# Response to Review Comment #2

Tian et al. present NoahPy, a differentiable reformulation of the Noah land surface model (LSM) aimed at improving the representation of permafrost thermo-hydrology. The authors rewrite the traditional Fortran-based Noah LSM into a PyTorch-based, partially differentiable framework and demonstrate that it reproduces the original model's numerical behavior while enabling gradient-based parameter optimization through backpropagation.

Strengths

1. The paper provides a clear and rigorous implementation of a differentiable land surface model using PyTorch.
2. The model reproduces the original Fortran Noah LSM with high fidelity (NSE > 0.99), indicating numerical equivalence.
3. The differentiable structure allows efficient gradient-based calibration (using Adam), showing faster and more stable convergence than traditional SCE-UA optimization.
4. The manuscript is well organized, the methodology transparent, and the validation experiments are convincing for the scope of the technical contribution.

Response：We sincerely appreciate the reviewer's positive and encouraging comments.

Major comments

1. Scope of differentiability vs. the claim of "fully differentiable LSM".

Although the abstract and conclusions describe NoahPy as a fully differentiable LSM, the actual implementation appears to make only the heat and moisture transport equations (the PDE solver) differentiable. Other key land-surface processes (e.g., those in Figure 1c) remain treated in their original, non-differentiable, piecewise form. As a result, the framework achieves gradient continuity for a subset of processes, but not necessarily full differentiability of the entire LSM.

The authors should clarify this scope explicitly in both the abstract and methods. Phrasing such as "a partially differentiable framework focusing on

the heat–moisture solver” or “a differentiable core of Noah LSM” would be more accurate and prevent reader misinterpretation.

Response:

After careful consideration on the reviewer’s concern, we respectfully maintain that the term "fully differentiable LSM" is appropriate in the context of modern automatic differentiation frameworks like PyTorch, upon which NoahPy is built. The reviewer correctly notes that many physical parameterizations in LSMs (e.g., for snow, vegetation, or surface roughness) are described by mathematically piecewise functions. In a strict sense, these are not continuously differentiable. However, our contribution is the re-implementation of the entire time-step solution of the modified Noah LSM—including its governing equations (as given in the process shown in Figure 1c) and all of its deterministic physical parameterizations—using native PyTorch operations.

In the context of deep learning frameworks, "differentiable" means that the automatic differentiation engine can compute a gradient for every operation in the computational graph. This is the same principle that allows for the training of deep neural networks using non-smooth activation functions like ReLU (Glorot et al, 2011). In Pytorch, non-differentiable points are handled using subgradient methods (https://docs.pytorch.ac.cn/docs/stable/notes/autograd.html), ensuring stable gradient propagation.

To make this important distinction clearer for all readers, we have added a clarifying statement to the Methods in Section 2.2. The new text reads:

“This is made possible by implementing every step of the numerical solution using the differentiable operations native to the PyTorch deep learning library (Paszke et al., 2019). It is important to note that this includes all physical parameterizations, such as those for vegetation and snow processes shown in Figure 1c. While some of these processes are mathematically not differentiable, re-implementing them within PyTorch ensures that a valid gradient can be computed for every operation via the automatic differentiation engine. This makes the entire model fully differentiable in the context of gradient-based optimization.”

**References:**

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S.: PyTorch: An Imperative Style, High-Performance Deep Learning Library, in: Proceedings of the 33rd International Conference on Neural Information Processing Systems, Curran Associates Inc., Red Hook, NY, USA, 2019.

Xavier, G., Antoine, B., and Yoshua, B.: Deep Sparse Rectifier Neural Networks, In Proceedings of the fourteenth international conference on artificial intelligence and statistics, 315-323, 2011.

2. Gradient continuity within phase-dependent processes

The thermal conductivity λ, volumetric heat capacity $C_s$, and latent heat term Q exhibit abrupt transitions near the freezing point due to phase change. These discontinuities can interrupt or distort the backpropagated gradients, even if the overall framework is formally differentiable. The authors are encouraged to clarify how such non-smooth terms are handled in the current implementation.

Response:

We thank the reviewer for this insightful question. The reviewer is correct that the idealized physics of phase change involves sharp discontinuities. Our implementation handles these non-smooth terms in a way that allows for continuous gradient propagation. To ensure this is clear to all readers, we have added a new paragraph to Section 2.2 (Implementation of NoahPy) that explicitly details how these phase-dependent processes are handled differentiably:

“A specific example of this is the handling of phase-dependent processes. The Noah LSM handles the latent heat of fusion using a source term method, as represented by the Q term in the heat conduction equation (Equation 1). This term explicitly calculates and applies the latent heat required to be released or absorbed to keep the soil temperature at the freezing point during a phase change. While this represents an abrupt physical transition, numerically, this

is not a true discontinuity but is implemented as a conditional logic check. In NoahPy, this entire conditional logic is re-implemented using a chain of native, computationally differentiable PyTorch operations, primarily torch. where, torch.min, and torch.max. PyTorch's automatic differentiation engine is designed to backpropagate through these subgradients, which is the same fundamental principle that enables the training of neural networks with ReLU activations. This numerical implementation avoids a mathematical discontinuity. Therefore, PyTorch's autograd engine can compute a valid gradient through this logic.”