Authors' Response to Reviewer 2

General Comments. I enjoyed reading this paper. Checking and correcting a compressor's output without having to modify the compressor itself makes a lot of sense. The missing-value and vorticity examples help show why this is useful, and being able to combine different safeguards is a real strength. I also appreciate that the authors show cases where safeguarding is expensive or substantially reduces the compression ratio, rather than only presenting favorable results.

Response: Thank you for your feedback.

We took in consideration all your comments in this document.

Comment 1

I would tone down some of the language about safety. Section 2 explains that the guarantee depends on the checks matching the user's requirements, but the abstract and introduction sometimes sound broader than that. Passing the chosen checks doesn't mean that every later analysis is protected. The discussion of bias and spectral errors makes this clear. I don't think the method needs to solve these additional problems for the paper to be useful. I would just make the limits more visible early on, and consistently describe the guarantee as applying to the declared requirements.

Response: Thank you for the comment.

We agree that it is important to clarify that the Compression Safeguards only safeguard the safety requirements that users declare. We have thus changed the abstract as follows:

> We introduce Compression *Safeguards*, a user-centric and domain-independent framework to overcome this trust gap, with which [...] (iii) *guarantee* that the user-declared safety requirements are *always* met by the safeguarded compressor.

> [...] With the appropriate application-specific safeguards, trust can grow in all lossy compressors and even untrusted, potentially unsafe compressors can be used safely and with confidence.

We have also changed the following sentence in the introduction based on the feedback from both reviewers:

> In short, the Safeguards provide hard guarantees that end users can rely on: they guarantee ~~safety~~ that user-declared safety requirements are upheld, have a simplified, consistent, and cross-compressor-reusable protect surface that builds trust, and are compatible with any compressor~~, and their worst-case outcome is at most a reduce~~.

We have amended the start of section 6 ("Discussion") with the following:

> Section 5 and appendix B have examined the effects of applying different lossy compressors across various datasets and analyses, and shown how the `compression-safeguards` reference implementation can safeguard these compressors to guarantee that important user-declared data invariants are preserved.

In section 7 ("Conclusions and Perspectives"), we have made the following changes:

> If the compressor violates the user's requirements, the Safeguards step in and correct them, guaranteeing ~~safety for users~~ that the user's safety requirements are upheld, always.

If users have already-lossy-compressed, and possibly safeguarded, data and only want to check whether it meets additional safety requirements, they can use existing APIs in the `compression-safeguards` implementation to check the compressed data against the uncompressed original reference data.

Comment 2

I was interested in the xarray frontend and would have liked to see it used in the evaluation. The authors explain why the example datasets are small, which is reasonable, but testing individual fields doesn't demonstrate the handling of stencil requirements across chunk boundaries. Could one of the existing examples be repeated with the data split into chunks? Checking the boundary values and reporting the runtime and memory use for a few chunk sizes would already be helpful. I'm not asking for a large multi-node study, just an example that connects the design in Section 3.6 to an actual result.

Response: Thank you for the comment.

The documentation already previously included an example that introduces the `xarray-safeguards` frontend and showcases why safeguarding non-pointwise quantities of interest across chunk boundaries requires specific support: `https://compression-safeguards.readthedocs.io/en/1.0.0rc1/examples/chunked/`. However, we wanted to explicitly investigate your question about the performance scaling of the `xarray-safeguards` frontend as the number of chunks increases, for which we have added a new appendix, B12 ("Performance scaling of chunked safeguarding with the `xarray-safeguards` frontend"). The appendix is based on the relative vorticity example from section 3.6 and examines the overheads from chunking and chunk-overlaps in a single-threaded environment. We have also included the corresponding new figure here – Figure B17 from appendix B12. The new appendix concludes that:

> Overall, this investigation highlights that runtime and memory usage increase as the number of chunks increase, likely caused by the per-computation and chunk-overlap, and per-chunk overheads in the safeguards and `xarray`, respectively. The increase in runtime is outweighed by the increase in the number of chunks across the evaluated chunk sizes, suggesting that this relative vorticity case would benefit from parallelised chunked safeguarding. However, as the runtime is growing faster

than the number of chunks increases, this would not hold forever. In particular, non-pointwise quantities of interest with larger stencils require greater chunk overlap, which significantly reduces the number of chunks where chunking and parallelisation still provide performance benefits.

Please refer to the revised manuscript for the complete discussion.

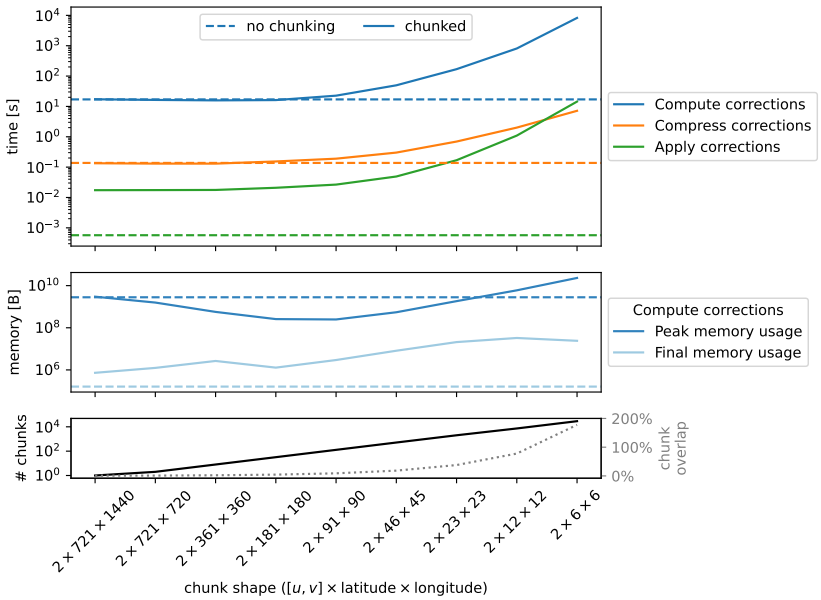


Figure B17: Scaling of single-threaded wall-clock runtime, memory footprint, and chunk overlap with the `xarray-safeguards` frontend when safeguarding the relative vorticity quantity of interest (see section 3.6) across an increasing number of chunks. The upper plot shows the wall-clock time, the middle the final and peak allocated memory usage in light and dark blue lines, respectively, and the lower plot the number of chunks and percentage of overlap between chunks. The baseline without chunking is shown as a dashed horizontal line and the measurements with the `xarray-safeguards` are given as solid lines. The performance is split into computing corrections in blue, losslessly compressing and decompressing corrections in orange, and applying the corrections in green.

Comment 3

The option to distribute corrections separately raises a practical question. How does a reader know that a correction file belongs to the data they are opening? Since the binary differences are computed against a particular decompressed approximation, applying them to a different reconstruction could give incorrect values. Section 3.5 describes a checksum for the combined stream, which is good. Is an equivalent check available for the separate-file workflow? Please explain what is stored to identify the matching data and detect a mismatch. Also, the random-access discussion should distinguish accessing the raw correction array from accessing corrections after lossless compression.

Response: Thank you for the comment.

When distributing corrections as a separate file, we recommend that the correction file should contain metadata to identify which data the correction should apply to, e.g. for labelled data the name and coordinates of the variable and a provenance entry that is based on the original data's provenance. This metadata could be used to automatically open both the data and corrections and to combine them. Ideally, further metadata would protect against the accidental application of corrections to the wrong data. The `numcodecs-safeguards` frontend includes a small checksum for this purpose, and a similar approach could be taken with separate files that are e.g. stored in HDF5 or Zarr format. Storing chunked data complicates the checksum approach, since all data used in the checksum would need to be loaded and corrected to check. However, since many compressors only support decompressing an entire chunk at once, i.e. have no support for partial decompression, the cost of reading the entire chunk and comparing the per-chunk checksum might be acceptable. While the `xarray-safeguards` frontend does not provide such per-chunk checksums itself, since it has no control over how the data is eventually stored and with which chunking, a future `zarr-safeguards` frontend could. We agree that these details should be clarified and have amended our manuscript as follows:

These corrections can then be stored and distributed as a separate file, and later applied after decompression, all without having to modify the compressed data. The corrections file should ideally have metadata to identify which original data it applies to and to guard against accidentally trying to apply corrections to different data. Importantly, storing corrections separately ensures that the unsafeguarded data can still be accessed directly whenever the safety requirements do not need to be satisfied.

We have also added the following about maintaining random access to compressed corrections:

Storing the pointwise binary corrections in addition to the lossy-compressed approximations will reduce the overall compression ratio, particularly if many unique corrections are needed. To reduce the storage overhead of the corrections, they should be compressed losslessly [Tao et al., 2017, Liang et al., 2023], e.g. by combining an entropy encoding and a lossless compressor such as `Zstandard` [Collet, 2021]. To maintain random access to the corrections, their lossless compression should support partial decompression [e.g., Lindstrom et al., 2025]. Often, more than one binary correction can re-establish the safety requirements, and implementations can choose to produce fewer unique corrections to reduce their entropy. In particular, data points that already fulfil the safety requirements should be left as-is, such that their correction is zero. If no corrections are required, i.e. all corrections are zero, corrections do not need to be stored at all.

Comment 4

I would be a little more careful with the comparisons to earlier work. I understand the difficulty with running QPET-SZ, and the reason given for leaving out MGARD-QoI, but the conclusions should stay within what was actually tested. The examples where masking, a ratio-error wrapper, or compressor tuning works better are useful and deserve attention. To me, the strongest contribution is that users can combine requirements and apply them across compressors, not that safeguards are always the most efficient way to preserve a particular property.

Response: Thank you for the comment.

We agree that the Compression Safeguards Framework and our `compression-safeguards` implementation are not the most efficient at preserving every property. We also did not design them with this goal in mind. The Safeguards are primarily designed for safety and flexibility, compressibility and runtime performance were only secondary and tertiary aims. In particular, while we focus on general purpose corrections and flexible and composable safeguards, other compressors specialise in compression ratios and throughput for specific problems and can naturally achieve better performance for those. For example, masking missing values (see appendix B2) or bounding pointwise relative errors (see appendix B4) are better solved by meta-compressors. Importantly, such problem-specific approaches can be used together with the Compression Safeguards to benefit from both. We have adjusted the manuscript to clarify our contribution and this symbiosis with existing methods.

In section 5.4 (“Example: NaN missing values”) we highlight the appendix more:

> While this example has focused on NaN missing values, Appendix B2 examines whether compressors preserve missing values encoded as non-NaN values. It also shows how a masking meta-compressor can also efficiently preserve missing data values.

In section 6 (“Discussion”), we have clarified that meta-compressors, where available,

typically perform better:

> At their most basic, the safeguards can ensure that pointwise absolute, relative, or ratio error bounds over the data are always met and that missing values are preserved, particularly when the wrapped compressor violates their error bound (see sections 5.4 and 5.8 and appendix B2). While such simple requirements can also be fulfilled using specialised meta-compressors, often with greater runtime performance and higher compression ratios (see appendices B2 and B4), the `compression-safeguards` further generalise to composed complex requirements, which can be composed arbitrarily, and across different compressors.

Section 7 ("Conclusions and Perspectives") is now clearer that Compression Safeguards can be combined with other methods:

> The Compression Safeguards are not meant to replace or discourage individual compressors or meta-compressors from implementing measures for safe lossy compression, such as support for missing values or quantities of interest. Indeed, many compressors or problem-specific meta-compressors may be able to integrate specific safety requirements like spectral errors [Ren et al., 2026] more efficiently and produce higher compression ratios. However, supporting the necessary diversity and combinatorial flexibility in safety requirements would require excessive duplication of effort for compression developers and for users who still would need to constantly re-verify which compressors safely fulfil their requirements and which meta-compressors are compatible. Instead, we provide a generally-applicable solution that can be used every time to keep users safe no matter the compressor they use and the requirements they have ~~, and which grows in efficiency with the compressor that is wrapped~~ . Importantly, the Compression Safeguards are symbiotically compatible with existing compressors and meta-compressors so that users can benefit both from high compression ratios and from guaranteed safety, for example by combining the Safeguards with `QPET` ~~or~~ , `OptZConfig` ~~with the~~

~~Safeguards~~ , or a meta-compressor for pointwise relative error bounds.

Comment 5

There are also a few small things to fix. Figure 1 has "to corrections are checked," and its caption seems to be missing some words. "We herefore utilise" should be "We therefore utilise," and "Table B1 shoes" should be "Table B1 shows." Around line 655, the explanation needs checking: cos(phi) tends to zero at the poles, whereas the magnitude of its derivative tends to one.

Response: Thank you for the comment.

We have corrected the typos as suggested.

We have also amended the first sentence of Figure 1's caption as follows:

> Overview of the dataflow in a safeguarded compressor, which guarantees that the user's safety requirements are upheld by an arbitrary compressor.

The mathematical explanation in line 655 was indeed wrong, we have adjusted it as follows:

> Figure 4 shows that the lossy absolute-data-error-bounded compressors, here SPERR, violate the error bound over the derived vorticity close to the poles, where the divisor $\cos\phi$ in Equation (3) tends to zero and the vorticity error-bound can thus only be satisfied by an increasingly small data error.

General Comments. These comments don't change my overall positive view. The paper offers a useful approach, and most of what I'm asking for is clarification and a more concrete demonstration of the chunked workflow. I recommend minor revisions.

Response: Thank you again for your instructive feedback.

References

Dingwen Tao, Sheng Di, Zizhong Chen, and Franck Cappello. Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1129–1139, 2017. doi: 10.1109/IPDPS.2017.115.

Xin Liang, Kai Zhao, Sheng Di, Sihuan Li, Robert Underwood, Ali M. Gok, Jiannan Tian, Junjing Deng, Jon C. Calhoun, Dingwen Tao, Zizhong Chen, and Franck Cappello. Sz3: A modular framework for composing prediction-based error-bounded lossy compressors. *IEEE Transactions on Big Data*, 9(2):485–498, 2023. doi: 10.1109/TBDATA.2022.3201176.

Y. Collet. *Zstandard Compression and the "application/zstd" Media Type*, 02 2021. URL `http://dx.doi.org/10.17487/RFC8878`.

Peter Lindstrom, Jeffrey Hittinger, James Diffenderfer, Alyson Fox, Daniel Osei-Kuffuor, and Jeffrey Banks. Zfp: A compressed array representation for numerical computations. *The International Journal of High Performance Computing Applications*, 39(1):104–122, 2025. doi: 10.1177/10943420241284023.

Congrong Ren, Robert Underwood, Sheng Di, Emrecan Kutay, Zarija Lukic, Aylin Yener, Franck Cappello, and Hanqi Guo. Ffcz: Fast fourier correction for spectrum-preserving lossy compression of scientific data, 2026.