- Line 34: The authors claim that ShellSet is entirely based on open source software, with a reference to a personal website of one of the authors. Unfortunately this is not sufficient to fulfil the definition of open source software as maintained by the Open Source Initiative (https://opensource.org/osd/). In particular the linked software does not include any license information (after a somewhat exhaustive search). This means that while the author clearly intends to make the software freely available, legally it is not spelled out what users are allowed to do (technically, the strictest copyright applies and users are not allowed to do any modification or distribute the software). This is not a problem for ShellSet itself though, since ShellSet is clearly licensed under GPL 3. I would suggest to either modify this statement to say "freely-distributed" dependencies, or to include an open source license on the linked website to make it clear to users what license this software is distributed under. I am aware that much of this software was written before clear definitions for open source software existed.

Due in part to this comment the author of the linked software is now in the process of updating the licencing of his works, beginning with the programs used in ShellSet. We will update the wording of the article to state that these components are "freely-distributed" but reserve the option to alter again if the licensing process is finalised before the final publication.

- Line 35: ShellSet requires Intel compiler, MPI, and MKL, but only provides a makefile with hard-coded include paths to Intel MKL. This approach is prone to problems on user systems since Intel MKL may be installed anywhere, in particular if the user is not an administrator of their system. A more portable solution would be to use cmake or autoconf as an operating system independent configuration system that allows to automatically find the location of Intel MKL. At least the authors should modify the Makefile so that it points to an include directory that also includes the Intel MPI headers and document how to change this directory. On my system mpif.h could not be found after adjusting the include paths, because the Makefile links to the MKL include directory, not the general Intel OneAPI include directory. I had to modify -I"/opt/intel/oneapi/mkl/2021.3.0/include" to -I/opt/intel/oneapi/2024.0/include, which is an unnecessary hurdle for new or inexperienced users. The (necessary) inclusion of version numbers in this path is another obstacle that results from using hand-written Makefiles.

An update will be made to the Makefile available in the package on GitHub which alters the include paths to use {MKLROOT} which is automatically defined during the installation of Intel MKL. The GitHub page is advertised on the Zenodo package page linked to the article. We will also add information within the user guide related to the issue noted and how/where to change the Makefile as exampled within the comment. Considering our plans for the software, as noted in the article and a later comment, in future we will alter the installation process to something such as cmake to simplify the installation process.

- Section 2.3 The description of OrbScore in the manuscript is insufficient. It is not laid out how exactly OrbScore compares to existing datasets and computes the scores of model results and the only reference to a description of OrbScore is given as a link to a personal website that cannot give a guarantee for future availability. At least the short text that is already available on the website should be included in the manuscript. In addition even on the linked website it is not spelled out "how" the score is computed (e.g. is it an RMS difference, or some other error norm? Is it the same norm for all criteria? How are the different criteria scaled against each other for the combined score?). I understand that some of the other tools have been published elsewhere, but there is no reference given to the original publication of OrbScore. **Later edit:** After continuing to read the manuscript I found the necessary reference in lines 181-182, which refer to the original description in Bird et al 2008. I think at least the paragraph that describes the grading procedure in line 179-183 should be generalized and moved into section 2.3 to explain OrbScore to the reader. My preference would be to also list all the available datasets (this gives the chance to explain any new datasets since the Bird et al. 2008 paper), the ways the individual scores are computed, and the procedure (and reasoning) for forming the final score as geometric mean. Lines 179-183 could then be shortened to say that OrbScore was run with the specific subset of datasets as in Bird et al. 2008.

We will add further details on OrbScore into section 2.3, including details on the scoring datasets provided in the package. For some clarity, OrbScore computes and reports a number of alternative metrics, but (based on years of experience) its author has selected one of these (for each kind of dataset) which is recommended* for model-ranking, and therefore that is the one reported to ShellSet. These choices can be changed within the OrbScore source code, by modifying a single line in each case.

***For example, RMS misfit is only preferred (because of its relation to likelihood) if the errors in the scoring dataset have Gaussian distributions. But for many scoring datasets there are non-Gaussian "outliers" so that another metric, such as mean (absolute value) of misfit is preferred.**

- Line 100: This sentence is somewhat confusing. I think what you are saying is that "by default" Intel MKL will automatically choose the maximum number of threads (e.g. as many as there are cores), but since you also run multiple MPI ranks in parallel it is better to let ShellSet choose the number of threads manually (e.g. as number of available cores divided by number of running MPI ranks). Please clarify the first part of the sentence.

The understanding within the comment is correct, however we agree that there could be some clarity in the wording we have used.

- Line 102+103: usually we speak of MPI processes (or ranks if you refer to the specific number), not threads. Threads are a different parallelization model, so MPI threads does not usually make sense (I think you refer to MPI processes in these two lines).

We will alter each "threads" reference to "processes" where necessary.

- Line 107+108: Either provide the name of the command line argument (if it is important), or remove the statement that it is a command line argument (because it is clear that it is activated somehow). The more interesting question here is how does a user know which "value" to choose to get a sufficiently accurate match. Is there a reasonable default value provided, or would a user have to perform manual testing to get "a feel" for a good value?

The value given to the program should be a decimal which represents the percentage change from the previous iteration (10% = 0.1), this is then used to generate a range from the previous model inside which the new model is deemed "close enough". Since this is an optional feature, we do not place a default value in the code and the user is expected to provide one when activating it. We also do not offer suggestions as to a good value, except to say that looking the full results in the tables within the appendix (currently C) the scores are relatively low and this (at least for this example) should be considered when deciding on a value. We will alter this part of the article to simplify it for the reader.

- Line 111: The grid search is an interesting addition, but its description is missing crucial details about the algorithm. In particular you should spell out somewhere that your grid search is actually a contracting grid search (which searches on multiple levels, narrowing in onto the optimal results). It would also be useful to provide some context on other possible search algorithms or why you chose a grid search (see my references on the conclusion for examples). I can think of several different ways to perform this grid search, and the choice of algorithm presumable influences how well it scales in parallel. E.g. does the algorithm create one grid level, then execute all models in this level on the available cores? Then once the level is finished it creates the next level (and how? use the best value as center point in parameter space for a finer grid? or try to find the neighboring points with best values and span a grid between them?) and finishes that level before refining further? And what happens if the algorithm identifies two disjunct regions with similar error values? **Later edit**: After reading the rest of the paper I found the algorithm description in Appendix A (which is not referenced from the paper). I would suggest to move Appendix A as a subsection into Section 3 (E.g. 3.1 could be the MPI parallelism part of Section 3, 3.2 could be the grid search algorithm, 3.3. could be the user interface changes). At the very least reference Appendix A from Section 3 when you mention the grid search algorithm.

We agree that perhaps the simplest option for the reader would be to move the description of the grid search algorithm from the appendix into section 3. We will rearrange section 3 to facilitate this while avoiding having only a section 3.1.

- Line 120: "theoretically" and "possible" is duplicative and makes the sentence harder to understand

This will be changed.

- Line 121: Please specify the conditions that are currently implemented and checked. As a simple user it is impossible to know what "the most general conditions" are.

We will add information about the encoded variable checks performed by ShellSet.

- Line 124: It is not clear what "its variable values" refers to. Do you mean the input control parameters?

We mean the values for the altered variables - those defining the address of the model within the parameter space. We will clarify this in the text.

- Line 125: "improved user satisfaction" is a very general term and hard to quantify without tools like user surveys etc. What are your sources? Maybe reword to "This combination ... has reduced error-prone manual operations and saved us and our collaborators valuable research time. This allows ... ." or similar

We will update this part in a manner similar to the suggestion.

- Line 137: unnecessary "the" before "Shells"

This will be removed.

- Line 140: Not necessarily a comment on this manuscript but for future versions of the software: You write that you have simplified the controls of the program, but you still require the user to prepare 3 (or so?) different files and up to 9 different command line arguments. In my opinion it should be possible to combine these parameters into a single input file and maybe one or two CLAs. Some of the CLAs look like they are runtime configuration options rather than something that should go into a CLA. And one of your input files only contains paths to other input files, which seems repetetive in design (I understand that this is for historic reasons because the software controls several submodules).

As noted in the comment one of the input files is simply a list of the input files for the three component programs – this is a historical feature which was done to leave the component programs as original as possible (as in response to Daniele Melini's comment). The three new input files are actually 2: the aforementioned input file list and 1 of either a grid search setup or list of models. Of the CLAs only 2 are required for the minimum grid search setup while 1 for listed input, with others controlling optional features or dependant upon the type of model, any defaults are noted in the user guide. We have tried to design the CLAs in a way that means the most commonly changed options and optional features are simply controlled & checked at launch time instead of checking & updating an input file, however in future updates we will continue to refine the setup when needed and depending on feedback.

- Line 145: This is what I was looking for before (how to compute the combined misfit), but you note that geometric mean is the new option, and you do not spell out what the old option was. Also the individual scores are still not explained. Later addition: After reading Section 4 and Bird et al. 2008 I am more confused. Bird et al 2008 already used a geometric mean to compute the combined score, how does this new method differ from the previously published one?

In the original OrbScore there were only 6 misfit scoring options. As noted in the comment the authors of Bird et al. 2008 did use the geometric mean to score their models, however it was computed manually by the

authors after each simulation. In ShellSet the geometric mean is now an encoded 7th scoring option, both for the list input and to select the best models within the grid search procedure. The user can decide to include any combination of 5 misfit scoring options in the calculation of the geometric mean, those which are calculated misfits (GV, SSR, FSR, SA, SD) but not the calculated score (SC). This information will be added to the text for clarity.

- Section 3.1 seems unnecessary as most of the important information is already given in line 35-37. Maybe include the remaining bits in the introduction (like the info about the specific oneAPI toolkits necessary) and delete this section. Also it is generally not considered good style to have a section number 3.1 if there is no 3.2. In addition it is ok to only list the dependency names in the manuscript, however the Github repository of the software should in addition list the minimum version numbers of all the dependencies (Fortran Compiler, MKL, MPI; or simply the combined Intel OneAPI version). I had to test the program on OneAPI 2024, but the authors clearly developed on some other version, so if 2024 wouldnt have worked for me I had no information on which version to use instead.

As noted, ShellSet was first developed using an older version of OneAPI and while we have had no issues using newer versions since then it is correct that version numbers be added to the GitHub repository. Section 3.1 can be summarised and moved into the introduction, while section 3 will change in line with previous comments.

- Line 177: It is not quite clear what "the authors" is referring to (I presume the authors of the original Bird et al paper), please clarify in the text

We mean the authors of Bird et al., this will be clarified in the text.

- Line 206 - 209: This is the description of the grid search algorithm that I would have expected in a general form in Section 3. However, the description is not specific enough in its current form to answer all of my questions above. E.g. You select the best 2 models for the creation of the next grid layer, forming new 3x3 grids on the lower level. From the statement that you have 18 models on the lower models I suppose the 2 selected models do not act as corner points of the next level, but you do not spell this out. Instead I suppose you create two independent 3x3 grids around the model parameters of the best models on the coarse level by varying the parameter values from their old values, e.g. as x_(i+1) = x_i +/- dx/2 if x is a model input parameter, i the level index of the grid hierarchy, and dx the step size of the parameter variation? I also assume that the same parameter combination that was run on the coarse grid is not repeated on the finer grid? This is implied in line 209, but it would be worth spelling out more clearly (e.g. add a sentence that explains how the input parameters on the lower level grids are chosen, and that one of the models on the lower level is identical to the "parent" model on the coarser level and therefore not recomputed). **Later edit**: This comment is now obsolete after I found Appendix A, maybe you can still use some of the ideas to improve upon Appendix A. One question that is still open to me: In Appendix A you describe a 2x2 grid search in the main text a 3x3 grid search. How does the algorithm proceed in a 2x2 grid search if the central model (the one from the coarser level) is the best model? This model is not associated with any of the 4 new cells that were generated. (this is not a problem in 3x3 grids, because the original model is also a cell center on the finer level).

As noted previously we will move appendix A into section 3 and add further detail on the algorithm's workings there.

- Fig. 1+2: The color scale label "Geometric mean" is clear to the authors, but to the reader it is not clear the geometric mean "of what" is shown here. Please reword to "Geometric mean score" or something similar. Also the tauMax axis label is missing its unit (I suspect fFric the other axis is unitless).

We will add the suggested detail to the two figures and add unit labels where necessary.

- Line 296: speedup performance -> performance speedup

This will be changed.

- Line 298+299: This performance result (as well as the table) while showing some benefit of the MPI parallelization is slightly concerning. I understand the complications of the performance measurement that were also already mentioned in community comment 1 (CC1) on the discussion page of the manuscript. I also agree with the authors that the final optimal setting for most users will be to let ShellSet automatically select the number of MKL threads to optimally use the available compute cores. However, I also think the request raised in CC1 (performance table for fixed number of MKL threads) needs to be addressed. My reasoning for this is the following: The MPI problem solved by ShellSet is very close to a scenario that is called 'embarassingly parallel', which means the number of models that need to be computed are independent from each other and require minimal communication. Therefore, we would expect the compute time to scale inversely proportional to the number of available MPI worker processes as long as a sufficient number of compute cores and models to be computed are available and the number of MKL threads per model is constant. However, due to the changing number of MKL threads between the rows of table 3 this is impossible to check from the table. The only case that can be used as a test for this is the transition from 8 workers / 8 models, to 16 workers / 16 models (both of which use 1 MKL thread according to table caption), for which we would expect the compute time to remain roughly constant within some uncertainties (due to changes in model setups). However, the table clearly shows a doubling of compute time from 15 to 29.5 minutes. This implies that either: (i) the MPI ranks are not optimally distributed among the available compute cores (e.g. all ranks are always distributed among the 8 performance cores, efficiency cores are ignored), (ii) the MPI implementation is incorrect or doesnt scale beyond 8 cores, (iii) the given number of MKL threads is incorrect. A rerun of the table with a fixed MKL thread number of 1 on the same hardware (no need for HPC) could distinguish between these cases. This could prove that the authors MPI implementation is correct and the weird timing results from the complications of modern consumer CPU architecture. This would also show that running ShellSet on HPC CPUs (either workstations or HPC clusters) will be more efficient than this table can show. This is because splitting a model into more and more MKL threads will decrease the parallel efficiency, while distributing more and more models among the available MPI workers will not (on modern HPC clusters we can run >100,000 MPI ranks efficiently in parallel, but for the foreseeable future we will not be able to split a model in >100,000 MKL threads efficiently).

The comment is correct that scaling is difficult to understand from the test performed. We had tried to perform a "realistic" test (behaving how a user might in using all cores) however true program scaling performance is then hidden.
We will re-run the performance test fixing the number of MKL threads to 1 in order to better understand the scaling performance of the program.

- Line 305: "We have shown in Sect. 4 ..."

Will be changed.

- Line 306: Specify which new data set

Will be noted.

- Line 311: Move the project name out of the conclusion, this is what the acknowledgments section is for. Keep the future application here.

The project name is not one which funded the creation of the program but one in which the program will be used later. However, we will remove mention of the project to avoid confusion and simply state our future targets.

- Line 314-316: This statement is very vague and seems disconnected from the earlier description of the search algorithm. In particular you have not described earlier which part about the grid search algorithm is currently not efficient. I assume it is the bottleneck of choosing the next grid level after one level has been completed, therefore limiting the total number of models that can be run in parallel. The current formulation of the sentence implies you already know the algorithms you want to try ("altering the search algorithm" instead of "exploring other search algorithms"), so you should either: name the alternatives you want to explore and why (see the introduction of Baumann et al. 2014 for a list of algorithms that have been used in geodynamics and Reuber 2021 for a wider list of sampling algorithms), or at least clarify which bottlenecks exist that you have to overcome with a new algorithm.

References:
https://doi.org/10.1016/j.tecto.2014.04.037.
https://doi.org/10.1007/s13137-021-00186-y

The conclusion will be reworded for clarity, including why we expect some or another search algorithm to offer better performance on a larger machine. Combined with the detail added about the grid search in a previous section and the performance test this will hopefully clarify our reasoning to a reader. We thank the reviewer for the suggested references.

- Appendix A and B seem like important additions to the manuscript, leaving them to the appendix made the main paper harder to read and understand. Appendix A should certainly move into Section 3. Depending on your decision on my comment about section 2.3 Appendix B should either move into Section 2.3 or at least should be referenced from there as a new dataset available for OrbScore.

Appendix A will be merged into section 3. Since appendix B is an outline of the creation of the dataset we prefer to keep that in the appendix in order to keep the article relatively focussed on the program and its performance and examples. However, we will add detail about this new (and other) datasets into section 2.3 and properly refer the reader to the appendix for detail on the creation of the new dataset.