Author response to reviewer comments on Ghosh and Regayre manuscript entitled "Optimizing Gaussian Process Emulation and Generalized Additive Model Fitting for Rapid, Reproducible Earth System Model Analysis." We thank both reviewers for their careful reading of the manuscript and for their constructive and insightful comments, which have helped improve the clarity and presentation of the paper. Reviewer comments are reproduced in blue, followed by our responses in black. Revisions to the manuscript are indicated in red.

In addition to the specific revisions made in response to the reviewers' comments, we have also made a number of minor editorial changes throughout the manuscript to improve clarity, readability, and consistency of presentation. These include improving paragraph flow, simplifying technical descriptions, correcting minor grammatical issues, and removing repeated citations. These changes were made in the spirit of addressing the reviewers' broader suggestions regarding clarity and presentation.

Reviewer1

Review for GMD of Ghosh et al. manuscript entitled "Optimizing Gaussian Process Emulation and Generalized Additive Model Fitting for Rapid, Reproducible Earth System Model Analysis."

This manuscript describes an approach to computer model emulation and sensitivity analysis using parallel and distributed computing on high performance computing (HPC). Specifically, they fit scalar Gaussian process (GP) emulators to grid-cell level Earth system model output in an embarassingly-parallel fashion, and use the fitted emulators to conduct main effects-style sensitivity analysis using generalized additive models (GAMs). Each model fit occurs on a core independently of the other fits and speedup is realized by running many such fits simultaneously on many cores, with no between-node communication required.

The resulting parallel speedups seem reasonable and are significant in magnitude. It might be easier to see the parallel efficiency if plotted differently, e.g. on a log-log plot, or with a theoretical maximum speedup curve (equal to number tasks) superimposed, or speedup as a fraction of theoretical. I am not sure I would say that optimized walltime is nearly constant up to three orders of magnitude in task count; we'd expect it to be constant when the tasks are less than the concurrency cap of 200 cores, but it should (and does) increase after that. The speedup graphs are not as informative as they could be, for that reason, because the baseline was never run (and the speedup calculated) for task sizes exceeding the number of cores, where different speedup behavior is expected (of course this is because of the increasing expense of running the baseline) - perhaps at least one long baseline analysis at (say) 400 cores could be run?

We thank the reviewer for this insightful and constructive comment regarding the presentation of parallel scaling performance. We agree that the original figures did not clearly

distinguish between ideal scaling behaviour and practical performance beyond the concurrency limit.

All optimised experiments were conducted with a fixed allocation of 200 CPU cores. Task counts exceeding this limit are therefore executed in batches rather than concurrently. We acknowledge that our previous statement that "walltime remains nearly constant" was imprecise; this behaviour applies only when the number of tasks is less than or equal to the concurrency limit. Beyond this point, walltime increases as expected due to batching, and this has now been clarified in the revised manuscript.

Following the reviewer's suggestions, we have substantially improved both the analysis and presentation:

We now include a theoretical maximum speedup curve (equal to min(N, 200)) to clearly indicate the concurrency limit. We add an explicit parallel efficiency plot, defined as the ratio of measured speedup to the theoretical maximum. We have performed additional long baseline (serial) experiments at larger task counts (512 and 1000 tasks), enabling direct comparison beyond the concurrency limit. Walltime scaling is now shown on logarithmic axes, which more clearly illustrates the linear scaling of the baseline and the deviation from ideal scaling in the optimised workflow.

These updates are reflected in both Fig.~4 and Fig.~6, which now clearly demonstrate the transition between regimes: near-ideal scaling for task counts below 200, and reduced efficiency at larger task counts due to batching and system-level constraints.

Revised text (Sect. 3.1, lines 213–222):

"To quantify the scaling behaviour, we define the theoretical maximum speedup as the minimum of the number of tasks and the 200-core concurrency limit. Measured speedup is calculated as the ratio of baseline to optimised walltime, and parallel efficiency is defined as measured speedup divided by the theoretical maximum.

At higher task counts, we performed additional baseline experiments at 512 and 1000 tasks to extend the comparison beyond the concurrency limit. These results confirm the expected deviation from ideal scaling due to batching and scheduling overheads. At low task counts, the measured speedup approaches the theoretical maximum, reaching approximately 130 times faster than the baseline. Parallel efficiency decreases beyond the concurrency limit and exhibits non-monotonic behaviour at large task counts, reflecting variability in walltime associated with HPC scheduling and I/O effects. Overall, the optimised workflow removes the sequential bottleneck in the GP stage, transforming the emulator workflow into a scalable, cluster-parallel process and enabling efficient utilisation of available HPC resources."

And (Sect. 3.2, lines 284–296)

"In the baseline workflow, runtimes increase more than proportionally with the number of tasks, quickly becoming impractical beyond a few hundred grid boxes, while peak memory use frequently approaches or exceeds node capacity.

In the optimised workflow, rBAM substantially reduces both runtime and memory requirements, achieving more than 20 times faster performance at approximately 10^3 tasks. However, the scaling departs from the theoretical maximum beyond moderate task counts. Although up to 200 CPU cores are available, full utilisation is not always achieved in practice because memory demands from the preceding GP emulation stage can limit the number of tasks that run simultaneously.

In particular, GAM fitting remains memory intensive, and the effective level of parallel execution is constrained by the total memory available on the JASMIN system, which typically ranges from about 100 to 190 GB depending on node availability and system load at runtime. As a result, fewer tasks can run at the same time than the nominal core limit would suggest, leading to batching effects and the observed flattening and fluctuations in speed-up and efficiency beyond approximately 100 tasks.

Despite these practical constraints, the optimised workflow maintains substantially lower walltimes and higher efficiency than the baseline, enabling scalable application of GAM-based variance decomposition at resolutions and ensemble sizes that were previously infeasible."

Other than that, a few other points to consider:

Could the GAM have been fit directly to the data instead of to GP emulator samples? The GAM essentially is a simple emulator (neglecting parameter interactions).

We thank the reviewer for this helpful comment. We agree that a GAM can, in principle, be fitted directly to the PPE data and may be viewed as a simple emulator under an additive (no-interaction) assumption. However, in our workflow the GP emulator and the GAM serve distinct purposes.

The GP emulator is first used to approximate the model response and to generate a dense sample of the high-dimensional parameter space from the original PPE. The available PPE consists of 221 simulations in a 37-parameter space, which provides only sparse coverage. Fitting a GAM directly to this limited sample therefore leads to less robust, noisier estimates of parameter effects, particularly at the grid-box level considered here.

While fitting a GAM directly to the original 221-member PPE is computationally inexpensive, the limited sample size provides only a coarse representation of the response surface. In contrast, using the GP emulator to generate a much larger sample (order 10^6) improves coverage of parameter space but makes GAM fitting computationally demanding, motivating the parallel HPC workflow developed in this study.

The GAM is therefore used as an interpretable post-processing tool applied to the emulator-generated ensemble, rather than as a replacement for the emulator itself. This distinction has been clarified in the revised manuscript (Sect. 2.2.2, Baseline GAM fitting).

Revised text (Sect. 2.2.2, lines 137–148):

“To clarify the distinct role of the GAM within the overall workflow, we briefly describe its relationship to the GP emulator. The GP emulator is first used to approximate the model response and generate a dense sampling of the 37-parameter space based on output from the original 221-member PPE. This dense sampling provides a smoother and more complete representation of the response surface. At the grid-box scale considered here, model responses are often highly non-linear and spatially heterogeneous, so fitting a GAM directly to the sparse PPE would provide only a coarse and potentially noisy estimate of parameter effects. The emulator-expanded sample therefore provides a more robust basis for variance decomposition and sensitivity analysis.

While a GAM fitted directly to the original PPE is computationally inexpensive and can be performed without HPC resources, it remains constrained by the limited sampling density of the parameter space. In contrast, the GP emulator enables efficient generation of a much larger ensemble (order 10^6), providing a smoother and more complete representation of the response surface. Fitting a GAM to this extended dataset becomes both memory-intensive and computationally demanding, motivating the need for the parallel HPC-based workflow developed in this study.”

It may be worth pointing out that the GAM sensitivity analysis is like a (unnormalized) first-order Sobol' sensitivity analysis with the assumption that there are no between-parameter interactions.

We thank the reviewer for this helpful and insightful suggestion. We agree that the GAM-based sensitivity analysis used in this study is closely related in interpretation to a first-order Sobol-type sensitivity measure under an additive (no-interaction) assumption.

In our approach, the contribution of each parameter is quantified through the variance of the corresponding one-dimensional response obtained by varying that parameter while holding all others fixed (“median-hold'' procedure). This provides a measure of first-order marginal importance that is analogous to an unnormalised first-order Sobol index.

We note, however, that this is not a formal Sobol decomposition. In particular, the quantities are not normalised by the total variance, and interaction terms are not explicitly represented because the fitted GAM is additive. The resulting sensitivity measures should therefore be interpreted as first-order, additive approximations of parameter importance.

We have clarified this relationship in the revised manuscript (Sect.~2.2.3, Baseline (median-hold) variance computation), including appropriate references to Sobol-type sensitivity analysis.

Revised text (Sect. 2.2.3, lines 164–169):

“This quantity provides a measure of first-order marginal importance for each parameter. It is analogous in interpretation to an unnormalised first-order Sobol-type sensitivity measure under an additive (no-interaction) assumption, as each parameter is varied independently

while all others are held fixed (Sobol, 2001; Saltelli et al., 2008). It is not a formal Sobol index, as it is not normalised by the total variance and does not include higher-order interaction effects.

Repeating this process for all 37 parameters yields a set of per-parameter variances $Var(f_i)$ and gradient signs given by $sign[Cov(x_i, f_i)]$, which can be compared to quantify the relative importance of model parameters in driving output variance."

Could this software be extended to analyze time series at a grid cell? This would require a different form of emulator and different distribution of data onto tasks. (Obviously a spatial analysis would be harder since communication across boundaries would become inevitable, although spatially-partitioned Gaussian processes can be applied.)

We thank the reviewer for this insightful comment. The current framework could, in principle, be extended to analyse time series at individual grid cells or across a region, although this would require modifications to both the emulator formulation and the task distribution strategy.

In the present study, each grid cell is treated independently, and the emulator is trained on scalar outputs, which enables an embarrassingly parallel workflow. Extending the approach to time series would require an emulator capable of representing temporal structure, such as a multi-output or functional Gaussian process, or an alternative time-resolved surrogate model. This would increase both the computational and memory requirements per task, but the overall task-parallel structure could still be retained, as grid cells would remain independent.

A spatially coupled analysis would present a greater challenge, as it would introduce dependencies between neighbouring grid cells and require communication across tasks. In such cases, the embarrassingly parallel structure would no longer strictly apply, and more advanced strategies, such as domain decomposition or spatially partitioned Gaussian processes, would be required.

Reviewer 2

Review for GMD of Ghosh et al. manuscript entitled "Optimizing Gaussian Process Emulation and Generalized Additive Model Fitting for Rapid, Reproducible Earth System Model Analysis."

This paper discusses how the speed of training of Gaussian Process emulators and associated fitting of Generalised Additive Models can be be improved by an optimised workflow, use of

in-memory I/O, and (I think) an alternative algorithm (as opposed to an alternative implementation) of the necessary prediction matrix.

The overall outcome is a significant improvement in performance, although it it is not clear to me whether the scale up goes beyond 200 cores. Is Figure 5, showing the results of running 10K tasks through 200 cores? In which case, how does the speedup go beyond what can be achieved in 200 cores alone? There is no point showing more than 200 tasks if the rest is linear?

We thank the reviewer for this important comment and agree that the original presentation did not sufficiently clarify the relationship between task count, concurrency, and speedup.

In all optimised experiments, the number of concurrent tasks was limited to a maximum of 200 CPU cores. Task counts exceeding this limit therefore correspond to workloads executed in batches rather than additional parallel resources. For example, a 10,000-task experiment is processed as multiple successive batches of up to 200 concurrent tasks.

The reported speedup is defined relative to the baseline serial implementation and does not exceed the concurrency limit. Apparent increases in speedup beyond 200 tasks reflect improved throughput relative to the baseline, rather than additional parallel scaling. In particular, the baseline workflow remains fully sequential, whereas the optimised workflow maintains high throughput through efficient batched execution.

We agree that this distinction was not clearly conveyed in the original manuscript. In the revised version, both Fig.~4 and Fig.~6 have been updated to explicitly show (i) the theoretical maximum speedup constrained by the 200-core limit, (ii) the measured speedup relative to this limit, and (iii) the corresponding parallel efficiency. The figures now include a clear indication of the concurrency cap and demonstrate the transition from near-ideal scaling (for task counts below 200) to batched execution at larger task counts.

In addition, we have clarified in the text that task counts exceeding 200 are necessary to represent realistic PPE workloads, where the number of emulator evaluations far exceeds available cores. These larger task counts therefore illustrate performance under practical HPC conditions rather than idealised scaling alone.

Revised text (Sect. 3.1, lines 210–222):

“In the optimised workflow, one emulator is launched per Slurm array task, enabling concurrent execution on up to **200 CPU cores**. For task counts below this concurrency limit, walltime remains approximately constant, indicating near-ideal parallel scaling. For larger task counts, walltime increases as tasks are executed in batches, reflecting the finite number of available cores.

To quantify the scaling behaviour, we define the theoretical maximum speedup as the minimum of the number of tasks and the 200-core concurrency limit. Measured speedup is calculated as the ratio of baseline to optimised walltime, and parallel efficiency is defined as measured speedup divided by the theoretical maximum.

At higher task counts, we performed additional baseline experiments at 512 and 1000 tasks to extend the comparison beyond the concurrency limit. These results confirm the expected deviation from ideal scaling due to batching and scheduling overheads. At low task counts, the measured speedup approaches the theoretical maximum, reaching approximately 130 times faster than the baseline. Parallel efficiency decreases beyond the concurrency limit and exhibits non-monotonic behaviour at large task counts, reflecting variability in walltime associated with HPC scheduling and I/O effects. Overall, the optimised workflow removes the sequential bottleneck in the GP stage, transforming the emulator workflow into a scalable, cluster-parallel process and enabling efficient utilisation of available HPC resources."

And (Sect. 3.2, lines 284–296)

"In the baseline workflow, runtimes increase more than proportionally with the number of tasks, quickly becoming impractical beyond a few hundred grid boxes, while peak memory use frequently approaches or exceeds node capacity.

In the optimised workflow, rBAM substantially reduces both runtime and memory requirements, achieving more than 20 times faster performance at approximately 10^3 tasks. However, the scaling departs from the theoretical maximum beyond moderate task counts. Although up to 200 CPU cores are available, full utilisation is not always achieved in practice because memory demands from the preceding GP emulation stage can limit the number of tasks that run simultaneously.

In particular, GAM fitting remains memory intensive, and the effective level of parallel execution is constrained by the total memory available on the JASMIN system, which typically ranges from about 100 to 190 GB depending on node availability and system load at runtime. As a result, fewer tasks can run at the same time than the nominal core limit would suggest, leading to batching effects and the observed flattening and fluctuations in speed-up and efficiency beyond approximately 100 tasks.

Despite these practical constraints, the optimised workflow maintains substantially lower walltimes and higher efficiency than the baseline, enabling scalable application of GAM-based variance decomposition at resolutions and ensemble sizes that were previously infeasible."

This little example shows the problem I have with the paper. I think the work done is very interesting, and worthy of publication, but the write-up doesn't quite do it justification. The authors assume a lot of the readers and the material is not always well ordered.

I would like to see this appear as it is a good piece of work, and a nice example of optimisation a workflow, but would recommend a substantial revision of the material in terms of order and clarity (I say substantial in that i think it would be of substantial benefit, but probably wouldn't take that long and I would expect the editor could decide on this without coming back to me).

In particular, I'd like to see

1. Some clarity in section 3.1 as to whether figure 4 is necessary at all? It appears that the innovation here was to take out a loop over grid points and use slurm job arrays? Is that it? The comment about "spawning competing rscript processes" is presumably

the key part of this? But if this is just making better use of the parallelism and isn't changing the algorithm, why do we need figure 4? And if it is changing the algorithm, how is it doing it, there are no details?

We thank the reviewer for this constructive comment and for recognising the value of the work. We agree that the original manuscript did not sufficiently distinguish between implementation-level changes and the broader restructuring of the workflow, and that this may have led to confusion regarding the nature of the innovation.

The optimised workflow is not limited to replacing a loop over grid points with Slurm job arrays. Rather, it involves a combination of changes to both execution strategy and data handling. In particular, the baseline implementation relied on sequential execution with multiple competing Rscript processes, repeated construction of large prediction matrices, and extensive disk-based I/O. In contrast, the optimised approach restructures the workflow to (i) eliminate competing processes by assigning one emulator per task, (ii) minimise I/O through in-memory data handling, and (iii) redesign the prediction matrix construction to avoid repeated recomputation. These changes together enable efficient scaling across many tasks and substantially reduce both runtime and memory usage.

We agree that this distinction was not clearly communicated in Sect.~3.1. In the revised manuscript, we have reorganised the section to more clearly separate (i) the limitations of the baseline workflow, (ii) the specific components of the optimised approach, and (iii) the resulting performance improvements. We have also clarified the role of Slurm job arrays as an enabling mechanism rather than the primary innovation.

Revised text (Sect. 3.1, lines 197–203):

“This optimisation is not limited to replacing a loop with Slurm job arrays; rather, it involves a restructuring of execution, data flow, and prediction matrix construction. In addition to task-level parallelisation, several key changes are introduced. First, competing Rscript processes are eliminated by assigning one emulator per task, avoiding contention within a node. Second, large intermediate files are avoided by replacing text-based input–output with in-memory data transfer or compact binary output, substantially reducing filesystem overhead. Third, prediction matrix construction is reorganised to minimise repeated recomputation, reducing both runtime and memory pressure. Together, these changes convert the GP emulation stage from a sequential, I/O-bound workflow into a scalable, task-parallel process with bounded per-task memory usage.”

Regarding Fig.~4, we agree that its purpose was not sufficiently clear in the original version. To improve focus in the main manuscript, we have removed this figure from the main text and moved the validation to the Appendix. We now summarise the key result in the revised manuscript and refer readers to the Appendix for the full comparison.

Revised text (Sect. 3.1, lines 204–206):

"Validation tests at representative grid points confirmed that the baseline and optimised workflows produced effectively identical emulator outputs (Appendix Fig.B1), demonstrating that the optimisation preserves statistical fidelity while substantially improving computational performance."

2. A better explanation in section 3.2 of of what is going on, that doesn't mix together algorithm and implementation. I found this really hard to understand. Pseudo function calls are mixed with algorithmic notation. (I said "I think" in my summary, because I have vaguely understood what was going on here, but I would like to actually understand it based on the text.)

We thank the reviewer for this constructive comment and agree that the original presentation of Sect.~3.2 was not sufficiently clear. In particular, the previous version mixed algorithmic description with implementation details (e.g. pseudo function calls), which made it difficult to follow the underlying method.

In the revised manuscript, Sect.~3.2 has been substantially reorganised to clearly separate the different components of the approach. We now distinguish explicitly between: (i) the mathematical formulation of the method, in which the vectorised computation of parameter contributions is introduced using standard notation; (ii) the interpretation of the resulting variance and sign metrics; and (iii) the implementation details, which are described separately in terms of mgcv::bam() and the use of predict(..., type="terms").

Pseudo-code-style descriptions and mixed notation have been removed, and the algorithm is now presented independently of the specific software implementation. Additional transitional sentences have also been introduced to clarify the purpose of each step, including validation against the baseline method and subsequent performance analysis.

We believe that these changes significantly improve the clarity and readability of Sect.~3.2, making the method easier to understand without requiring prior knowledge of the implementation details.

Revised text (Sect. 3.2, lines 226–233):

"The optimised approach replaces this with an R-based Big Additive Model (rBAM), which is designed for efficient estimation on large datasets. In simple terms, the key change is that all parameter effects are evaluated simultaneously in a single pass, rather than recomputing model predictions separately for each parameter as in the baseline approach.

This is achieved through fast restricted maximum likelihood (fREML) estimation, discrete smoothing, and multithreaded execution. In addition, the workflow is restructured to avoid repeated prediction and design-matrix reconstruction, allowing

parameter contributions to be computed in a single batched (vectorised) operation. Together, these changes substantially reduce both runtime and memory requirements while preserving the statistical formulation of the model."

3. The tables in B1-B4 need some thinking about. I understand that these are exemplars showing the difference between the two implementations at four points, but these sort of things are notorious. Why not show scatter plots (or whatever) of the difference across the domain (of however many points it was), for some of the parameters, not all, so we can have confidence that these are not the "best" results, and are typical?

We thank the reviewer for this helpful suggestion and fully agree that demonstrating agreement across the full domain is more robust than relying on a small number of exemplar grid boxes.

Following this recommendation, we have replaced the original four-point comparison and the associated Tables B1–B4 with a domain-wide analysis. Specifically, we now present a scatter plot of variance contributions across 100 grid boxes (3,700 parameter–grid point combinations) in Figure 6 (Section 3.2). In this figure, each point represents the contribution of a single parameter at a single grid cell, allowing a comprehensive comparison between the baseline (pyGAM) and optimised (rBAM) implementations.

This updated analysis demonstrates that the agreement between the two methods is consistently high across the domain (overall Pearson correlation $r = 0.999$), with points tightly aligned along the 1:1 line. This confirms that the results are representative and not dependent on a small number of selected locations.

To further support this, we include an additional spatial diagnostic in the Supplement (Appendix B, Figure C1), which shows the distribution of correlation coefficients across all analysed grid boxes. This indicates that only 4 out of 100 grid boxes exhibit slightly reduced agreement ($r < 0.99$), while the remaining majority show near-perfect correspondence.

As a result of these improvements, the original four-grid-box figure and Tables B1–B4 have been removed to avoid any potential perception of selective reporting. The revised presentation provides a more comprehensive and transparent assessment of the agreement between the baseline and optimised workflows.

Revised text (Sect. 3.2, lines 266–280):

"To verify that the optimised formulation preserves the statistical fidelity of the baseline workflow, we compare variance contributions across a representative spatial sample of grid boxes distributed over the study domain. Figure 5 shows that variance

contributions from the optimised rBAM workflow closely match those from the baseline pyGAM implementation across all sampled grid boxes and parameters. The points lie tightly along the 1:1 line, with an overall Pearson correlation of r = 0.999, demonstrating that the optimised approach reproduces the baseline variance decomposition with high fidelity.

Small deviations from the 1:1 relationship are visible for a limited number of parameters and grid boxes. These arise from numerical and algorithmic differences between the two implementations rather than from any change in the underlying statistical formulation. In particular, pyGAM and mgcv::bam differ in spline basis construction, smoothing parameter estimation (cross-validation versus fREML), and numerical optimisation strategies. In addition, the optimised workflow uses discrete smoothing and block-wise evaluation of the prediction matrix, which can introduce minor numerical approximations at large sample sizes (~10^6). However, these differences are too minor to affect the overall ranking or interpretation of parameter importance. To provide additional context, the spatial distribution of correlation coefficients across all analysed grid boxes is included in the Appendix (Fig. C)."

I should say that Appendix A is excellent and very useful.

We thank the reviewer for this positive comment and are pleased that Appendix A is found to be useful.

Minor comments:

P 2, line 28; the sentence beginning "However" implies that somehow the size of PPEs is linked the complexity of the model. Might be cleaner to say something like: "However, complex models are expensive, and costs are prohibitive beyond o(100) members"

(There is also an assumption in this sentence about how big PPEs need to be, which is surely problem related).

The paragraph then carries on and conflates PPES with the assumption that they are only useful if they have to be expanded in size ... this paragraph probably just needs reordering, since the justification for the expansion in size follows the assumption.

We thank the reviewer for this helpful observation. We agree that the original sentence could be interpreted as implying a direct link between PPE size and model complexity, and that the paragraph structure could be improved.

We have revised the text to clarify that computational cost is driven by model complexity, while the need to expand ensembles arises from the requirements of statistical analysis rather than an inherent property of PPEs. The paragraph has also been reordered for improved clarity.

Revised text (Sect. 1, lines 29–33):

“Complex Earth system models are computationally expensive, which typically limits PPEs to O(100) members. To enable robust statistical analysis of parameter sensitivities and uncertainty, these ensembles are often augmented using model emulation, allowing exploration of millions of parameter combinations at relatively low computational cost (Sexton et al., 2012). While this expansion improves statistical robustness, it introduces additional computational demands during analysis, particularly for variance decomposition and sensitivity analysis across the enlarged parameter space.”

P2 ,line 39/40 bottlenecks are or bottleneck is

We thank the reviewer for noting this issue. The sentence has been revised to correct subject–verb agreement and improve clarity. In addition, we have slightly rephrased the paragraph to streamline the description of computational costs and improve readability.

Revised text (Sect. 1, lines 41–43):

“The major computational bottleneck in the widespread implementation of statistical emulation of PPEs and subsequent variance decomposition arises from the cost of evaluating multiple model variables within each model grid box, a necessary step to establish interpretable functional relationships.”

P 6 On page 5 we were told that each task had 16 GB of memory, but here we are told that peak memory use exceeded 80 GB. I've gotten confused as to what is going on. Is this an example of the "otherwise noted"?

We thank the reviewer for highlighting this point. The apparent discrepancy arises because the reported memory values refer to different stages of the workflow.

The 10–16 GB refers to the per-task memory allocation in the optimised workflow, where each task is executed independently within a Slurm job array. In contrast, the peak memory usage exceeding 80 GB corresponds to the baseline implementation, where pyGAM constructs large spline design matrices within a single process.

We have clarified this distinction in the manuscript (section 2.2.2 ~ Baseline GAM fitting) and now explicitly state that the baseline pyGAM implementation is memory-intensive (~80 GB per process), whereas the optimised workflow uses mgcv::bam (rBAM), which is significantly more memory-efficient. This reduction in per-task memory is critical for enabling large-scale parallel execution across HPC systems and is a key factor in making the optimised workflow scalable.

Revised text (Sect. 2,2,2, lines 150–155):

“In the baseline implementation, which uses pyGAM, memory usage within a single process regularly exceeded 80 GB during spline matrix assembly, reflecting the cost of constructing large design matrices in memory. In contrast, the optimised workflow employs rBAM, which is substantially more memory-efficient and enables block-wise computation and distributed

execution across independent tasks. This allows each task to operate within the typical 10–16 GB memory allocation, making the approach well suited for large-scale parallelisation."

P 7. It is not necessary to keep citing Lawrence et al after the first time JASMIN was mentioned.

We thank the reviewer for this suggestion. Repeated citations of Lawrence et al. after the first mention of JASMIN have been removed.