as inaccurate. In practice, these libraries operate on array abstractions that can be backed by either standard NumPy arrays or JAX arrays, and interoperability can be achieved through lightweight interface layers (e.g., Coordax). Adaptation of Xarray to JAX through Coordax is trivial. In our research, we directly interface JAX-based components with a PyTorch emulator using Coordax, in a nearly “drop-in” workflow. This specific example is not yet published (although it will become publicly available on our GitHub). We agree that, of course, JAX is not universally interchangeable/compatible with all libraries, but we do not claim that it is. Thus we believe these statements, which are intended to highlight the strength of the Python ecosystem as a whole rather than claim strict drop-in compatibility across all applications, are appropriate.

Finally, it is true that models written in non-Python languages do not prevent the use of Python tools for pre- and post-processing. We do not make any statements in our text about the usefulness of other languages (many of which can be differentiated and/or deployed on GPUs). Our point in this section is that JCM is written in a programming language that many scientists, engineers, and developers are already familiar with. Integrating JCM with another Python program or existing analysis pipeline also does not require any additional I/O, which can be necessary to move data between programs from different software ecosystems.