Responses to Reviewers' Comments for Manuscript egusphere-2026-4266

Compression Safeguards: Building Trust into Lossy Data Compression

Addressed Comments for Publication to

Geoscientific Model Development

by

Juniper Tyree, Robert Underwood, Clément Bouvier, Daniel Köhler, Tim Reichelt, Peter Dueben, Sara Faghih-Naini, Heikki Järvinen, and Milan Klöwer

Dear Dr. Rolf Sander,

Please find enclosed the revised version of our previous submission entitled "Compression Safeguards: Building Trust into Lossy Data Compression" with manuscript number egusphere-2026-4266. We would like to thank you and the reviewers for the valuable comments which helped improving the quality of our manuscript. In this revision, we have carefully addressed the reviewers' comments. A summary of main modifications and a detailed point-by-point response to the comments from Reviewers 1 and 2 are given below.

Sincerely,

Juniper Tyree, Robert Underwood, Clément Bouvier, Daniel Köhler, Tim Reichelt, Peter Dueben, Sara Faghih-Naini, Heikki Järvinen, and Milan Klöwer

Note: To enhance the legibility of this response letter, all the editor's and reviewers' comments are typeset in boxes. Rephrased or added sentences are typeset in color. The respective parts in the manuscript are highlighted to indicate changes.

Authors' Response to Reviewer 1

General Comments. Overall, I think the paper is in a good place. I have a few points of feedback that I think ought to be addressed but otherwise see no big defect that should require major changes.

The problem is well motivated, and legitimate. I know from my own research that this area is of import, and this work is meaningful towards a goal of improving confidence when it comes to lossy compression.

Overall, great ideas and engineering.

Response: Thank you for your valuable feedback.

We took in consideration all your remarks in this document.

Comment 1

The fact that, for example, an error bound passed to ZFP is ignored by the safeguards if it is not specifically asked to be protected is a bit counterintuitive to me. If the user is supplying an error bound to the base compressor, it should be respected by the safeguards. Otherwise, why supply an error bound to the base compressor at all? I feel a user would expect both the base compressors guarantee and the safeguards guarantee to be respected. I think this should be made extra clear to the user and ideally be changed to have the initial bounds respected.

Response: Thank you for this thoughtful comment.

We agree that automatically upholding the compressor-provided safety guarantees would be ideal. Fundamentally, the Compression Safeguards are designed to be compressor-independent at their core, explicitly so that they do not need to know about how different compressors work or which configuration options they support, all so that the Compression Safeguards can be used with any compressor. This design decision is incompatible

with preserving, e.g. ZFP's error bound automatically. Importantly, frontends for the Safeguards have no such constraint and can integrate much more tightly with specific compression frameworks or compressors. For example, the `LibPressio` framework [Underwood et al., 2021] has the `pressio:abs`, `pressio:rel`, and `pressio:pw_rel` options that automatically translate the provided absolute, range-relative, or pointwise relative error bound to the compressor-specific option. Therefore, a future `libpressio-safeguards` frontend could automatically read these options to automatically pass their corresponding safety requirements to the Compression Safeguards.

We also agree that the behaviour of the Compression Safeguards is counter-intuitive and a potential pitfall. We have therefore added the following to section 2.2 ("Meta-compression with safeguards") in the manuscript:

> The Compression Safeguards Framework can be used in a meta-compressor to construct a safeguarded compressor wherein one or more safeguards are wrapped around some compressor to guarantee that the user's safety requirements are always satisfied. Since the Safeguards are designed to be compressor-independent, they do not automatically know whether the inner compressor provides any safety guarantees itself, and users have to manually inform the Safeguards about these requirements to ensure that the Safeguards know to uphold them.

We further want to clarify that configuring a different base compressor error bound can still be valuable, e.g. to require fewer corrections for higher-order safety requirements that cannot be expressed as just a simple error bound. Or, to give an even simpler example, ZFP typically produces errors that are significantly lower than the error bound requires [Lindstrom, 2017, Reichelt et al., 2026]. A higher compression ratio could be achieved by giving ZFP an unsafe looser error bound, while guaranteeing that the actual error bound is upheld by wrapping ZFP in Safeguards.

Comment 2

I have some concerns RE: speed. I do agree that paying once at compression time is reasonable, and the decompression overhead is very low. That said, many of these compressors are chosen because they are either fast and/or compress extremely well, this must be the case to keep up with the data as it is being output. I would like to see some throughput numbers for the safeguarded vs native compressors. It's written that they may perform better when compiled natively, but that hand-waves the issue of performance without really addressing it. Even the relative results can't really be trusted because the compressors themselves are in WebAssembly.

Response: Thank you for the comment.

We investigated the performance overhead of using the `numcodecs-wasm-*` compressors instead of their native counterparts and found it to be around 30× across compression and decompression across the three compressors. This was *substantially* higher than we had expected. Since the primary overhead of the safeguards is computing the corrections, which is mostly independent from the performance of the inner compressor, the safeguarded native compressors would perform roughly the same as before; in other words the reported overhead numbers would increase for native compressors by roughly 30× as well.

Our `compression-safeguards` reference implementation was not designed for high performance, focusing instead on flexibility and ease of initial implementation. We had already planned to investigate a more performance-focused implementation of the Compression Safeguards Framework in future work, and thus feel that performance optimisations or changing the evaluations themselves are out of scope for this manuscript. However, we also want to ensure that we do not give the now-misleading impression that safeguarding would have low overhead compared to native compressors. We have thus rewritten most of section 5.7 ("Performance evaluation"), except for the analysis of compression ratios which remains unchanged, and made several further adjustments throughout the text.

The new performance evaluation focuses on comparing the absolute throughput for the different stages of safeguarding – computing corrections, encoding and decoding corrections, and applying them – across the different examples. Our conclusions in this comparison remain the same, and we have replaced the performance evaluation figures to showcase only this comparison and to not invite comparisons between the throughput of a compressor and its safeguarded equivalent. We have also included the new figures here – Figures 5 from the main text and B11 from a new appendix. We have further adjusted the corresponding text to remove all relative comparisons and better highlight the inherent performance limitations of the Compression Safeguards, including:

> Altogether, this comparison shows that the throughput of safeguarding is clearly dependent on the complexity of the safety requirements and the algorithm with which corrections are produced. Fundamentally, safeguarded compression performs significantly more work than unsafeguarded compression. In addition to compressing with the underlying compressor, the data also needs to be decompressed, and corrections need to be computed, checked, and losslessly encoded. Therefore, safeguarding during compression most likely bottlenecks and significantly reduces the compression throughput for modern compressors. If a compressor is optimised for higher performance, the overhead of safeguarding is likely to mask the performance gain. This is in contrast to `OptZConfig`, which needs to compress and check for several iterations and whose performance thus likely improves alongside the compressor's as long as the check of safety requirements does not bottleneck the execution time. In real-world applications where a sufficient target compression ratio might be known in advance, `OptZConfig` can be further sped up by terminating the tuning early once the target has been safely met [Underwood et al., 2022]. Therefore, while safeguarding may offer competitive runtime performance compared to `OptZConfig` in very simple cases, `OptZConfig` outperforms safeguarding for more complex requirements.

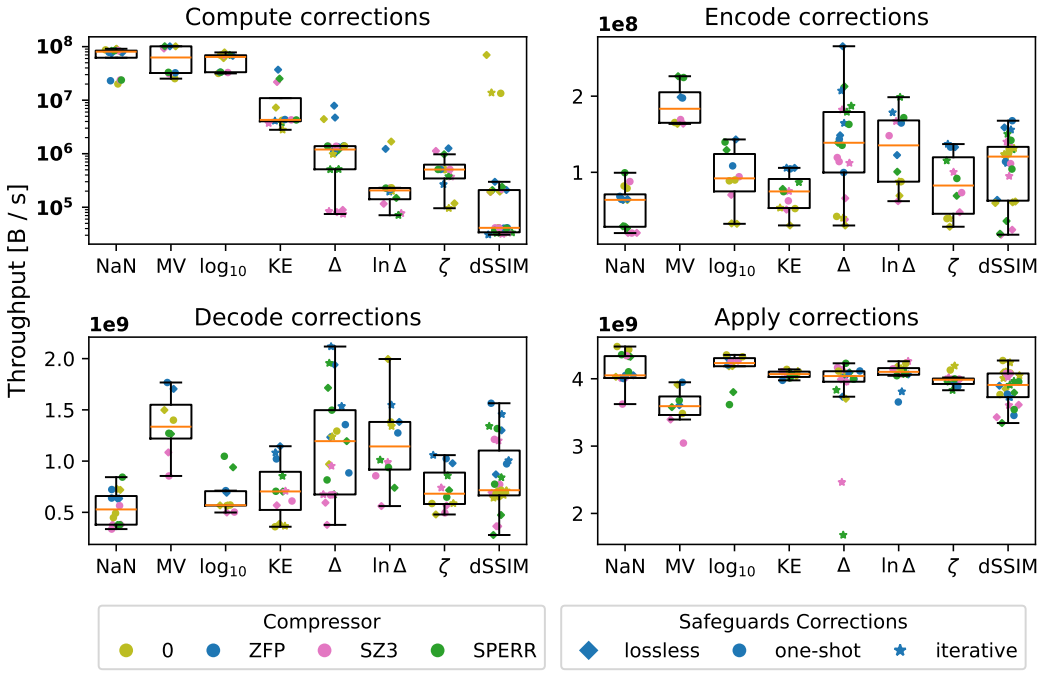


Figure 5: Throughput of the `numcodecs-safeguards.SafeguardedCodec`'s safeguarding stages across all examples in section 5 and appendix B. The throughput is reported separately for computing corrections, encoding (compressing) and decoding (decompressing) corrections, and applying corrections. The distribution of the measurements is visualised with box plots that show the median in orange, the inter-quartile range as a box, and use whiskers to display 1.5 times the inter-quartile range, bounded by the minimum and maximum values. The throughputs are split by the wrapped compressor – the all-zero predictor in lime, `ZFP` in blue, `SZ3` in pink, and `SPERR` in green – and by the mode of corrections – lossless as diamonds, one-shot (default) as dots, and iterative as stars. The examples are abbreviated as NaN (missing values), MV (non-NaN missing values), $\log_{10}$ (pointwise logarithm), KE (kinetic energy), Δ (Laplacian with x=const), $\ln \Delta$ (logarithm of Laplacian), ζ (relative vorticity), dSSIM (dSSIM with 3×3 kernel). The throughput for the non-native WASM codecs, which should not be directly compared with the safeguarding throughputs, are given separately in Figure B11.

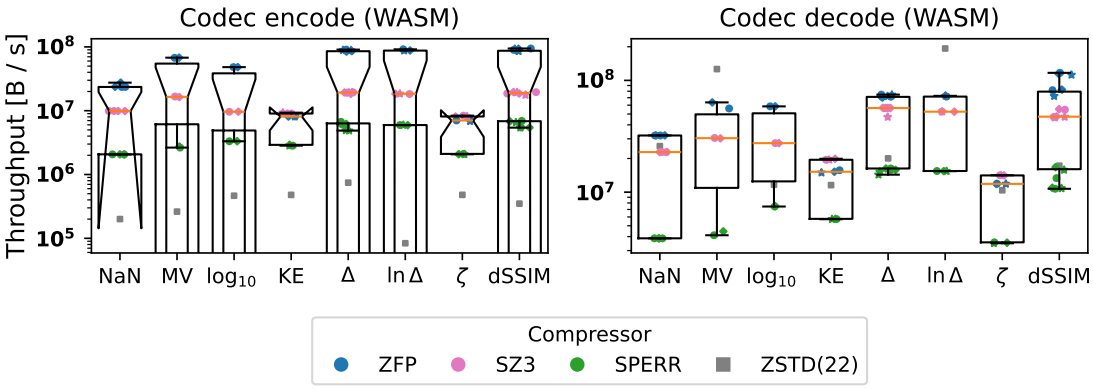


Figure B11: Throughput of compression and decompression, measured in B s^{-1}, with `ZFP`, `SZ3`, `SPERR`, and maximum-level `Zstandard` across all examples in section 5 and appendix B. The throughput is measured for the non-native single-threaded `numcodecs-wasm-*` compressors (see section 5.2), which have significant overhead to guarantee reproducible execution. Therefore, the throughput measurements should not be directly compared with the safeguarding throughputs in Figure 5. The distribution of the measurements is visualised with box plots, as in Figure 5. The examples are abbreviated as NaN (missing values), MV (non-NaN missing values), $\log_{10}$ (pointwise logarithm), KE (kinetic energy), Δ (Laplacian with x=const), $\ln \Delta$ (logarithm of Laplacian), ζ (relative vorticity), dSSIM (dSSIM with 3×3 kernel).

We would like to again thank the reviewer for expressing their concern, which led us to investigate further and to make it much clearer in the revised manuscript that the Compression Safeguards add significant overhead during compression.

We address the following two comments together:

Comment 3

The worst case is worse than suggested. In the abstract the manuscript says, "at most the cost of a reduced compression ratio". While this is still technically correct when the compression ratio falls below 1, it seems to imply that it will still be >1, just closer to 1. Wouldn't it make more sense to just fall back to lossless compression at that point?

Comment 4

"The impact on compression ratio [varies but] is small in many cases" is only really true when the safeguard isn't really doing much work and isn't needed as much. When the safeguard actually performs the safeguarding, the compression ratio impact is fairly large.

Response: Thank you for these two comments.

We agree that the mentioned sentences were not sufficiently clear, and we have changed them as follows:

[...] (iii) *guarantee* that the safety requirements are *always* met by the safeguarded compressor~~, at most at the cost of a reduced compression ratio~~. The Compression Safeguards represent a paradigm shift: [...] We showcase how our reference implementation of the Compression Safeguards can be flexibly applied to safeguard important properties across several real-world examples from weather and climate sciences for different compressors. ~~The impact on compression ratio varies but is small in many cases. The computational load increases during compression but is negligible during decompression.~~ The cost of the Safeguards is a reduced compression throughput, significantly when safeguarding complex safety requirements, and lower compression ratios if the Safeguards have to correct the decompressed data to uphold its safety. There is only minor overhead during decompression.

> Altogether, Compression Safeguards provide a key modular tool that gives users the confidence to use lossy compression safely across scientific disciplines.

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 reduced compression factor~~. [...] The Compression Safeguards are designed to be always used, just like a car seat belt: They incur only minimal overhead in storage and time if no corrections are needed, but automatically intervene to restore ~~safety~~ the user's safety requirements when necessary.

The manuscript already discusses the case of sub-lossless compression ratios towards the end of section 5.7 ("Performance evaluation"), which we have amended as follows:

> Finally, Figure 6b also provides insight into whether safeguarded lossy compression is worthwhile or if lossless compression using `Zstandard` achieves higher compression ratios. For most examples and most compressors, one-shot safeguarded compressors are better, though there are some exceptions. `ZFP` typically provides comparatively low lossy compression ratios, and safeguarded `ZFP` produces sub-lossless compression ratios in the NaN missing value and vorticity examples. Similarly, in other cases where the base-compression ratio is low, e.g. for the unstructured data with non-NaN missing values, or when many corrections are needed, e.g. when preserving the logarithm over the Laplacian, lossless compression can also outperform safeguarded lossy compression. In the worst case, where the compressor produces adversarial random noise and the Safeguards have to produce incompressible unique corrections for every data point, the safeguarded

compression ratio could even drop below one. Such sub-lossless compression ratios could be avoided by falling back to fully lossless compression if necessary [e.g., Zhao et al., 2021, Liang et al., 2023]. However, lossless compression does not take advantage of the tolerable information loss allowed by the safety requirements. A better alternative could be to fall back to safeguarding an all-zero predictor, which instead allows for lossy compression. In all our examples, this consistently provides better-than-lossless compression ratios.

We also extended the earlier discussion of the cost of corrections in section 2.3 ("Implementation-agnostic and hardware-portable corrections") to highlight prior work on the trade-off in producing corrections when necessary:

Even with these remedies, in the worst case of incompressible unique corrections for every data point, corrections may take as much additional space as losslessly compressing the original data. While such a safeguarded compressor may not be efficient, it would still be safe; section 5.7 discusses towards the end how such cases could be handled. If, in contrast, the lossy compressor is well-behaved and configured appropriately, only a few sparse corrections may be required, whose storage cost scales linearly. Safeguarded compressor configurations could even be tuned for higher combined compression ratios using existing compression optimisation frameworks such as `OptZConfig` [Underwood et al., 2022], all while avoiding the fear and optimisation complexity of violating the user's safety requirements. This trade-off between cheap approximate compression / prediction and costly corrections is the foundation of many existing error-bounded compressors like SZ and SZ3 [Di and Cappello, 2016, Zhao et al., 2021], SPERR [Li et al., 2023] and QPET [Liu et al., 2025], which use more specialised and less generally-applicable approaches than the Safeguards.

Comment 5

The novelty is a bit overstated per the references within the paper. For example Gorski et al is essentially the same thing as safeguards but for topology. Additionally, the paper cites FFCz, pMSz, and QPET, all of which are after compression corrections. To be clear, there is still plenty of novelty (e.g., you could easily say yours is the only that does not need to be tightly coupled) but a lot of the language reads as overstating towards being the only post-hoc correction engine.

Response: Thank you for the comment.

We understand that we did not communicate the novelty of the Compression Safeguards clearly enough and appreciate being able to clarify it in the manuscript. We have clarified the introduction as follows:

With Compression Safeguards, users can declare any composition of safety requirements, such as simple pointwise error bounds, complex regional quantities of interest such as preserving the wind divergence inside a hurricane, or global requirements. These requirements are automatically checked for violations after compression and re-established if necessary by producing per-point corrections. Instead of requiring trust in the complex compressor or the complex method for correcting violations, the Safeguards, like `QPET` [Liu et al., 2025], check again to verify their own corrections, thereby guaranteeing that neither safety bugs in the compressor [e.g., Fallin and Burtscher, 2024, Reichelt et al., 2026] nor the corrections can slip through. Compression Safeguards build on the well-established concept of first lossily approximating data before correcting violations [e.g., Liang et al., 2023, Liu et al., 2025, Li et al., 2023, Ren et al., 2026, Li et al., 2026, Fallin et al., 2026, Huang et al., 2025] but provide novel flexibility in fully generalising the concept by (1) delegating approximation to the lossy compressor and (2) producing corrections in a standardised format that (3) supports different implementations and (4) can protect any safety requirement. In short, the Safeguards provide hard

guarantees that end users can rely on: [...]

We have also extended section 4 ("The `compression-safeguards` in Comparison with Prior Work") to specifically address the mentioned examples:

The Compression Safeguards Framework follows the increasingly popular approach of correcting errors from an initial compression step, whether by storing smooth residuals [e.g., Huang et al., 2025] or by encoding outliers losslessly [e.g., Liang et al., 2023, Liu et al., 2025] or ~~lossily~~ quantising errors [e.g., Liang et al., 2023, Li et al., 2023, Ren et al., 2026, Fallin et al., 2026].

We first briefly compare the Compression Safeguards with several methods that specialise on specific safety requirements. **`pMSz`** and **`FFCz`** both wrap existing compressors and iteratively produce post-hoc corrections, to preserve the Morse-Smale complex topological descriptor in `pMSz` [Li et al., 2026], and to preserve spatial and spectral error bounds using both spatial and spectral corrections in `FFCz` [Ren et al., 2026]. **`LOPC`** is an integrated quantisation-based compressor that iteratively computes quantisation sub-bins to preserve topology via local order [Fallin et al., 2026]. These three methods can be seen as three implementations of two safeguards, topology and spectral errors, which produce corrections in three application-specific non-general and incompatible formats that provide, for their specific safety requirement, better correction compression ratios. If the three were changed to output their corrections in the implementation-agnostic binary corrections format from section 2.3, they could indeed be seen as three alternative implementations of the Compression Safeguards Framework, where each only supports its specific safety requirement. What sets our `compression-safeguards` reference implementation apart from the above methods is its support for novel and composable safety requirements, supported through near-arbitrary pointwise and stencil quantities of interest.

We now compare in Tables 1 and 2 ~~compare~~ the characteristics of the

`compression-safeguards` reference implementation ~~and their support for quantities of interest~~ with `QoI-SZ3`, `QPET`, `MGARD-QoI` / `MGARD-λ`, and `OptZConfig`, which showcase prior work on supporting non-specific safety requirements through quantities of interest.

`QoI-SZ3` [Jiao et al., 2022] extends the `SZ3` compressor by analytically deriving a priori per-point absolute data error bounds, which are passed to `SZ3`, that bound the absolute error over a derived quantity of interest. These per-point data error bounds have to be stored as part of the compressed data so that they can be used in `SZ3`'s quantiser.

`QoI-SZ3` supports quantities of interest that contain polynomials, logarithms, or square roots, and may also preserve regional averages as well as isosurfaces over the QoI (see Table 2).

`QPET` [Liu et al., 2025] is the successor to `QoI-SZ3` and bounds the absolute or range-relative error over a derived quantity of interest. It derives approximate a priori per-point absolute data error bounds using the symbolic derivative over the quantity of interest, which are passed to the compressor. `QPET` can be coupled to existing compressors, e.g. in `QPET-SZ` and `QPET-SPERR` (see section 5.2). In case the~~se~~ approximate data error bounds exceed the quantity of interest's error bound, `QPET` post-hoc losslessly encodes the outlier data points. `QPET` supports pointwise and blockwise quantities of interest that contain polynomials, exponentiation, logarithm, trigonometric and hyperbolic functions, but not their inverses, the sign and absolute value functions, as well as isosurfaces (see Table 2).

[...]

`OptZConfig` [Underwood et al., 2022] is a meta-compressor in the `libpressio` compression framework [Underwood et al., 2021] that allows auto-tuning the a priori configuration of a lossy compressor via iterative black-box optimisation to

target or preserve some metric. `OptZConfig` can optimise for any user-defined metric that produces a single scalar output, such as a specific compression ratio or quantity of interest, using a variety of efficiently parallelised search functions. While `OptZConfig` may sometimes produce higher compression than other approaches, it cannot guarantee that a safe configuration can be found, and would thus require a global lossless fallback.

A clear distinction of the Compression Safeguards is that they operate entirely based on post-processing after compression, which is treated as a black box and not manipulated by the Safeguards, whereas `QoI-SZ3`, `QPET`, `MGARD-QoI`, and `OptZConfig` all ~~other methods~~ operate mainly before compression, typically to inject data error bounds, and thus require some level of integration and further fallbacks. While the aforementioned `FFCz` and `pMSz` are also post-hoc methods, they only support specific non-composable safety requirements.

[...]

Overall, Compression Safeguards provide a previously missing last line of defence for end users. They guarantee that the user's safety requirements are always met, support novel safety requirements, and can be used with any compressor, including `FFCz`, `pMSz`, `LOPC`, `QoI-SZ3`, `QPET`, `MGARD-QoI` / `MGARD-`λ, and `OptZConfig`, to combine their strengths with the guaranteed safety that the Compression Safeguards provide.

We further adjusted section 5.8 ("ClimateBenchPress") and section 7 ("Conclusions and Perspectives") as follows:

We can also observe that the safeguarded zero-predictor provides good compression, better than `ZFP` and `BitRound`, though with much larger still-safe errors, which result in especially high spectral errors. Therefore, it would be worth investigating a spectral error safeguard in the future, e.g. based on Ren et al. [2026].

In this work, we have presented the Compression Safeguards Framework, a user-centred approach that empowers users to control which safety invariants of the data they need to be preserved by lossy compression. The Compression Safeguards Framework extends existing ~~predict~~ approximate-then-correct compression approaches by going beyond lossless outlier corrections and safety-requirement-specific correction formats. The Safeguards work with any compressor and data predictor, are portable across platforms and implementations and data formats, and provide novel flexibility in the safety requirements that users can preserve and how they can be composed. Since the Safeguards define portable corrections with respect to the binary representation of the data, they are generally applicable for any kind of data from any scientific domain. The Compression Safeguards are designed to act as a final layer of defence against potentially unsafe compression, to build trust in lossy compression, and to unlock the benefits of lossy compression for the entire computational science community.

The Compression Safeguards are not meant to replace or discourage individual compressors from implementing measures for safe lossy compression, such as support for missing values or quantities of interest. Indeed, many compressors may be able to integrate specific safety requirements like spectral errors [Ren et al., 2026] more efficiently and produce higher compression ratios.

Comment 6

Typos etc.

- 375: “byte-aligned symbols that (are) well-suited”
- 628: “better compression that tuning” -> “than”
- 655: Possible math typo (tends to 1, I think)
- 768: “would would not be competitive”
- 786: “shown in Figure 7)” Closing paren w/ no open
- The DOI 10.24381/cds.92db7fef is in the bib thrice
- Reference on line 1801 seems to have first and last names swapped

Response: Thank you for the comment.

We have corrected the typos in lines 375, 628, 768, 786, and 1801 as suggested.

The DOI 10.24381/cds.92db7fef was indeed repeated too many times. One of the three was a duplicate, but the other references are distinct, even though they use the same DOI, which is recommended by the Climate Data Store’s citation and attribution guideline at this DOI’s web page.

As reviewer 2 also noted, the explanation in line 655 was incorrect. 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. Once again, overall good work. I am definitely open to be persuaded on pretty much all of the above.

Response: Thank you again for all of your valuable feedback.

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.

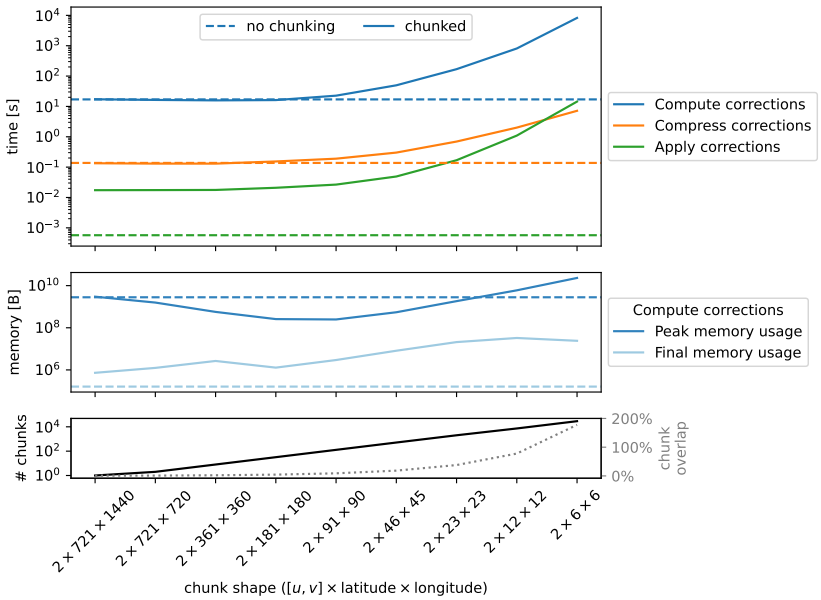


Figure B17: Scaling of single-threaded wall-clock runtime, memory footprint, and chunk overlap with the `xarray-safeguards` frontend when safeguarding the relative vorticity quantity of interest (see section 3.6) across an increasing number of chunks. The upper plot shows the wall-clock time, the middle 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, please refer to change 1 for the correction.

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

Robert Underwood, Victoriana Malvoso, Jon C. Calhoun, Sheng Di, and Franck Cappello. Productive and performant generic lossy data compression with libpressio. In *2021 7th International Workshop on Data Analysis and Reduction for Big Scientific Data (DRBSD-7)*, pages 1–10. IEEE, 2021. doi: 10.1109/DRBSD754563.2021.00005.

P Lindstrom. Error distributions of lossy floating-point compressors. In *2017 Joint Statistical Meetings*, 10 2017. URL `https://www.osti.gov/biblio/1526183`.

T. Reichelt, J. Tyree, M. Klöwer, P. Dueben, B. N. Lawrence, A. H. Baker, S. Faghih-Naini, T. Hoefler, and P. Stier. Climatebenchpress (v1.0): a benchmark for lossy compression of climate data. *Geoscientific Model Development*, 19(13):5933–5960, 2026. doi: 10.5194/gmd-19-5933-2026.

Robert Underwood, Jon C. Calhoun, Sheng Di, Amy Apon, and Franck Cappello. Optzconfig: Efficient parallel optimization of lossy compression configuration. *IEEE Transactions on Parallel and Distributed Systems*, 33(12):3505–3519, 2022. doi: 10.1109/TPDS.2022.3154096.

Kai Zhao, Sheng Di, Maxim Dmitriev, Thierry-Laurent D. Tonellot, Zizhong Chen, and Franck Cappello. Optimizing error-bounded lossy compression for scientific data by dynamic spline interpolation. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 1643–1654, 2021. doi: 10.1109/ICDE51399.2021.00145.

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.

Sheng Di and Franck Cappello. Fast error-bounded lossy hpc data compression with sz. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 730–739, 2016. doi: 10.1109/IPDPS.2016.11.

Shaomeng Li, Peter Lindstrom, and John Clyne. Lossy scientific data compression with

sperr. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1007–1017, 2023. doi: 10.1109/IPDPS54959.2023.00104.

Jinyang Liu, Pu Jiao, Kai Zhao, Xin Liang, Sheng Di, and Franck Cappello. Qpet: A versatile and portable quantity-of-interest-preservation framework for error-bounded lossy compression. *Proc. VLDB Endow.*, 18(8):2440–2453, 9 2025. ISSN 2150-8097. doi: 10.14778/3742728.3742739.

Alex Fallin and Martin Burtscher. Lessons learned on the path to guaranteeing the error bound in lossy quantizers. *arXiv*, 2024. doi: 10.48550/ARXIV.2407.15037.

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.

Yuxiao Li, Mingze Xia, Xin Liang, Bei Wang, Robert Underwood, Sheng Di, Hemant Sharma, Dishant Beniwal, Franck Cappello, and Hanqi Guo. pmsz: A distributed parallel algorithm for correcting extrema and morse smale segmentations in lossy compression, 2026.

Alex Fallin, Nathaniel Gorski, Tripti Agarwal, Bei Wang, Ganesh Gopalakrishnan, and Martin Burtscher. Fast topology-aware lossy data compression with full preservation of critical points and local order, 2026.

Langwen Huang, Luigi Fusco, Florian Scheidl, Jan Zibell, Michael Armand Sprenger, Sebastian Schemm, and Torsten Hoefler. Error bounded compression for weather and climate applications. *arXiv*, 2025. doi: 10.48550/arXiv.2510.22265.

Pu Jiao, Sheng Di, Hanqi Guo, Kai Zhao, Jiannan Tian, Dingwen Tao, Xin Liang, and Franck Cappello. Toward quantity-of-interest preserving lossy compression for scientific data. *Proc. VLDB Endow.*, 16(4):697–710, 12 2022. ISSN 2150-8097. doi: 10.14778/3574245.3574255.

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.

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.