Response to Reviewer #1's Comments

I think the manuscript is concise and well-written. It presents a code that seeks to fill a gap in the current shallow-water modeling landscape and leverages novel numerical implementations to improve accuracy. The code has been tested on commercial hardware, making it a valuable tool for education and research when high-performance computing resources are unavailable. I have some questions and comments regarding the code's efficiency and parallelism, as well as the results presented in the manuscript. Once these are addressed, I think the manuscript will be acceptable for publication.

R:> We thank the reviewer for their positive assessment of our manuscript and their constructive feedback. As requested, we have addressed all questions and comments regarding the code's efficiency, parallelization strategies, and the verification results presented in the study.

To complement these technical clarifications and enhance the solver's utility for research and education, we have developed a comprehensive *User Manual and Technical Reference*, now available at: www.hydrology.cl/swepy

This documentation provides a step-by-step guide for users to reproduce the benchmarks discussed in this work, covering environment setup, input data formats, and execution workflows.

1) Does the code feature any CPU parallelism through either multi-core or multi-thread capabilities? I infer from table 1 that the code is not MPI capable, is it thus safe to say that the code has no CPU parallelism outside the vectorized SIMD instructions? If this is the case it should be clearly stated especially when discussing simulation execution time.

R:> The reviewer raises a very good point. Unfortunately, the code doesn't currently support CPU parallelism. Although the speedups are relative to a serial implementation, we have revised the text to clarify this.
The reviewer raises an important point regarding the interpretation of our performance benchmarks. At present, SWEpy does not implement explicit CPU parallelism (e.g., via MPI or OpenMP). Therefore, the reported CPU timings correspond to single-core, serial execution, subject only to automatic compiler-level optimization and NumPy's internal vectorization.

We agree that this distinction is crucial for a fair comparison. We have revised Section 5.3 and the captions of Table 7 to explicitly state that the reported speedup factors are relative to a serial, single-core CPU implementation. This highlights the acceleration achieved by shifting the workload to the GPU, rather than comparing against a fully optimized multi-core CPU solver, which aligns with our goal of providing high-performance computing (HPC) capabilities on standard workstations.

2) Timing results are presented for the Maule test case. Is there any reason that no timing results are presented for the other benchmarks? If there isn't, it might be valuable to include a table with with timings for CPU and GPU for the different benchmarks and at different resolutions.

R:> We agree that including performance metrics across a broader range of idealized benchmarks provides a more comprehensive view of the solver's scalability. As suggested, we have performed additional timing tests for the Conical Island and Gaussian Bump benchmarks (using the RK4,3+WENO configuration) at three different resolutions.

The results, summarized in the table below, reveal a characteristic behavior of GPU-accelerated solvers:

a) Kernel latency vs. throughput: For smaller meshes, the fixed overhead of launching CuPy kernels and memory orchestration dominates the execution time, leading to speedups below unity. This is expected, as GPUs are optimized for high-throughput tasks rather than low-latency serial operations.

b) Parallel efficiency: As the number of elements increases (moving toward "Fine" resolutions), the solver reaches higher GPU occupancy. This is evidenced in the Gaussian Bump benchmark, where the speedup scales from 0.76X (latency-bound) to 26.20X (throughput-bound) as the element count reaches approx 1.2 million.

Test	Resolution	#Elements	GPU Time/step (s)	CPU Time/step (s)	Speedup
C. Island	Coarse	2,958	1.03	0.076	0.07x
C. Island	Medium	11,682	1.03	0.411	0.40x
C. Island	Fine	45,924	1.04	2.579	2.47x
G. Bump	Coarse	18,992	0.96	0.73	0.76x
G. Bump	Medium	74,848	1.22	8.39	6.90x
G. Bump	Fine	1,184,128	8.37	219.29	26.20x

Table 1: Average execution time per time-step for idealized benchmarks (NVIDIA GTX 1650 vs. Intel Core i5-10300H Serial)

These findings have been incorporated into Section 5, clarifying that the advantages of SWEpy are most pronounced in high-resolution research scenarios where traditional serial solvers become computationally prohibitive.

3) It is mentioned in the manuscript that the code has high throughput on modest hardware. I find this to be very valuable, but there doesn't seem to be any quantification of this throughput. Would be it possible to include a plot or a table describing the code's throughput compared to the maximum throughput of the hardware it was tested on? I think this would make a compelling case for the code's efficiency.

R:> We thank the reviewer for this valuable suggestion. Quantifying throughput on a high-level, array-based GPU framework like CuPy presents unique challenges compared to low-level CUDA C++ implementations. Because SWEpy uses a "layered" kernel approach, where each array operation triggers a specialized kernel launch, the primary bottleneck at small scales is the API dispatch overhead rather than the hardware's theoretical FLOP limit. In addition, profiling hundreds of kernel launches per cycle is highly demanding and becomes realistically infeasible on consumer hardware.

However, to address the reviewer's request, we have performed a detailed system trace using NVIDIA profiling tools to quantify the solver's effective utilization of available hardware resources. We report these metrics in Table 2 (see below), focusing on GPU Utilization (Throughput) and VRAM Occupancy:

Test	Resolution	#Elements	GPU Time (s)	GPU Ops	GPU Throughput	VRAM Throughput
C. Island	Coarse	2,958	155.76	3,103,522	4.0%	2.51%
C. Island	Medium	11,682	337.41	6,477,322	7.2%	3.76%
C. Island	Fine	45,924	549.27	12,982,607	25.0%	7.05%
G. Bump	Coarse	74,848	36.38	524,831	22.9%	4.8%
G. Bump	Medium	297,152	50.11	564,031	65.5%	11.8%
G. Bump	Fine	1,184,128	148.30	564,031	91.1%	34.3%

Table 2: GPU performance metrics derived from system traces for the Conical Island and Gaussian Bump benchmarks.

As shown in the table, SWEpy demonstrates excellent strong scaling for the "Gaussian Bump" case. As the number of elements increases, the GPU utilization rises from 22.9% to 91.1%, indicating that the hardware becomes fully saturated with arithmetic operations once the problem size is large enough to mask the Python-to-CUDA dispatch latency. This confirms that, for real-world scenarios (approx. 1.2M elements), the code effectively approaches the hardware's maximum throughput. The Conical Island case, in contrast,

shows that the wetting-drying algorithm adds significant overhead not because of GPU or memory usage, but because its arithmetic complexity results in higher kernel launch density. This suggests clear paths for CuPy-based optimization via custom kernel writing (cupy.ElementWiseKernel) or kernel fusing (cupy.fuse).

We have added this table and a new subsection 5.1.4 Computational Performance and Scalability to provide a transparent quantification of the solver's efficiency and a discussion of the results.

4) Following my first question, if there isn't any CPU parallelism outside SIMD instructions, is the code eventually intended to feature MPI or multi-threading capabilities? Most consumer hardware has CPUs with multiple cores that are multi-threading capable and I think many users who might not have access to a GPU could benefit from this (even on consumer hardware, GPUs remain a quite expensive component).

R:> We appreciate the reviewer's perspective on accessibility for non-GPU users. Our current design philosophy prioritizes a "Python-native" approach to GPU acceleration to minimize the barrier to entry for researchers familiar with the NumPy ecosystem.

While we have explored JIT-compilation for CPUs using Numba and process-based parallelism via Python's native multiprocessing, implementing a robust MPI layer for unstructured grids in pure Python introduces significant communication overhead that often negates the benefits of parallelization without resorting to hybrid C++/Python architectures (e.g., mpi4py wrapping C routines). Therefore, our roadmap focuses on optimizing GPU efficiency. However, we have clarified in the Conclusions that the modular architecture of SWEpy allows for community contributions, and we welcome future developments that might bridge the gap for multi-core CPU execution.

5) Swepy uses the CuPy library for it's GPU implementation and CuPy utilizes CUDA. Does this make the code only operational on CUDA compatible GPUs? If so, it should be clearly stated in the text.

R:> We thank the reviewer for pointing this out. The current release of SWEpy relies on the CuPy library, which is primarily designed for the NVIDIA CUDA ecosystem. We have updated the Introduction to explicitly state that an NVIDIA GPU is currently required for accelerated execution mode. While CuPy offers experimental support for AMD GPUs via the HIP/ROCm interface, we have focused our validation efforts strictly on the stable CUDA backend to ensure reproducibility and stability for this initial release.

6) Following my previous question, if the code only runs on CUDA compatible GPUs, do you plan to make the code functional on GPUs that aren't CUDA compatible?

R:> Regarding cross-platform compatibility, our development roadmap is strictly tied to the evolution of the CuPy ecosystem. We deliberately chose CuPy to prioritize code readability and maintainability. While CuPy is actively expanding support for AMD GPUs via the ROCm (Radeon Open Compute) platform, this feature is currently designated as experimental by the CuPy development team (https://docs.cupy.dev/en/stable/install.html#installing-cupy-on-amd-rocm).

Consequently, we do not plan to develop a separate, manual backend for non-CUDA devices, as this would bifurcate the codebase. Instead, we have adopted a strategy of "inherited compatibility": as CuPy's support for the HIP/ROCm stack stabilizes and reaches feature parity with CUDA, SWEpy will become AMD-compatible with minimal changes to the source code. We are closely monitoring the active development and intend to officially validate and support AMD hardware in future versions once the underlying library guarantees the numerical stability required for scientific modeling. For completeness, we have included a comment on this point in the conclusion.

7) For the Maule test case, the manuscript says that the discrepancies between the HySea simulation and the Swepy simulation are due to interpolation errors, projection errors, and resolution differences. The manuscript then states that studying these factors is left to future work. However, given the timings mentioned in the manuscript, it does not seem infeasible to compare the Swepy and HySea simulations at the same resolution. Would this be possible? If it is I think it would be valuable to see the impact of the additional resolution on the errors observed.

R:> The reviewer raises a good point. Exactly matching resolutions are difficult to achieve because HySea uses structured rectangular meshes on spherical latitude and longitude, whereas SWEpy uses triangles on flat coordinates. A right-triangle structured mesh that matches the geometry of HySea's grids is possible, but it would have twice as many elements. Considering these limitations, meshes with resolutions closer to those of the HySea simulations have been run, and their results have been incorporated into the manuscript.

8) The use of the WENO reconstruction clearly shows better preservation of gradients and flow features, from the manuscript it seems that this can be done using polynomials of any order. Is there a reason outside of keeping the stencils simpler that only second order polynomials were used? Have you made any attempts using higher order polynomial reconstruction (for example with order 3 or 4) and if so did you see any additional improvements?

R:> We thank the reviewer for this important clarification. The decision to employ quadratic reconstruction (leading to 2nd/3rd-order spatial accuracy) was driven by a trade-off among computational cost, numerical stability, control of numerical diffusion, and GPU memory access patterns.

Implementing reconstruction polynomials of higher order (e.g., 3rd or 4th order) on unstructured grids requires significantly larger stencils (accessing neighbors of neighbors). This drastically increases memory latency due to non-contiguous indirect addressing and elevates register pressure per thread. Furthermore, ensuring the positivity of water depth and handling wet/dry transitions becomes increasingly stiff with higher-order polynomials without strict slope limiting, which can degrade the effective order near shocks. We are currently conducting a theoretical study of high-order stability for future work, but for the scope of this paper and the first version of \texttt{SWEpy}, the quadratic WENO scheme offers the optimal balance between accuracy and GPU throughput.

9) Technical comments/corrections:

- **Figure 1: The figure implies that data is saved for output every time the RK iterations are completed. The text itself says that the output time is user determined. I think the figure could be improved by including this information.**

  R:> We thank the reviewer for these detailed observations. Assuming the reviewer is referring to Figure 5 (the flowchart describing the model's algorithmic logic) rather than Figure 1, we have updated the diagram to explicitly show that the output frequency is a user-defined parameter, decoupled from the internal Runge-Kutta time-steps.

- **Figure 1: I think the figure could benefit from also highlighting that the code can perform divergence and stagnation detection.**

  R:> We have also added this to the flowchart, highlighting the runtime stability checks (divergence and stagnation detection) as suggested.

Response to Reviewer #2's Comments

The article introduces SWEpy, open-source, Python-based GPU-accelerated finite-volume solver for the shallow-water equations on unstructured triangular grids, targeting tsunami propagation, flooding, and dam-break scenarios. The code uses a central-upwind flux formulation with WENO spatial reconstruction and SSP Runge–Kutta time integration to reduce numerical diffusion while attempting to ensure well-balancing and positivity preservation. The paper details the numerics and some algorithmic designs of the CuPy based code, providing additional validation and verification examples.

Considering the article and the code as available today, there are several shortcomings that do not yet support the conclusions and ambition set out in the article. The largest issues include (details are in the comments):

R:> We appreciate the reviewer's careful review and constructive feedback on the manuscript. Their comments on the state of the art in GPU-accelerated solvers, the clarity of our modular architecture, and the rigor of our numerical verification have been key to improving the quality and focus of this work. Furthermore, to address the reviewer's concerns about software accessibility and usability, we have developed a comprehensive User Manual and Technical Reference (now available at: www.hydrology.cl/swepy) that guides users through mesh preprocessing, model configuration, and results visualization. We have addressed the major concerns by revising the introductory context, providing deeper technical details on the GPU implementation, and refining the validation section.

1) The background provided lacks many references regarding the state of shallow water solvers, especially the HPC field. The packages that are mentioned are not entirely representative and also have some errors in the stated properties. For instance, a major code that is GPU enabled and has many of the advantages listed for SWEpy has been authored by a group including Manuel Castro-Diaz.

R:> We appreciate the reviewer's guidance in expanding the background of this manuscript. These suggestions have been key in refining our introductory framework to better reflect the diversity of numerical schemes and hardware acceleration strategies currently available.

In response, we have substantially revised the Introduction and Table 1 to (i) expand the coverage of state-of-the-art SWE solvers with an explicit emphasis on diverse parallelization strategies, and (ii) place SWEpy more carefully within the existing HPC ecosystem. These solvers are now contextualized within the broader landscape of MPI, CUDA, OpenMP, and Kokkos-based implementations. To improve accuracy, Table 1 has been revised to harmonize and correct reported properties (parallelization model, GPU support, numerical formulation, grid type, and scope), and is now presented as a representative overview of the trade-offs among performance, flexibility, numerical formulation, and accessibility.

Furthermore, we would like to clarify that the foundational work of the EDANYA group (University of Málaga) and their Tsunami-HySEA solver was indeed considered in our

original manuscript. Specifically, the model was featured in Table 1 and served as the primary benchmark for the Maule Tsunami validation (Section 5.3). We recognize that Tsunami-HySEA, developed by a team including Manuel J. Castro-Díaz and often published with Jorge Macías as lead author, has become a cornerstone of the field due to its exceptional fast-computing capabilities. It has been the engine for major international efforts such as the NEAMTHM18 project (Basili et al., 2021) and has been at the forefront of real-time probabilistic tsunami forecasting (e.g., Selva et al., 2021).

Importantly, the revised manuscript now emphasizes that SWEpy's contribution is not the introduction of GPU acceleration per se, but rather the combination of: unstructured triangular grids, higher-order central-upwind finite-volume schemes with WENO reconstruction, and a high-level Python/CuPy implementation that lowers the barrier to GPU-accelerated SWE modeling while retaining competitive performance. These changes are primarily in the Introduction and Table 1, where the discussion of HPC solvers and GPU-enabled frameworks has been significantly expanded and refined. We believe these revisions address the reviewer's concerns and provide a more balanced, accurate, and up-to-date overview of the current state of shallow water solvers in the HPC field.

2) The modularity of the code and the algorithm design is mentioned various times as an advantage, but this is never fully explained or demonstrated. Reading through the code itself, it is not obvious how this modularity it designed or how one would go about implementing additional modules beyond the idea that Python allows this. Furthermore, the design of the algorithms on the GPU is not discussed, especially how this would enable easy extensions on a GPU. The design of the GPU components need to be more fully discussed, including how CuPy works and communicates between the CPU and GPU seamlessly. For example, what were the design decisions that needed to be made for the GPU?

R:> We thank the reviewer for highlighting our original manuscript's shortcomings. Guided by their remarks, we have revised the document to better showcase our code's modularity and the design decisions that came with the selection of the GPU backend. In addition, to offload the presentation of the deeper technical aspects, we have developed a separate user manual with more detailed descriptions of the software's inner workings, use, and extensibility. Our work related to the reviewer's specific topics is outlined below:

a) Regarding the code's modularity, to better showcase this feature, we have revised section 4 of the manuscript to better link the explanation of the software to the code itself, illustrating the connections between the various modules and functions thereof in the library. In addition, the user manual includes sections on the software architecture and execution workflow, offering a more hands-on approach to understanding the library.

b) Regarding the GPU design decisions, we have included in the manual (sect. 4) thorough descriptions of the most important functions that SWEpy uses, the algorithms they represent, and how they differ from what would be traditional serial implementations. This illustrates to users how they can, if needed, build their own modules to suit their needs.

c) With respect to the inner workings of CuPy itself, we believe technical aspects regarding the CuPy library are far better explained in their documentation. However, some comments on explicit CPU-GPU synchronization for performance are present in the user manual section 4.

3) The verification and validation results are incomplete in several critical ways. There are tests showing conservation and well-balancing missing. The results also show a degradation in order, but this issue is not addressed satisfactorily, especially for a smooth problem. There are other issues that are mentioned, but not satisfactorily explained, such as the loss of stability in the Forward Euler and WENO runs in the Maule Tsunami.

R:> We appreciate the reviewer's comments regarding a more complete verification. We have substantially expanded Section 5 to provide the quantitative evidence required to support the solver's reliability:

- Well-balancing and conservation: We have added Section 5.1.2, featuring a "Lake at Rest" test over a white-noise bathymetry. This is a severe test for hydrostatic reconstruction. \texttt{SWEpy} successfully maintains the steady state, with residuals in water surface elevation and discharge reaching machine precision (approx. 10^{-15}). Furthermore, we have included a mass balance analysis in the Technical Reference using the Synolakis Soliton Runup benchmark. Our results indicate that even during the complex wetting and drying phases of the wave run-up and backwash, the relative global mass error, defined as (|M(t) - M_0|/M_0), remains below 10^{-5} per time step.

- Order of Convergence on unstructured grids: We acknowledge the observed reduction from the theoretical 3rd-order accuracy to approximately 2nd-order in the presented benchmarks. This is not an implementation error but a documented characteristic of central-upwind schemes on a non-orthogonal unstructured grid. As noted by *Bryson et al. (2011)* and *Nguyen (2023)*, the loss of stencil symmetry and the geometric distortion of triangular elements inherently degrade the effective reconstruction order. However, it is important to distinguish our results from linear reconstructions. Even though it shares a similar 2nd-order asymptotic convergence rate, our quadratic WENO implementation exhibits a significantly lower error constant and reduced numerical dissipation compared to linear alternatives. This allows \texttt{SWEpy} to preserve steep gradients and wave amplitudes more effectively than traditional second-order solvers, especially in far-field propagation scenarios.

- Stability of forward Euler vs. SSP-RK: Instabilities may arise from a combination of different factors and their interactions. For instance, the absence of explicit dissipation and dispersion mechanisms, together with upstream wave superposition effects, may combine to generate spurious waves whose amplitudes increase over time. In the SWEpy User Manual, soliton experiments are presented alongside a discussion of the possible mechanisms underlying these instabilities. From these results, it is observed that WENO reconstruction reduces numerical diffusion. Using

this reconstructor, the effect of mesh resolution was studied together with the influence of the temporal integrator. In this context, the FE+WENO configuration with the most refined mesh was found to produce the largest amplitudes of spurious waves among the cases considered. Such oscillations do not develop in the Runge–Kutta (RK) configurations, as expected given the strong stability preserving (SSP) property of the temporal scheme employed.

These additions, supported by new figures in the manuscript and the Technical Reference, provide a transparent and physically grounded explanation of the solver's numerical behavior.

While these are significant, it is clear that this is a significant step towards a working code that demonstrates the goals as laid out in the article. The article could address these concerns directly, or change the focus of the paper to be more about the implementation of the GPU code and the performance characteristics, minimizing the discussion of numerical methods that have been well established. For that reason, I would recommend at least major revisions that address the over-arching issues with the article as outline above (and the code) and a reconsideration or a resubmission with a change in focus to be more focused on the implementation details.

Comments and Suggestions:

[a.] The introduction is missing many details. A lot of recent work in the CPU/GPU, especially in Python, is out of date. For example, CuPy has undergone a major effort from NVIDIA to bring in direct support.

R:> We thank the reviewer for the comment. The Introduction intentionally focuses on positioning SWEpy within the context of accessible GPU acceleration for scientific computing, rather than providing a comprehensive survey of recent Python CPU/GPU developments. As discussed in Section 1, the manuscript explicitly highlights CuPy as a NumPy-compatible, CUDA-based framework that leverages the mature NVIDIA ecosystem to provide stable, reproducible GPU acceleration, support for custom kernels, and scalable execution without exposing users to low-level GPU programming. The goal of this discussion is not to catalog all recent developments, but to motivate the design choice of a CuPy-based implementation of finite-volume shallow-water solvers on unstructured grids. In this sense, the Introduction now reflects the technical scope and objectives of the work rather than an exhaustive review of the rapidly evolving Python GPU landscape.

[b.] (line 54) The reasons for unstructured vs. structured grids is well known, but this characterization omits adaptive grids, something that both structured and unstructured grids can take advantage of.

R:> We appreciate the reviewer's observation regarding the omission of adaptative grids. We agree that grid adaptivity is not exclusive to unstructured meshes. In fact, both structured and unstructured discretizations can employ adaptive refinement strategies. Our intention was not to suggest otherwise, but to emphasize that unstructured triangular grids provide flexibility, enabling localized refinement in complex domains (e.g., irregular coastlines, urban features, and rapidly varying bathymetry) since triangulation via Delaunay is extremely robust, and it can be applied without the additional grid-management machinery typically required by adaptive structured approaches.

To avoid ambiguity, we have revised the text around line 54 to explicitly acknowledge adaptive structured grids and to clearly distinguish between adaptive refinement techniques and native geometric flexibility. The revised wording clarifies that SWEpy's unstructured-grid formulation complements rather than replaces adaptive strategies, which remain an important direction for future development.

[c.] (table 1) There are a few notable packages missing from this list, such as MOST, ADCIRC, and exaHYPe, and some inaccuracies in the table, such as GeoClaw's support for GPUs (GeoClaw also does not use a HLLE solver, but that's more complex). I might suggest that you have a column that distinguishes parallelism support and what type of support is provided (e.g., OpenMP, MPI, OpenACC, CUDA, CuPy, kokkos). You could still maintain the GPU column to distinguish this, as that is the focus of \texttt{SWEpy}.

R:> We thank the reviewer for this feedback, which has provided an opportunity to strengthen the technical context of our study. We agree that a comprehensive and accurate overview of the existing modeling landscape is essential. Consequently, we have refined our comparative framework by focusing on two primary areas:

- Inclusion of representative models and technical refinement: We have expanded the content of Table 1 to include MOST, ADCIRC, and exaHYPe. Simultaneously, we have performed technical refinements to existing entries, such as GeoClaw, to better reflect their numerical formulations and current hardware compatibility. This ensures the table serves as a high-fidelity reference for the state-of-the-art.
- Categorization of parallelism paradigms: Following the reviewer's suggestion, we have revised Table 1 to explicitly distinguish parallelization frameworks from GPU acceleration, identifying the primary parallel strategy used (e.g., MPI, OpenMP, CUDA, Kokkos, CuPy) while retaining a dedicated GPU column that reflects SWEpy's central focus. These changes enhance the accuracy and transparency of the comparison without expanding the table beyond its intended scope.

To complement these changes, we have also added a paragraph in the Introduction discussing the role of these established HPC solvers in the current landscape of geophysical

fluid dynamics. We believe these revisions significantly strengthen SWEpy's contextual positioning.

[d.] (line 61) This is a bit of an over-generalization and overstates the cost of the common types of Riemann solvers that are used in the packages in table 1. The point about central schemes being simpler are generally true, however.

R:> We thank the reviewer for this important clarification. We agree that our original wording may have overstated the computational burden of commonly used approximate Riemann solvers. Our intention was not to suggest that Riemann-solver-based methods are inefficient or inappropriate, but rather to highlight an alternative formulation that avoids characteristic decompositions and eigenstructure evaluations. In the revised manuscript, we have softened this statement and clarified that central-upwind schemes offer algorithmic simplicity and implementation advantages, while acknowledging that modern approximate Riemann solvers (e.g., Roe, HLL, HLLC) are well-optimized and widely used in high-performance SWE solvers. The revised Introduction now presents the central-upwind approach as a complementary alternative rather than a computationally superior replacement.

[e.] (line 86) Here, high-order is mentioned in the context of near-field wet/dry fronts. This ends up being limited to first order, and for good reason. It is also not clear that a high-order reconstruction in the near-shore is helpful in most cases anyway when using the shallow water equations without non-hydrostatic effects included. This high order reconstruction is a good argument for using GPUs, but this point is not directly made.

R:> We thank the reviewer for this insightful observation. We have addressed this by explicitly linking the numerical complexity of the high-order scheme with the necessity of GPU acceleration in the revised manuscript:

- Numerical stability at the front: We agree that in the immediate vicinity of wet/dry fronts, the reconstruction effectively reverts to first-order (via the correction procedure) to ensure positivity and prevent spurious oscillations. This "fallback" is a standard and necessary practice to ensure the stability of shallow-water solvers in moving-shoreline problems.
- High-order methods as a driver of GPU adoption: As the reviewer noted, the high arithmetic intensity of WENO reconstruction is one of the strongest justifications for using GPU hardware. While the computational overhead of these schemes can be prohibitive for serial CPU execution, the massively parallel nature of GPUs enables high-efficiency stencil-based calculations. We have updated the Introduction to explicitly state that the adoption of high-order spatial reconstruction is precisely what necessitates the high-performance throughput provided by SWEpy.
- Utility in hydrostatic models: We emphasize that high-order reconstruction remains crucial for tsunami modeling even without non-hydrostatic effects. Its main role is to preserve wave phase and amplitude in the far field during transoceanic propagation.

Reducing numerical dissipation in the open ocean ensures that the energy reaching the near-shore environment is accurate, regardless of the order of reduction at the final inundation stage.

We believe these additions clarify the "why" behind our architectural decisions and align with the reviewer's suggestion to link numerical complexity with hardware choice.

[f.] While mentioned that it would be "easy", it is not clear how additional source terms would be added, either from the article or the code, especially with higher order reconstructions that may be important to these additional source terms.

R:> We thank the reviewer for pointing this out. We agree that the term "easy" was imprecise and potentially misleading. Our intention was simply to indicate that the modular structure of the solver allows extensions, not to imply that incorporating additional source terms requires minimal effort or no methodological care. In the revised manuscript, we have removed this wording and rephrased the statement to more neutrally indicate that the framework is designed to allow the inclusion of additional source terms within the existing modular structure, without implying simplicity of implementation. The changes are presented in Section 4. "Architecture & parallel structure."

[g.] (line 310) The near-shore well-balancing is often where the largest errors can occur. Quantifying how much a "slight imbalance" is seemingly extremely relevant.

R:> We thank the reviewer for this pertinent observation, and we have therefore added the following clarification: "However, it does not ensure well-balancing near fronts, where slight imbalances may occur (Liu et al., 2018). This aspect was investigated using Synolakis's analytical solution for wave run-up, as described in the Analytical Benchmarks section of the User Manual & Technical Reference. In these numerical experiments, the proposed approximations achieve good accuracy in the wet–dry region. Within this framework, a clear dependence of the results on the temporal discretization and on the Courant number is also observed. These results are currently being prepared for presentation in a dedicated study, primarily focused on the numerical and theoretical properties of wet–dry reconstruction techniques, with particular attention to the suitability of the underlying model for the phenomenon under consideration". Finally, within the initial scope of this first version of SWEpy, we show that the imbalances introduced by the numerical implementation do not significantly affect the results presented on wave generation and propagation, as the conical island experiment demonstrates.

[h.] It appears that the time steps are adaptive? If so, what's the penalty for having to take a time step. This is especially relevant for GPU computing.

R:> The reviewer raises an excellent point. Since the timestep calculation is vectorized, its execution time should be low. Indeed, via system trace profiling of a simulation over the finest mesh (1.2M elements) of the Gaussian bump test, it can be seen that (on our available hardware, NVIDIA GeForce GTX 1650) this time amounts (on average, n=5) to a sum of 3.85 s, of a total average execution time of 142.12 s, i.e. 3% of the time was spent calculating the adaptive timestep. In contrast, by forcing a constant timestep equal to the minimum timestep (0.000175 s) calculated adaptively to ensure stability, the average total execution time jumps to 148.299 s. So, while some overhead is introduced by adaptively calculating the timestep, the penalty is effectively bypassed by a more efficient timestepping. We've revised the manuscript to include a note about this performance point in the new section (5.1.4) regarding performance findings.

[i.] The CFL restriction is significant for higher-order WENO schemes. Can you say what the empirical stability is in practice for these simulations?

R:> The reviewer highlights a critical operational aspect of high-order schemes on unstructured grids. From a theoretical standpoint, ensuring positivity-preserving properties for Central-Upwind schemes often requires a conservative CFL limit (e.g., CFL < 1/6 for certain 2D configurations). However, in the development of SWEpy, we have optimized this through the following empirical and numerical strategies:

- SSP Runge-Kutta integration: We utilize the Strong Stability Preserving (SSP) Runge-Kutta scheme. Theoretically, these multi-stage methods expand the stability region compared to the Forward Euler. In our simulations, we have found that using SSP-RK3 allows us to safely double the effective CFL number while maintaining the TVD (Total Variation Diminishing) property, consistently achieving stable runs with CFL approx 0.25 to 0.6 for WENO-3 configurations.
- Mesh quality and geometry: On unstructured triangular grids, the stability is intrinsically linked to the ratio between the inscribed circle radius of the smallest element and the characteristic wave speed. We have observed that for "smooth" benchmarks (e.g., Conical Island), a higher CFL of up to 0.45 is attainable. Conversely, for highly dynamic scenarios with complex bathymetry (e.g., Malpasset or Maule), a more conservative CFL between 0.15 and 0.25 is preferred to minimize spurious oscillations at the wet/dry interface or due to wave steepening.

While a formal von Neumann stability analysis for WENO on arbitrary unstructured grids is beyond the scope of this implementation paper, our empirical results confirm that SWEpy remains robust under standard tsunami modeling constraints.

[j.] (fig 5) Not sure that this is necessary, or maybe can be simplified for the article? In its stead, it may be more helpful to illustrate the modularization by showing how the code itself is split up into functions in the code.

R:> We thank the reviewer for this suggestion. The purpose of the diagram is to provide readers with a high-level overview of the software architecture, illustrating how the main components of the numerical method, data flow, and GPU/CPU interactions are organized. Our intention was to help readers quickly understand the code's structure and design philosophy rather than document individual functions. We have followed the common practice in GMD publications, in which overview architecture diagrams are typically included to summarize the model structure at the system level. To address the reviewer's concern, we have clarified in the caption and text that the figure represents a conceptual overview rather than a function-level code map. More detailed descriptions of module organization and internal function structure are provided in the SWEpy user manual and technical documentation.

[k.] (fig 5) Imposition of the boundary conditions is essential, and with a wider stencil, can be tricky to implement. Later, you mention that there are issues with the BCs. Can you better quantify what is being done and how the "leaky" boundary effect could be better addressed rather than expanding the boundaries away from the area of interest? This is often not practical in many cases of interest. A common example that can be difficult is imposition of an incoming wave while allowing a wave to exit without reflection.

R:> We find the reviewer's comment pertinent, and we thank them for their insight. Regarding the quantification of what is being done, we have included in the User Manual a detailed explanation of the boundary conditions that can be simulated with SWEpy, along with considerations for their use, and an explanation of the `leaky' problem.

In this first version, the way boundary conditions are imposed does not account for the reconstruction scheme's treatment. This limitation, without modifying the boundary conditions routine, imposes an additional consideration for the user, namely either flattening the bathymetry near the boundary or extending the numerical domain.

Regarding the effect of a permeable boundary, the construction of an appropriate boundary condition can be understood as a synthesis of how the flow across the boundary of a general control volume (i.e., the computational domain) is represented. Consequently, addressing this aspect would require a dedicated analysis, potentially involving low-complexity models, in order to derive a suitable formulation. We consider that such an analysis lies beyond the initial scope of this first version of SWEpy.

[l.] (line 423) A stiff (implicit) solver would be very non-trivial to implement on a GPU.

R:> We fully agree with the reviewer's assessment regarding the complexity of implementing implicit solvers in a massively parallel environment. The challenges associated with global matrix assembly, the iterative nature of stiff solvers, and the communication overhead often present significant bottlenecks for GPU architectures.

However, we believe the modular design of SWEpy provides a suitable foundation for addressing these challenges in future development. As suggested, we have updated Section 4.2.3 to clarify how the solver's architecture facilitates such extensions:

"The modularity of the timestep function enables user-defined integration schemes to be incorporated into the GPU workflow for future extensions, such as predictor–corrector methods or the implementation of a modified Newton–Raphson subroutine for implicit formulations, although implicit formulations are currently less well suited for efficient GPU execution."

By utilizing a high-level Python/CuPy interface, developers can leverage existing GPU-accelerated linear algebra libraries to experiment with implicit kernels without re-engineering the entire data-loading and preprocessing pipeline of SWEpy. While we acknowledge that achieving high efficiency for stiff problems on GPUs remains a non-trivial task, SWEpy's "separation of concerns" between the numerical operator and the time-integration logic lowers the technical barrier for such research.

[m.] Modularity does enable additional capabilities, but saying that this would be easy while maintaining efficiency is rarely true. Some trade-off needs to be made, so a discussion of this would be warranted if this is a major feature of the code as proposed. This design question is very difficult to evaluate given the current state of the code. Diagrams illustrating how the code is modularized and composed would make this a salient point, but at present this is very difficult to evaluate.

R:> We thank the reviewer for this thoughtful comment. We agree that our previous wording may have overstated the ease of extending the code while maintaining computational efficiency. Our intention was to emphasize that the software architecture was designed to support extensibility, not to suggest that incorporating new physics or capabilities is straightforward or without trade-offs. In the revised manuscript, we have removed language implying simplicity and clarified that extensions must be implemented consistently with the existing numerical discretization and performance-oriented design.

To make the modular structure more evident, we have also clarified the description and caption of the architecture diagram (Figure 5), emphasizing that it illustrates the main software components and their interactions. This figure provides a system-level view of the code organization and data flow, highlighting the structure of numerical methods, GPU execution, and module boundaries. Moreover, additional detailed documentation of internal module organization is provided in the SWEpy user manual.

[n.] On an old GPU or even high-performance GPUs the data related to the grid can get large and lead to significant slow-downs. Was this addressed, or were the problems run not large enough for this to occur? Furthermore, memory locality is not assured for even moderately complex unstructured grids and can be particularly challenging for a GPU. Did this have an impact on the results presented?

R:> We thank the reviewer for this critical observation on GPU hardware constraints. We have addressed the concerns regarding memory capacity and locality by performing scalability stress tests and analyzing the kernel performance patterns:

- Memory capacity and VRAM limits: For the validation cases presented (e.g., Malpasset with ~26k elements and Maule with ~1.1M elements), the memory footprint remained well within the 4GB VRAM limit of our consumer-grade NVIDIA GTX 1650 hardware, leaving results unharmed. To probe the solver's limits however, we conducted "toy problem" simulations with up to 6.5 million elements. These tests confirmed that the current memory bottleneck is not the runtime flux calculation, but the precomputation of WENO least-squares matrices. This step requires handling large stacked matrix inversions, which eventually reach the VRAM capacity for very large grids.
- Memory locality and coalescence: Unstructured grids fundamentally challenge memory locality due to indirect addressing (scatter/gather patterns), which can disrupt coalesced access. However, SWEpy mitigates this through massive parallelism. By mapping the element-wise operations to vectorized CuPy kernels, the solver launches a sufficient number of threads to saturate the GPU's memory bandwidth and compute units. While the current implementation relies on split kernels (separate operations for reconstruction steps), the GPU's hardware scheduler effectively interleaves active warps to hide memory latencies. This ensures that even with the overhead of indirect memory access, the solver achieves significant speedups over serial execution (as evidenced in Table 5 of new section 5.1.4) by exploiting the aggregate throughput of the device. Future development targets further optimization via kernel fusion to maximize arithmetic intensity and reduce kernel launch overhead.

These findings indicate that while memory locality is a challenge for unstructured grids, the WENO scheme's computational weight is an effective mechanism for maintaining high GPU throughput, throttled mainly by high kernel launch density. Regarding future scalability, we have identified two primary optimization paths regarding memory use: (i) implementing memory-efficient batching for the matrix precomputation phase, or (ii) transitioning to 'on-the-fly' coefficient calculations within the kernels to reduce the VRAM footprint at the expense of higher arithmetic load. While these strategies could further extend the solver's capabilities, they often involve lower-level CUDA kernels that may compromise the 'Pythonic' accessibility and modularity central to SWEpy's design philosophy. Consequently, the current implementation prioritizes a balance between high performance and throughput on consumer-grade hardware and code maintainability for the broader geoscientific community.

[o.] It would be much easier to evaluate the output in 2D rather than 3D in most of the cases presented.

R:> We thank the reviewer for this helpful suggestion and apologize if the presentation made interpretation less convenient in some cases. Our intention in using three-dimensional surface visualizations was to remain consistent with how results are commonly presented in the literature for the same benchmark tests and real-case scenarios, where free-surface elevation is often shown in perspective to highlight wave propagation, run-up, and inundation patterns. These plots are intended primarily to provide physical intuition for the flow evolution, while the quantitative assessments in the manuscript (e.g., gauge comparisons, error norms, and time series) are based on the underlying two-dimensional solution fields.

[p.] (table 2) The loss of order of convergence is significant and needs to be better addressed.

R:> We fully agree with the reviewer regarding the significance of the convergence order. As now detailed in the revised Section 5.1.1, the observed reduction to 2nd-order is consistent with the behavior of Central-Upwind schemes on distorted unstructured meshes due to stencil irregularities (*Bryson et al., 2011; Nguyen, 2023*). Consequently, the motivation for employing the quadratic WENO reconstruction was not to strictly achieve 3rd-order convergence, but to leverage its superior dispersive properties and lower error constant compared to linear schemes. This reduction in numerical diffusion is the primary driver of our choice, as it ensures better preservation of long-range wave structures, which is a priority for SWEpy's application capabilities. Exploring the mathematical reason for this loss of order in the extended scheme would require a formal analytical derivation of its order, while, of much value, it represents a distinct theoretical study beyond the scope of this work.

[q.] (fig 9) Label the figure axes with the actual thing being plotted, e.g., surface depth. This is also not a gauge plot, so I am not sure that the caption is correct.

R:> We thank the reviewer for pointing out the error. We've updated the labels and captions to better reflect the image's content.

[r.] (fig 9) Why is there water being plotted on top of the island? There are additional suspicious looking areas that are wet for all the methods.

R:> We thank the reviewer for this careful observation. The apparently ``wet'' cells over the island are a visualization artifact related to the tolerance used in post-processing rather than a physical or numerical failure of the solver. In SWEpy, cells are classified as wet or dry for plotting purposes using a small threshold on the computed water depth. Near wet/dry interfaces, the numerical solution may contain extremely small residual water depths due to

discretization and reconstruction effects. These values are dynamically insignificant but can still appear as wet if the visualization tolerance is set too low.

For the conical island test case, the suspicious cells correspond to water depths on the order of 10^{-5} - 10^{-4} m, while the incident wave height is approximately $3*10^{-2}$ m. Thus, these values are several orders of magnitude smaller than the physically relevant solution and do not affect the flow dynamics. This behavior is further illustrated in the following figure: Panel (a) shows a comparison between the coarse original triangular mesh and a finer grid, while panel (b) displays the corresponding solution where tiny residual depths near the shoreline are visualized as wet cells. Panel (c) shows the same test on a refined mesh. With increased resolution, the residual depths near the wet/dry front drop to 10^{-10} - 10^{-9} m, effectively eliminating the visible artifact. This confirms that the effect is resolution and tolerance related rather than a systematic wetting of dry land. The images in the manuscript have been fixed accordingly, and this issue has been addressed.

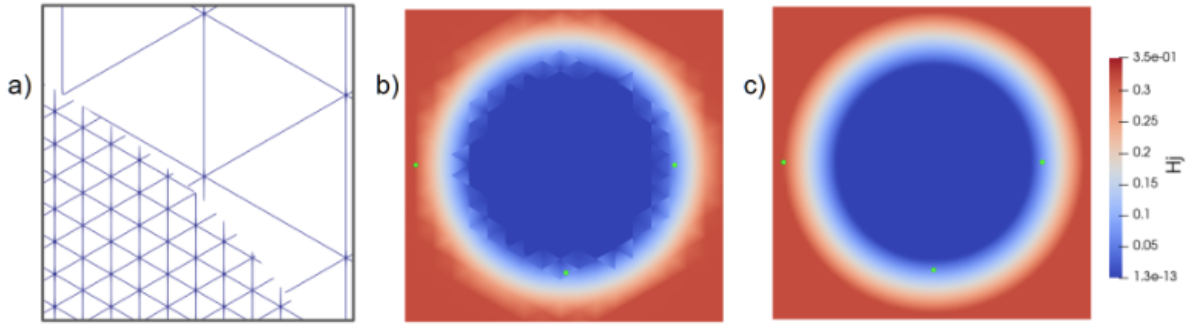

Figure 1: (a) Grid size comparison closeup. Water column heights (Hj) around protrusion of conical island diminish from the (b) coarse grid to the (c) fine grid.

[s.] (table 3) It would be helpful to report L$^\infty$ or max errors for these types of problems (you could use a shifted norm for these)

R:> We thank the reviewer for this valuable suggestion. As noted in other responses, the model formulated through the Saint-Venant equations is subject to well-known limitations arising from its derivation based on depth-averaging, simplifying assumptions (such as hydrostatic pressure), and the aggregation of different physical contributions within the Manning friction term. In this context, we consider that a central objective of this class of models is to describe the qualitative behavior (i.e., trends) of the flow variables rather than to achieve strict pointwise numerical accuracy. From this perspective, the L^2 error, as is presented in Table 3, provides a consistent and meaningful measure of comparison, allowing us to assess whether the phenomenon has been captured with a representative level of accuracy in an applied setting, such as the conical island analysis. Nevertheless, we agree that reporting L^∞ errors could be informative; such metrics are used elsewhere in the manuscript as complementary indicators, for instance, in the interpretation of the mass-preservation test.

[t.] The conical island problem is using a first order time method, why? To isolate the spatial reconstruction, a high order time marching scheme should be used instead.

R:> The reviewer raises a good point. While it's true that spatial reconstruction significance is most effectively isolated by using a higher-than-spatial order time marching scheme, the simpler conditions of the experiment allow us to showcase a higher effective CFL number that still produces high-quality results, demostrating the superiority of the WENO scheme. This reasoning is now better explained in the conical island experiment section of the revised manuscript.

[u.] A test demonstrating well-balancing and conservation is critical for verification of the methods.

R:> We fully agree with the reviewer that demonstrating exact well-balancing and mass conservation is a prerequisite for any robust shallow water solver. To address this, we have incorporated a new section (Section 5.1.2: Well-balancing test) and a comprehensive mass-balance analysis in the User Manual & Technical Reference:

- Well-Balancing stress test: We have implemented a "Lake at Rest" benchmark over a white-noise-generated bathymetry (see Figure 8). This test introduces high-frequency oscillations in the bottom topography. As shown in Figure 8a, SWEpy preserves the static equilibrium, with zero spurious oscillations and residuals reaching machine precision. This confirms that our bathymetry source term discretization is exactly well-balanced with the flux gradients.
- Conservation: Mass conservation is examined in Section 8 of the validation and benchmark of the User Manual & Technical Reference. The results are based on the Synolakis wave runup experiment, focusing on the propagation of a soliton. The experiment was carried out considering several reconstruction methods. The temporal evolution of mass is presented, showing mass conservation except during constant reconstruction. Relative errors are also provided.

These results, now detailed in the revised manuscript, provide numerical evidence that SWEpy adheres to the fundamental physical principles required for reliable geophysical modeling. Furthermore, we have included these verification cases in the newly created User Manual to allow users to replicate these benchmarks.

[v.] Malpasset Benchmark:

- **Were refinement studies done given the claim that the resolution has an impact on the comparison?**

  R:> Yes, a finer, midpoint subdivision mesh (also part of the TELEMAC dataset) was run and results improved slightly. The coarse mesh was constructed using available topobathymetric data from EDF, however the fine one appears to be a linear

interpolation of the coarse one. This means that true higher-resolution terrain isn't achieved, which certainly might affect the accuracy of the results. However, the TELEMAC guide doesn't report results using this mesh, so we opted for the fairest comparison.

- **The boundary problem and false wet/dry problems seem significant. Can this be better addressed than what is listed?**

  R:> We acknowledge that models based on the Saint-Venant equations may be insufficient for the description of problems of this type. This limitation also applies to the modeling of the friction term and the representation of pressure effects. From an application-oriented perspective, the main objective of this work was to generate flow variables that allow for comparison with measurements available at specific locations; that is, to perform simulations with the lowest possible level of complexity while still capturing the fundamental qualitative behavior of the flow variables. These results are intended to support decision-making in scenarios such as dam-break disasters. This aspect is emphasized in the plan-view descriptions shown in Figure 15, together with the comparative results reported in Table 5.

  The purpose of Figure 14 is to illustrate how certain descriptions can be improved using the tools implemented in this first version of \texttt{SWEpy}. However, since the wet–dry treatment criterion may lead to the artificial water influx problem, addressing this issue would require modifications to the governing model, the conditions imposed in these regions, and the introduction of an adaptive reconstruction procedure. We are currently investigating these aspects in the context of an ongoing numerical and theoretical study.

- **I am not sure that the Malpasset benchmark shows that SWEpy could be used to replicate results from more careful validation studies. Additional comparisons may need to be done, and a demonstration that the boundary issues are not impactful would be helpful.**

  R:> We appreciate the reviewer's concern regarding the robustness of the Malpasset benchmark. To address this, we have revised Section 5.2 to include a better discussion on the boundary condition. Specifically, we demonstrate that the "leaky" boundary artifacts, arising from the interaction between the high-order reconstruction stencils and the ghost cells in steep and dry topography, are small, fairly localized and do not propagate quickly into the primary analysis region.

  As illustrated in Figures 15 and 16 of the manuscript, these spurious inflows occur outside the domain sections where the flood wave is actively propagating during the measurement window. Crucially, their limited impact is confirmed by the quantitative metrics in Table 7 where SWEpy achieves wave arrival times and peak water heights that match or exceed the accuracy of the reference TELEMAC data. Also, the new sentence around line 704 quantifies the smallness of the introduced water heights (~0.01m) versus the height of the inundation wave (~10m). This

confirms that while the boundary treatment requires future refinement for long-term stability in confined basins, it does not compromise the solver's ability to accurately reproduce the catastrophic flooding dynamics in this benchmark. We chose to report these limitations transparently to provide a realistic assessment of the current solver capabilities in realistic topographic scenarios.

[w.] Maule Benchmark:

- **Make a note to the reader that FE == forward Euler, not finite element, which is implied but may lead to confusion.**

  R:> As indicated in line 633, we have revised the notation and now refer to the forward Euler scheme as EE (explicit Euler) to avoid ambiguity with the finite element method.

- **Why is the CFL 0.25 rather than at least 1/3?**

  R:> The choice of CFL = 0.25 was driven by the stability constraints inherent to the minmod limiter with a high diffusion parameter. While a theoretical CFL of 1/3 is the maximum for the Central Upwind method, in this study we employed a Minmod limiter with a diffusion parameter ϑ=1.4 (where ϑ > 1 reduces numerical diffusion but increases the risk of oscillatory behavior). To ensure a fair and consistent comparison across all solver configurations (Explicit Euler vs. RK3, Minmod vs. WENO), we adopted a conservative CFL=0.25. This value provided a stable "common denominator" that prevented the EE+minmod from blowing up, allowing us to evaluate the effects of all reconstruction and time-marching configurations without varying the time-step stability limit between runs.

- **The explanation of high-frequency waves in the FE+WENO at low resolution does not make sense (fig 18). Instead, the low-resolution in both time and space should have stabilized the run. What is going on here?**

  R:> We totally agree about the coarser space resolution tends to smooth out spurious oscillations. However, it is our understanding that lower temporal resolution increases the risk of amplified spurious instabilities in long-range wave-propagation problems, especially in the absence of diffusion and dispersion terms, due to nonlinear wave steepening.

  Numerical experiments were conducted considering traveling solitons in order to investigate the development of these instabilities. It was observed that results obtained with the explicit Euler (EE) scheme are highly unstable in regions far from the soliton peak, possibly due to the generation of superimposed reflected waves. It was also observed that a smaller CFL coefficient could control these instabilities in the EE case, but would result in higher numerical dissipation. These unstable oscillations could explain the results shown in Fig. 18(b), where the superposition of

upstream oscillations ahead of the propagation front leads to this unstable configuration.

In this context, we note that in SWEpy's User Manual, the soliton experiments described above are presented together with a discussion of the possible mechanisms underlying these instabilities. From these results, it is also observed that WENO reconstruction reduces numerical diffusion. Using this reconstructor, additional experiments were performed with different mesh resolutions to analyze the development of instabilities arising from spurious oscillations. In particular, the FE+WENO configuration with the most refined mesh was found to produce larger amplitudes of spurious waves among the cases considered. Such oscillations do not develop in the Runge–Kutta (RK) configurations, as expected given the strong stability preserving (SSP) property of the temporal scheme employed.

Therefore, the instabilities observed in Fig. 18 may be related to the absence of an explicit dissipation mechanism for spurious waves, combined with upstream wave superposition effects, mesh structure, and the numerical flux formulation. A paragraph addressing these points has been added to the manuscript.

- **The quantification as to why the high-order WENO scheme looks more diffusive than HySEA seems important for demonstrating the advantages of the method proposed. Again, a convergence study would help to illuminate the issue.**

  R:> We thank the reviewer for this thoughtful suggestion. We agree that a detailed convergence and diffusion analysis would provide valuable quantitative insight into the relative behavior of different numerical schemes. However, the primary aim of this paper is to present and document the SWEpy software framework and to demonstrate how higher-order WENO reconstructions can reduce the excessive numerical diffusion that affects central-upwind (CU) schemes in long-distance wave propagation. This issue has been noted in previous studies of CU methods, where maintaining wave amplitude and phase over long distances in the far field is challenging. That said, our results aim to illustrate that incorporating high-order spatial reconstruction within the CU framework improves the representation of far-field wave propagation compared to lower-order reconstructions, rather than to provide a rigorous numerical analysis of diffusion mechanisms or formal convergence rates.

  As an exploratory example, we've included a simulation using a coarser mesh, which more closely matches the resolution of a HySEA simulation. The results show that, at this lower, matching resolution, SWEpy is capable of providing results comparable to HySEA, and that the degradation is similar for both softwares and consistent with the lower resolution.

- **Given the importance of GPU computing, having only one benchmark comparing the implementations is insufficient. We also are left wondering if this is a good comparison. Was the CPU code really tuned to a GPU? What**

about a code that is fast on a CPU (HySEA)? Scalability is essential for these problems and nothing is presented. What about a better CPU and GPU, would these differences be the same?

R:> We thank the reviewer for this important observation and agree that a comprehensive and hardware-optimized performance study would be valuable. The primary goal of the performance section in this paper, however, is not to present a fully optimized GPU algorithm or a systematic CPU–GPU scalability study, but rather to illustrate the practical benefits of GPU acceleration within the SWEpy framework.

In pursuit of this, we've added subsection 5.1.4 benchmarking the performance of the Gaussian bump and Conical island tests on our hardware, which further adds to the comparison between the CPU serial version of SWEpy and the presented GPU-accelerated software. While full FLOPs-targeted profiling remains unfeasible due to high kernel launch density, we hope that the results illustrate SWEpy's performance patterns.

To provide a clear, reproducible baseline, the comparison was performed on a single CPU core versus a single consumer-grade GPU, a common reference point in many GPU-accelerated scientific computing studies. The CPU implementation in SWEpy was not specifically tuned for GPU-like parallel execution, and we acknowledge that highly optimized multi-core CPU solvers, such as HySEA, can achieve strong performance. Our intention was therefore not to claim universal speedups across all CPU implementations, but to demonstrate that even without extensive low-level optimization, the GPU-enabled version of SWEpy can significantly reduce runtime for large-scale shallow-water simulations.

We agree that performance results will depend on the specific hardware used, including newer GPU architectures or high-performance multi-core CPUs. A detailed scalability and cross-platform performance study, including strong and weak scaling analyses, would be an important next step, but is beyond the scope of the present paper, which focuses on the numerical framework and its GPU-enabled implementation rather than on hardware-specific optimization.

We have revised the manuscript to clarify the intent and scope of the performance comparison so that it is understood as an illustrative demonstration rather than a definitive benchmark of GPU versus CPU performance across architectures.

- **Discussion of how you used CuPy in replacement of NumPy is very sparse. This is an critical factor that should be held up as a reason to do what you did and would significantly add to the impact of the article. What exactly needed to change (e.g., was it just dropping in `import cupy as np`) or something more?**

  R:> We thank the reviewer for their observation. It was indeed more than just dropping in CuPy as a replacement for NumPy. In the first place, the original version of the code depended on loops over the mesh for the minmod and wet-dry reconstruction procedures. This doesn't port well to the GPU because of memory

synchronization, so all algorithms were vectorized to depend only on SIMD commands. While this is also possible with regular NumPy (in fact, the serial implementation used to test GPU speed up also uses this), CuPy far better exploits vectorization in high mesh resolution cases, as evidenced by the new benchmark results section 5.1.4 of the manuscript. Furthermore, the algorithm for WENO stencil generation was developed using CuPy's complex indexing capabilities rather than iteration or geometric search algorithms to leverage optimized vector masking kernels; and the precomputation of matrix inversion operates on whole stacks of matrices with depth equal to the number of cells rather than iterating through them. Also, regarding data saving, since .vtk files require a specific XML structure, the software writes them line-by-line sequentially, which is an unavoidable loop. To overcome this, explicit GPU-CPU memory transfers before looping were coded in to reduce device synchronization. All these design aspects were taken into account while writing the SWEpy user manual and have been explicitly included in Section 4.

[x.] Code suggestions:

- **The code has no documentation. It was not clear how to even run the examples, which scripts need to be called? How does the input work? Jupyter notebooks or something similar would be perfect for this. There is also no explanation of the input format.**

  R:> We sincerely thank the reviewer for these observations, which are highly pertinent to the standards of the open-source and geoscientific modeling communities. We acknowledge that the initial submission lacked the necessary documentation for a seamless user experience. In response, we have improved the repository and developed a comprehensive User Manual & Technical Reference for SWEpy to ensure full transparency, accessibility, and reproducibility. Our updates address each of your points as follows:

  - [a] Documentation and Onboarding: We have authored a comprehensive User Manual & Technical Reference and a README.md file that serves as the central documentation hub. The new documentation includes detailed sections on hardware and software prerequisites (NVIDIA CUDA, Miniconda), step-by-step and installation instructions.
  - [b] Execution and reproducibility: To clarify "which scripts need to be called," we introduced *run_sim.py* as the unified main entry point in the root directory. This script utilizes an argument parser that allows users to execute benchmarks by simply specifying the input directory and the desired numerical method (e.g., *fe*, *rk3weno*) directly from the terminal. Furthermore, we developed *check_installation.py*, a diagnostic tool that performs a hardware handshake, verifies the CuPy environment, and runs arithmetic test kernels to ensure the system is properly configured before a full simulation.
  - [c] Input framework and format: We have included an *input_example* directory containing a complete, ready-to-run setup. The input logic is now centered on *Config.swe*, a readable configuration file where users specify

physical parameters (gravity, roughness) and numerical controls. The repository now provides explicit templates for the text-based formats required for node coordinates (*NodeCoords.txt*), element connectivity (*ElemNodes.txt*), and boundary conditions (*GhostCells.txt*). Furthermore, we have added a dedicated section in the User Manual & Technical Reference explaining the input data format required for unstructured grids.

- [d] Software architecture: The core solver logic has been organized into a dedicated *src/* directory, maintaining a strict "separation of concerns". This structure isolates file I/O (*cuFileLoader.py*), numerical flux kernels (*cuCentralUpwindMethod.py*), and time-integration engines (*cuSolver.py*), facilitating future community extensions such as new source terms or alternative solvers.

We believe these significant improvements transform SWEpy from a research prototype into a robust community tool. By providing a clear onboarding path, from automated environment diagnostics to standardized input templates, we have ensured that the software is immediately usable after cloning the repository. This restructured framework not only addresses the current usability concerns but also establishes a sustainable foundation for collaborative development.

- **The GitHub repository is large, and the files included seem like they could be auto-generated.**

  R:> We appreciate the reviewer's feedback regarding the repository's structure and size. We recognize that the initial inclusion of high-resolution datasets (e.g., the Maule and Malpasset cases) led to unnecessary size increase and a lack of clarity. In the revised version, we have implemented a strict decoupling of source code and large-scale data, following best practices for scientific software repositories:

  - [(a)] Repository downsizing: We have removed all large-scale simulation outputs and high-density mesh files from the main GitHub repo. The repository now contains only the core *src/*engine and a lightweight *input_example/* directory (approx. 12 MB) intended for quick verification and onboarding.
  - [(b)] Clarification of "auto-generated" files: We clarify that the text-based files in the input folders (e.g., *ElemNeighs.txt*, *NodeCoords.txt*) are the specific mesh topology formats required by our solver. While these are generated by pre-processing scripts, they are essential for the solver's execution. To prevent further confusion, the User Manual & Technical Reference now includes a dedicated "Data Formats" section that describes the structure and origin of each file.

  These changes significantly improve the repository's maintainability and provide a clearer distinction between the SWEpy engine and its application datasets.

- **There are several issues with the code itself, such as remaining hard-coded links.**

  R:> Thank you for pointing this out. We have removed the remaining hard-coded paths and links from the codebase. All file paths are now handled using relative paths with respect to the repository root, and any user-specific configuration has been moved to input files or configuration variables. This ensures that the code runs correctly after cloning the repository on different systems.

- **How does one run the code? Cloning the repository gives no indication of how to actually run anything.**

  R:> We apologize for the lack of clarity in the initial submission regarding the execution of the solver. We have completely restructured the repository to provide a standard, "plug-and-play" experience for the user. These improvements are documented in the new User Manual and the README.md:

  - [(a)] Unified entry point: We have introduced *run_sim.py* in the root directory as the primary interface for the solver. Users no longer need to navigate internal folders or guess which script to call. The simulation can be started with a single command: *python run_sim.py*.
  - [(b)] Commandline interface (CLI): The entry script now includes an argument parser (*argparse*) that allows users to define the input folder and the numerical scheme directly. For example, to run the high-order WENO solver with RK3, the user simply executes: *python run_sim.py ./input_example/ -m rk3weno*.
  - [(c)] Automated diagnostics: To avoid common setup errors related to CUDA/CuPy, we added *check_installation.py*. This script performs a system check, verifies GPU visibility, and confirms that the environment is ready for execution before the user attempts a full run.

  The README.md file and the User Manual now include a "Quick Start" guide that walks the user through installing prerequisites with Miniconda and running a baseline benchmark, ensuring immediate reproducibility.

- **The license choice can significantly limit the adoption and use beyond academic and government entities. Not to suggest you need to change this, but in the intro of this package you may want to consider changing it to a more permissive license.**

  R:> Thank you for this thoughtful comment. We agree that license choice influences the potential adoption and use of scientific software. The decision to release SWEpy under the GPL license was made to ensure that future modifications and derivative works remain openly available to the community, in line with the project's emphasis on transparency, reproducibility, and collaborative development.

We recognize that more permissive licenses can facilitate broader industrial adoption, and we have revised the Introduction to clarify that the GPL license reflects a deliberate choice that prioritizes open scientific development. This clarification has been added to better communicate the rationale behind the licensing decision.

- **Jupyter notebooks would greatly enhance this demonstration.**

  R:> We thank the reviewer for this excellent suggestion. We agree that Jupyter Notebooks are powerful tools for interactive learning and data exploration. While we have not included notebooks in this specific revision, we have prioritized making the standalone execution workflow as intuitive and "notebook-like" as possible through the following structural improvements:

  - [(a)] Command-Line Interface (CLI): Instead of multiple scripts, we have unified the execution into a single entry point: *run_sim.py*. This allows users to run the included benchmarks with a single, clear command (e.g., *python run_sim.py*), providing a "plug-and-play" experience that rivals the simplicity of a notebook cell.
  - [(b)] Automated environment setup: To lower the entry barrier (a common use for notebooks), we developed *check_installation.py*. This tool automates verification of the GPU, CUDA, and CuPy environments, ensuring users can proceed with the simulation without the common friction of manual library debugging.
  - [(c)] Comprehensive technical documentation: The updated User Manual and *README.md* provide a linear, step-by-step tutorial that mirrors the notebook's narrative. It guides the user from hardware diagnostics to result visualization in ParaView, using the ready-to-use data in the *input_example/* directory.

  We consider a robust CLI to be the necessary foundation for the solver's long-term stability. However, we fully intend to release a suite of Jupyter Notebooks for interactive post-processing and parameter sensitivity studies in the next minor update of SWEpy, leveraging the modular architecture established during this revision.