Compression Safeguards: Building Trust into Lossy Data Compression

Juniper Tyree[1], Robert Underwood[2,3], Clément Bouvier[1], Daniel Köhler[1], Tim Reichelt[4], Peter Dueben[5], Sara Faghih-Naini[5], Heikki Järvinen[1], and Milan Klöwer[4]

[1]INAR, University of Helsinki, Helsinki, Finland
[2]Argonne National Laboratory, Lemont, IL, United States of America
[3]University of Chicago, Chicago, IL, United States of America
[4]University of Oxford, Oxford, United Kingdom
[5]European Centre for Medium-Range Weather Forecasts, Bonn, Germany

Correspondence: Juniper Tyree (juniper.tyree@helsinki.fi)

Abstract. The growth in data volumes produced by scientific models is accelerating. Data production of high-resolution weather and climate models is outpacing the methods and budgets for storing, sharing, and analysing this data, posing a threat to scientific progress. Lossy data compression greatly reduces data sizes but loses some quality, detail, or precision of the original data. Even though some lossy compressors promise size reductions of $100\times$ or more, the lack of trust in lossy compression, rooted in the risk of losing important information, has thus far limited their adoption. We introduce Compression *Safeguards*, a user-centric and domain-independent framework to overcome this trust gap, with which (i) scientists declare their precise safety requirements for what lossy compression must preserve, e.g. regionally varying error bounds on quantities derived from the decompressed data, then (ii) wrap a compressor of their choice in the corresponding safeguards, which then (iii) *guarantee* that the user-declared safety requirements are *always* met by the safeguarded compressor. The Compression Safeguards represent a paradigm shift: Data producers and users no longer carry the risks of lossy compression, having to manually check for problems after compression, but instead control up-front what needs to be safeguarded. 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. Users thus no longer need to re-verify each new compressor for each new use case, or to restrict themselves to few safe compressors and supported use cases. 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 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. Safeguards can unlock the data reduction benefits of lossy compression and therefore solve many data storage problems across the scientific community that otherwise hinder research.

1 Introduction

The data volumes produced by scientific simulations are increasing rapidly (ECMWF, 2019) as computational resources have grown with industrial, societal, and scientific demand. For example, weather and climate models have grown in resolution and scope (Stevens et al., 2019; Rackow et al., 2025; Segura et al., 2025; Hadade et al., 2025). However, methodological improvements and stagnating budgets for data storage, access, sharing, and analysis have not kept up with the typical exponential data growth (Baker et al., 2014; Kunkel et al., 2014; Hart, 2017; Peters-von Gehlen et al., 2025), especially as storage costs have increased. As the catastrophic impacts of extreme weather and climate are becoming increasingly apparent (Intergovernmental Panel on Climate Change (IPCC), 2023), the societal demand for better and higher-resolution models, which produce even more data, will grow further. Tackling this data volume challenge by reducing data storage requirements, transfer, and processing costs at every step is thus an important task in supporting and unblocking research and operations in weather, climate, and many other fields.

Data compression methods reduce the space required to store data. Lossless compression methods reproduce the bitwise equivalent data during decompression, but no longer provide sufficiently high compression factors to handle the growth in scientific data (Baker et al., 2014, 2016). Lossy data reduction can produce higher compression factors by only reproducing an approximation, where some information from the original data has been lost. Lossy compressors can be based on methods such as rounding, sampling, dimensionality reduction, model fitting, and more. Despite continuing advances in lossy data compression that promise compression ratios of $100\times$ (e.g., Liu et al., 2024; Huang et al., 2025; Han et al., 2026) or even $1000\times$ (e.g., Huang and Hoefler, 2023; Mirowski et al., 2024; Liu et al., 2024), lossy compression has not been widely taken up by the weather and climate modelling community (Hammerling et al., 2019; Baker et al., 2016; Tintó Prims et al., 2024) or generally across the computational sciences. Most importantly, the fear of losing information that may be relevant in a later analysis or application of the data has been blocking its adoption (Baker et al., 2014). For instance, there is concern that lossy compression may systematically distort the data, e.g. by smoothing out extreme weather events or violating the conservation of some derived quantities (Baker et al., 2016). The failure to adopt lossy compression for model outputs because of these concerns and the broader lack of trust is already hindering crucial scientific research: As scientific computing is pushing against storage limits, e.g. with high-resolution Earth system modelling projects such as the Destination Earth Climate Change Adaptation Digital Twin (Hadade et al., 2025), model outputs instead have to be quickly downscaled or discarded (Klocke et al., 2025; Cappello et al., 2025).

Many lossy compressors are error-bounded and allow limiting the error introduced by lossy compression. While most scientific compressors support bounding the pointwise error (e.g., Lindstrom, 2014; Delaunay et al., 2019; Liang et al., 2023; Huang et al., 2025; Li et al., 2023; Fallin and Burtscher, 2024; Liang et al., 2022), either directly or by first approximating and then correcting errors through encoding residuals or outliers, others are designed for reducing global errors (e.g., Christopoulos et al., 2000; Seitola et al., 2014; Li et al., 2023; Ballester-Ripoll et al., 2020; Liang et al., 2022). There have also been several efforts to establish and discuss criteria for which levels of compression error can be tolerated without losing important information (e.g.,

Klöwer et al., 2021; Baker et al., 2024, 2017; Cappello et al., 2025; Baker et al., 2014, 2016; Tintó Prims et al., 2024; Reichelt et al., 2026; Poppick et al., 2020; Baker, 2021; Huang et al., 2023; Sather et al., 2022; Pinard et al., 2021b, a). However, the large variety in Earth system data and their downstream applications creates complex and diverse safety requirements that often go beyond unified, simple error bounds (Cappello et al., 2025; Huang et al., 2023).

For some safety requirements, e.g. preserving topology, several new specialised compressors (e.g., Soler et al., 2018; Yan et al., 2024; Agarwal et al., 2026; Fallin et al., 2026; Li et al., 2026) and some compressor-independent adaptors (e.g., Gorski et al., 2025) have been proposed. Even though such specialised approaches can provide high efficiency, they need significant effort from compression developers, do not cover all use cases, and often do not compose well to preserve several requirements, while still asking users to find and trust one specific compressor implementation. There have also been efforts to preserve arbitrary requirements across different compressors: The state-of-the-art `QPET` framework can preserve quantities $f(x)$ derived from the data x (Liu et al., 2025) but requires tight integration with compressors, i.e. cannot be used with compressors for which specific software support has not yet been developed. The meta-compressor `OptZConfig`, in contrast, can wrap around any compressor to tune its configuration to uphold any safety requirement, but would need to fall back to lossless compression if no suitable configuration can be found (Underwood et al., 2022), e.g. because the compressor does not preserve missing values.

Altogether, there is currently no user-centric mechanism that (i) allows scientific users to formally specify their diverse safety requirements, (ii) ensures that all of these requirements are consistently enforced simultaneously (Cappello et al., 2025), (iii) works with any compressor without requiring tight software integration, all (iv) without falling back to globally-lossless compression. In practice, users themselves have to take on the risks of lossy compression. Data producers and users are often advised to always manually verify that lossy compression has not violated important properties of their data (e.g., Reichelt et al., 2026). This approach is time-consuming, consequently only applied to data samples in practice and therefore error-prone, particularly around edge cases, and does not foster confidence in the reliability of lossy compression. In particular, trust is typically only gained with one compressor and in one configuration – changing either and the user is required to repeat the verification. Without a reliable mechanism to automatically uphold safety requirements across different compressors and datasets, trust in lossy compression naturally remains limited within the scientific community, which hinders its adoption.

To address this trust gap, we introduce Compression *Safeguards* – a user-focused framework designed to enforce user-defined safety requirements and build trust in safe lossy compression. Instead of proposing yet another compressor or requiring users to learn about, check, and trust each new compressor, the Compression Safeguards are fully general: They require no integration and thus work with any current and future compressor, with any type of data, in any scientific domain, allowing users to configure their Safeguards once and then benefit from improvements in future compressors without needing to configure or check again. 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: they guarantee 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. Even compressors that are not error bounded can be used safely with Safeguards – an especially important application as increasingly many modern lossy compression methods are based on neural networks (e.g., Huang and Hoefler, 2023; Mirowski et al., 2024; Han et al., 2026), which are inherently more difficult to validate and trust. We see Compression Safeguards as a necessary paradigm shift to address the understandable lack of trust in lossy compression that exists as a psychological problem throughout the computational science community.

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 the user's safety requirements when necessary. We imagine Safeguards to become a new default component in modular lossy compression pipelines (e.g., Liang et al., 2023). These pipelines may consist of the following components: preprocessing using meta-compressors, tuning error bounds (e.g., Liu et al., 2025), lossy approximation (e.g., Delaunay et al., 2019; Liang et al., 2023; Li et al., 2023; Lindstrom, 2014), safeguarding (ours), lossless compression of the approximation and any corrections, and full-stack optimisation of this pipeline (e.g., Underwood et al., 2022). With such a modular design, the responsibility for highly efficient and safe lossy compression is divided: The responsibility of the compressor is to provide high compression ratios, even if safety requirements are sometimes violated. The responsibility of the Safeguards is to guarantee arbitrary safety requirements, ideally only having to produce corrections rarely. Duplicate implementation work is thereby reduced across the ecosystem, and users are empowered to switch out individual modular components depending on availability or other data production constraints.

To enable the adoption of safeguarded lossy compression, we provide the `compression-safeguards` reference implementation, an open-source cross-platform Python package, which supports composing different safety requirements, works with any compressor, supports novel non-pointwise safety requirements, and produces corrections that do not simply revert to lossless compression. Our implementation works best with gridded integer or floating-point data but can also be repurposed for unstructured and categorical data, e.g. integer-encoded time series data with missing values. We have also created two user-facing frontends, i.e. easy-to-use interfaces for the `compression-safeguards` for specific use cases: `numcodecs-safeguards` and `xarray-safeguards` integrate with the popular `numcodecs` compression application programming interface (API) (Zarr developers, 2025) and the `xarray` ecosystem for labelled multidimensional and chunked data (Hoyer and Hamman, 2017), respectively. All packages are published to the Python Package Index (PyPI).

In this manuscript, we present the Compression Safeguards for (a) scientists who would work with compressed data and require confidence that lossy compression does not hinder their research, (b) producers of scientific data who would apply Safeguards when compressing the data, and (c) developers of compression algorithms. Since not all sections may be relevant to every reader, we mark which are focused on **data-users [U]**, **data-producers [P]**, and **technical details [T]**. In particular, we

- introduce the Compression Safeguards Framework in **section 2 [U, P, T]**,
- present the `compression-safeguards` reference implementation in **section 3 [T]** and its data-producer-facing frontends in **sections 3.5 and 3.6 [P]**,
- compare the `compression-safeguards` with prior work in **section 4 [P, T]**,
- evaluate safeguarding different lossy compressors for several real-world, primarily weather and climate use cases in **section 5 and appendix B [U, P]**, including their (de)compression speed in **sections 5.7 and 5.8 [P, T]**,
- discuss the results of the evaluation in **section 6 [U, P, T]**, and
- conclude in **section 7 [U, P, T]** that the Compression Safeguards are a key modular component towards safe, trusted, and fearless lossy compression, while also outlining where there is need for future work.

The `compression-safeguards` reference implementation (Tyree et al., 2026c) has a continuously updated online documentation (Tyree et al., 2026a) with additional examples and analyses.

2 The Compression Safeguards Framework

Compression Safeguards enable users to declare their safety requirements for lossy compression and guarantee that they will be upheld. We distinguish between the abstract concept of the Safeguards, which we refer to as the *Compression Safeguards Framework*, and specific implementations of this framework. While this section introduces the abstract framework in detail, section 3 presents our `compression-safeguards` reference implementation and its user-facing frontends.

The fundamental idea of the Compression Safeguards Framework is (i) to automate the process of checking user safety requirements and (ii) to produce portable pointwise binary corrections that are applied to the lossy-decompressed data to re-establish any violated requirements. To verify and guarantee that the produced corrections are safe, the corrections are automatically re-checked. So long as the checks correctly represent the user's intended requirements, trusting the Compression Safeguards Framework therefore does not require trust in the compressor or how corrections are produced.

The Compression Safeguards Framework is designed to be implementation-independent and work with any digital data format in any domain. It supports floating-point arrays of regularly spaced dimensions as well as integer or categorical data on unstructured grids or dimensions. While lossy compressors typically rely on ordinal data to formulate an approximation, this is not a requirement for the Safeguards framework per se. The framework only specifies the interoperable format of the pointwise binary corrections and how to apply them. This limited specification allows different implementations of the Compression

Safeguards to co-exist and explore different use cases, strategies, and trade-offs for producing corrections, while also ensuring that users can read safeguarded data produced by any implementation. Having an interoperable format additionally opens up a future where popular data formats like HDF5 (The HDF Group, 2025) or Zarr (Miles et al., 2025) could support applying such corrections transparently at decompression time. Data-users could then read safeguarded lossy compressed data, e.g. produced by weather forecasting centres using different implementations of the Safeguards, without needing to use custom software or interact with the Safeguards directly.

2.1 What is a safeguard?

Each safeguard represents a user-provided safety requirement. It combines

1. a declarative specification of the safety requirement,
2. a procedure to check if this requirement is met for some decompressed approximation $\hat{x}$ relative to the original data x, and
3. a safeguard- and implementation-specific procedure to compute the corrected data $\tilde{x}$ that fulfils the requirement.

A binary correction $\tilde{b}$ is derived from this corrected data $\tilde{x}$. This correction $\tilde{b}$ can later be applied to the approximation $\hat{x}$ to correct it and reproduce the corrected data $\tilde{x}$. Importantly, while the approximation may not meet the safety requirements, the corrected data is guaranteed to satisfy them.

Safety requirements of arbitrary complexity can be expressed as safeguards. For instance, a simple pointwise absolute error bound safeguard over finite data can be specified as $|\hat{x}_i - x_i| \leq \epsilon_{abs}$ for all indices i.

2.2 Meta-compression with safeguards

Meta-compressors are compressor adaptors that wrap one or more internal compressors, allowing their functionality to be extended or combined (Alted, 2010; Underwood et al., 2021). 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.

An overview of the meta-compression process is shown in Figure 1. At compression time, a safeguarded compressor requires additional computation: the data x is first compressed, then decompressed to produce the approximation $\hat{x}$, corrected if necessary, and finally checked again to validate the corrections. The corrections need to be already computed during compression when the original data is still known. At decompression time, the corrections can be cheaply applied without any knowledge of how they were computed.

The Compression Safeguards Framework does not specify whether or how the compressed data and the corrections should be stored together. This flexibility allows the framework to be adopted both ad hoc, when compressing new data with Safeguards,

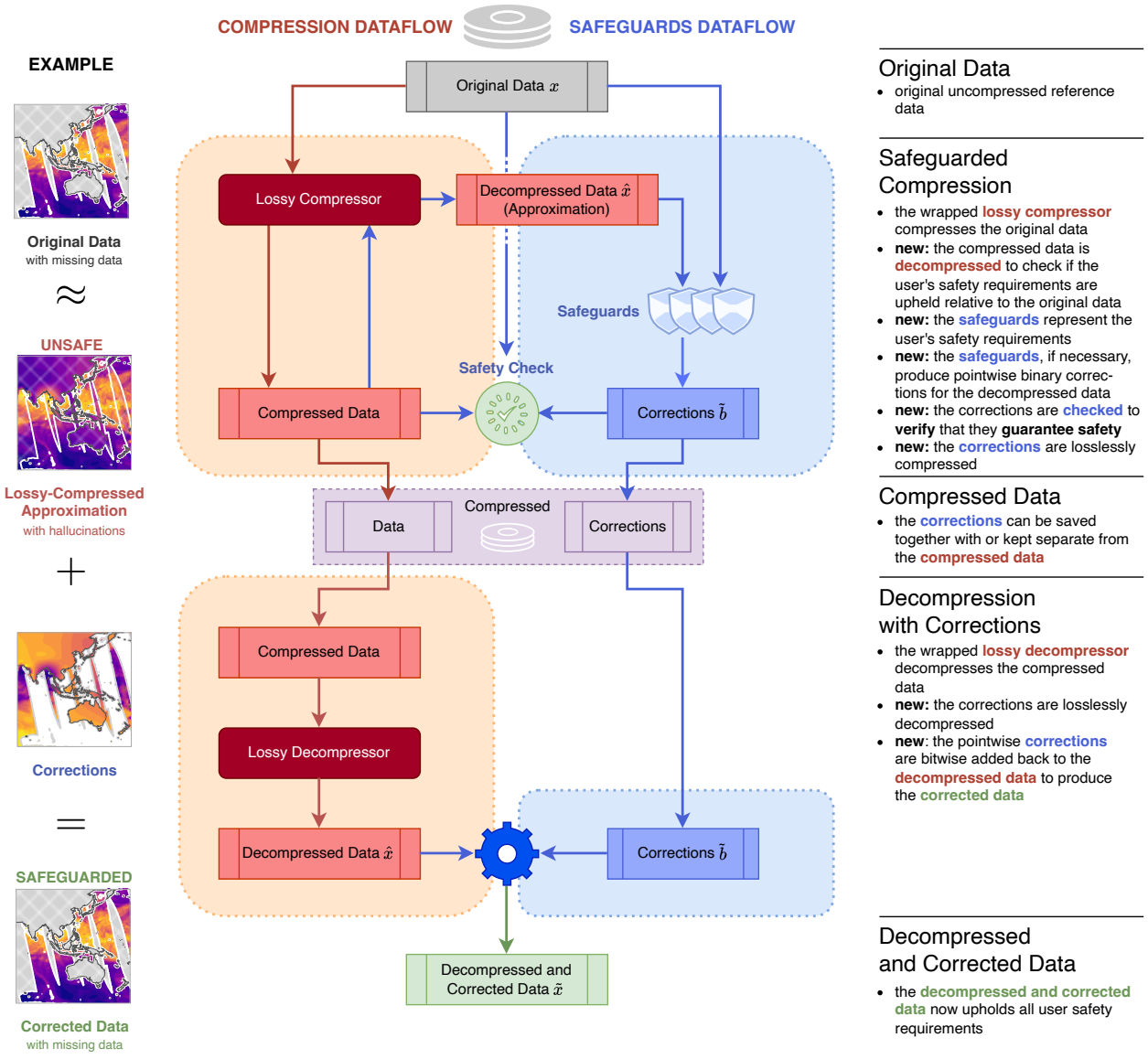


Figure 1. Overview of the dataflow in a safeguarded compressor, which guarantees that the user's safety requirements are upheld by an arbitrary compressor. During compression, the corrections for the decompressed data are produced and validated, and then later applied during decompression to correct the decompressed data. In the provided example on the left, the compressor hallucinates values for missing data, and the Safeguards remove these hallucinations.

or post hoc, when computing corrections for data that had been compressed before. Post hoc safeguarding is possible as long as the original uncompressed data can still be accessed while computing the corrections. 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. Furthermore, separate storage would also allow the Safeguards to be integrated with Magri and Lindstrom (2024)'s progressive decompression framework, which allows users to choose data accuracy during decompression.

2.3 Implementation-agnostic and hardware-portable corrections

This technical section defines the interoperable binary format of the corrections. Given the approximation $\hat{x}$ and corrected data $\tilde{x}$, which may be integer or floating-point, the corrections $\tilde{b}$ are stored in the following binary format, indexed with i:

$$\tilde{b}_i = \text{as_bits}(\tilde{x}_i) - \text{as_bits}(\hat{x}_i) \tag{1}$$

where the `as_bits` function converts from the data's original type to its binary unsigned integer representation, the binary difference is computed using wrapping unsigned integer subtraction. Importantly, $\tilde{b}$ is pointwise for each element in $\hat{x}$ and has the same n-dimensional shape as the original data x.

The binary correction can later be applied to the approximation to reproduce the corrected data, which only requires a pointwise wrapping unsigned integer addition per data point:

$$\tilde{x}_i = \text{from_bits}(\text{as_bits}(\hat{x}_i) + \tilde{b}_i) \tag{2}$$

where the `from_bits` function converts the binary representation back to the data's original type. Since the Compression Safeguards define their corrections with respect to the binary representation of the data, they can be applied to any data format, from gridded floating-point to unstructured categorical, and can correct for any type of safety requirement, whether local, regional, or global. Furthermore, this pointwise fixed-length binary format also naturally supports random-access.

Why is the correction encoded as a binary difference? For integer data, the binary difference is bitwise equivalent to the integer difference. For floating-point data, conversely, the binary difference is generally less concise since values with the same floating-point difference may have different binary differences, which increases the entropy of the corrections, thus decreasing their lossless compressibility. However, the difference between any two floating-point numbers cannot always be represented with the same precision, e.g. while $(a - b) + b = a$ mathematically, in 16-bit IEEE 754-2008 floating-point arithmetic (IEEE, 2008), $(0.00000006 - 0.0001222) + 0.0001222 = 0.0000001 \neq 0.00000006$, which could cause safety requirements to be violated (Fallin and Burtscher, 2024). While using two floating-point additions, with special rules for propagating non-finite values, would be able to represent all differences, it would also double the correction's size footprint. Furthermore, some processors may flush subnormal floating-point values to zero and could thus cause two machines to apply the corrections

differently. While linear quantisation provides an efficient encoding for absolute error bound corrections (e.g., Di and Cappello, 2016; Li et al., 2023), it cannot easily be extended to more complex safety requirements without needing to also store a per-point-varying absolute error bound (Liu et al., 2025). Using binary differences sidesteps all of these problems, prioritising hardware- and implementation-portable correctness, both in storage and when applying corrections, over a more compressible corrections format.

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.

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.

2.4 The printer problem in safeguarding lossy compression

The Compression Safeguards Framework needs to know about all safety requirements that should be upheld. For example, if the data is first safeguarded with an absolute error bound, and then later this safeguards-corrected data is safeguarded again with a relative error bound, the resulting data may violate the original absolute error bound. Even applying the same safeguard twice in a row can violate the guarantees. This is also known as the printer problem or generation loss (Reich and Heathcote, 2022): Every time a document is copied from a previously copied and printed document, new artefacts are added, and errors may accumulate over time. To avoid the printer problem, the Compression Safeguards must only be used on the original, uncompressed, and unsafeguarded reference data, and they must apply all safety requirements at once. When the safety requirements need to change after safeguarding, the Compression Safeguards must be applied again to the original and decompressed data to produce the new corrections. In the future, standardised metadata for lossy-compressed data could help with warning about such compression error accumulation. A prototype for guarding against the printer problem is implemented in

the `xarray-safeguards` frontend (see section 3.6). Additionally, future implementations of the Compression Safeguards framework could provide the additional guarantee that the accumulated error is bounded, as with rounding, linear quantisation, or `ZFP` (Fox et al., 2020).

3 The `compression-safeguards` Reference Implementation

We provide a reference implementation of the Compression Safeguards Framework, the `compression-safeguards` Python package. Its design was guided by our goal of providing a cross-platform portable implementation that allows different safety requirements to be combined arbitrarily while keeping the runtime cost predictable and limited.

The `compression-safeguards` package is published open-source under the Mozilla Public License 2.0 on GitHub (Tyree et al., 2026c) and PyPI (Tyree et al., 2026f). To be portable, the `compression-safeguards` are written in pure Python and only require six dependencies from the Python Package Index, of which only two need platform-specific wheels: `numpy`, (Harris et al., 2020) Python's fundamental package for scientific computing over n-dimensional arrays, which is vectorised to achieve data parallelism, and `numpy-quaddtype` for IEEE 754 128-bit quadruple precision floating point calculations. Future work could explore using the Python array API standard (Meurer et al., 2023) to support non-`numpy` array computation backends such as `CuPy` (Okuta et al., 2017) for CUDA-accelerated computations on NVIDIA GPUs.

We have also published two frontend packages to PyPI. The `numcodecs-safeguards` package (Tyree et al., 2026h) implements a meta-compressor for the `numcodecs` (Zarr developers, 2025) compression framework. The `xarray-safeguards` frontend (Tyree et al., 2026i) provides support for the `xarray` (Hoyer and Hamman, 2017) ecosystem for labelled multidimensional and chunked data, which enables parallelising the computation and application of corrections across multiple processors and machines. The documentation for the reference implementation and its frontends are published in Tyree et al. (2026a).

3.1 Composing multiple safeguards using safe intervals

Instead of providing specialised safeguards for each specific use case, the reference implementation provides basic building block safeguards, which can be composed using logical combinators to represent the user's diverse safety requirements. When computing the corrections for such a composed safeguard, care needs to be taken to ensure that the corrections from different sub-safeguards do not conflict with and invalidate each other. Since the number of possible combinations increases quickly when preserving multiple safety requirements, it is infeasible to support each combination with a custom correction algorithm. While fixed-point iteration could be used to find corrections that satisfy all requirements, the runtime cost of such an iterative approach would not be predictable.

Instead, the `compression-safeguards` implementation produces corrections in a chunking-friendly, communication-free, and iteration-free one-shot algorithm that uses safe intervals to represent and compose corrections from different safeguards. A safe per-point interval, $l_i \leq \tilde{x}_i \leq h_i$, specifies a continuous range of data values that all satisfy the safety requirements, from an inclusive lower bound l up to an inclusive upper bound u. In our reference implementation, each safeguard produces a union of safe intervals for each data point. These unions are more expressive than absolute error bounds and can

represent exact, asymmetric, and disjoint value ranges, which allows producing better corrections. Since the original data x always fulfils all requirements, the union of safe intervals must always contain x. Multiple safeguards can be combined, without requiring explicit collaboration, by intersecting, unioning, or selecting between their unions of safe intervals. After the safe intervals from all safeguards have been combined via intersection, the corrected value is chosen from each data point's union of disjoint safe intervals. To reduce the entropy of the resulting binary corrections, the `compression-safeguards` implementation tries to choose fewer unique corrections with a HyperLogLog-inspired heuristic (Flajolet et al., 2007) that maximises the number of leading and trailing zero bits in the total order binary difference between the approximation and corrected value.

3.2 Preserving pointwise and stencil safety requirements

The `compression-safeguards` implementation can preserve a variety of safety requirements. Pointwise requirements are checked independently for each data point and can directly be translated into pointwise safe intervals. For example, a pointwise maximum absolute error bound $|x_i - \hat{x}_i| \leq \epsilon_{abs}$ can be transformed into the equivalent safe interval $(x_i - \epsilon_{abs}) \leq \hat{x}_i \leq (x_i + \epsilon_{abs})$. The `compression-safeguards` implementation currently support the following pointwise safeguards that bound the error over the data x for all indices i:

1. maximum absolute error: $(x_i - \epsilon_{abs}) \leq \hat{x}_i \leq (x_i + \epsilon_{abs})$ for $\epsilon_{abs} \geq 0$.
2. maximum relative error: $(x_i - \epsilon_{rel} \cdot |x_i|) \leq \hat{x}_i \leq (x_i + \epsilon_{rel} \cdot |x_i|)$ for $\epsilon_{rel} \geq 0$.
3. maximum ratio error: $\min(x_i \,/\, \epsilon_{ratio}, x_i \cdot \epsilon_{ratio}) \leq \hat{x}_i \leq \max(x_i \cdot \epsilon_{ratio}, x_i \,/\, \epsilon_{ratio})$ for $\epsilon_{ratio} \geq 1$.

   The ratio error bound is sometimes also known as a decimal error bound, where $\epsilon_{decimal} = \log_{10}(\epsilon_{ratio})$.

The error over a pointwise derived quantity of interest (QoI) $f(x)$ can similarly be bounded (see section 3.4). Additionally, the `compression-safeguards` implementation also has safeguards to preserve a specific value c inclusively, $(x_i = c) \rightarrow (\hat{x}_i = c)$, or exclusively, $(x_i = c) \leftrightarrow (\hat{x}_i = c)$, and to preserve the sign of the data, optionally relative to a zero-by-default isovalue.

Even for these simple safeguards, special care must be taken to handle edge cases correctly, e.g. floating-point rounding errors, non-finite values, and wrapping integer arithmetic (Fallin and Burtscher, 2024). The error-bounding safeguards always preserve infinite values with the same bit pattern, and can preserve NaN values with the same or with an arbitrary payload.

Some safety requirements are defined as a function over a small local neighbourhood around each data point in a regular grid. This neighbourhood is called a stencil, and it can be thought of as a window that moves over the data, e.g. to compute the data derivative using finite differences. Whereas each point is independent in pointwise safeguards, stencils may overlap, and so in stencil safeguards the points and their corrections are interdependent. Furthermore, the data's boundary conditions have to be taken into account to ensure that the requirements are correctly preserved, e.g. along a periodic axis. The `compression-`

`safeguards` implementation provides stencil safeguards to preserve error bounds over user-defined derived quantities of interest $f(\boldsymbol{x})$ over local neighbourhoods.

Producing pointwise-independent safe intervals for stencil safeguards results in suboptimal corrections, since satisfying a safety requirement without knowing how the neighbouring points will be corrected requires conservatively restricting all corrections. For instance, an affine transformation such as $\hat{x} = 3x + 1$ preserves the monotonicity of the input but this cannot be represented by safe intervals. An implementation of the Compression Safeguards Framework that uses different strategies, such as incremental (e.g., Jiao et al., 2022) or iterative corrections, may be able to produce less pessimistic corrections and higher compression ratios. We explore iterative and incremental corrections in section 5.7 and appendix B9, respectively.

3.3 Logical and regional compositions of safeguards

Users may have regionally varying safety requirements that are stricter in areas with important domain-specific features, e.g., around local extrema or critical points. In the `compression-safeguards` implementation, safeguards can be made regionally-varying using late-bound constant parameters. When declaring their safety requirements during the one-time data-independent safeguard configuration, users only need to provide the name of a late-bound parameter but not yet its value. The values of all late-bound parameters only have to be provided when later computing the corrections for some specific data, allowing these values to be data-dependent, e.g. the data's coordinates or a mask. These late-bound parameter values are treated as constant.

The error-bounding safeguards can be configured with a late-bound error bound array, such that the error bound can vary arbitrarily per-point across the data domain. Furthermore, the value-preserving and sign-preserving safeguards allow their sentinel / isovalue to be late-bound. Late-bound parameters can also be used in a logical select combinator, which takes a late-bound index array to select, per point, which of a collection of combined-over safeguards must fulfil its requirements for this point. The select combinator can be used to represent a discrete set of regions of interest.

The `compression-safeguards` implementation also provides an "any" combinator for logical disjunctions over multiple safeguards, an "all" combinator for logical conjunctions, and an assumed-safe safeguard. Intuitively, for each data point, at least one safeguard in the any combinator must fulfil its requirements, whereas all safeguards must fulfil their requirements in the all combinator. While the "all" combinator directly maps to the intersection of safe intervals, the "any" combinator over non-pointwise safeguards is not equivalent to the pointwise union. For the assumed-safe combinator, every data point is assumed to be safe, which can be used together with the select combinator to apply some safeguard only in particular regions of interest.

The logical any, all, and select combinators can combine over both pointwise and stencil safeguards, e.g. to preserve isosurfaces and gradients inside a hurricane. They are themselves pointwise if they only combine over pointwise safeguards. When the combinators combine over stencil safeguards, the influence of every point is explicitly tracked since a failure of the per-point check may be related to any data point in the stencil, and equivalently, one data point correction can impact all neighbouring points in whose stencils it is.

3.4 Safeguards for derived quantities of interest

Derived quantities of interest (QoIs) refer to domain- and use-case-specific functions $f(x)$ that are computed over the data x. The `compression-safeguards` implementation can preserve pointwise absolute, relative, and ratio error bounds over both pointwise and stencil QoIs. The error bounds are first translated into safe intervals over the evaluated QoI. This user-given quantity of interest, which is written in Python-like syntax, is then traversed as a symbolic expression tree to propagate the safe intervals. At each expression node, which represents a function $g(a)$, the inverse function g^{-1} is applied to determine the safe intervals over a. For functions where no unique inverse exists, e.g. the periodic $\sin(a)$, the piecewise inverse is used instead. While periodic functions may produce a union of disjoint safe intervals, only one branch is currently followed to bound the runtime cost. For functions with several data-dependent parameters, e.g. a weighted sum, the flexibility in the safe intervals is distributed conservatively amongst all data-dependent parameters such that all possible combinations remain safe.

The quantities of interest are evaluated using `numpy` in a configurable floating-point data type that provides sufficient precision to losslessly represent all values of the data x, e.g. at least `float64` for `int32` data. Special care is taken to handle all edge cases correctly, including NaN, infinity, and -0.0. At each expression node, the propagated safe intervals are checked for these edge cases and possible rounding errors, followed by the final check of the corrections, to guarantee that the produced corrections are indeed safe. Since `numpy` does not guarantee correctly-rounded floating point operations, future work could investigate quantity of interest evaluation that is cross-platform reproducible.

The `compression-safeguards` implementation supports many novel quantity of interest features that are listed and compared with prior work in Table 2 in section 4. For instance, they support forward / central / backwards finite differences of arbitrary order and with an arbitrary order of accuracy (Fornberg, 1988), over any axis on a regular grid, with a constant grid spacing or arbitrary constant gridpoint positions, and an optional grid period such as $360°$ longitude, even for chunked data. Furthermore, comparisons and logical combinators can be composed to preserve monotonic sequences or critical points and local order (Fallin et al., 2026) and to optionally limit requirements to dynamically identified regions of interest such as hurricanes. More broadly, different safeguards, regionally varying, with quantities of interest and late-bound constants, can be composed to describe and enforce many diverse safety requirements, or can often at least preserve a more conservative invariant, which makes the `compression-safeguards` more flexible and reusable than if it only supported specific safety requirements. The online documentation provides some general tips and tricks for composing these various features to translate safety requirements into safeguards (Tyree et al., 2026b).

3.5 The `numcodecs-safeguards` frontend

The `numcodecs` Python package defines a minimal interface for compressing n-dimensional arrays (Zarr developers, 2025). The `numcodecs-safeguards` frontend provides the `SafeguardedCodec`, a meta-compressor for the `numcodecs` API, which follows the data flow in Figure 1 and works with any deterministic `numcodecs`-compatible codec. It produces one combined byte stream that includes the compressed data, the losslessly compressed corrections, and a checksum of the

corrected data, which is used to guard against non-deterministic decompression that might invalidate the safety requirements. If no corrections are required, they are excluded from the output entirely.

Since the Compression Safeguards Framework does not prescribe how corrections should be losslessly compressed, the `SafeguardedCodec` provides a reasonable and overridable default: The corrections are first tokenised, then the most frequent correction is extracted into a bitmap, followed by a byte shuffle, and the resulting bytes are compressed with `Zstandard` (Collet, 2021). This pipeline is applied to the corrections or the 1D binary deltas over the corrections, depending on which produces better compression. In contrast to standard Huffman coding, these transformations are easily vectorisable and produce byte-aligned symbols that are well-suited for the byte-based `Zstandard` compressor.

The `SafeguardedCodec` extends the `compression-safeguards` by providing an experimental flag to iteratively reduce the one-shot corrections to improve compression ratios at the cost of extra compute time (see section 5.5 and appendices B6 and B7). In contrast to incremental corrections, which are briefly explored in appendix B9, this iterative process does not compute new corrections but instead checks if some data points can safely skip their corrections. The `SafeguardedCodec` can also produce lossless corrections that reproduce the original data exactly, which may also be iteratively reduced.

Compression in `numcodecs` operates on unlabelled data without explicit support for data chunking, and `numcodecs`-based tools thus typically compress each chunk independently. Since only pointwise safety requirements can be preserved independently across chunks, chunked data should instead be compressed with the `xarray-safeguards` frontend.

3.6 The `xarray-safeguards` frontend for parallelisation over chunked data

Large datasets often no longer fit into a computer's working memory and are instead split into chunks such that computations can be distributed in time and across multiple processors and machines. The `xarray-safeguards` frontend expands the `compression-safeguards` to support parallelisation over chunked data by integrating with the `xarray` ecosystem (Hoyer and Hamman, 2017) for labelled multidimensional and chunked data. Unlike the `SafeguardedCodec`, the `xarray-safeguards` can safely produce per-chunk corrections even for non-pointwise stencil safeguards.

To avoid the need for synchronisation between the chunks, preserving stencil safeguards across chunk boundaries requires read-only access to data and approximation elements from adjacent chunks. Computing chunked corrections is, however, non-trivial since each data point is not only affected by the data values inside its local neighbourhood, but also by the safety requirements that all points inside this neighbourhood have and impose back on it. Therefore, computing the corrections for a chunk requires access to the stencils of the boundary points as well as all the neighbourhoods of the points in these stencils, which are likely outside the original chunk. For example, when safeguarding a second-order finite central difference, which uses a stencil of size $[x-h; x+h]$, a double-stencil of size $[x-2h; x+2h]$ is needed. This is further complicated when multiple safeguards are combined, all of which may use different stencils with different boundary conditions.

In contrast to the all-in-one ad-hoc high-level meta-compression API of the `numcodecs-safeguards` package, the `xarray-safeguards` provide a lower-level functional API where corrections are produced and applied lazily and data coordinates are automatically made available as auxiliary parameters. Corrections also remain separate from the compressed approximation such that the user can freely choose how to losslessly compress corrections and where and how to store them,

e.g. in the same or a different dataset, using any data format of their choice. This makes the `xarray-safeguards` great for post hoc adoption of the Compression Safeguards: Corrections can be computed long after the original data was compressed – as long as the original data is still available – then distributed to users in separate files, which are then recombined at dataset-load time. Since the `xarray-safeguards` frontend works with labelled data arrays that support metadata, it also implements some simple protections against accidental occurrences of the printer problem, e.g. producing corrections for already-corrected data or without the original data (see section 2.4).

4 The `compression-safeguards` in Comparison with Prior Work

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 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 the characteristics of the `compression-safeguards` reference implementation 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

	QoI-SZ3	**QPET**	**MGARD-QoI/λ**
Method	Per-point absolute error bounds	Iteratively tuned per-point absolute error bounds, with per-point lossless fallback	Finite-element decomposition
Compressor	SZ3	SZ3, SPERR, ...	MGARD
-Integration	Coupled, pre-processing	Coupled, pre-processing	Coupled, pre-processing
Data Domain	Structured grid, pointwise-only unstructured	Structured grid, pointwise-only unstructured	**Structured and unstructured grid**
Error Norm	Pointwise absolute	Pointwise absolute, range-relative	**Arbitrary**
QoIs based on	Symbolic analysis	**Symbolic & numeric analysis**	Symbolic analysis
... with input	Multi-variable	Multi-variable	Single-variable
... with output	Pointwise, block mean	Pointwise, block mean	Pointwise, bounded
... with safety	Symbolic	**Numeric from fallback**	Symbolic
... across chunks	Only pointwise	Only pointwise	Only pointwise
Complexity	$O(\text{QoI}_N)$	$O(\text{QoI}_N) + O(\text{QoI}_k) \cdot it$	$O(\text{QoI}_N) \cdot N$

	OptZConfig	**Safeguards (ours)**
Method	Iterative compressor config black-box optimization	Per-point non-lossless corrections
Compressor	**Arbitrary*** (black box)	**Arbitrary** (black box)
-Integration	Meta-compressor, pre-processing	Meta-compressor, post-processing
Data Domain	Structured grid, **extensible***	Structured Grid, pointwise-only unstructured
Error Norm	From `libpressio`, **extensible***	Pointwise absolute, relative, ratio
QoIs based on	Black-box evaluation	**Symbolic and numeric analysis**
... with input / output	Stacked variables / Scalar global result	Stacked variables / Pointwise, stencil
... with safety	Not guaranteed without lossless fallback	**Numeric from corrections**
... across chunks	Only pointwise	**Pointwise and stencil**
Complexity	$O(\text{QoI}_N) \cdot it$	$O(\text{QoI}_N) \cdot O(\log O(\text{QoI}))$

Table 1. Comparison of the characteristics of the `compression-safeguards` reference implementation and quantity-of-interest-aware error-bounded lossy compressors. The complexity is expressed in terms of evaluating a quantity of interest QoI over N elements in $O(\text{QoI}_N)$, while evaluation over a subset k for a number of iterations it takes $O(\text{QoI}_k) \cdot it$. The QoI's expression tree depth is denoted $O(\log O(\text{QoI}))$. $(*)$: `OptZConfig` can be manually extended to support arbitrary compressors, data domains, error norms, and quantities of interest.

	QoI-SZ3	QPET	MGARD-QoI/λ	OptZConfig	Safeguards (ours)	Evaluation
Non-finite QoIs	✗	✓	✓	✓*	✓	
Polynomials	✓	✓	✓	✓*	✓	5.5, 5.6, B6 & B8
Symbolic integers	✗	✗	✗	✓*	✓	
Exp and Log	only log	✓	✗	✓*	✓	B4
Abs and Sign	✗	✓	✗	✓*	✓	
Trigonometric	✗	no inverse	✗	✓*	✓	5.6
Hyperbolic	✗	no inverse	✗	✓*	✓	
Rounding	✗	✗	✗	✓*	✓	B8
Modulo	✗	✗	✗	✓*	✓	
Isosurface	✓	✓	✗	✓*	✓	
Mean	block	block	arbitrary	arbitrary*	stencil	B9
Finite Difference	✗	✗	✗	✓*	✓	5.6 & B6
Comparison	✗	✗	✗	✓*	✓	
Classification	✗	✗	✗	✓*	✓	B8
Combinators	all	all	✗	arbitrary*	not, any, all, where	
Auxiliary Data	✗	✗	✗	✓*	✓	5.6 & B6
Data Boundaries	✗	✗	✗	✓*	✓	5.6 & B12
Summary	Linear +	Differentiable	Bounded linear	Arbitrary*	Piecewise-invertible	

Table 2. Supported features by quantity-of-interest-aware lossy compressors, `QoI-SZ3`, `MGARD-QoI`, and meta-compressors, `QPET`, `OptZConfig`, and the `compression-safeguards`. For each feature, the evaluation sections where the feature is used are listed. $(*)$: `OptZConfig` can support arbitrary quantities of interest but requires the user to manually implement them, typically in C++. QoIs have to return a single scalar error metric, e.g. to describe whether the QoI is satisfied for all data points. Crucially, `OptZConfig` cannot guarantee that a safe configuration can be found, and safely using `OptZConfig` requires a possibly-external global lossless fallback.

`QPET-SPERR` (see section 5.2). In case the 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).

`MGARD`, Multi-Grid Adaptive Reduction of Data (Liang et al., 2022), is a lossy compression framework that recursively decomposes the data grid, using linear interpolation to approximate the residuals from coarser levels, until a user-chosen error bound for an arbitrary error norm is met. The `MGARD-QoI` variant (Ainsworth et al., 2019) can bound errors over those quantities of interest that can be expressed as bounded linear functionals, e.g. the arithmetic mean over the data, but not a non-linear function such as $\sin(x)$ or the percentage of points exceeding an error threshold (Baker et al., 2017; Underwood et al., 2022).

The experimental MGARD-Lambda variant (Banerjee et al., 2022) additionally supports preserving those non-linear quantities of interest that can be converted into linear constraints under simplifying assumptions.

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 OptZ-Config all 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.

Most of the above QoI-aware compressors support pointwise absolute error bounds, including over the quantities of interest, though MGARD-QoI and OptZConfig stand out for supporting arbitrary error norms. Similarly, while most QoI-aware compressors only support structured grids, or unstructured data packed into a 1D array, MGARD supports unstructured grids and OptZConfig can work with any data. The compression-safeguards implementation can handle unstructured 1D data when using only pointwise safeguards, but stencil safeguards still require a structured grid to define their local neighbourhoods.

The compression-safeguards implementation of quantities of interest builds on the symbolic analysis of QoIs as expression trees in QoI-SZ3 and QPET. QPET and the compression-safeguards both combine symbolic and numeric analysis of the quantity of interest and guarantee correctness using corrections. In contrast, QoI-SZ3 and MGARD-QoI only provide symbolic correctness, which may be violated through inaccurate floating-point arithmetic, and OptZConfig does not guarantee safety. While QoI-SZ3 and QPET derive symmetric data error bounds, with QPET correcting outliers losslessly, the compression-safeguards produce asymmetric safe intervals from which non-lossless corrections are chosen. All QoI-aware compressors support at least pointwise quantities of interest. QoI-SZ3 and QPET support multi-variable compression out of the box, but OptZConfig and the compression-safeguards require multiple variables to be stacked into one array. While QoI-SZ3 and QPET support block means over a quantity of interest, the compression-safeguards provide the more expressive moving window stencil safeguards.

In terms of complexity and runtime cost, QPET is expected to be more efficient than QoI-SZ3 and the compression-safeguards since the combination of a small number of compression iterations with lossless outlier correction is often faster than propagating error bounds through a quantity of interest. For the compression-safeguards implementation, the complexity comes primarily from having to evaluate the quantity of interest at each node of the expression tree to guarantee that numeric rounding errors do not cause violations of the error bounds.

Like `OptZConfig`, the Compression Safeguards are an end-user-focused solution that is compatible with any compressor and works across domains. Overall, the black-box optimiser `OptZConfig` provide the greatest flexibility for the end-user, as any data with any quantity of interest can be preserved for any compressor, so long as it is able to find a safe compressor configuration, which may be impossible if the wrapped compressor cannot be configured to provide lossless compression, e.g. with a zero error bound. However, extending `OptZConfig` to support e.g. boundary conditions requires manual implementation, whereas many such features are directly provided by the `compression-safeguards` implementation. Common tasks like preserving missing values, regionally varying error bounds, simple quantities of interest using auxiliary variables and boundary conditions can be directly and concisely configured, and more complex safety requirements can be composed from several safeguards. Furthermore, as `OptZConfig` can only tune global parameters, it would have to fall back to fully lossless compression to provide safety in cases where the compressor cannot be configured to preserve a requirement, e.g. missing values, whereas the Compression Safeguards always guarantee safety and would, in this case of missing values, only need to produce a few corrections.

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.

5 Experimental Evaluation

We evaluate the `compression-safeguards` reference implementation on a variety of real-world datasets and analyses using the `numcodecs-safeguards.SafeguardedCodec` meta-compressor. While some examples are generic, others are taken from weather and climate analyses. The goal of our evaluation is to (i) explore how different lossy compressors perform across diverse use cases, (ii) showcase how the `compression-safeguards` can protect important safety requirements, and (iii) examine the impact of safeguarding on compression ratios, compression throughput, and compression error distributions. All evaluation codes and datasets are available in Tyree et al. (2026e). The specific version of the code and analyses used to produce the results in this manuscript has also been published in Tyree et al. (2026g) under version 1.0.0rc1.post1.

This evaluation does not cover all features of the `compression-safeguards` implementation. In particular, preserving regionally varying safety requirements, topological features such as isosurfaces and critical points are all excluded for brevity. The online documentation provides additional examples and guides for these use cases and more (Tyree et al., 2026a).

5.1 Datasets

In the evaluation, we choose publicly available datasets that reflect common difficulties in compressing weather and climate data: humidity and precipitation for non-normal distributions, satellite and fire danger data for missing values, and wind because vorticity is a common derived quantity, which uses wind gradients:

1. ERA5 is a comprehensive reanalysis of the global climate and weather, which is produced by the European Centre for Medium-Range Weather Forecasts (ECMWF). ERA5 combines observations with the IFS numerical weather prediction model to reproduce the global state of the atmosphere, land surface, and ocean waves, filling gaps in observational data to produce a complete dataset, gridded regularly in space and time (Hersbach et al., 2020). We use

   (a) the specific humidity (Hersbach et al., 2018a) for 02 April 2024 at 12:00 at $850\mathrm{hPa}$ ($4.0\mathrm{MiB}$), and

   (b) the u and v wind components (Hersbach et al., 2018a) for 02 April 2024 at 12:00 at $500\mathrm{hPa}$ ($7.9\mathrm{MiB}$), and

   (c) the single-level hourly total precipitation (Hersbach et al., 2018b) for 02-04 April 2024 ($285.2\mathrm{MiB}$).

2. The energy release component of the ERA5-derived global daily fire danger index (Copernicus Climate Change Service, Climate Data Store, 2019) for 18 July 2024 ($4.1\mathrm{MiB}$).

3. The global 6-hourly total column water vapour over ocean derived from satellite observations (Copernicus Climate Change Service, 2022; Andersson et al., 2021) for 02 August 2020 ($3.5\mathrm{MiB}$).

We have also made these datasets, their licenses, and download scripts available in Tyree et al. (2026d). The datasets are also deliberately kept small. Weather and climate datasets are usually chunked in practice, often along the time dimension, with chunks typically of size 1-10MiB (Hassell and Cimadevilla Alvarez, 2026). We therefore evaluate mostly on single timesteps of 2D fields, which are sufficient to demonstrate cases of unsafe compression and assess how the Safeguards resolve them. However, we acknowledge that scientific compressors are often evaluated on much larger datasets (e.g., Zhao et al., 2020; Reichelt et al., 2026; Han et al., 2026) where they can produce higher compression ratios.

5.2 Compressors

We compare the `compression-safeguards` with `ZFP-ROUND`, abbreviated henceforth as `ZFP`, and `SZ3`, `SPERR`, `OptZConfig`, and `QPET-SPERR`. **`ZFP`** (Lindstrom, 2014; Diffenderfer et al., 2019) is a transform-based error-bounded lossy compressor that processes n-dimensional data in $4 \times \ldots \times 4$ blocks. The values in each block are decorrelated, reordered by magnitude, and concatenated into a bitstream that can be truncated to satisfy a user-provided compression target, e.g. a specific compression ratio or an absolute error tolerance. The `ZFP-ROUND` variant eliminates the error bias in the original implementation (Diffenderfer et al., 2019; Hammerling et al., 2019). **`SZ3`** (Zhao et al., 2021; Liang et al., 2023; Liu et al., 2024) is a modular prediction-based error-bounded lossy compressor that consists of preprocessing, prediction, lossy quantisation, entropy coding, and lossless compression. `SZ3` supports bounding the pointwise absolute or range-relative error, combinations of the two, the peak signal to noise ratio, or the L_2 error, all of which are transformed into pointwise absolute error bounds. Several domain-optimised configurations and expansions of SZ3 have been published, including `CliZ` for climate data (Jian et al., 2024). The **`SPERR`** compressor (Li et al., 2023) combines wavelet-based lossy compression with quantised outlier corrections to bound the pointwise absolute error. `SPERR` is tuned to produce around 2% outliers, which minimises the combined cost of compression and corrections. **`OptZConfig`** (Underwood et al., 2022) is a meta-compressor that automatically tunes the wrapped compressor's configuration to optimise a user-defined metric, e.g. to bound the error over a quantity of interest

while also increasing compression ratios. While OptZConfig cannot guarantee safety, it can often identify safe and efficient compressor setups. **QPET-SPERR** (Liu et al., 2025) integrates SPERR with QPET to bound the absolute or range-relative error over a derived quantity of interest. It derives and auto-tunes an approximate global absolute error bound over the data that is passed to SPERR, and losslessly encodes data points to correct violations in the quantity of interest (see section 4 for further details). These compressors cover important classes of scientific lossy compression with a focus on reducing local errors: ZFP is block-transform-based, SZ3 is predictive, SPERR is wavelet-transform-based, and OptZConfig and QPET represent the state of the art in preserving quantities of interest. Please refer to Di et al. (2024) for a survey and taxonomy of further error-bounded lossy compression methods.

For compression, we use the numcodecs-wasm (Tyree, 2026) project, which adapts many popular scientific compressors to the numcodecs compression API (Zarr developers, 2025), compiles them to platform-independent WebAssembly bytecode, published as Python packages, and then executes each compressor in its own sandbox, making compression fully reproducible across different platforms. Please refer to Reichelt et al. (2026) for further details on numcodecs-wasm.

Table A1 lists the versions of all compressors that we evaluate. We test v3.3.2 of SZ3 unless otherwise specified, but additionally compare against v3.2.2 in section 5.4 to highlight how compression-safeguards would have protected users from a bug in v3.2.2. We intended to compare both QPET-SPERR, where QPET derives a global absolute error bound for SPERR, and QPET-SZ, where QPET derives per-point absolute error bounds for SZ3. Unfortunately, QPET-SZ is tightly coupled to an outdated version of SZ3 and crashed in our evaluations, and the authors were unable to fix the underlying bug in time. We also do not compare with the quantity-of-interest-aware MGARD-QoI compressor because of its high computational complexity. Please refer to Underwood et al. (2022) for a comparison between MGARD-QoI and OptZConfig, which concludes that OptZConfig outperforms MGARD-QoI.

In the following examples, we use OptZConfig's FRaZ-based Fixed Metric Fidelity Search (FMFS) tuning algorithm (Underwood et al., 2020, 2022) to maximise compression ratios while eliminating all violations of the example's safety requirements. We have extended OptZConfig such that it can be used with numcodecs-wasm compressors and can reuse safety requirement validation routines written in Python, ensuring that OptZConfig results are comparable. Since OptZConfig struggled with directly finding safe absolute compressor error bounds, whose search space typically spans many orders of magnitude, we instead tune the logarithm of the error bound for 25 iterations.

The Compression Safeguards Framework is intended to safeguard an existing lossy compressor and the SafeguardedCodec was not specifically designed to be a compressor in and of itself. Nevertheless, we also evaluate the effects of safeguarding a constant-zero-predictor, denoted as 0, to stress-test the lossless compressibility of the corrections that are needed for nearly every point of the all-zero decompressed data. Additionally, we compare, for each safeguarded compressor, the default one-shot corrections with iteratively-reduced corrections (see section 3.5) and iteratively-reduced lossless corrections to assess the impact of the one-shot corrections. We also compress every dataset losslessly using Zstandard (Collet, 2021) at maximum compression, level 22, to provide a comparison for safe compression ratio and throughput.

5.3 Evaluation metrics and visualisations

We compute the following metrics to compare the `compression-safeguards` implementation with different compressors, where x is the original data of size N, $\hat{x}$ is the decompressed data and $\tilde{b}$ is the binary corrections:

1. $L_\infty(\hat{x}) = \max_i |\hat{x}_i - x_i|$ measures the maximum pointwise absolute error. If L_∞ is NaN or infinite, we also give the finite-only maximum pointwise absolute error in parentheses.

2. $L_2(\hat{x}) = \sqrt{\frac{\sum_i^N (\hat{x}_i - x_i)^2}{N}}$ measures the root mean square error. If L_2 is NaN or infinite, we also give the finite-only root mean square error in parentheses.

3. $ok_{abs}(x_i, \hat{x}_i, \epsilon_{abs}) = (|\hat{x}_i - x_i| \leq \epsilon_{abs}) \vee (\hat{x}_i \equiv x_i)$ defines whether a point i satisfies a pointwise absolute error bound ϵ_{abs}. The equivalence cases in the disjunction accounts for infinite and NaN values, such that infinite values must be preserved exactly and $\hat{x}_i$ must be NaN if and only if x_i is NaN. In examples that use a quantity of interest $f(x)$, ok_{abs} is computed over $f(x)$ and $f(\hat{x})$ instead.

4. $\mathrm{V} = \frac{|\ \{i|\neg ok(x_i, \hat{x}_i)\}\ |}{N}$ measures the percentage of data points that violate their safety requirements, often a pointwise absolute error bound. We report the percentage rounded to one digit of precision, and use $V < 0.05\%$ to denote that there were some but very few violations. If no points violate the requirements, we give $V = 0$.

5. $\mathrm{C} = \frac{|\ \{i|\tilde{b}_i \neq 0\}\ |}{N}$ measures the percentage of data points for which the safeguards produce a non-zero correction. If no corrections were produced, we give $C = 0$.

6. $\mathrm{CR} = \frac{|\mathrm{bytes}(x)|}{|\mathrm{bytes}(c)|}$ measures the compression ratio as the ratio between the in-memory byte-size of the original data x over the byte-size of the compressed data c. For instance, $\times 10$ describes compression that reduced the size by a factor of ten. For the original data, we report the in-memory byte-size in binary notation, i.e. $10\mathrm{MiB}$ refers to $10 \cdot 2^{20}$ bytes.

Each figure contains the compression ratio CR and the percentage of violations V, while the other metrics are reported in auxiliary tables. For each compressor, we also include a map of the violations, if there were any, and a map of the corrections when using the `compression-safeguards`. Many auxiliary figures also show the distribution of the data, quantity of interest, or its error as a histogram above the colour bar to assess if compressors introduce bias. On these histograms, we mark the 25%, 50%, and 75% quantiles with line markers, and the mean with a dotted line. If the colour bar includes error values that violate the safety requirements or data values that are outside the original range, they are marked on the colour bar with white hatches. Missing or NaN values are highlighted with pink hatches.

5.4 Example: NaN missing values

Scientific datasets often contain missing values, e.g. from sometimes-sparse missing observation values or from ocean-only model variables over land. Missing values are typically represented as NaNs in memory but may have a different representation on disk. To examine how compressors handle NaN missing data, we compress four 6-hourly satellite observations of sea-only

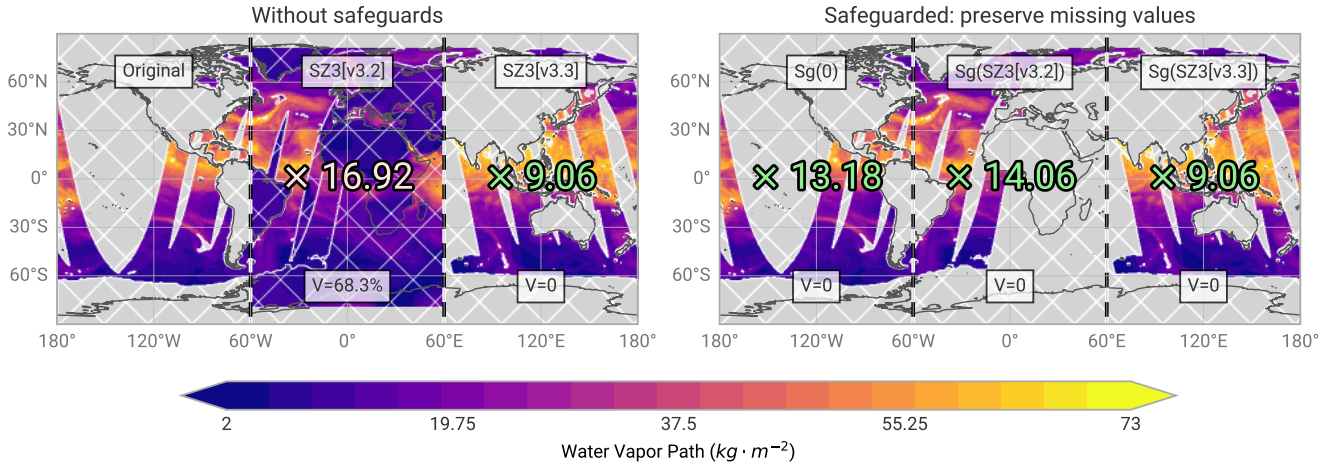


Figure 2. Compressed water vapour path maps from satellite data with NaN missing values using different lossy compressors with absolute error bounds. In the left map, the original data and the decompressed data from `SZ3` v3.2 and v3.3 are shown. The right map shows the `compression-safeguard`ed data from an all-zero predictor and `SZ3`. For each compressor, the compression ratio and percentage of violations V, are given as well. Areas with originally missing values, above land and in between satellite swaths, are outlined and hatched.

satellite observation data, where data is statically missing on land and temporarily missing where there is no satellite coverage. We configure each compressor and the safeguards with an absolute error bound of $\epsilon_{abs} = 0.1\text{kg}\,\text{m}^{-2}$ (see listing C1).

Not all lossy compressors support or handle non-finite values correctly (Fallin and Burtscher, 2024). Figure 2 shows that `SZ3` [v3.2] hallucinates plausible non-NaN values where data was originally missing. We reported and fixed this bug in `SZ3` pull request #91 (Tyree, 2025), and `SZ3` [v3.3] now correctly preserves the missing values.

Error-bounding safeguards always preserve NaN values, configurable with the same or any NaN payload bit pattern, and the safeguarded compressors thus have no violations. When the safeguards have to correct hallucinated non-missing values, the compression ratio drops, though the decrease is lower when not preserving NaN payload bit patterns exactly (see Table B1). For `SZ3` [v3.3], which has no violations, the safeguards produce no corrections and thus only have a negligible impact on the compression ratio. Overall, the safeguards are able to correctly and efficiently preserve missing values and would have protected users from the bug in `SZ3` [v3.2].

Interestingly, the safeguarded `SZ3` [v3.2] compressor compresses better than `SZ3` [v3.3]. While this may point to a more efficient NaN correction scheme in the safeguards than in `SZ3`, the difference may also come from unrelated algorithmic changes between the two versions of `SZ3`. The safeguarded all-zero predictor produces the second-highest compression ratio.

Appendix B1 further compares the preservation of missing values with `ZFP` and `SPERR`. It also showcases that `OptZ-Config` is only able to safely preserve NaN missing values if the underlying compressor can preserve them itself. 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.

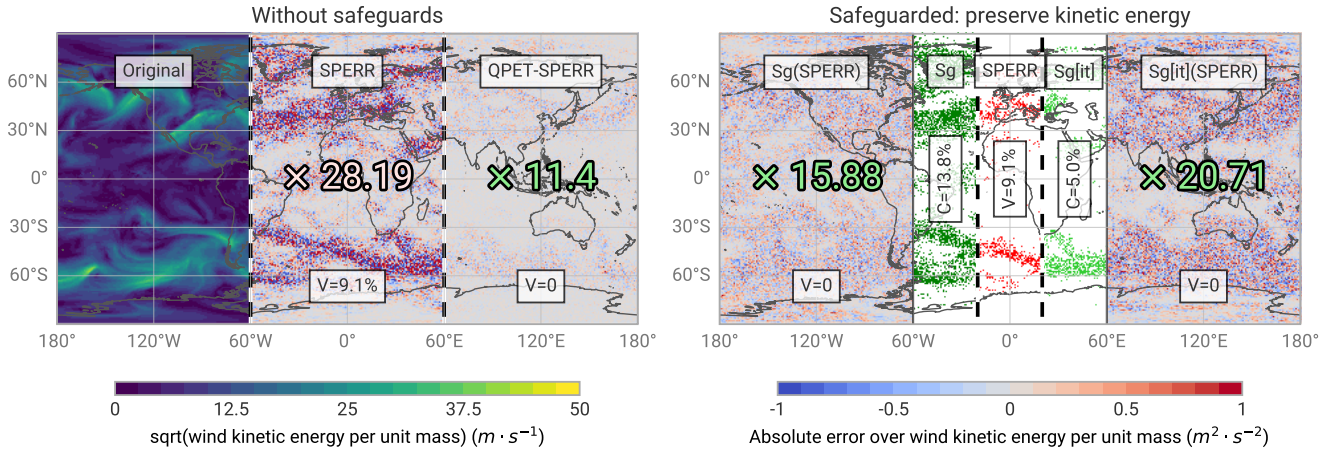


Figure 3. Pointwise kinetic energy error maps derived from compressed wind u and v components using different lossy compressors. In the left map, the original data and the decompression error from `SPERR` and `QPET-SPERR` are shown. The right map shows the compression error for the `compression-safeguarded` `SPERR` using one-shot corrections, `Sg(SPERR)` / `Sg`, on the left and iteratively-reduced corrections, `Sg[it](SPERR)` / `Sg[it]`, on the right. The middle section of the right map compares the two correction distributions with the distribution of `SPERR`'s error-bound violations. The compression ratio and percentage of violations V are also given for each compressor, as well as the percentage of corrections C for the safeguarded compressors.

5.5 Example: kinetic energy as a multi-variable pointwise quantity of interest

Preserving error bounds and other properties over quantities of interest $f(x)$ derived from the data x is an important gap in lossy scientific compression (Cappello et al., 2025). Appendix B4 provides an introductory example about preserving the single-variable quantity $\log(x)$, which can also be expressed as a ratio error bound. In this more complex example, we compute the wind kinetic energy per unit mass, $\mathrm{KE} = \frac{1}{2}(u^2 + v^2)$, from a dataset of wind components u and v, examine whether it is preserved after lossy compressing u and v, and demonstrate how the `compression-safeguards` can bound the absolute error over the derived kinetic energy. Kinetic energy is a pointwise quantity of interest that is computed over two variables at once. For simplicity, we stack both variables into one array, such that they can be compressed together. Stacking u and v is possible because they have very similar data distributions. Since u and v are independent, we stack them along a new first, slowest varying axis. We aim to bound the absolute error over the kinetic energy to $\epsilon_{abs(\mathrm{KE})} = 1\mathrm{m}^2\,\mathrm{s}^{-2}$ and configure each compressor with an absolute error bound of $0.125\mathrm{m}\,\mathrm{s}^{-1}$ over the stacked array.

The safeguards are directly configured to bound the pointwise absolute error over the derived kinetic energy, which is translated into a quantity of interest. Since u and v are stacked into one array, we now compute the kinetic energy over a two-element (u, v) local neighbourhood along the stacked first axis (see listing C4). Since `QPET-SPERR` can preserve polynomials, we can similarly configure `QPET-SPERR` to bound the pointwise absolute error over the derived kinetic energy. We therefore utilise `QPET-SPERR`'s block-mean mode to equivalently bound the mean of x^2 over the stacked (u, v) axis.

Figure 3 shows that simply bounding the absolute error over u and v is not sufficient in general, as `SPERR` violates the targeted absolute error bound over the kinetic energy. Similar violations can also be seen in appendix B5 for the `ZFP` and `SZ3` compressors. With the safeguards and `QPET-SPERR`, no violations are produced. In this example, safeguarded `SPERR` compresses the best and even outperforms `QPET-SPERR`, which in turn beats `OptZConfig`-tuned `SPERR`. The safeguarded compressors also provide better compression than tuning with `OptZConfig` (see appendix B5). The compression ratios for the safeguards can be further improved by switching from one-shot to iteratively-reduced corrections (see also Table B5).

5.6 Example: relative vorticity as a stencil quantity of interest

While pointwise quantities of interest can independently consider and correct each data point, many quantities of interest depend on a local neighbourhood around each point, e.g. to approximate spatial derivatives using finite differences. In this example, we compute the relative vorticity

$$\zeta(\phi, \lambda) = \left(\frac{\partial v}{\partial \lambda} - \frac{\partial \left(u \cdot \cos \phi \right)}{\partial \phi} \right) \cdot \left(R_e \cdot \cos \phi \right)^{-1} \tag{3}$$

for a dataset of wind u, v vectors on a regular latitude-longitude grid, which requires taking the spatial derivative along both the latitude ϕ and longitude λ axes, where R_e is Earth's radius. Note that the derivative along the longitude is computed along a periodic axis, which requires handling the periodic boundary condition when computing and safeguarding the finite differences. We compare how the derived vorticity is affected after compressing u and v with different error-bounded lossy compressors, and demonstrate how the `compression-safeguards` can bound the absolute error over the derived vorticity. As in section 5.5, u and v are stacked into one variable along a new first, slowest varying axis.

We configure each compressor with an absolute error bound of $0.01 \mathrm{m\,s^{-1}}$ over the u-v array with the aim of bounding the absolute error over the derived vorticity to $\epsilon_{abs(\zeta)} = 10^{-8} \mathrm{s^{-1}}$; the safeguards are configured directly with this bound.

For the safeguards, Equation (3) is translated into a quantity of interest over a small local neighbourhood, where first-order, second-order-accuracy finite central differences over an arbitrary grid approximate the spatial derivatives, taking into account the period of 360 degrees along the longitude axis. For the derivative along the latitude axis, we switch to forward and backward finite differences at the polar boundaries, as in the second case in appendix B6. The complete configuration of the vorticity quantity of interest for the `SafeguardedCodec` is given in listing C7.

The `compression-safeguards` implementation goes beyond the state-of-the-art. While `QPET` can bound the mean error over non-overlapping blocks of data, this is insufficient for bounding the error over a finite difference, where data points and local neighbourhoods are not independent. Since `QPET` also does not support either boundary conditions over the data or referencing auxiliary variables in a quantity of interest, here the latitude and longitude coordinates, we do not compare with `QPET-SPERR` in this example. We do, however, evaluate `OptZConfig`, which requires manual extension to support the relative vorticity quantity of interest.

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. Similar violations can also be seen in appendix B7 for the `ZFP` and `SZ3`

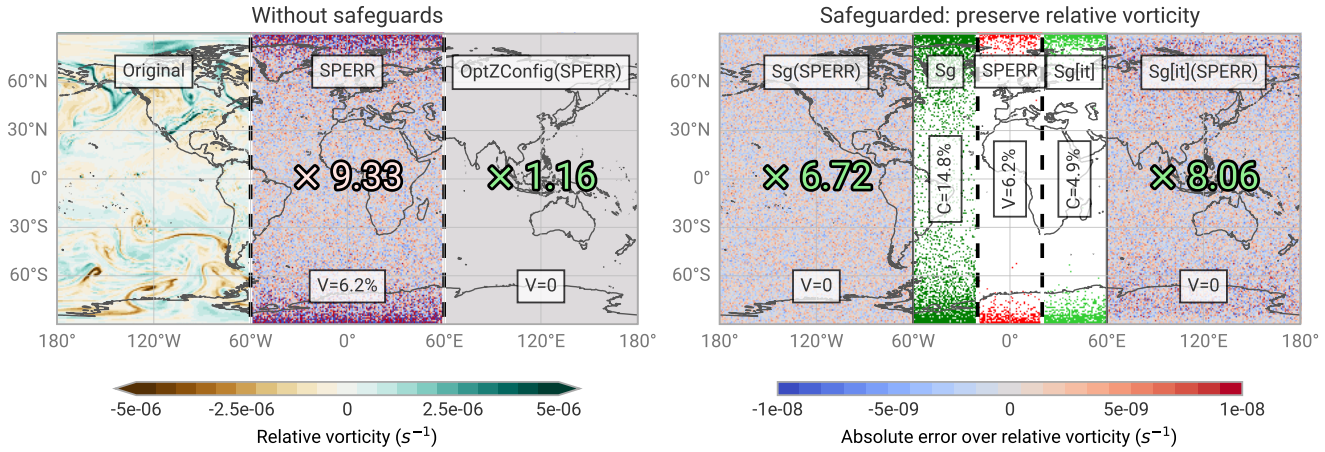


Figure 4. Relative vorticity error maps derived from compressing wind u and v components using different lossy compressors. In the left map, the original data and the decompression error from `SPERR` and `OptZConfig`-tuned `SPERR` are shown. The right map shows the compression error for the `compression-safeguard`ed `SPERR` using one-shot corrections, `Sg(SPERR)` / `Sg`, on the left and iteratively-reduced corrections, `Sg[it](SPERR)` / `Sg[it]`, on the right. The middle section of the right map compares the distribution of the two correction distributions with the distribution of `SPERR`'s error-bound violations. The compression ratio and percentage of violations V are also given for each compressor, as well as the percentage of corrections C for the safeguarded compressors.

compressors. With the safeguards, all violations are corrected, though at the cost of decreasing the compression ratio. This decrease is particularly significant for `SPERR` (see also table B8). Crucially, while `OptZConfig` is also able to safely bound the vorticity error, it only achieves very little compression that is clearly outperformed by safeguarding. This is likely because `OptZConfig` has to reduce the global error bound to account for localised violations near the poles, whereas the Safeguards only need to produce localised corrections.

The safeguarded compression ratio can again be improved by switching from one-shot to iteratively-reduced corrections. Interestingly, in this example, there is a clear pattern in where some of the one-shot corrections can be removed. Whereas the one-shot corrections have to be conservative everywhere, the iterative corrections are only needed closer to the poles, where the wrapped lossy compressors violate the safety requirements.

The example of relative vorticity exercises the support for several novel features: boundary conditions, coordinates as auxiliary data, and finite differences over arbitrary grids. Appendix B6 provides an additional derivative-focused example about safeguarding the Laplacian. To further stress test the `compression-safeguards` implementation, appendix B8 examines how to guarantee a minimum score for the data structural similarity index (dSSIM) measure (Baker et al., 2024).

5.7 Performance evaluation

The previous sections have evaluated how our reference implementation can preserve important data invariants across a variety of weather and climate use cases, showcasing the flexibility and novel features of the `compression-safeguards` im-

plementation. We now evaluate the runtime performance of the `numcodecs-safeguards.SafeguardedCodec` across the examples from sections 5.4 to 5.6 and appendices B2, B4, B6 and B8. In each example, we measure the throughput that each safeguarded compressor takes to (de)compress, and compare the throughputs across examples and stages of safeguarding. We later also compare the compression ratios across the examples. The measurements were performed on an Apple M2 CPU running Python 3.11 with `numpy` 2.4.6. We exclude cases where the safeguards did not have to produce any corrections.

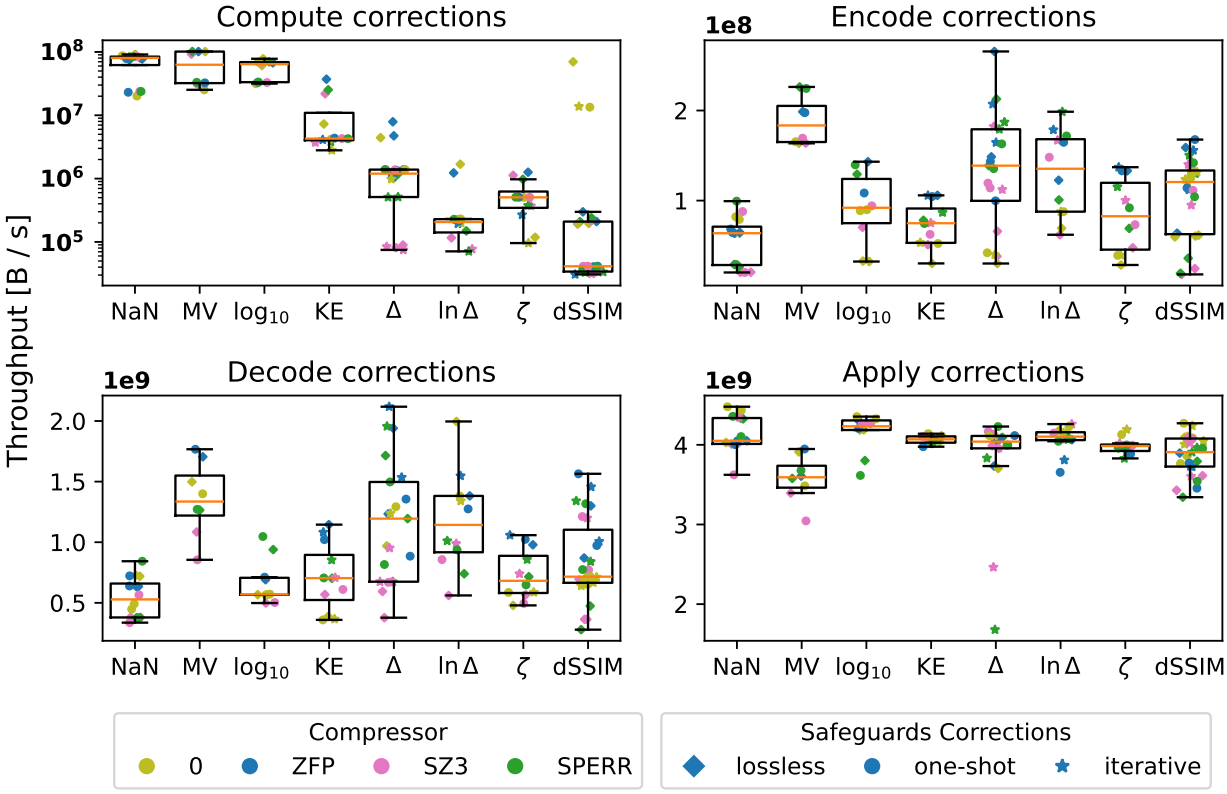


Figure 5. Throughput of the `numcodecs-safeguards.SafeguardedCodec`'s safeguarding stages across all examples in section 5 and appendix B. The throughput is reported separately for computing corrections, encoding (compressing) and decoding (decompressing) corrections, and applying corrections. The distribution 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 5 compares the throughput of the `numcodecs-safeguards.SafeguardedCodec` during compression and decompression across of the examples. The most significant overhead comes from computing the corrections, which increases drastically as the complexity of the safety requirements grows, especially for the examples using non-pointwise quantities of interest. Iteratively reducing corrections has the lowest throughput, while producing lossless corrections that are iteratively reduced typically has higher throughput than producing the one-shot corrections. While the time taken to compute the corrections is typically similar across different compressors, in the Laplacian example where `SZ3` requires more corrections Table B6, correcting `ZFP` is significantly faster and `SZ3` is significantly slower. In the dSSIM example, the top three safeguarded zero-predictor outlier throughputs are produced by the much-simplified `bins(256)` safeguard (see appendix B8).

In contrast to the example-specific performance for computing corrections, the encoding, decoding, and application of corrections all perform similar across examples. Importantly, applying the corrections has the highest throughput of all steps, since it only requires the binary addition of the corrections (see section 2.3) and, specifically in `numcodecs-safeguards`, the check of the corrected data's checksum (see section 3.5).

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.

While safeguarding can drastically reduce compression throughput for complex non-pointwise safety requirements, data is typically only compressed and written once but decompressed and read many times by the data users (Cappello et al., 2025). To allow users to benefit from safe and fearless decompression at little additional computational cost, data producers may be able to spend more on compression time, to parallelise computing corrections, e.g. with the `xarray-safeguards` frontend, or to offload them, e.g. to the GPU (e.g., Li et al., 2026; Ren et al., 2026). Since corrections can be computed long after compression and can distributed separately from the compressed data (see section 2.2), safeguarding can also be moved entirely off the performance-critical compression path. We plan to specifically investigate a performance-focused implementation of the Compression Safeguards Framework in future work.

While this evaluation focused on the runtime performance of the `numcodecs-safeguards` frontend, Appendix B12 briefly investigates the performance scaling of the `xarray-safeguards` frontend over chunked data.

Several compressors internally use lossless corrections (e.g., Liang et al., 2023; Liu et al., 2025) to guarantee error bounds over data or quantities of interest. The `compression-safeguards` implementation instead produces non-lossless corrections in an expensive one-shot algorithm that uses a simple heuristic to improve their compressibility (see section 3.1). Figure 6 compares the compression ratios of these one-shot corrections with lossless corrections and iteratively-reduced corrections. Across most examples, one-shot corrections offer higher compression ratios than lossless corrections, typically by a factor of two or more. The non-NaN missing values example from Appendix B2 is the exception, since the error-bounded compressors require corrections only for the missing value sentinel, for which only lossless corrections suffice. The one-shot corrections also outperform the compression ratios from `QPET-SPERR`, also when using lossless corrections for safeguarding `SPERR`. Importantly, safeguarding with one-shot corrections achieves better compression than tuning with `OptZConfig` across most examples. The only exceptions are non-NaN missing values with `SZ3` and the Laplacian quantity of interest with `SZ3` and `SPERR`. While `OptZConfig` can compress better when preserving an affine quantity of interest with uniform global

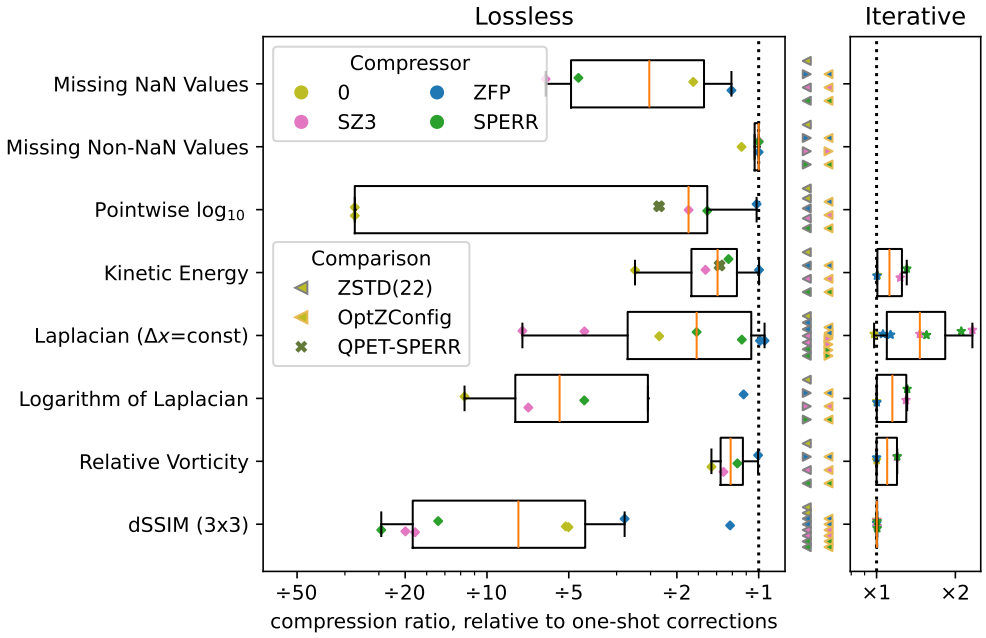


Figure 6. Impact of different correction methods on the safeguarded compression ratios for each of the examples, including for the all-zero predictor in lime. The compression ratio for the lossless and iteratively-reduced corrections is shown relative to the compression ratio with the default one-shot corrections. The distribution of the measurements is visualised with box plots, as in Figure 5. The compression ratio of `QPET-SPERR` relative to one-shot safeguarded `SPERR` is also given as a dark green "x" for the $\log_{10}$ and kinetic energy examples. For all examples, grey-outlined and yellow-outlined triangles in the middle, coloured like the wrapped compressors, show whether lossless compression with `Zstandard` and tuning with `OptZConfig` produced lower or higher compression ratio than the one-shot corrections for the safeguarded compressors.

error bound and a highly-efficient compressor, the safeguards outperform tuning whenever safety requirements are non-linear or when the wrapped compressor cannot be configured to preserve a requirement by itself.

In the examples with non-pointwise safeguards, the iteratively-reduced corrections provide smaller benefits, typically an increase by less than a factor of two. As the above-analysed runtime cost of these iterative corrections is significant, one-shot corrections can be seen as a valuable trade-off between compression speed and compression ratio.

Finally, Figure 6 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.

5.8 ClimateBenchPress

We further investigate the performance of the `compression-safeguards` implementation using the ClimateBenchPress (v1.0) benchmark, which is designed to fairly compare the compression ratio, compression errors, and execution cost of lossy compressors across a diverse variety of datasets and variables (Reichelt et al., 2026). The benchmark compares compressors across three different levels of absolute and relative error bounds to investigate trade-offs between compression ratios and distortion errors. Please refer to Reichelt et al. (2026) for a description of the datasets, benchmark setup, evaluation metrics, and the code and data availability.

We extend ClimateBenchPress with several new compressors. First, we add safeguarded versions of bit rounding and Pcodec, abbreviated henceforth as `BitRound`, and of `EBCC` (Huang et al., 2025), `SPERR`, `SZ3`, and `ZFP-ROUND`, abbreviated henceforth as `ZFP`. These safeguarded compressors preserve the pointwise absolute or pointwise relative error bound, depending on the variable being compressed. Reichelt et al. (2026) uses `SZ3` v3.2.2 in the benchmark, abbreviated henceforth as `SZ3[v3.2]`, which does not handle NaN values correctly. While most of section 5 evaluates v3.3.2, v3.2.2 is compared once in section 5.4. Table A2 lists the versions of all compressors that we evaluate with ClimateBenchPress. Second, we also benchmark the safeguarded all-zero predictor. As ClimateBenchPress evaluates the data structural similarity index (dSSIM) metric (Baker et al., 2024), we also include a safeguarded all-zero predictor that additionally guarantees a perfect dSSIM score using the cheap quantisation-bin preservation method from appendix B8. Since the dSSIM score is computed independently for each 2D latitude-longitude slice, the minimum and maximum values for each slice are provided as late-bound parameters.

Third, however, we do not benchmark `QPET-SPERR`, which does not support pointwise relative error bounds and would thus achieve the same results as `SPERR` but with lower compression throughput. We also exclude `OptZConfig`, whose performance at maximising compression ratios for such simple safety requirements without early stopping would not be competitive and entirely dominated by having to compress and decompress for multiple iterations.

Figure 7 compares the mean absolute error, compression ratio, and percentage of pointwise error bound violations across all variables and compressors. Fallin and Burtscher (2024) and Reichelt et al. (2026) have shown that many available scientific error-bounded compressors have some edge cases where the error bounds are violated. Importantly, *all violations are eliminated* by safeguarding the compressors, thus providing safety. This safety comes with smaller mean absolute errors and reduced compression ratios, particularly for `SZ3`, where many missing value and pointwise relative error bound violations need to be corrected (Reichelt et al., 2026). `BitRound`, `ZFP`, and their safeguarded variants typically produce the smallest compression

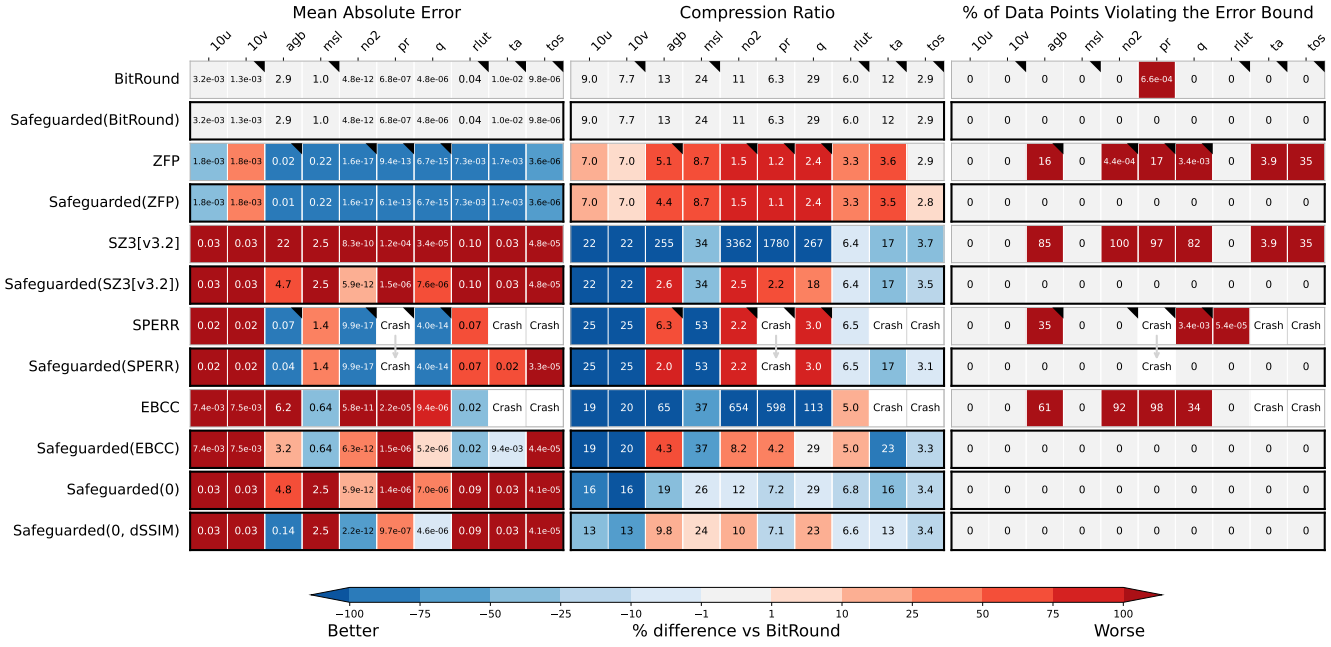


Figure 7. Scorecard of the mean absolute error, compression ratio, and error bound violations for several error-bounded lossy compressors and their outlined `compression-safeguarded` equivalents for the *mid* level error bound. Each row contains the results for a different compressor, each column for a different variable, and each panel for a different metric. Black triangles indicate cases where an absolute or relative error bound needed to be conservatively transformed since the compressor lacked support for it. Combinations where the compressor raised an exception are denoted with "Crash". The cells are coloured to indicate the difference – blue for better, red for worse – compared to the `BitRound` compressor, which is used as a baseline. The data variables are abbreviated as in section 2.1 in Reichelt et al. (2026): "10u": IFS 10 meter u-wind component, "10v": IFS 10 meter v-wind component, "agb": ESA CCI above ground biomass estimate, "msl": IFS mean sea level pressure, "no2": CAMS nitrogen dioxide mass fraction, "pr": NextGEMS precipitation rate, "q": IFS specific humidity, "rlut": NextGEMS outgoing longwave radiation flux, "ta": CMIP6 air temperature, "tos": CMIP6 sea surface temperature.

errors. SZ3, safeguarded SZ3, and the safeguarded zero-predictor produce much larger errors but also compress better, especially SZ3 without safeguards. EBCC and SPERR raise exceptions when compressing the "ta" and "tos" variables, which both contain NaN values. When safeguarding both compressors, we first replace NaNs with the data mean and then use the safeguards to restore the NaN values, as in appendix B1, which resolves the crashes. For the "pr" variable, ClimateBenchPress converts the relative error bound, which SPERR does not support, to a conservative tiny absolute error bound, causing SPERR to fail. This crash is unchanged when wrapping SPERR in safeguards. The complete scorecards for the low, middle, and high error bounds are provided in Figures B12 to B14, respectively.

We further investigate the trade-off between compression ratio, also called compression rate, and several distortion error metrics using rate-distortion plots in Figure 8. Safeguarded SPERR typically provides the best *safe* trade-off without violations. Safeguarding generally tends to reduce the compression ratio, though the difference is only significant for SZ3's maximum absolute error, likely because the safeguards have to correct many violations, as described above and shown in Figure 7. 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). The dSSIM-preserving safeguarded zero-predictor produces, by construction, $\text{dSSIM} = 1$ across all variables, though at the cost of some lower compression ratios than the basic safeguarded zero-predictor. All in all, for the simple use cases of preserving only pointwise absolute or relative error bounds and NaN missing values, ClimateBenchPress shows that the safeguards provide safety while maintaining high compression ratios when few corrections are needed.

We also evaluate the runtime performance using ClimateBenchPress in Figure B16. The results mirror the outcomes in section 5.7 and appendix B10, which compares safeguarding a larger variety of more complex safety requirements.

6 Discussion

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. 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.

When safeguarding pointwise safety requirements like error bounds or missing values, only the points that violate the requirements need corrections. For lower-dimensional data, e.g. 2D data slices, losslessly compressing the safeguards-produced corrections can often minimise their impact on the combined compression ratio. For higher-dimensional data and highly effective compressors such as SZ3 or SPERR, frequent per-point corrections can, however, bottleneck the compression ratio. While

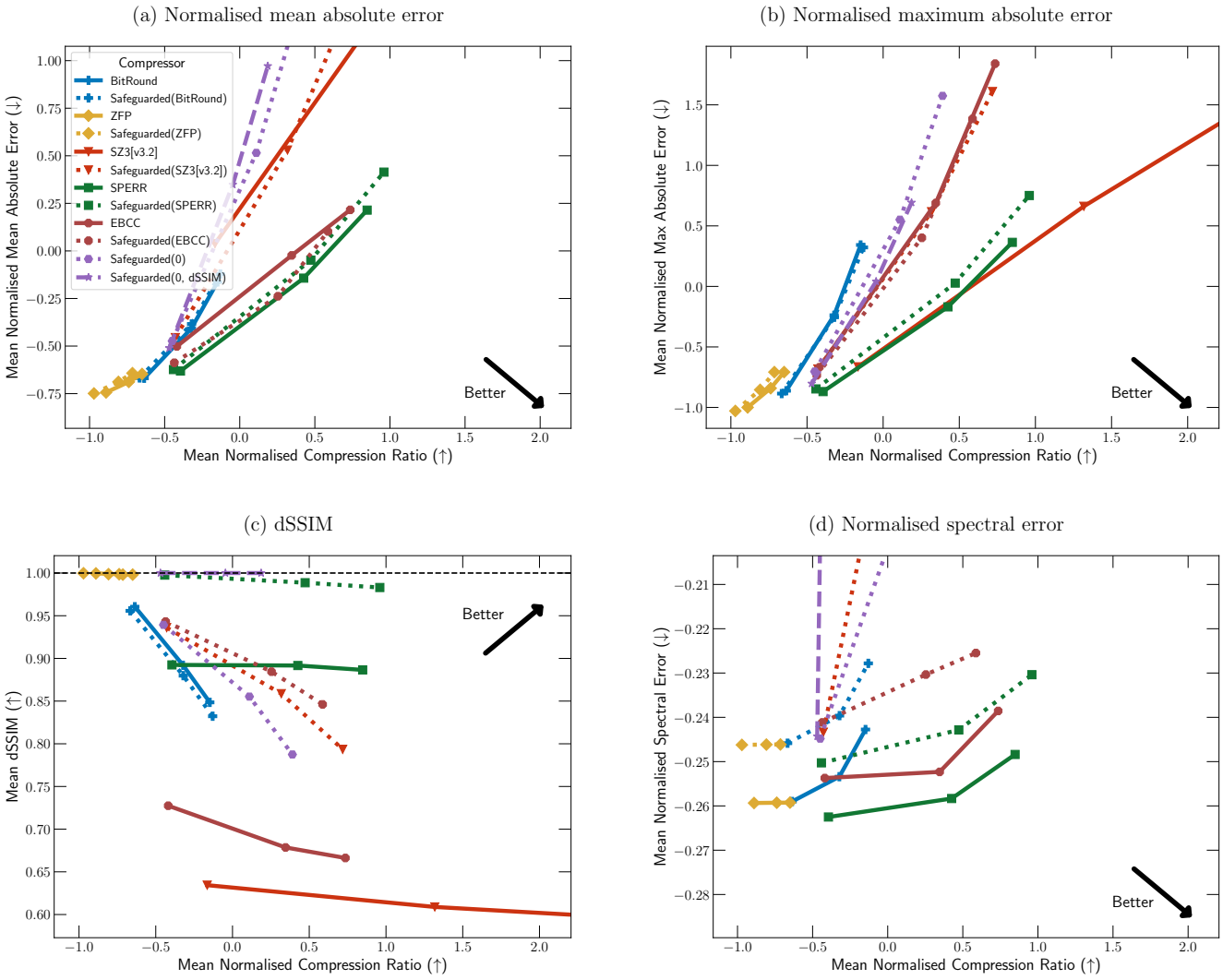


Figure 8. Comparison of the rate-distortion plots, for four distortion metrics, for several error-bounded lossy compressors, shown as solid lines, and their `compression-safeguarded` equivalents, shown as dotted lines. Each plot shows the compression ratio on the x-axis and a different distortion metric on the y-axis. The compression ratios and rate-distortions for each compressor are normalised across all variables as in Figure 3 of Reichelt et al. (2026), except for dSSIM. In the dSSIM scores panel, the optimal dSSIM $= 1$ is highlighted as a dashed line for comparison. Each plot's axes are selected to exclude outliers; the uncropped rate-distortion curves are provided in Figure B15.

safeguarded compressors have near-constant overhead during decompression, the overhead during compression depends heavily on the complexity of the safety requirements and can easily bottleneck and drastically reduce compression throughput (see section 5.7).

The `compression-safeguards` implementation also supports bounding the error over quantities of interest that are computed per-point or over local neighbourhoods. For example, higher order derivatives over regular grids with arbitrary point spacing, or arbitrary functions over these derivatives, can be preserved using their finite difference approximations (see appendix B6 and section 5.6). Here, the `compression-safeguards` provide *novel* support for data with boundary conditions, e.g. the periodic boundary along Earth's longitude axis, and can even correctly preserve non-pointwise safety requirements across chunk boundaries in chunked data. Overall, the `compression-safeguards` provide sufficient flexibility to preserve even complex safety requirements, such as a lower bound on the dSSIM score (see appendix B8).

The evaluation has also highlighted that the `compression-safeguards` implementation comes with several limitations.

The errors produced by lossy compression should ideally be unbiased and without spatial patterns, e.g. to ensure that two-point statistics or integrals over the data are not disturbed (Kouznetsov, 2021). The `compression-safeguards` implementation does not yet provide any safeguard to guarantee that the compression errors are always unbiased. Appendix B3 shows that the safeguarded compression errors typically have a similar distribution to the errors from the wrapped compressor and may propagate its bias, though the choice of producing fewer unique corrections to improve their compressibility may also introduce additional bias. While the implementation of a bias safeguard should be investigated in the future, there does not seem to exist a general and composable non-lossless method to produce error distributions that are unbiased with respect to both the data and an arbitrary quantities of interest. For instance, an unbiased error in the data x is incompatible with an unbiased error in $\log(x)$.

Non-pointwise safety requirements create dependencies between different data points, where correcting one might violate the safety requirements for another. The one-shot corrections produced by the `compression-safeguards` implementation are necessarily conservative and may affect too many data points, hurting compression ratios (see sections 5.5 to 5.7 and appendix B8). Iteratively reducing the number of corrections can increase compression ratios at the cost of lower throughput during compression (see section 5.7). Incrementally computed corrections can produce less conservative corrections and may provide further compression improvements in the future (see appendix B9).

Still, safeguarding lossy compressors compares favourably with the state-of-the-art `OptZConfig` and `QPET-SPERR` methods, even when using only one-shot corrections. Across our evaluation, safeguarded compression ratios were higher than `QPET-SPERR`'s, likely because one-shot corrections compress significantly better than lossless corrections (see section 5.7). Furthermore, while `OptZConfig` tunes the configured compressor error bound, which might be dragged down to satisfy sparse outliers with stricter requirements, the Safeguards are able to provide localised corrections that more efficiently handle outliers while benefitting from higher compression ratios elsewhere. Conversely, safeguarded compression can still benefit from tuning with `OptZConfig` when requirements are uniform, e.g. in an affine derivative-based quantity of interest (see appendix B6), and violations can be efficiently reduced with a lower error bound.

Finally, the `compression-safeguards` implementation also does not yet support preserving global safety requirements. Instead of preserving a global mean error, for example, appendix B8 has to bound the stricter pointwise error everywhere, which produces more conservative corrections. Since global requirements can be expressed as stencil safeguards that cover the entire data, iterative or incremental corrections would likely be required to produce better corrections with good compressibility.

7 Conclusions and Perspectives

The data volumes produced in computational sciences, including in numerical weather prediction and climate modelling, have been growing exponentially. While lossy compression methods promise reductions in data size by factors ranging from ten to more than a thousand, their adoption has thus far been limited. A key barrier is the concern among scientists that compression may lead to the loss of critical information, potentially affecting the reliability and interpretability of results derived from the decompressed data. Consequently, this lack of trust continues to hinder the uptake of lossy compression, contributing to the ongoing growth of data volumes and hindering scientific progress.

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 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 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. 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`, `OptZConfig`, or a meta-compressor for pointwise relative error bounds. When safeguarding a compressor that is already safe for a particular application, the Safeguards simply automate the verification of safety. If the compressor violates the user's requirements, the Safeguards step in and correct

them, guaranteeing that the user's safety requirements are upheld, always. Particularly as compressors get more complicated and increasingly use less-constrained machine-learning, trust and safety can be provided by the Safeguards while allowing compression developers and users to focus on achieving the best compression ratios.

We have provided a reference implementation for the Compression Safeguards, the `compression-safeguards` Python package, as well as user-facing integrations with the `numcodecs` compression API and `xarray` ecosystem for chunked data. Our implementation supports a larger variety of safety requirements than the state-of-the-art `QPET` compressor. The compressor-tuning `OptZConfig` meta-compressor can be manually extended for even greater flexibility, so long as the user knows the specialised meta-compressors or compressor-specific settings for *how* to preserve their safety requirements. However, `OptZConfig` uses global black-box optimisation to handle user-provided requirements and thus cannot guarantee safety if it cannot find a safe compressor configuration. In contrast, the `compression-safeguards` always guarantee safety even when generalising to complex composed requirements, are configured concisely based only on *what* needs to be preserved, analyse these requirements symbolically and numerically to produce targeted corrections, and support novel features out of the box, including data boundary conditions and higher order finite differences over chunked data. We have evaluated the reference implementation across a variety of examples, highlighting the expressiveness of the `compression-safeguards` but also their limitations, the use of expensive one-shot pointwise corrections with low throughput and lack of safeguards that bound global errors or bias, which we hope to address in future work.

Our `compression-safeguards` implementation provides flexibility and portability while focusing on gridded data and local or regional safety requirements. Since the Compression Safeguards Framework specifies an interoperable format for corrections, other implementations can co-exist, and safeguarded corrections can be applied across different hardware without knowing how they were produced. This guaranteed interoperability eases the adoption of the Compression Safeguards and ensures their long-term compatibility, as corrections can be computed even for already-compressed data, delivered to users as separate files, and then applied when reading the data without requiring complex new software. For instance, we hope to extend the NetCDF Climate and Forecast (CF) Metadata Conventions (Eaton et al., 2024) to natively support the application of corrections. Users could then transparently read safeguarded data in CF-based formats and software such as NetCDF or `xarray`.

While we hope that the Compression Safeguards provide an important tool in fostering trust in safe lossy compression, the Safeguards are just one piece of a larger puzzle. The question of which safety requirements need to be preserved for different variables is still unresolved and subject to ongoing research and debate, particularly as data producers often do not know about all downstream uses of their data. We therefore hope that high-resolution data-producing institutions and projects, such as ECMWF, Destination Earth, or the Coupled Model Intercomparison Project (CMIP), may use the Safeguards to optionally provide safely-lossy-compressed data to their end users – with user-chosen safety requirements that could be collected to inform community discussions – particularly where data storage and sharing constraints already impose limits, and safely-lossy-compressed data can provide scientists with higher-quality datasets than what can currently be made available. Importantly, the

Compression Safeguards are similarly capable of guaranteeing that more niche, individual-user-specific safety requirements are preserved when users compress their own data.

As lossy compression is adopted more, we believe that the community should also develop new provenance metadata standards for lossy compression, such that lossy compressors and the Compression Safeguards can record the safety requirements that they fulfil. Such metadata would also offer protection from repeatedly compressing the same data over and over, accidentally accumulating errors beyond the specified error bounds with each iteration.

Overall, we hope that the Compression Safeguards will contribute to building trust in lossy compression, enabling its broader adoption across scientific computing to mitigate the rapid growth of data volumes, and to make higher-resolution datasets more accessible. Where data volumes impose limits on scientific research, as in high-resolution climate modelling, we hope that trusted lossy compression can empower scientists to continue advancing our understanding of and informing our responses to growing societal challenges.

Code and data availability. All code and data are available open source from the GitHub repository at https://github.com/juntyr/compression-safeguards, which is published under the Mozilla Public License 2.0. The exact version of the code that was used to produce the results and plots presented in this paper is archived on Zenodo under doi:10.5281/zenodo.23776642 (Tyree et al., 2026g). The `compression-safeguards` reference implementation and its `numcodecs-safeguards` and `xarray-safeguards` frontends are published to the Python Package Index at https://pypi.org/project/compression-safeguards/, https://pypi.org/project/numcodecs-safeguards/, and https://pypi.org/project/xarray-safeguards/, respectively. They are published under the Mozilla Public License 2.0, which is a weak, file-based copyleft free and open-source license that permits commercial use and integration with differently-licensed code but ensures that modifications to the original source code files remain open-source and available for the public benefit.

Appendix A: Experimental Evaluation: Compressor Versions

Compressor	Version	Python package
compression-safeguards	1.0.0rc1	numcodecs-safeguards==0.1.0rc1
	1.0.0rc1	xarray-safeguards==0.1.0rc1.post1
		numcodecs-wasm==0.2.5
mask	-	numcodecs-mask==0.1.0
pw_rel	-	numcodecs-pw-ratio==0.2.1
replace	-	numcodecs-replace==0.1.3
OptZConfig via libpressio	0.15.4+ via 1.0.7+	numcodecs-wasm-pressio==0.1.0b2
QPET-SPERR	-	numcodecs-wasm-qpet-sperr==0.2.0
SPERR	0.8.2+	numcodecs-wasm-sperr==0.2.0
SZ3 [v3.2]	3.2.1+	numcodecs-wasm-sz3==0.7.0
SZ3	3.3.2+	numcodecs-wasm-sz3==0.8.1
ZFP (ROUND)	1.0.1+	numcodecs-wasm-zfp==0.6.0
Zstandard	1.5.7	numcodecs-wasm-zstd==0.4.0

Table A1. Versions of the compressors used in the evaluations in section 5 and appendix B. For each compressor, we provide the published version, if applicable, and the name and version of the corresponding Python package. The published version is suffixed by a '+' if the Python package wraps a compressor code in between published versions.

Compressor	Version	Python package
compression-safeguards	1.0.0b6*	numcodecs-safeguards==0.1.0b6
		numcodecs-wasm==0.2.2
replace	-	numcodecs-replace==0.1.0
Bit Rounding	-	numcodecs-wasm-bit-round==0.4.0
EBCC	0.1.5-alpha	numcodecs-wasm-ebcc==0.3.1a0
PCO	0.4.6	numcodecs-wasm-pco==0.3.0
SPERR	0.8.2+	numcodecs-wasm-sperr==0.2.0
SZ3 [v3.2]	3.2.1+	numcodecs-wasm-sz3==0.7.0
ZFP (ROUND)	1.0.1+	numcodecs-wasm-zfp==0.6.0

Table A2. Versions of the compressors used in ClimateBenchPress in section 5.8. For each compressor, we provide the published version, if applicable, and the name and version of the corresponding Python package. The published version is suffixed by a '+' if the Python package wraps a compressor code in between published versions. (*) Version 1.0.0b6 of the `compression-safeguards` is functionally equivalent to version 1.0.0rc1.

Appendix B: Experimental Evaluation: Auxiliary Examples and Results

B1 Example: NaN missing values

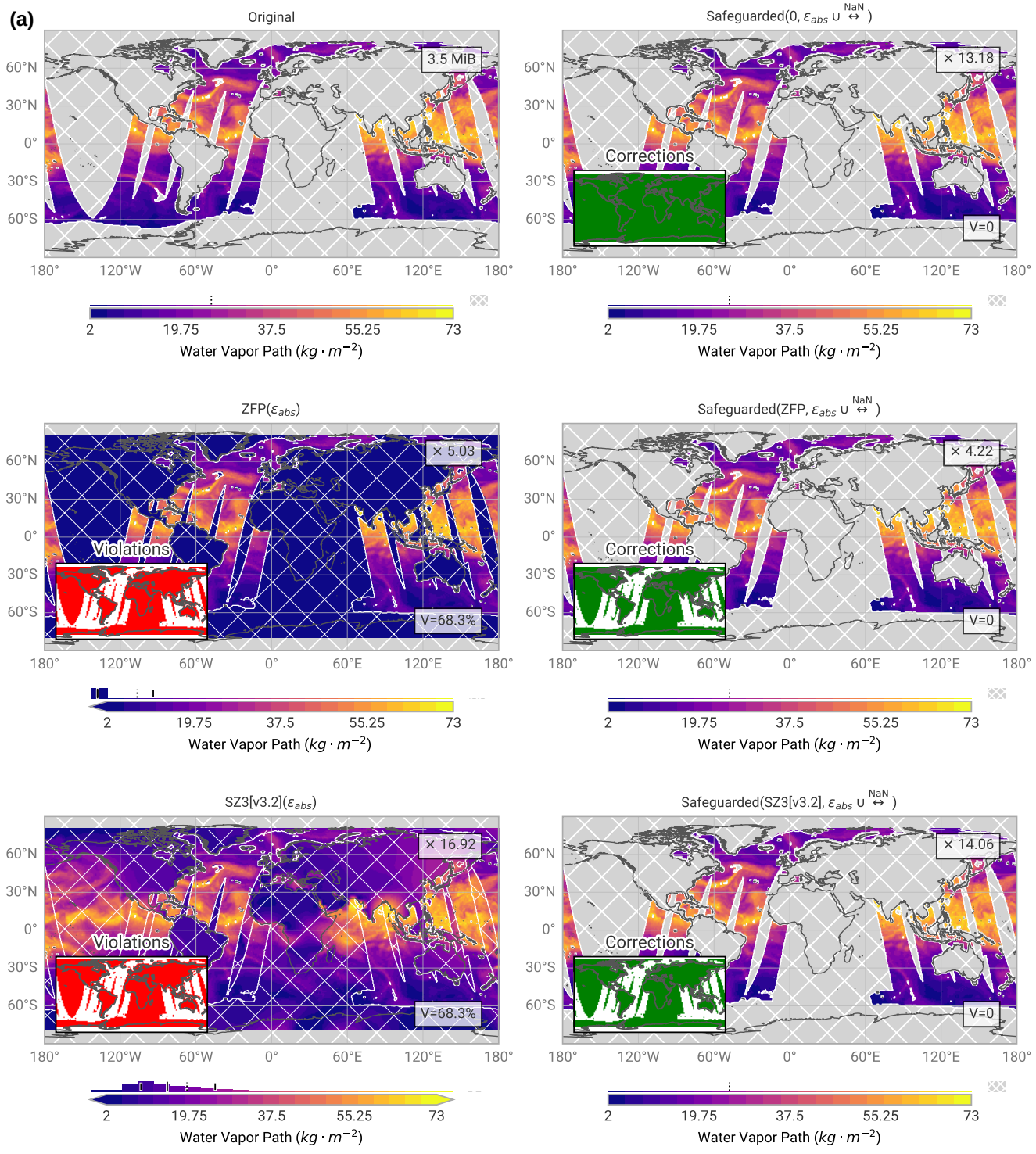
This appendix contains additional results for compressing water vapour path satellite data with NaN missing values using different absolute-error-bounded lossy compressors in section 5.4. Figure B1 and table B1 compare how missing values are preserved, including the results for `ZFP`, `SPERR`, and `OptZConfig`.

`ZFP`, where NaN values trigger undefined behaviour in lossy compression, fills missing values with compiler-chosen values, here zeros. Recently, Lindstrom (2025) has outlined several approaches for how `ZFP` could support encoding special values such as NaN, infinity, or other sentinel values, none of which have been implemented at the time of writing.

`SPERR` raises an exception when compressing data with NaN values, `H5Z-SPERR` (Li, 2026). The HDF5 plugin for `SPERR`, replaces the NaNs with the mean data value before compressing with `SPERR`, and keeps a bitmask of NaN locations to recover the NaNs after decompression. For this example, we similarly replace NaN values with the mean, but leave the corrections up to the safeguards. Since `SPERR` interpolates over this mean fill-value, it produces silhouette-like results like `ZFP`.

Table B1 shows that `OptZConfig` does not find safe configurations for `ZFP`, `SZ3` v3.2 and `SPERR` since none of these compressors can be configured such that they correctly preserve NaN missing values. For `SZ3` v3.3, `OptZConfig` performs like `SZ3` without tuning.

We do not compare with `QPET-SPERR` in this example because it is so tightly coupled with `SPERR` that `QPET-SPERR` also raises an exception when compressing NaN values.


(a)
Original
3.5 MiB
Safeguarded(0, ε_{abs} ∪ $\overset{NaN}{\leftrightarrow}$)
× 13.18
Corrections
V=0
ZFP(ε_{abs})
× 5.03
Violations
V=68.3%
Safeguarded(ZFP, ε_{abs} ∪ $\overset{NaN}{\leftrightarrow}$)
× 4.22
Corrections
V=0
SZ3[v3.2](ε_{abs})
× 16.92
Violations
V=68.3%
Safeguarded(SZ3[v3.2], ε_{abs} ∪ $\overset{NaN}{\leftrightarrow}$)
× 14.06
Corrections
V=0
60°N
30°N
0°
30°S
60°S
180°
120°W
60°W
0°
60°E
120°E
180°
2
19.75
37.5
55.25
73
Water Vapor Path ($kg \cdot m^{-2}$)

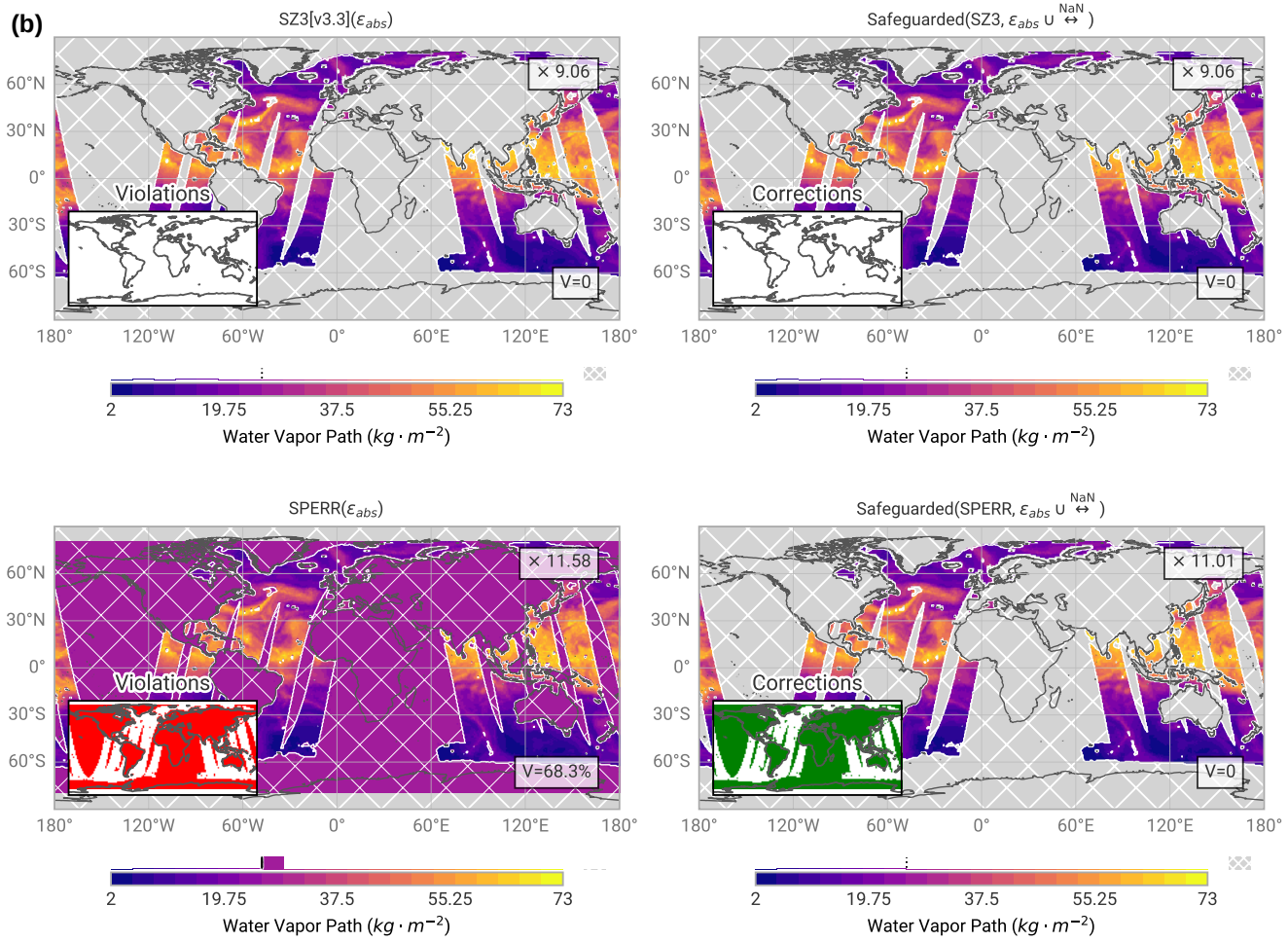


Figure B1. (a)-(b) Decompressed water vapour path maps with NaN missing values from section 5.4. In the left column, the maps for the original data and decompressed data from `ZFP`, `SZ3` [v3.2], `SZ3` [v3.3], and `SPERR` are shown. In the right column, the `compression-safeguarded` equivalents are given. For each compressor, the compression ratio and percentage of violations V are shown as well. For compressors with violations, a map of the violations in red is added. For the safeguarded compressors, a map of the one-shot corrections in dark green is overlaid. Areas with originally missing values are outlined and hatched.

Compressor	Safeguarded	Corrections	L_∞	L_2	V	C	CR	
							same NaN	any NaN
0	$\epsilon_{abs} \cup \overset{\text{NaN}}{\leftrightarrow}$	lossless	0.0	0.0	0	100.0%	× 7.57	× 7.57
		one-shot	0.1	0.05	0	100.0%	× 13.18	× 13.18
ZFP(ϵ_{abs})	-		0.03	0.0047	68.3%			× 5.03
	$\epsilon_{abs} \cup \overset{\text{NaN}}{\leftrightarrow}$	lossless	0.03	0.0047	0	68.3%	× 3.35	× 3.35
		one-shot	0.03	0.0047	0	68.3%	× 3.35	× 4.22
OptZConfig(ZFP)	$\epsilon_{abs} \cup \overset{\text{NaN}}{\leftrightarrow}$		3.8e-06	7.9e-08	68.3%			× 2.05
SZ3[v3.2](ϵ_{abs})	-		0.1	0.057	68.3%			× 16.92
	$\epsilon_{abs} \cup \overset{\text{NaN}}{\leftrightarrow}$	lossless	0.1	0.057	0	68.3%	× 2.31	× 2.31
		one-shot	0.1	0.057	0	68.3%	× 2.31	× **14.06**
OptZConfig(SZ3[v3.2])	$\epsilon_{abs} \cup \overset{\text{NaN}}{\leftrightarrow}$		2.4e-07	3.5e-09	68.3%			× 5.13
SZ3[v3.3](ϵ_{abs})	-		0.1	0.049	0			× 9.06
	$\epsilon_{abs} \cup \overset{\text{NaN}}{\leftrightarrow}$	lossless	0.1	0.049	0	0	× 9.06	× 9.06
		one-shot	0.1	0.049	0	0	× 9.06	× 9.06
OptZConfig(SZ3[v3.3])	$\epsilon_{abs} \cup \overset{\text{NaN}}{\leftrightarrow}$		0.1	0.049	0			× 9.06
SPERR(ϵ_{abs})	-		0.1	0.041	68.3%			× 11.58
	$\epsilon_{abs} \cup \overset{\text{NaN}}{\leftrightarrow}$	lossless	0.1	0.041	0	68.3%	× 2.39	× 2.39
		one-shot	0.1	0.041	0	68.3%	× 2.39	× 11.01
OptZConfig(SPERR)	$\epsilon_{abs} \cup \overset{\text{NaN}}{\leftrightarrow}$		4.8e-07	4.2e-08	68.3%			× 2.59
ZSTD(22)	-		0.0	0.0	0			× 6.0

Table B1. Compression metrics for water vapour path data with NaN missing values from section 5.4. For each compressor, the L_∞ and L_2 errors, the percentage of violations V, and the compression ratio CR are given. If the compressor cannot preserve NaN values, there are as many violations as NaN values, 68.3%, that need to be corrected. For each safeguarded compressor, the percentage of corrections C is given as well. Since the safeguards can be configured to preserve NaN values with the same payload bit pattern or to allow any NaN, compression ratios for both cases are shown. The highest *safe* compression ratio is highlighted in bold.

B2 Example: non-NaN missing values

This appendix extends section 5.4 with an example of compressing data with non-NaN missing values, here a single time step of the energy release component of the fire danger index. The data is on an unstructured grid, in a 1D array with matching 1D latitude and longitude arrays, and contains missing values over sea. It is stored in Gridded Binary (GRIB) format, which typically encodes missing values on disk as the integer-representable value 9999. Since high-level tools like `cfgrib` (Amici, 2018) replace missing values with NaNs when loading the data, we manually read the GRIB file using the lower-level `eccodes` tool (Santoalla and Najm, 2017).

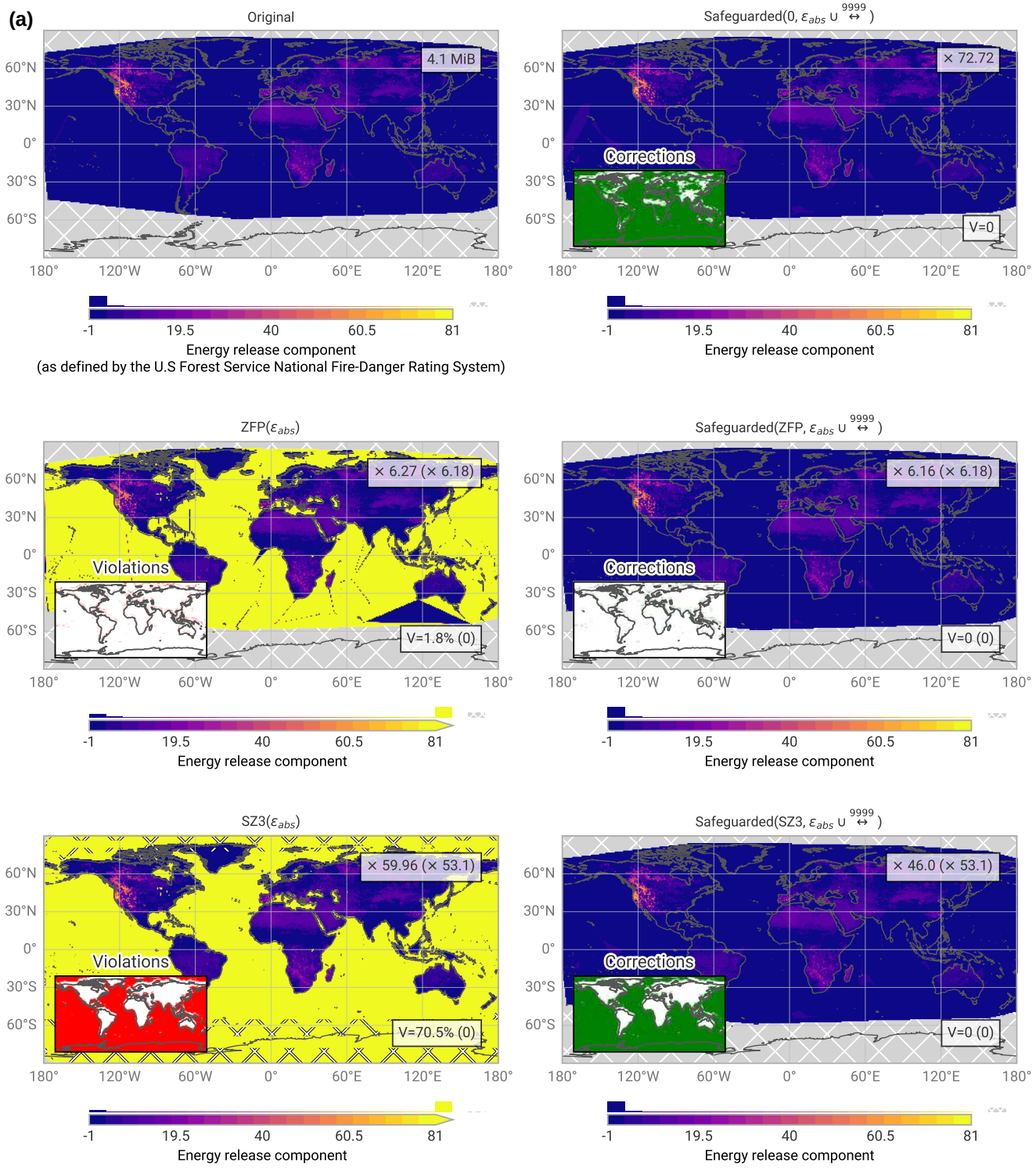
We configure each compressor and the safeguards with an absolute error bound of 1, the step size between the discrete metric values. The safeguards are further configured to preserve the sentinel value 9999, so that missing values are remain missing and no non-missing values take on the sentinel value (see listing C2). `OptZConfig` is configured to maximise compression while eliminating violations.

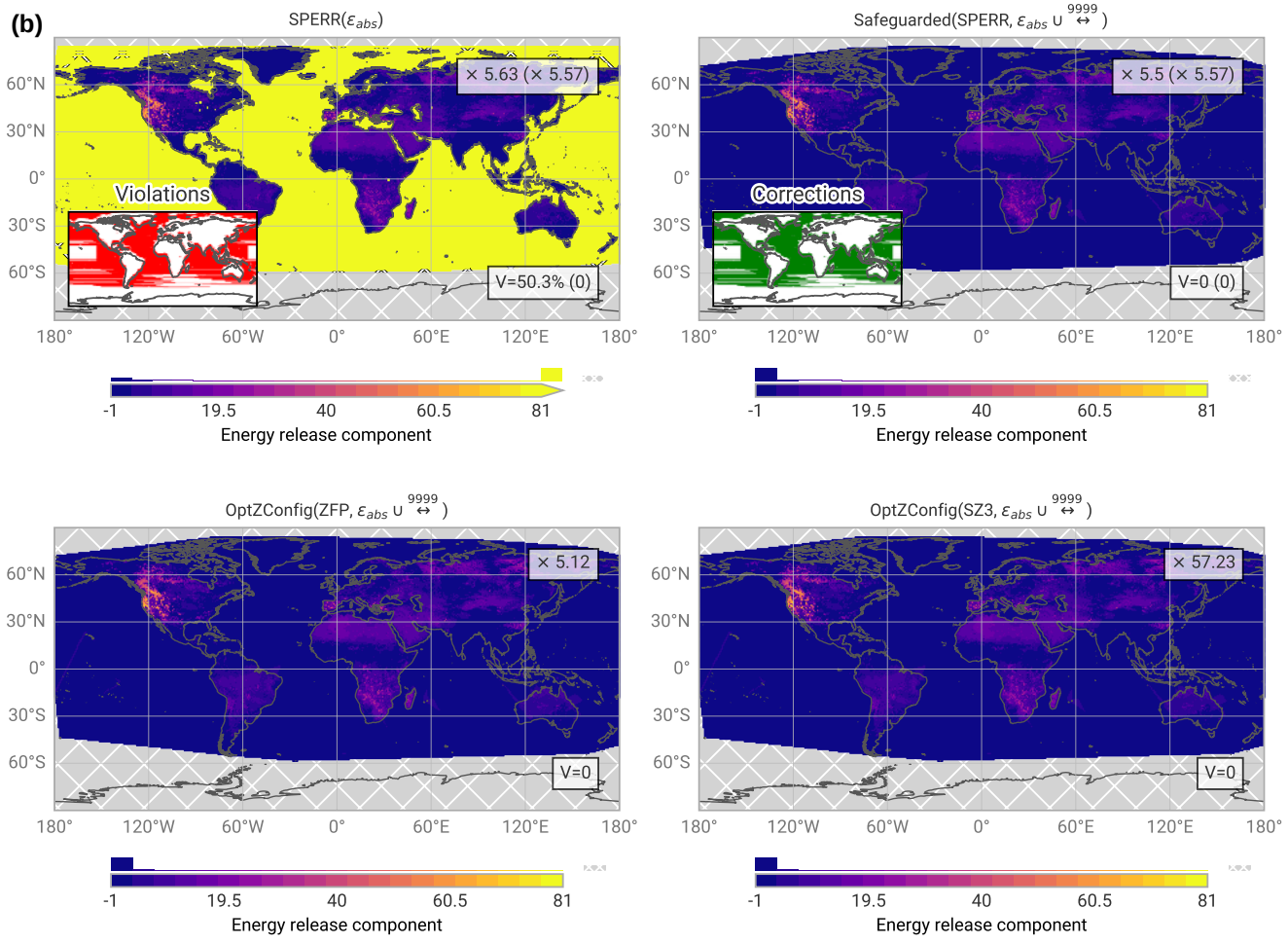
Figure B2 and Table B2 show that none of the lossy compressors handles the non-NaN missing values correctly. `ZFP` preserves most missing values as missing as it only interpolates the large 9999 value into some neighbouring values, owing to its small 4^1 block size in 1D. `SPERR` also preserves some missing values but has significantly more violations. For `SZ3`, seemingly no missing values are preserved correctly, likely because non-missing values get interpolated towards 9999 and originally-missing values only need to change slightly to no longer count as missing. Since `ZFP`, `SZ3`, and `SPERR` have no knowledge that the value 9999 is special, it is expected that they would be unable to preserve sentinel-encoded missing values. Requiring all compressors to support such special values would require significant duplicated development effort.

With the safeguards, the missing values are correctly preserved, at the cost of a small reduction in compression ratio. The safeguarded all-zero-predictor produces the best compression for this simple example.

A specialised meta-compressor that stores a mask of the missing values during compression and restores them during decompression can also be used to preserve missing values, e.g. the `mask_interpolation_compressor_plugin` in the `LibPressio` C++ compression framework (Underwood et al., 2021) or the `numcodecs-mask` Python package. Figure B2 and table B2 show that such a meta-compressor can provide higher per-compressor compression ratios than the safeguards.

Tuning the compressors using `OptZConfig` allows `ZFP` and `SZ3` to correctly preserve the missing values by drastically lowering their error bounds. While safeguarded `ZFP` compresses better than tuned `ZFP`, tuning `SZ3` outperforms both safeguarding and the masking meta-compressor. However, `OptZConfig` fails to find a safe configuration for `SPERR`, since `SPERR` approximates the sentinel even for increasingly small error bounds and does not produce lossless compression as the error bound tends to zero.


(a)
Original
4.1 MiB
Safeguarded(0, ε_{abs} ∪ $\overset{9999}{\leftrightarrow}$)
× 72.72
Corrections
V=0
60°N
30°N
0°
30°S
60°S
180°
120°W
60°W
0°
60°E
120°E
180°
-1
19.5
40
60.5
81
Energy release component
(as defined by the U.S Forest Service National Fire-Danger Rating System)
ZFP(ε_{abs})
× 6.27 (× 6.18)
Violations
V=1.8% (0)
Safeguarded(ZFP, ε_{abs} ∪ $\overset{9999}{\leftrightarrow}$)
× 6.16 (× 6.18)
Corrections
V=0 (0)
SZ3(ε_{abs})
× 59.96 (× 53.1)
Violations
V=70.5% (0)
Safeguarded(SZ3, ε_{abs} ∪ $\overset{9999}{\leftrightarrow}$)
× 46.0 (× 53.1)
Corrections
V=0 (0)


(b)
SPERR(ϵ_{abs})
× 5.63 (× 5.57)
Violations
V=50.3% (0)
Safeguarded(SPERR, $\epsilon_{abs} \cup \overset{9999}{\leftrightarrow}$)
× 5.5 (× 5.57)
Corrections
V=0 (0)
OptZConfig(ZFP, $\epsilon_{abs} \cup \overset{9999}{\leftrightarrow}$)
× 5.12
V=0
OptZConfig(SZ3, $\epsilon_{abs} \cup \overset{9999}{\leftrightarrow}$)
× 57.23
V=0
60°N
30°N
0°
30°S
60°S
180°
120°W
60°W
0°
60°E
120°E
180°
-1
19.5
40
60.5
81
Energy release component

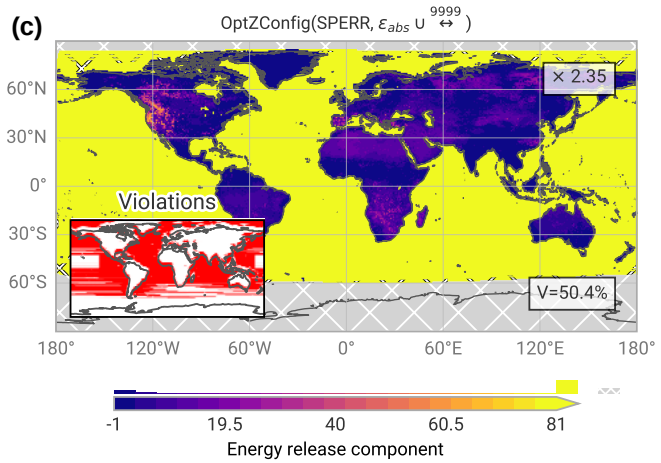


Figure B2. (a)-(c) Decompressed energy release component maps with non-NaN missing values. In the left column, the maps for the original data and decompressed data from `ZFP`, `SZ3`, and `SPERR` are shown in the first four rows. In the right column, the `compression-safeguarded` equivalents are given, which specifically preserve the missing value sentinel. The last two rows present the maps for the `OptZConfig`-wrapped compressors, which are tuned to preserve the missing value. For each compressor, the compression ratio and percentage of violations V are shown as well. For compressors with violations, a map of the violations in red is added. For the safeguarded compressors, a map of the one-shot corrections in dark green is overlaid. Areas with originally missing values are hatched. The results from first wrapping `ZFP`, `SZ3`, and `SPERR` in the `numcodecs-mask` meta-compressor are given in parentheses.

Compressor	Meta	Safeguarded	Corrections	L_∞	L_2	V	C	CR
0	-	$\epsilon_{abs} \cup \overset{9999}{\leftrightarrow}$	lossless	1.0	0.12	0	87.6%	× 62.89
			one-shot	1.0	0.31	0	87.6%	× **72.72**
ZFP(ϵ_{abs})	-	-		0.75	0.16	1.8%		× 6.27
		$\epsilon_{abs} \cup \overset{9999}{\leftrightarrow}$	lossless	0.75	0.15	0	1.8%	× 6.16
			one-shot	0.75	0.15	0	1.8%	× 6.16
	$\overset{9999}{\leftrightarrow}$	-		0.75	0.15	0		× 6.18
		$\epsilon_{abs} \cup \overset{9999}{\leftrightarrow}$	one-shot	0.75	0.15	0	0	× 6.18
OptZConfig(ZFP)	$\overset{9999}{\leftrightarrow}$			0.0	0.0	0		× 5.12
SZ3(ϵ_{abs})	-	-		1.0	0.89	70.5%		× 59.96
		$\epsilon_{abs} \cup \overset{9999}{\leftrightarrow}$	lossless	1.0	0.31	0	70.5%	× 46.0
			one-shot	1.0	0.31	0	70.5%	× 46.0
	$\overset{9999}{\leftrightarrow}$	-		1.0	0.31	0		× 53.1
		$\epsilon_{abs} \cup \overset{9999}{\leftrightarrow}$	one-shot	1.0	0.31	0	0	× 53.1
OptZConfig(SZ3)	$\overset{9999}{\leftrightarrow}$			6.4e-05	1.3e-06	0		× 57.23
SPERR(ϵ_{abs})	-	-		0.75	0.38	50.3%		× 5.63
		$\epsilon_{abs} \cup \overset{9999}{\leftrightarrow}$	lossless	0.75	0.24	0	50.3%	× 5.5
			one-shot	0.75	0.24	0	50.3%	× 5.5
	$\overset{9999}{\leftrightarrow}$	-		0.75	0.24	0		× 5.57
		$\epsilon_{abs} \cup \overset{9999}{\leftrightarrow}$	one-shot	0.75	0.24	0	0	× 5.57
OptZConfig(SPERR)	$\overset{9999}{\leftrightarrow}$			7.5e-07	3.9e-07	50.4%		× 2.35
ZSTD(22)	-	-		0.0	0.0	0		× 49.94

Table B2. Compression metrics for energy release component data with the non-NaN missing value 9999. The table compares the results of different absolute-error-bounded lossy compressors, with and without a masking meta-compressor, with and without missing value safeguards. For each compressor, the L_∞ and L_2 errors, the percentage of violations V, and the compression ratio CR are given. For each safeguarded compressor, the percentage of corrections C is given as well. The highest *safe* compression ratio is highlighted in bold.

B3 Example: bias in the error distribution

This appendix investigates if the `compression-safeguards` implementation propagates or introduces bias in the safeguarded error distribution. We use several synthetic approximation distributions for accumulated precipitation as examples. The safeguards are configured to bound the absolute error to $0.1\mathrm{mm\,h^{-1}}$ or below, and we examine the per-grid-cell mean-in-time and the global mean biases.

Figure B3 show that the `compression-safeguards`, in the absence of a safeguard for unbiased and symmetric errors, do not guard against bias in the compression error. For instance, the constant zero approximation produces negatively biased corrections for the non-negative precipitation dataset, as many values near zero lie within the error bound and thus require no corrections. Biased approximations that still fulfil the safety requirements are not corrected and propagate their bias into the safeguarded compression error. For this particular dataset, using an unbiased mean approximation does not resolve the issue but just makes the bias positive. The bias can, however, be reduced by making the approximation itself unbiased, e.g. by adding uniform random noise. While a noisy approximation produces the lowest Moran's I score in Table B3, the bias is still unexpectedly often outside the expected range, and the required additional corrections reduce the compression ratio. This example thus highlights the need for future work to provide a way for users to control the bias introduced by lossy compression and safeguarding.

Safeguarded(..., ϵ_{abs})	L_∞	L_2	bias	I	$E_{>95\%}$	V	C
0	0.1	0.031	-0.014	0.45	68.1%	0	15.6%
$\overline{PR}$	0.1	0.084	0.068	0.77	98.0%	0	9.5%
$PR + (\epsilon_{abs} \,/\, 2)$	0.05	0.05	0.05	0.48	100.0%	0	0
$PR + \mathrm{U}(\pm(\epsilon_{abs} \cdot 3))$	0.1	0.042	-0.0077	0.074	52.1%	0	66.7%

Table B3. Compression metrics for different synthetic approximation distributions. For each safeguarded compressor, the L_∞ and L_2 errors over the precipitation PR, the global bias $E[\hat{PR} - PR]$, the global Moran's I score, the percentage of violations V and corrections C are given. Moran's I is a measure of spatial autocorrelation in range $[-1; +1]$, where $|I| \to 0$ indicates no correlation (Moran, 1950). Here, we compute Moran's I for the error distribution $\hat{PR} - PR$. Finally, the percentage of latitude-longitude pairs where the mean error is unexpectedly large, $E_{>95\%}$, is given. A location has an unexpectedly large error when the mean error is outside two standard deviations of the expected spread of the Irwin Hall distribution that models uniformly distributed i.i.d. compression errors.

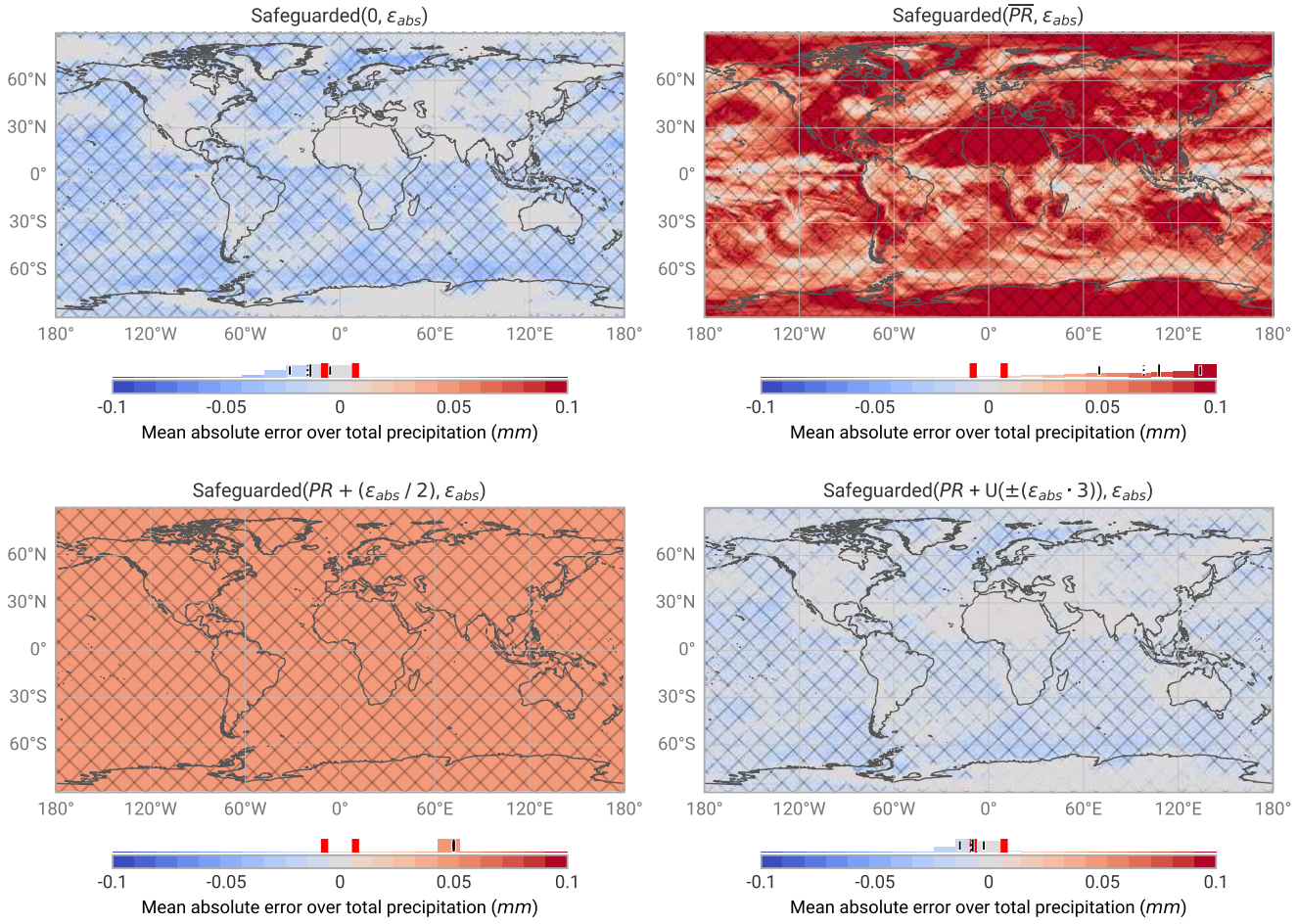


Figure B3. Mean error / bias maps of the compression error produced by the `compression-safeguards` for different synthetic approximations for three days of hourly total precipitation: constant zero in the top left, constant mean in the top right, biased by half the error bound in the bottom left, and with added uniform random noise in the bottom right. The expected range of biases for uniformly distributed i.i.d. compression errors, computed as two standard deviations of the Irwin Hall distribution, is shown as red bars over the colour bar. Each error map is hatched where the mean error is outside this expected range.

B4 Example: logarithm as a pointwise quantity of interest

In this appendix, we introduce safeguarding pointwise quantities of interest by bounding the absolute error over the log-scale visualisation of specific humidity. $\log(x)$ is one of the simplest quantities of interest since it can be preserved in various ways:

(a) a compressor can directly support bounding an absolute error over the $\log(x)$ quantity of interest, or

(b) it can support a ratio error bound over the data to bound the logarithmic error (see item 3), or

(c) an absolute-error-bounding compressor can be wrapped in Liang et al. (2018)'s meta-compressor to bound the ratio error by compressing the logarithm of the data while taking care of non-positive data values. This meta-compressor is implemented as the `pw_rel_compressor_plugin` in the `LibPressio` C++ compression framework (Underwood et al., 2021), and in the `numcodecs-pw-ratio` Python package.

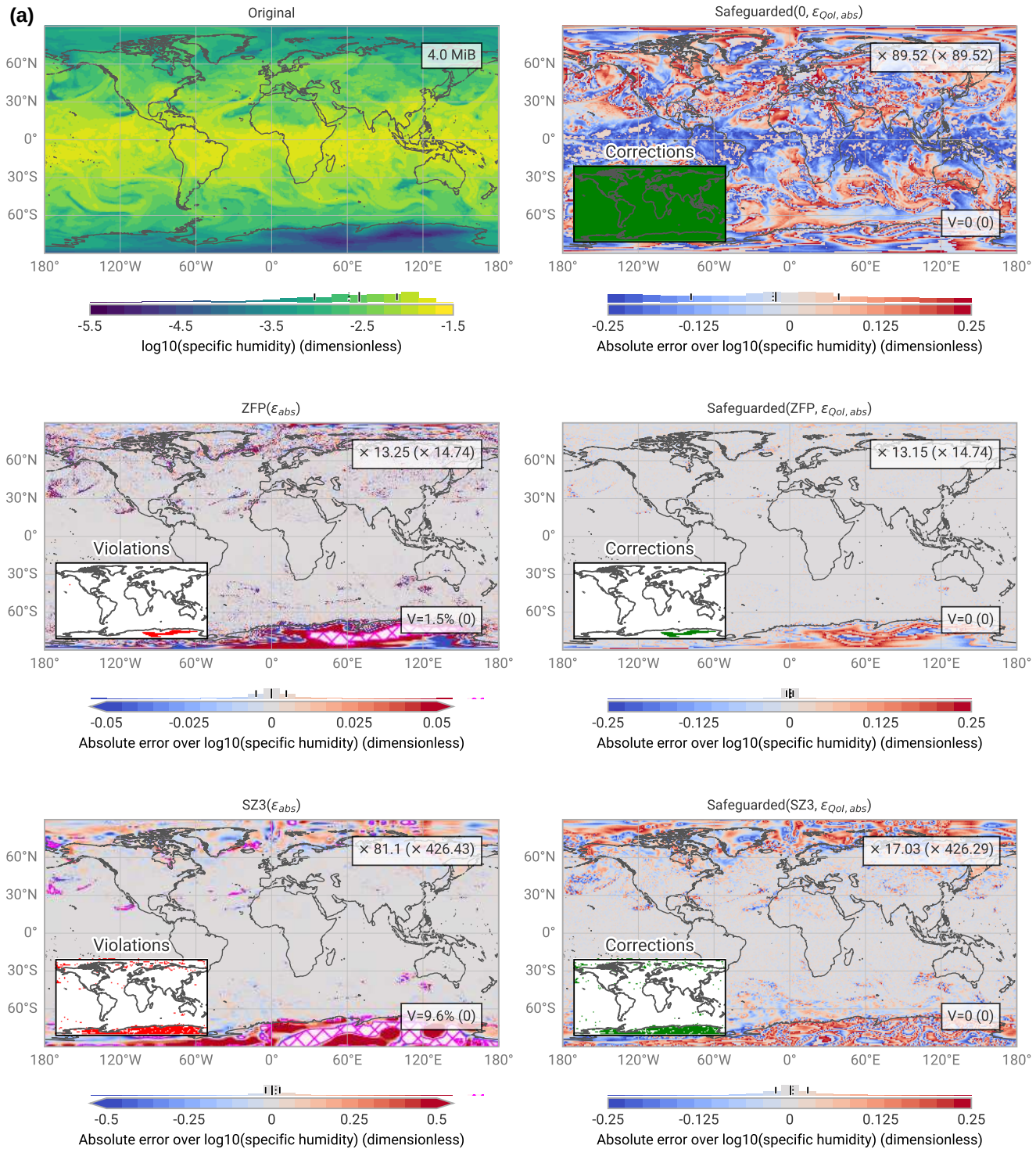
Support for bounding the logarithmic error is sparse amongst error-bounded compressors: `ZFP` requires export mode to bound the relative error for normal-only values, `SZ3` only supports a range-relative / normalised absolute error bound, and `SPERR` lacks support entirely. Therefore, we fall back to configuring each compressor with a reasonable absolute error bound of $5{\cdot}10^{-4}\mathrm{kg\,kg^{-1}}$, but also compare the results from using the `numcodecs-pw-ratio` meta-compressor. We configure `QPET-SPERR` and the safeguards to directly bound the pointwise absolute error on the decimal logarithm to 0.25 (see Listing C3). `OptZConfig` is configured to maximise compression while eliminating violations.

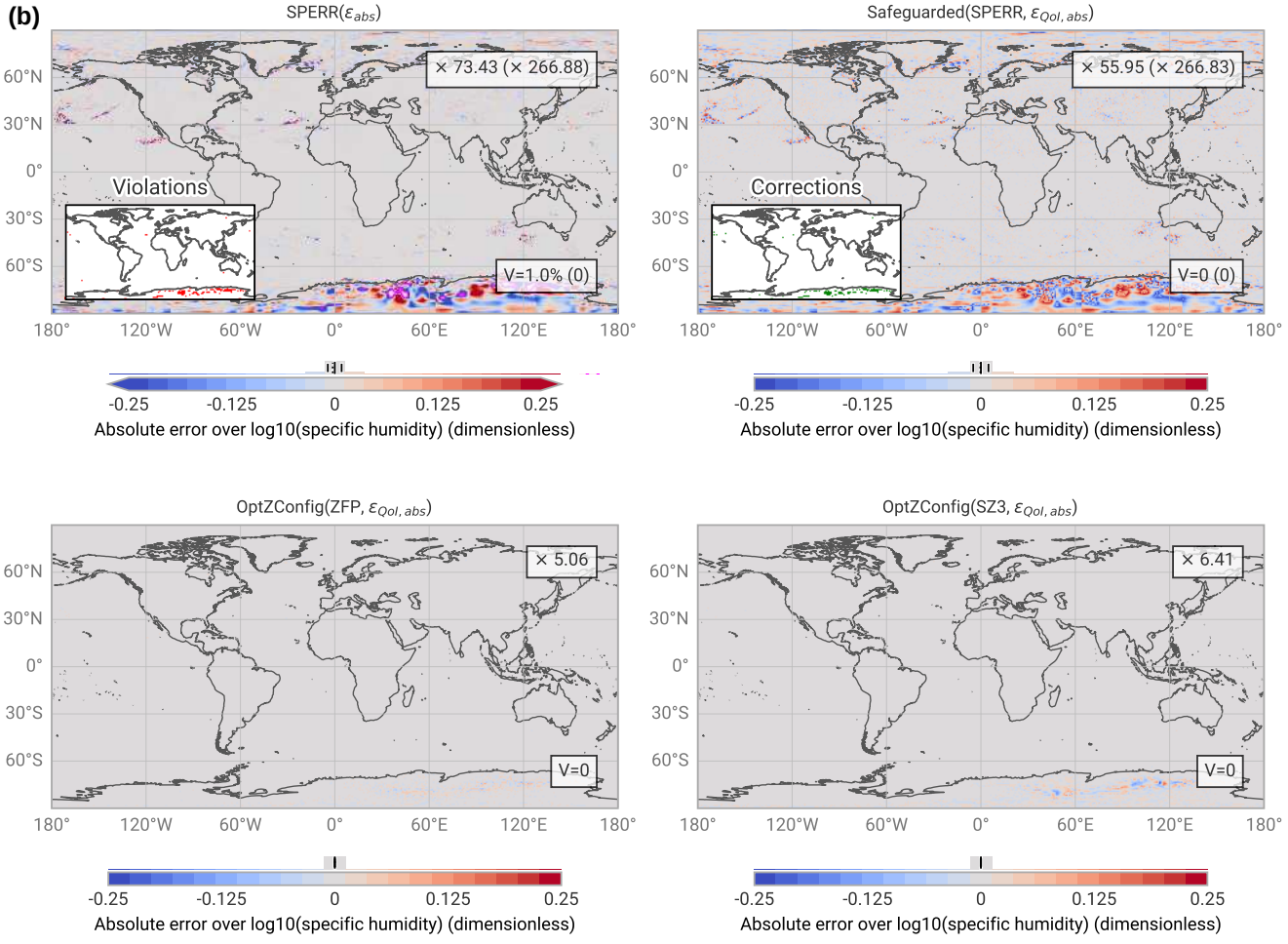
Figure B4 shows that data-error-bounded `ZFP`, `SZ3`, and `SPERR` all fail to preserve the logarithm, since they produce negative values whose real logarithm is undefined. Safeguarding these compressor eliminates the violations, though compression ratios decrease, particularly for `SZ3`. `OptZConfig` and `QPET-SPERR` also correctly preserve the quantity of interest.

The violations can also be eliminated by equivalently bounding the data ratio error with $\epsilon_{ratio} = 10^{0.25}$ using the meta-compressor. In this case, the safeguards do not need to produce any corrections, and compression ratios increase drastically, as can be seen in Figure B4 and table B4. While the safeguarded all-zero predictor produces the best compression without using the meta-compressor, whether using a ratio error bound over the data or an absolute error bound over the quantity of interest, it is significantly outperformed by the `numcodecs-pw-ratio`-wrapped `SZ3`.

Safeguarded `SPERR` outperforms `QPET-SPERR`, likely because `QPET-SPERR` needs to find a near-zero global absolute error bound that avoids negative values. Similarly, all safeguarded compressors outperform the compressors tuned with `OptZ-Config`. While we were unable to compare with `QPET-SZ` (see section 5.2), which derives per-point absolute error bounds, we would expect it to perform better than `QPET-SPERR`, possibly as well as the `numcodecs-pw-ratio`-wrapped `SZ3`.

This example highlights that, in some simple cases with a single safety requirement, specialised approaches such as meta-compressors can extend the functionality of existing compressors while achieving high compression ratios. However, it may not always be possible to compose several such specialised approaches, whereas the `compression-safeguards` support composing arbitrary safety requirements. Importantly, approaches like the `numcodecs-pw-ratio` meta-compressor can be used together with the `compression-safeguards`, providing the best of both worlds with extended compressor functionality, higher compression ratios, and guaranteed safety.


(a)
Original
4.0 MiB
60°N
30°N
0°
30°S
60°S
180°
120°W
60°W
0°
60°E
120°E
180°
-5.5
-4.5
-3.5
-2.5
-1.5
log10(specific humidity) (dimensionless)
Safeguarded(0, $\epsilon_{QoI,abs}$)
× 89.52 (× 89.52)
Corrections
V=0 (0)
-0.25
-0.125
0
0.125
0.25
Absolute error over log10(specific humidity) (dimensionless)
ZFP(ϵ_{abs})
× 13.25 (× 14.74)
Violations
V=1.5% (0)
-0.05
-0.025
0
0.025
0.05
Absolute error over log10(specific humidity) (dimensionless)
Safeguarded(ZFP, $\epsilon_{QoI,abs}$)
× 13.15 (× 14.74)
Corrections
V=0 (0)
-0.25
-0.125
0
0.125
0.25
Absolute error over log10(specific humidity) (dimensionless)
SZ3(ϵ_{abs})
× 81.1 (× 426.43)
Violations
V=9.6% (0)
-0.5
-0.25
0
0.25
0.5
Absolute error over log10(specific humidity) (dimensionless)
Safeguarded(SZ3, $\epsilon_{QoI,abs}$)
× 17.03 (× 426.29)
Corrections
V=0 (0)
-0.25
-0.125
0
0.125
0.25
Absolute error over log10(specific humidity) (dimensionless)


(b)
SPERR(ε_{abs})
× 73.43 (× 266.88)
Violations
V=1.0% (0)
Safeguarded(SPERR, $\varepsilon_{Qol,\,abs}$)
× 55.95 (× 266.83)
Corrections
V=0 (0)
OptZConfig(ZFP, $\varepsilon_{Qol,\,abs}$)
× 5.06
V=0
OptZConfig(SZ3, $\varepsilon_{Qol,\,abs}$)
× 6.41
V=0
60°N
30°N
0°
30°S
60°S
180°
120°W
60°W
0°
60°E
120°E
180°
-0.25
-0.125
0
0.125
0.25
Absolute error over log10(specific humidity) (dimensionless)

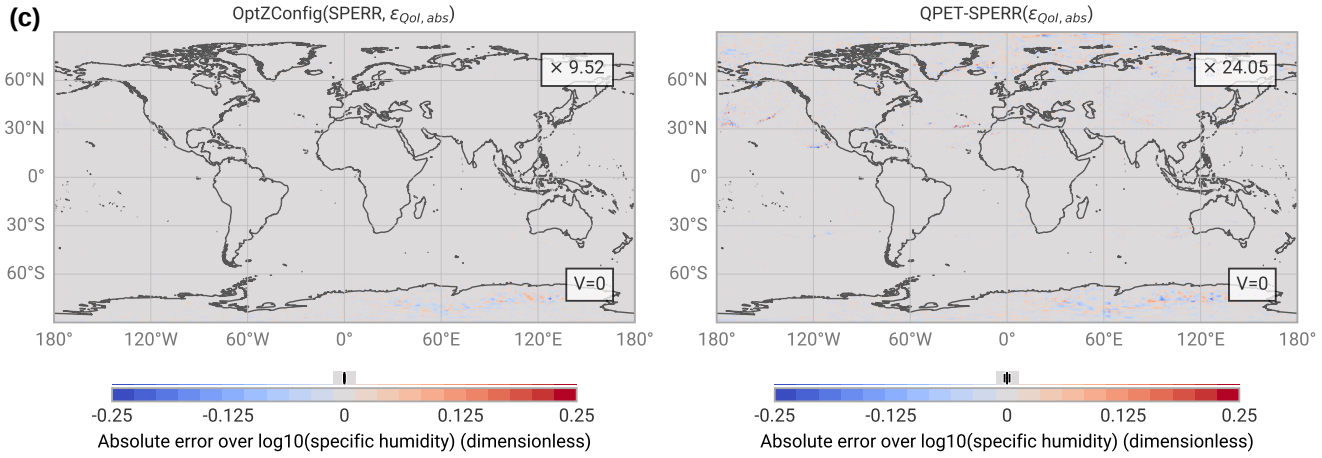


Figure B4. (a)-(c) Error maps for the logarithm over compressed specific humidity. The left column shows in the first four rows the maps for the original data and the errors over the logarithm of the decompressed data from `ZFP`, `SZ3`, and `SPERR`. In the right column, the `compression-safeguarded` equivalents are given, which have been configured to directly preserve the logarithmic scale. The last two rows present the maps for the `OptZConfig`-wrapped compressors, which are tuned to bound the error over the data logarithm, and for `QPET-SPERR`, which is configured like the safeguards. For compressors with violations, a map of the violations in red is added. For the safeguarded compressors, a map of the one-shot corrections in dark green is overlaid. Areas where lossy compression introduces non-positive specific humidity values, whose logarithm is undefined, are highlighted with pink outlines and hatching. The results for bounding the ratio error by first wrapping `ZFP`, `SZ3`, and `SPERR` in the `numcodecs-pw-ratio` meta-compressor are given in parentheses.

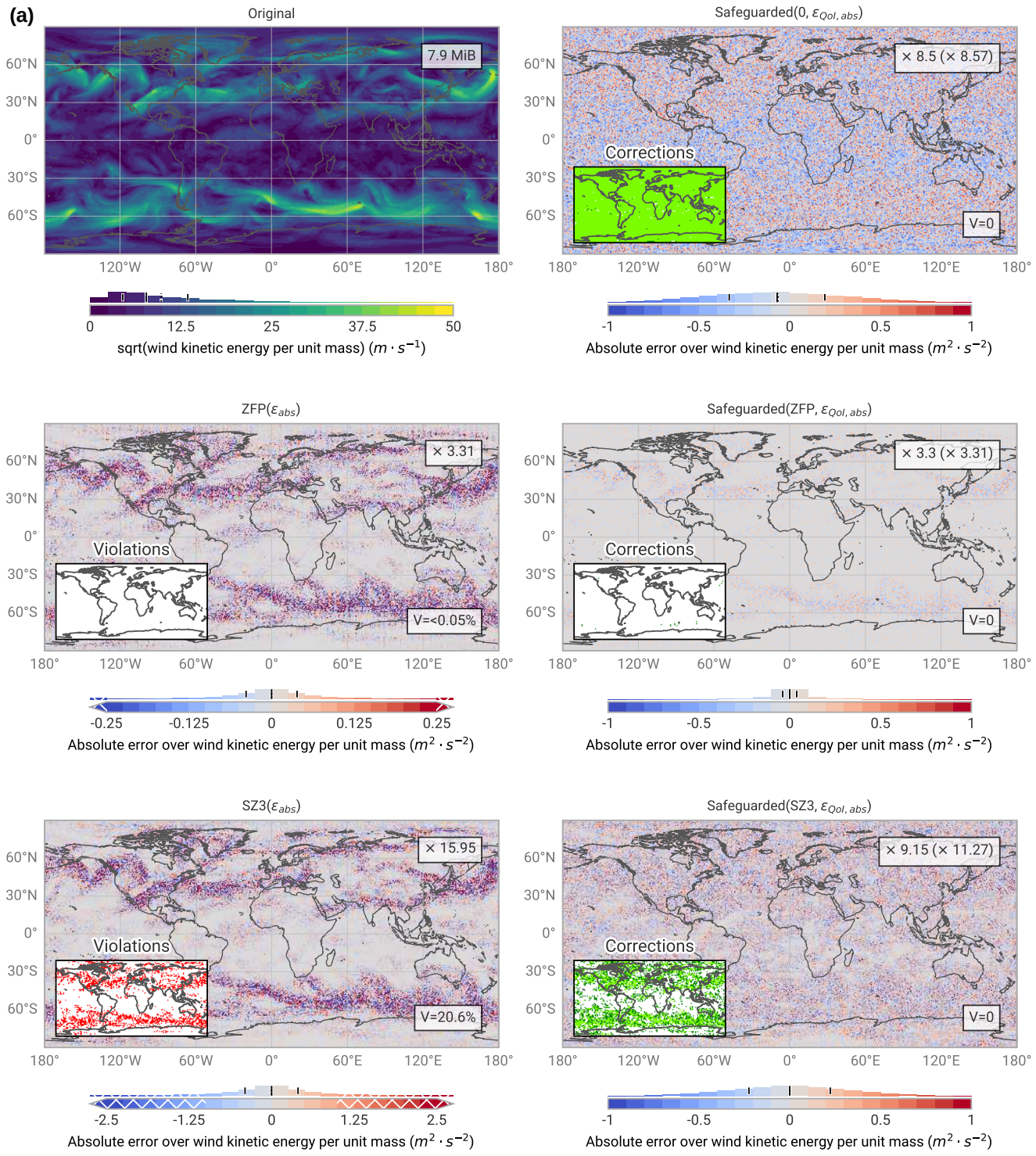
Compressor	(Bound)	Safeguarded	Corrections	$L_\infty(\hat{q})$	$L_\infty(\log_{10}(\hat{q}))$	$L_2(\log_{10}(\hat{q}))$	V	C	CR
0		$\epsilon_{QoI,abs}$	lossless	0.0	0.0	0.0	0	100.0%	× 2.92
			one-shot	0.013	0.25	0.136	0	100.0%	× 89.52
		ϵ_{ratio}	lossless	0.0	0.0	0.0	0	100.0%	× 2.92
			one-shot	0.013	0.25	0.136	0	100.0%	× 89.52
ZFP	ϵ_{abs}	-		0.00029	NaN [1.1]	NaN [0.0321]	1.5%		× 13.25
		$\epsilon_{QoI,abs}$	lossless	0.00029	0.25	0.0279	0	1.5%	× 12.92
			one-shot	0.00029	0.25	0.0328	0	1.5%	× 13.15
	$\epsilon_{ratio} \to \epsilon_{abs}$	-		0.0025	0.0943	0.0133	0		× 14.74
		$\epsilon_{QoI,abs}$	one-shot	0.0025	0.0943	0.0133	0	0	× 14.74
OptZConfig(ZFP)	ϵ_{abs}	$\epsilon_{QoI,abs}$		9.4e-06	0.137	0.00346	0		× 5.06
SZ3	ϵ_{abs}	-		0.0005	NaN [4.74]	NaN [0.165]	9.6%		× 81.1
		$\epsilon_{QoI,abs}$	lossless	0.0005	0.25	0.0601	0	9.6%	× 9.4
			one-shot	0.0005	0.25	0.0737	0	9.6%	× 17.03
	$\epsilon_{ratio} \to \epsilon_{abs}$	-		0.01	0.25	0.0684	0		× **426.43**
		$\epsilon_{QoI,abs}$	one-shot	0.01	0.25	0.0684	0	0	× 426.29
OptZConfig(SZ3)	ϵ_{abs}	$\epsilon_{QoI,abs}$		3.6e-06	0.229	0.00606	0		× 6.41
SPERR	ϵ_{abs}	-		0.0005	NaN [4.28]	NaN [0.0615]	1.0%		× 73.43
		$\epsilon_{QoI,abs}$	lossless	0.0005	0.25	0.0353	0	1.0%	× 36.19
			one-shot	0.0005	0.25	0.038	0	1.0%	× 55.95
	$\epsilon_{ratio} \to \epsilon_{abs}$	-		0.006	0.247	0.0313	0		× 266.88
		$\epsilon_{QoI,abs}$	one-shot	0.006	0.247	0.0313	0	0	× 266.83
OptZConfig(SPERR)	ϵ_{abs}	$\epsilon_{QoI,abs}$		1.4e-05	0.226	0.0049	0		× 9.52
QPET-SPERR	ϵ_{abs}	$\epsilon_{QoI,abs}$		0.00027	0.25	0.0129	0		× 24.05
ZSTD(22)		-		0.0	0.0	0.0	0		× 2.23

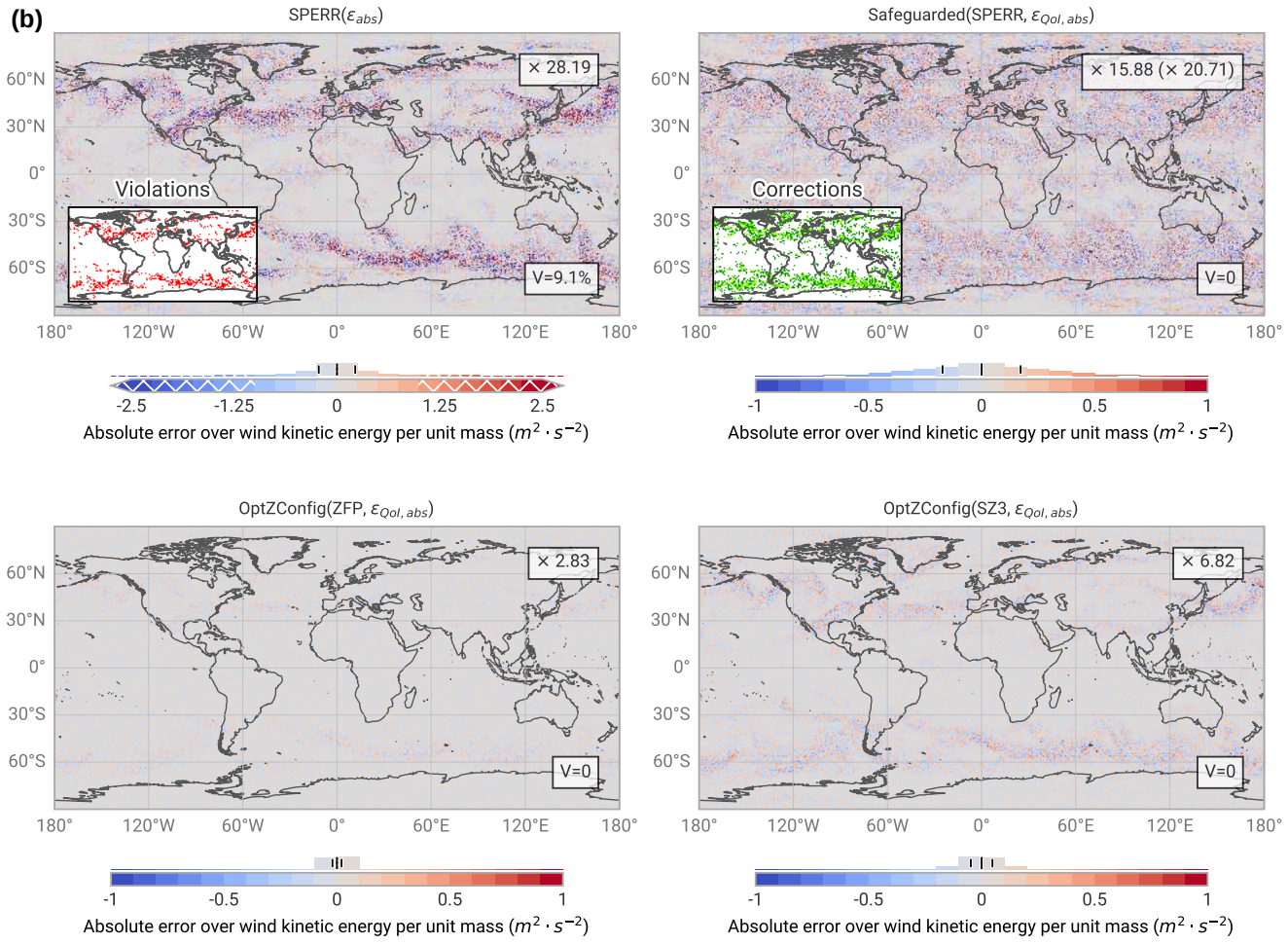
Table B4. Compression metrics for specific humidity data q. The table compares the results of different absolute-error-bounded lossy compressors, with and without the pointwise ratio error bound meta-compressor, with and without safeguards for the specific humidity logarithm. For each compressor, the L_∞ errors over q and $\log_{10}(q)$ as well as the L_2 error over $\log_{10}(q)$, the percentage of violations V, and the resulting compression ratio CR are given. For each safeguarded compressor, the percentage of corrections C is provided as well. The comparison also includes the safeguarded zero-predictor, first with a $\log_{10}(q)$ quantity of interest and second with a ratio error bound, which produce equivalent results. The highest *safe* compression ratio is highlighted in bold.

B5 Example: kinetic energy as a multi-variable pointwise quantity of interest

This appendix contains additional results for computing the kinetic energy over compressed wind u and v components using different absolute-data-error-bounded lossy compressors in section 5.5. Figure B5 and table B5 compare how the derived kinetic energy is preserved, including the results for `ZFP`, `SZ3`, and `OptZConfig`.

Figure B5 shows that all data-error bounded compressors violate the error bound over the derived kinetic energy, which are resolved with using either the safeguards, `OptZConfig`, or `QPET-SPERR`. Table B5 details that iteratively-reduced corrections improve the safeguarded compression ratio, by requiring fewer corrections as seen in the corrections maps in Figure B5, and producing higher errors that only just stay below the error bound, thus making better use of the safety constraints. Even with one-shot corrections, the safeguards outperform `OptZConfig` and `QPET-SPERR`. `QPET-SPERR` also compresses better than `SPERR` tuned with `OptZConfig`, which indicates that correcting violations or outliers, instead of finding a smaller safe error bound, is likely superior for a non-linear quantity of interest like kinetic energy.


(a)
Original
7.9 MiB
60°N
30°N
0°
30°S
60°S
120°W
60°W
0°
60°E
120°E
180°
0
12.5
25
37.5
50
sqrt(wind kinetic energy per unit mass) ($m \cdot s^{-1}$)
Safeguarded(0, $\varepsilon_{QoI,abs}$)
× 8.5 (× 8.57)
Corrections
V=0
-1
-0.5
0
0.5
1
Absolute error over wind kinetic energy per unit mass ($m^2 \cdot s^{-2}$)
ZFP(ε_{abs})
× 3.31
Violations
V=<0.05%
-0.25
-0.125
0
0.125
0.25
Absolute error over wind kinetic energy per unit mass ($m^2 \cdot s^{-2}$)
Safeguarded(ZFP, $\varepsilon_{QoI,abs}$)
× 3.3 (× 3.31)
Corrections
V=0
-1
-0.5
0
0.5
1
Absolute error over wind kinetic energy per unit mass ($m^2 \cdot s^{-2}$)
SZ3(ε_{abs})
× 15.95
Violations
V=20.6%
-2.5
-1.25
0
1.25
2.5
Absolute error over wind kinetic energy per unit mass ($m^2 \cdot s^{-2}$)
Safeguarded(SZ3, $\varepsilon_{QoI,abs}$)
× 9.15 (× 11.27)
Corrections
V=0
-1
-0.5
0
0.5
1
Absolute error over wind kinetic energy per unit mass ($m^2 \cdot s^{-2}$)


(b)
SPERR(ϵ_{abs})
× 28.19
Violations
V=9.1%
60°N
30°N
0°
30°S
60°S
180°
120°W
60°W
0°
60°E
120°E
180°
-2.5
-1.25
0
1.25
2.5
Absolute error over wind kinetic energy per unit mass ($m^2 \cdot s^{-2}$)
Safeguarded(SPERR, $\epsilon_{Qol,\,abs}$)
× 15.88 (× 20.71)
Corrections
V=0
-1
-0.5
0
0.5
1
Absolute error over wind kinetic energy per unit mass ($m^2 \cdot s^{-2}$)
OptZConfig(ZFP, $\epsilon_{Qol,\,abs}$)
× 2.83
V=0
Absolute error over wind kinetic energy per unit mass ($m^2 \cdot s^{-2}$)
OptZConfig(SZ3, $\epsilon_{Qol,\,abs}$)
× 6.82
V=0
Absolute error over wind kinetic energy per unit mass ($m^2 \cdot s^{-2}$)

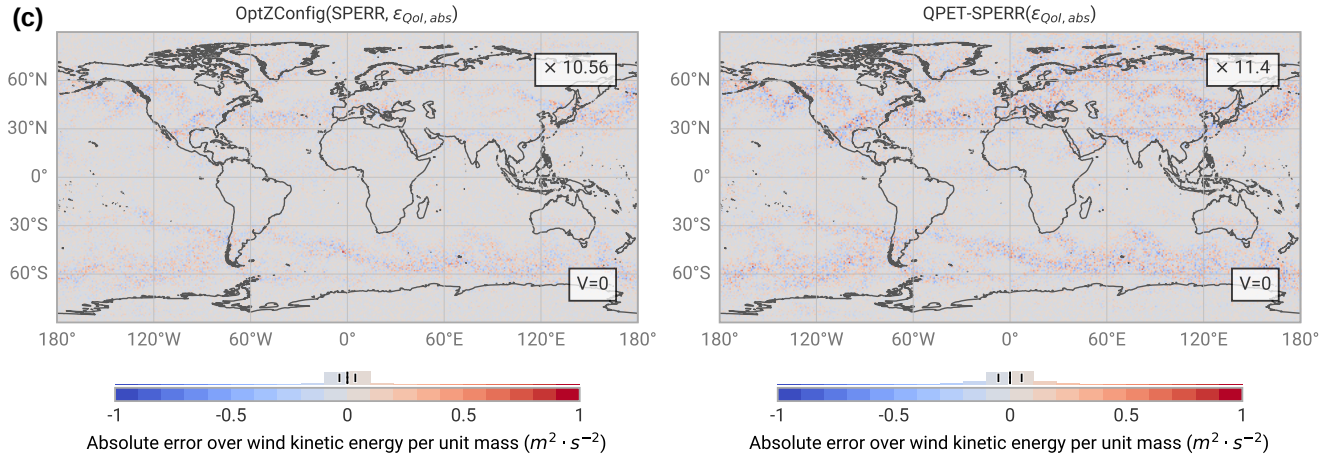


Figure B5. (a)-(c) Error maps for kinetic energy derived from compressed wind components from section 5.5. The left column shows in the first four rows the maps for the original data and the errors for the kinetic energy of the decompressed data from `ZFP`, `SZ3`, and `SPERR`. In the right column, the `compression-safeguarded` equivalents are given, which have been configured to bound the absolute error over the kinetic energy. The last two rows present the maps from the `OptZConfig`-wrapped compressors, which are tuned to similarly bound the error, and for `QPET-SPERR`, which is configured like the safeguards. For the safeguarded compressors, the compression ratios from iteratively-reduced corrections are given in parentheses. For compressors with violations, a map of the violations in red is added. For the safeguarded compressors, a map of the corrections is overlaid. The map is light green where the there are both the one-shot and iterative corrections and dark green where only the one-shot corrections are needed, i.e. where the iterative method was able to remove corrections.

Compressor	Safeguarded	Corrections	$L_\infty(\hat{u})$	$L_\infty(\hat{v})$	$L_\infty(\hat{\mathrm{KE}})$	$L_2(\hat{\mathrm{KE}})$	V	C	CR
0	$\epsilon_{QoI,abs}$	lossless	1.41	1.41	1.0	0.199	0	89.5%	× 2.98
		one-shot	1.0	0.999	0.999	0.375	0	89.7%	× 8.5
		iterative	1.41	1.73	1.0	0.4	0	87.1%	× 8.57
ZFP(ϵ_{abs})	-		0.028	0.0349	1.19	0.103	<0.05%		× 3.31
	$\epsilon_{QoI,abs}$	lossless	0.028	0.0349	0.998	0.103	0	<0.05%	× 3.31
		one-shot	0.028	0.0349	0.926	0.1	0	0.1%	× 3.3
		iterative	0.028	0.0349	0.998	0.103	0	<0.05%	× 3.31
OptZConfig(ZFP)	$\epsilon_{QoI,abs}$		0.0145	0.0148	0.641	0.0533	0		× 2.83
SZ3(ϵ_{abs})	-		0.125	0.125	9.29	0.985	20.6%		× 15.95
	$\epsilon_{QoI,abs}$	lossless	0.125	0.125	1.0	0.428	0	13.1%	× 5.83
		one-shot	0.125	0.125	0.998	0.327	0	26.8%	× 9.15
		iterative	0.125	0.125	1.0	0.451	0	11.9%	× 11.27
OptZConfig(SZ3)	$\epsilon_{QoI,abs}$		0.0123	0.0123	0.971	0.111	0		× 6.82
SPERR(ϵ_{abs})	-		0.125	0.125	8.36	0.62	9.1%		× 28.19
	$\epsilon_{QoI,abs}$	lossless	0.125	0.125	1.0	0.359	0	5.3%	× 12.3
		one-shot	0.125	0.125	0.999	0.277	0	13.8%	× 15.88
		iterative	0.125	0.125	1.0	0.372	0	5.0%	× **20.71**
OptZConfig(SPERR)	$\epsilon_{QoI,abs}$		0.0143	0.0143	0.93	0.0888	0		× 10.56
QPET-SPERR	$\epsilon_{QoI,abs}$		0.0385	0.0386	0.999	0.128	0		× 11.4
ZSTD(22)	-		0.0	0.0	0.0	0.0	0		× 2.12

Table B5. Compression metrics for the kinetic energy example in section 5.5. For each compressor, the L_∞ errors over u and v and the kinetic energy KE as well as the L_2 error over KE, the percentage of violations V, and the resulting compression ratio CR are given. For each safeguarded compressor, the percentage of corrections C are provided as well. The highest *safe* compression ratio is highlighted in bold.

B6 Example: laplacian as a stencil quantity of interest

In this appendix, we introduce safeguarding higher-order spatial derivatives using the example of the Laplacian, which we have adopted from the `ZFP` documentation on preserving derivatives (Lawrence Livermore National Laboratory, 2024). In particular, we evaluate how the Laplacian $\Delta u = \frac{\partial u^2}{\partial x^2} + \frac{\partial u^2}{\partial y^2}$ of two different 2D functions $u(x, y)$, discretised over a regular grid covering $[-1, 1]$ with 1024 grid points along both axes, is affected by compressing the discretised $u[x, y]$ array. We evaluate the Laplacian numerically using second-order finite central differences of second-order-accuracy.

First, we take $u(x, y) = \frac{(x^2+y^2)^{\frac{3}{2}}}{9}$, which has an analytical Laplacian of $\Delta u = \sqrt{x^2 + y^2}$. We configure each lossy compressor with an absolute error bound of 10^{-7} over the u array with the aim of bounding the absolute error over the Laplacian to $\epsilon_{abs(\Delta u)} = 0.01$. The safeguards are configured to directly bound the pointwise absolute error over the derived Laplacian. To simplify the quantity of interest for this first case, we assume a constant grid spacing and only evaluate the Laplacian where the second-order finite differences are defined but ignore the boundary points (see listing C5). `OptZConfig` is configured to maximise compression while eliminating violations. Since `QPET` cannot bound the error over finite differences, we do not compare with `QPET` in this example (see section 5.6 for further details).

Figure B6 shows that all absolute-data-error-bounded lossy compressors violate the absolute error bound of 0.01 over the Laplacian, with `SZ3` producing the fewest violations while also having the highest compression ratio. With the safeguards, all violations are eliminated, though at a considerable reduction in compression ratio, particularly for `SZ3`. The iteratively-reduced corrections improve the compression ratios, as shown in Table B6, particularly for safeguarded `SZ3`. While the correction maps highlight a clear improvement in correcting fewer points, the iteratively-reduced corrections still likely overcorrect.

`OptZConfig` is also able to compress without violations. While tuning `ZFP` underperforms compared to safeguarding, `OptZConfig` achieves much better compression ratios with `SZ3` and `SPERR`, clearly outperforming even iteratively-reduced safeguard corrections.

For this first case, we also investigate the impact of tuning the data error bound to produce the best safeguarded compression by testing two more compressor-dependent absolute error bounds that produce $\sim 1\%$ and zero violations, respectively. Table B6 shows that, for this example, using lower data error bounds that need fewer corrections improves the overall compression ratios. For `SZ3` and `SPERR`, but not for `ZFP`, using a sufficiently low error bound that requires no corrections produces the highest compression that is still safe, which matches the results from `OptZConfig`. We hypothesise that the linear cost of corrections can easily bottleneck the overall compression ratio for highly efficient compressors with sub-linear cost like `SZ3` and `SPERR`, which gives compressor-tuning an edge for this affine quantity of interest.

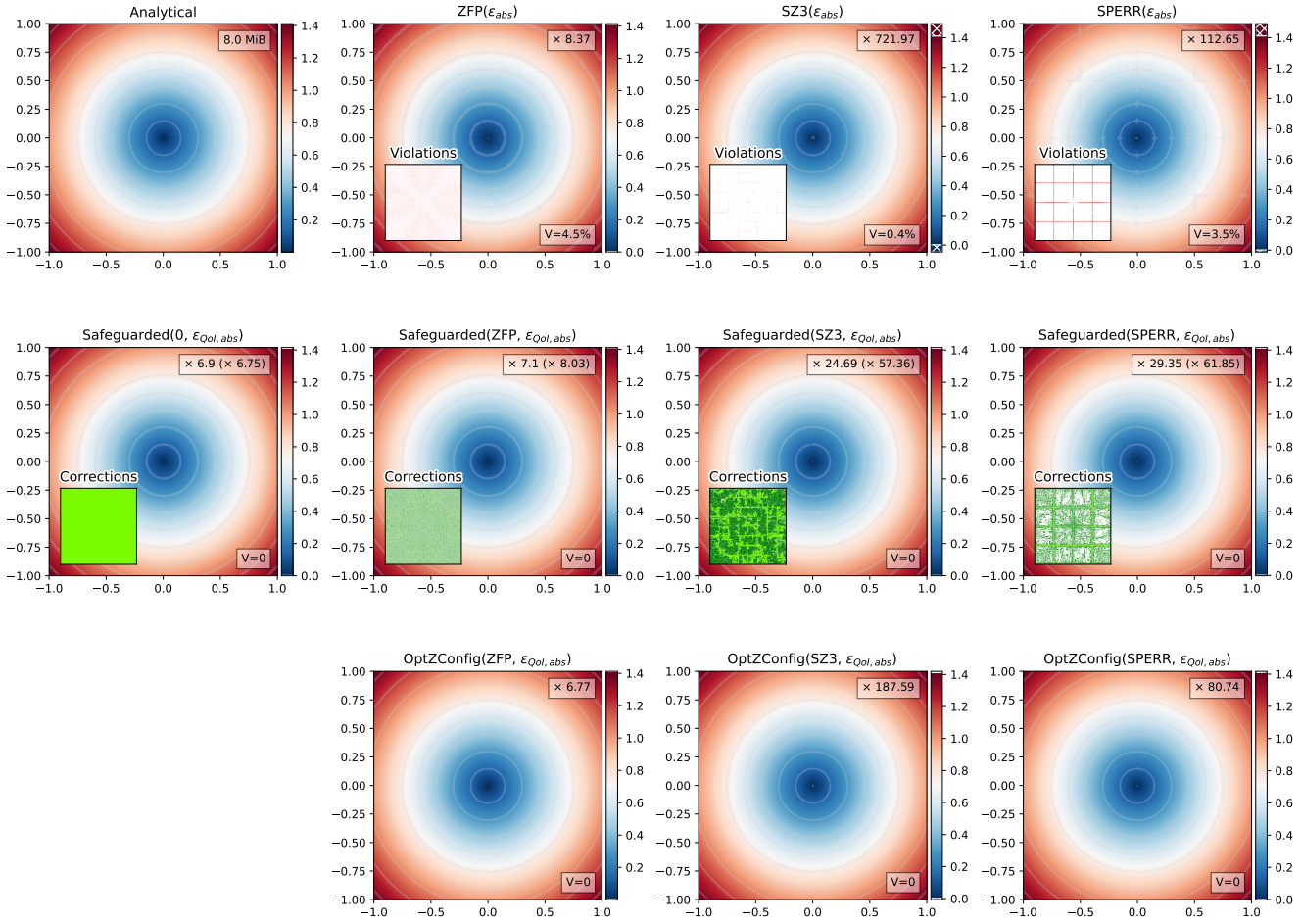


Figure B6. Laplacian of $u(x,y) = (x^2+y^2)^{3/2}\,/\,9$ after applying lossy compression to u. The top row shows the analytical Laplacian as well as the numerically computed Laplacians over the $\hat{u}$ array compressed with `ZFP`, `SZ3`, and `SPERR`. The middle row shows the `compression-safeguarded` equivalents that bound the absolute error over the Laplacian. The bottom row presents the `OptZConfig`-wrapped compressors, which are tuned to similarly bound the error. Contour lines in light grey are shown on top of the Laplacian. For compressors with violations, a map of the violations in red is added. For the safeguarded compressors, a map of the corrections is overlaid. The map is light green where the there are both the one-shot and iterative corrections and dark green where only the one-shot corrections are needed, i.e. where the iterative method was able to remove corrections. For the safeguarded compressors, the compression ratios from using iteratively-reduced corrections are given in parentheses.

(a)

Compressor	ϵ_{abs}	Safeguarded	Corrections	$L_\infty(\hat{u})$	$L_\infty(\Delta\hat{u})$	$L_2(\Delta\hat{u})$	V	C	CR
0		$\epsilon_{QoI,abs}$	lossless	0.31	0.0099	3.7e-05	0	100.0%	× 2.97
			one-shot	0.31	0.0092	0.0028	0	100.0%	× 6.9
			iterative	0.31	0.0092	0.0028	0	100.0%	× 6.75
ZFP(ϵ_{abs})	1e-07	-		2.7e-08	0.028	0.0045	4.5%		× 8.37
		$\epsilon_{QoI,abs}$	lossless	2e-08	0.01	0.0036	0	5.5%	× 7.18
			one-shot	2e-08	0.0086	0.0024	0	40.0%	× 7.1
			iterative	2e-08	0.01	0.0036	0	5.4%	× 8.03
	5e-08	-		1.5e-08	0.015	0.0025	0.2%		× 7.91
		$\epsilon_{QoI,abs}$	lossless	1.3e-08	0.01	0.0025	0	0.2%	× 7.83
			one-shot	4.8e-09	0.0073	0.0019	0	12.1%	× 7.45
			iterative	1.3e-08	0.01	0.0025	0	0.2%	× 7.88
	2e-08	-		7.8e-09	0.0077	0.0015	0		× 7.43
		$\epsilon_{QoI,abs}$	lossless	7.8e-09	0.0077	0.0015	0	0	× 7.43
			one-shot	7.8e-09	0.0077	0.0015	0	0	× 7.43
			iterative	7.8e-09	0.0077	0.0015	0	0	× 7.43
OptZConfig(ZFP)		$\epsilon_{QoI,abs}$		4.1e-09	0.0041	0.00079	0		× 6.77
SZ3(ϵ_{abs})	1e-07	-		1e-07	0.086	0.0022	0.4%		× 721.97
		$\epsilon_{QoI,abs}$	lossless	9.4e-08	0.01	0.0017	0	25.1%	× 5.64
			one-shot	7.6e-08	0.0094	0.0024	0	86.8%	× 24.69
			iterative	9.4e-08	0.01	0.0021	0	25.0%	× 57.36
	1.5e-07	-		1.5e-07	0.13	0.0032	0.9%		× 821.12
		$\epsilon_{QoI,abs}$	lossless	1.2e-07	0.01	0.0018	0	49.9%	× 3.0
			one-shot	1.2e-07	0.0091	0.0026	0	91.0%	× 22.22
			iterative	1.2e-07	0.01	0.0026	0	51.8%	× 32.51

(b)

Compressor	ϵ_{abs}	Safeguarded	Corrections	$L_\infty(\hat{u})$	$L_\infty(\Delta\hat{u})$	$L_2(\Delta\hat{u})$	V	C	CR
SZ3(ϵ_{abs})	5e-09	-		5e-09	0.0093	0.00076	0		× 176.04
		$\epsilon_{QoI,abs}$	lossless	5e-09	0.0093	0.00076	0	0	× 176.02
			one-shot	5e-09	0.0093	0.00076	0	0	× 176.02
			iterative	5e-09	0.0093	0.00076	0	0	× 176.02
OptZConfig(SZ3)		$\epsilon_{QoI,abs}$		5.2e-09	0.0098	0.00075	0		× **187.59**
SPERR(ϵ_{abs})	1e-07	-		9.7e-08	0.12	0.0049	3.5%		× 112.65
		$\epsilon_{QoI,abs}$	lossless	8.7e-08	0.01	0.0013	0	6.7%	× 17.35
			one-shot	8.7e-08	0.0087	0.0014	0	41.2%	× 29.35
			iterative	8.7e-08	0.01	0.0014	0	6.5%	× 61.85
	3.5e-08	-		3.5e-08	0.041	0.0017	0.9%		× 98.06
		$\epsilon_{QoI,abs}$	lossless	3.5e-08	0.01	0.001	0	1.3%	× 46.54
			one-shot	1.1e-08	0.0088	0.00083	0	8.7%	× 53.68
			iterative	3.5e-08	0.01	0.0011	0	1.3%	× 83.29
	8e-09	-		8e-09	0.0083	0.00041	0		× 80.65
		$\epsilon_{QoI,abs}$	lossless	8e-09	0.0083	0.00041	0	0	× 80.65
			one-shot	8e-09	0.0083	0.00041	0	0	× 80.65
			iterative	8e-09	0.0083	0.00041	0	0	× 80.65
OptZConfig(SPERR)		$\epsilon_{QoI,abs}$		8e-09	0.0085	0.00041	0		× 80.74
ZSTD(22)		-		0.0	0.0	0.0	0		× 4.26

Table B6. (a)-(b) Compression metrics for the first Laplacian example, $u(x,y) = (x^2+y^2)^{3/2}/9$. For each compressor and for three different absolute data error bounds, the L_∞ errors over u and Δu as well as the L_2 error over Δu, the percentage of violations V, and the resulting compression ratio CR are given. For each safeguarded compressor and the three safeguard correction methods, the percentage of corrections C is provided as well. The highest *safe* compression ratio is highlighted in bold.

Second, we take $u(x, y) = e^{4x+3y}$, which has an analytical Laplacian of $\Delta u = 25u$. We then plot the natural logarithm of the Laplacian, making the quantity of interest non-linear. We configure each compressor with an absolute error bound of $5 \cdot 10^{-6}$ over the u array with the aim of bounding the absolute error over the natural logarithm of the Laplacian to 0.1. For this second case, we no longer assume a constant grid spacing but instead pass the coordinate arrays `X` and `Y` to the `SafeguardedCodec` as late-bound constant parameters. Furthermore, we now use second-order central finite differences on non-boundary points and second-order forward/backwards finite differences on boundary points to evaluate the Laplacian everywhere. The complete configuration is given in listing C6.

Figure B7 shows that all absolute-data-error-bounded lossy compressors violate the quantity of interest, since they can produce negative Laplacian values whose logarithms are undefined. While a similar problem could be resolved in appendix B4 by wrapping the compressors in a meta-compressor that bounds the ratio error, this is not possible here where a ratio error over the Laplacian, a complex weighted sum, would have to be preserved. The safeguards still eliminate all violations, but are required to correct most points and thus reduce the compression ratio significantly for `SZ3` and `SPERR`. In this case, using the iteratively-reduced corrections provides less benefit. As the correction maps and Table B7 show, the reduction in the number of corrections works less well, and the safeguards very likely over-correct.

Tuning the absolute error bound of each compressor to produce no violations does not help either, since a near-zero error bound is required, and `OptZConfig` produced worse compression ratios than using one-shot safeguard corrections. In contrast, the safeguarded all-zero-approximation achieves the highest safe compression ratio, even outperforming unsafeguarded `SZ3`. Since the safeguards are able to propagate the error bound over the logarithmic quantity of interest through the logarithm and the Laplacian, the heuristic to choose well-compressible corrections can make full use of the greater range of possible corrections when correcting the well-behaved zero-entropy all-zero approximation.

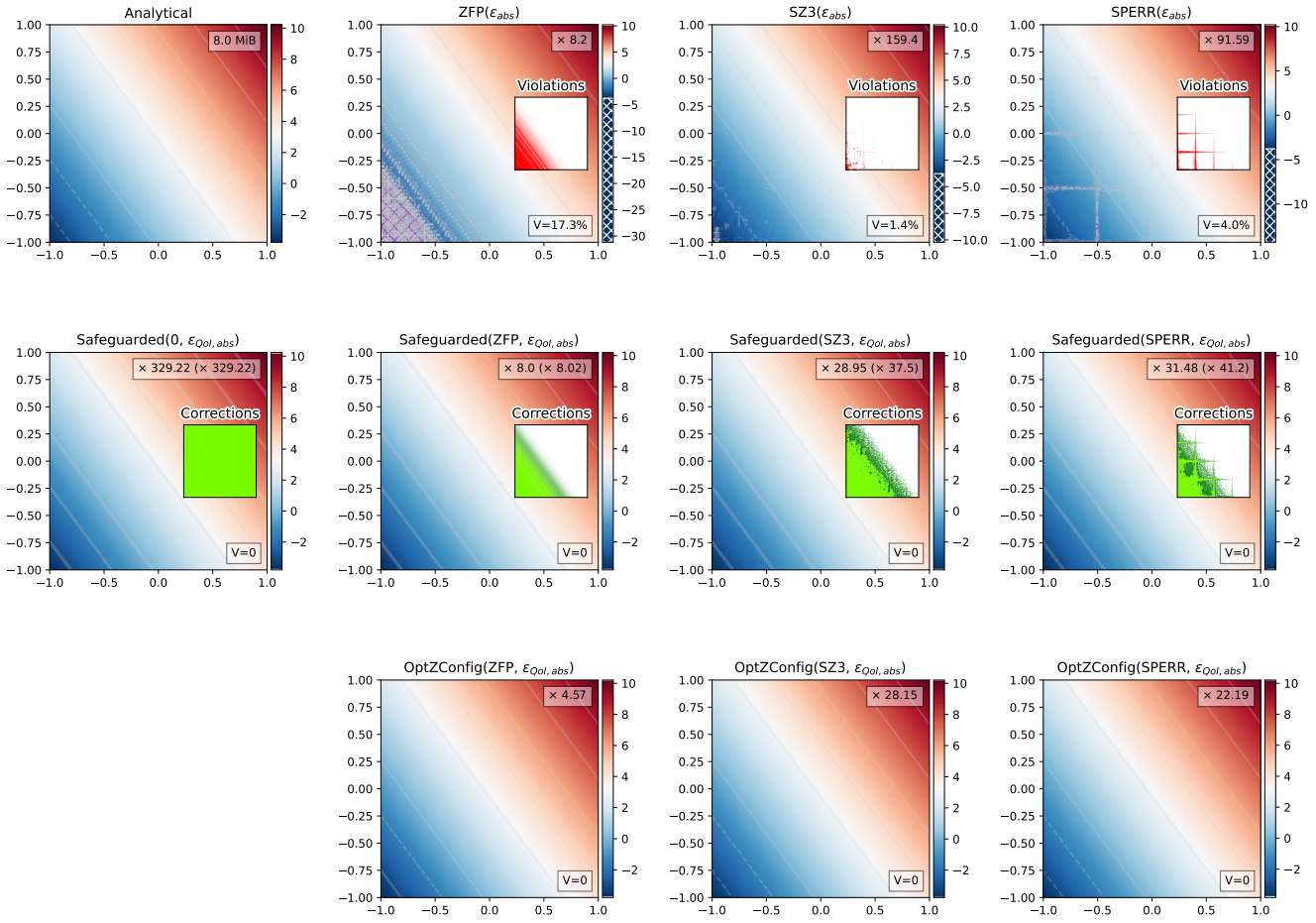


Figure B7. Logarithm of the Laplacian of $u(x, y) = e^{4x+3y}$ after compressing u. The top row shows the logarithm of the analytical Laplacian as well as the numerical results from `ZFP`, `SZ3`, and `SPERR`, whereas the middle row shows their `compression-safeguarded` equivalents, which bound the absolute error over the logarithm of the Laplacian. The bottom row presents the `OptZConfig`-wrapped compressors, which are tuned to similarly bound the error. Contour lines in light grey are shown on top of the Laplacian. For compressors with violations, a map of the violations in red is added. For the safeguarded compressors, a map of the corrections is overlaid. The map is light green where the there are both the one-shot and iterative corrections and dark green where only the one-shot corrections are needed, i.e. where the iterative method was able to remove corrections. Areas where the logarithm is undefined are marked with pink hatching. For the safeguarded compressors, the compression ratios from using iteratively-reduced corrections are given in parentheses.

Compressor	Safeguarded	Corrections	$L_\infty(\hat{u})$	$L_\infty(\ln(\Delta\hat{u}))$	$L_2(\ln(\Delta\hat{u}))$	V	C	CR
0	$\epsilon_{QoI,abs}$	lossless	0.0	0.0	0.0	0	100.0%	× 27.22
		one-shot	0.0012	0.087	0.028	0	100.0%	× **329.22**
		iterative	0.0012	0.087	0.028	0	100.0%	× **329.22**
ZFP(ϵ_{abs})	-		2e-06	NaN (2.7e+01)	NaN (1.3)	17.3%		× 8.2
	$\epsilon_{QoI,abs}$	lossless	2e-06	0.1	0.019	0	23.4%	× 7.04
		one-shot	2e-06	0.087	0.018	0	30.1%	× 8.0
		iterative	2e-06	0.1	0.023	0	22.6%	× 8.02
OptZConfig(ZFP)	$\epsilon_{QoI,abs}$		4.2e-09	0.084	0.0024	0		× 4.57
SZ3(ϵ_{abs})	-		5e-06	NaN (6.6)	NaN (0.052)	1.4%		× 159.4
	$\epsilon_{QoI,abs}$	lossless	5e-06	0.1	0.0085	0	29.7%	× 4.11
		one-shot	5e-06	0.089	0.018	0	45.6%	× 28.95
		iterative	5e-06	0.1	0.017	0	29.5%	× 37.5
OptZConfig(SZ3)	$\epsilon_{QoI,abs}$		1.7e-08	0.099	0.002	0		× 28.15
SPERR(ϵ_{abs})	-		5e-06	NaN (1.3e+01)	NaN (0.16)	4.0%		× 91.59
	$\epsilon_{QoI,abs}$	lossless	5e-06	0.1	0.009	0	14.8%	× 7.18
		one-shot	5e-06	0.092	0.014	0	28.7%	× 31.48
		iterative	5e-06	0.1	0.014	0	14.6%	× 41.2
OptZConfig(SPERR)	$\epsilon_{QoI,abs}$		3.5e-09	0.079	0.0007	0		× 22.19
ZSTD(22)	-		0.0	0.0	0.0	0		× 73.21

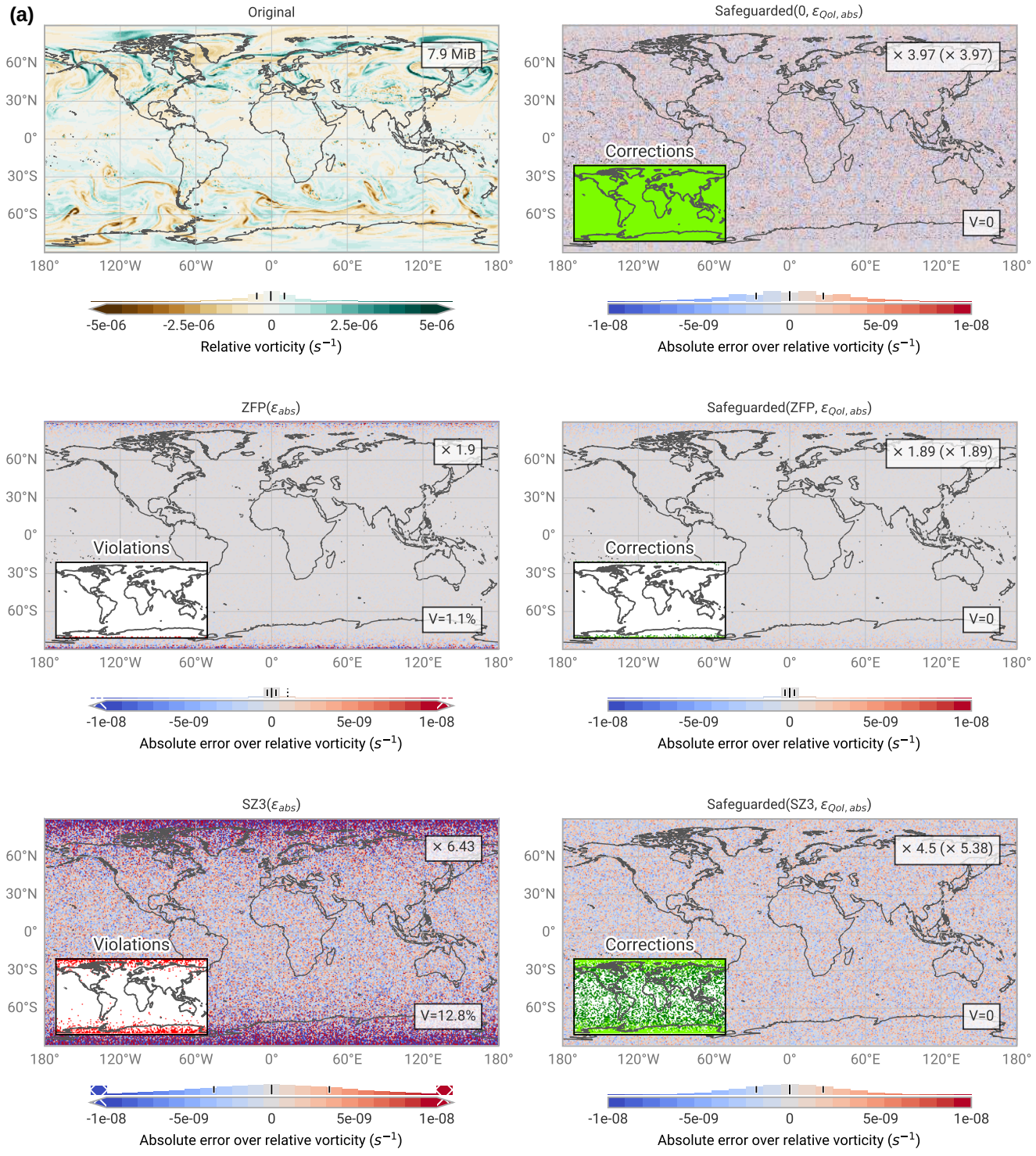
Table B7. Compression metrics for the second, logarithm of the Laplacian, example, $u(x, y) = e^{4x+3y}$. For each compressor, the L_∞ errors over u and $\ln(\Delta u)$ as well as the L_2 error over $\ln(\Delta u)$, the percentage of violations V, and the resulting compression ratio CR are given. For each safeguarded compressor and the three safeguard correction methods, the percentage of corrections C is provided as well. The highest *safe* compression ratio is highlighted in bold.

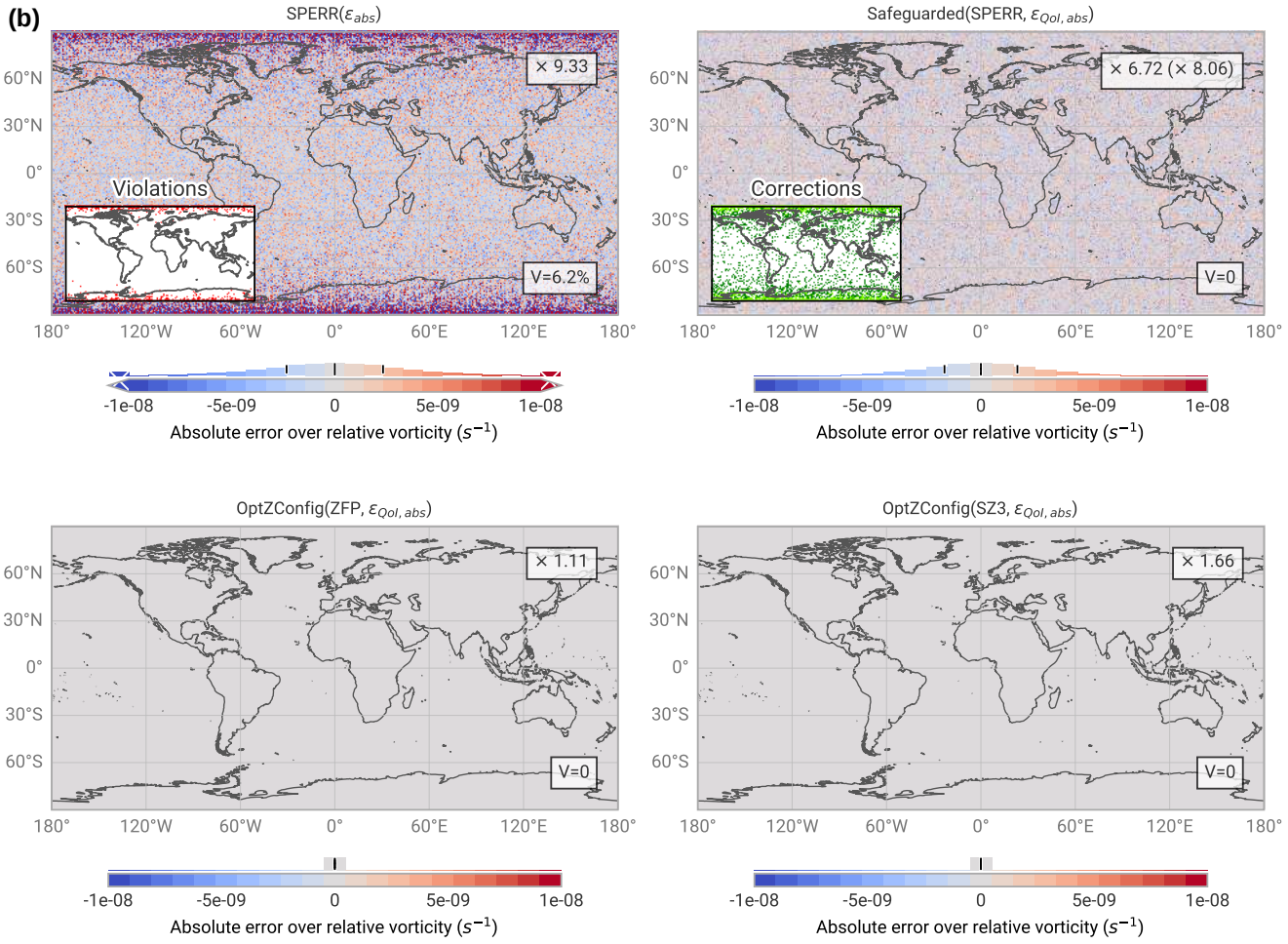
B7 Example: relative vorticity as a stencil quantity of interest

This appendix contains additional results for computing the relative vorticity over compressed wind u and v components using different absolute-data-error-bounded lossy compressors in section 5.6. Figure B8 and table B8 compare how the derived vorticity is preserved, including the results for `ZFP` and `SZ3` and their `OptZConfig`-tuned counterparts.

Figure B8 shows that all lossy absolute-data-error-bounded compressors exceed the absolute vorticity error bound near the poles. Since `ZFP` only has a few violations, its small compression ratio is almost unaffected by the safeguards corrections. For `SZ3` and particularly `SPERR`, which provides the highest compression ratio, safeguarding results in significant decreases in compression ratios, which can be improved by using the iteratively-reduced corrections instead of one-shot corrections. The correction maps clearly show how the iterative approach is able to better match the corrections to the violations and limit them to near the poles.

`OptZConfig` is also able to compress without violations, though it produces very low compression ratios that are clearly outperformed by even one-shot safeguard corrections. The worse performance likely comes from `OptZConfig` having to drastically reduce the compressor error bound globally to account for violations near the poles.


(a)
Original
7.9 MiB
60°N
30°N
0°
30°S
60°S
180°
120°W
60°W
0°
60°E
120°E
180°
-5e-06
-2.5e-06
0
2.5e-06
5e-06
Relative vorticity (s^{-1})
Safeguarded(0, $\varepsilon_{QoI,abs}$)
× 3.97 (× 3.97)
Corrections
V=0
-1e-08
-5e-09
0
5e-09
1e-08
Absolute error over relative vorticity (s^{-1})
ZFP(ε_{abs})
× 1.9
Violations
V=1.1%
Absolute error over relative vorticity (s^{-1})
Safeguarded(ZFP, $\varepsilon_{QoI,abs}$)
× 1.89 (× 1.89)
Corrections
V=0
Absolute error over relative vorticity (s^{-1})
SZ3(ε_{abs})
× 6.43
Violations
V=12.8%
Absolute error over relative vorticity (s^{-1})
Safeguarded(SZ3, $\varepsilon_{QoI,abs}$)
× 4.5 (× 5.38)
Corrections
V=0
Absolute error over relative vorticity (s^{-1})


(b)
SPERR(ϵ_{abs})
× 9.33
Violations
V=6.2%
60°N
30°N
0°
30°S
60°S
180°
120°W
60°W
0°
60°E
120°E
180°
-1e-08
-5e-09
0
5e-09
1e-08
Absolute error over relative vorticity (s^{-1})
Safeguarded(SPERR, $\epsilon_{QoI, abs}$)
× 6.72 (× 8.06)
Corrections
V=0
OptZConfig(ZFP, $\epsilon_{QoI, abs}$)
× 1.11
V=0
OptZConfig(SZ3, $\epsilon_{QoI, abs}$)
× 1.66
V=0

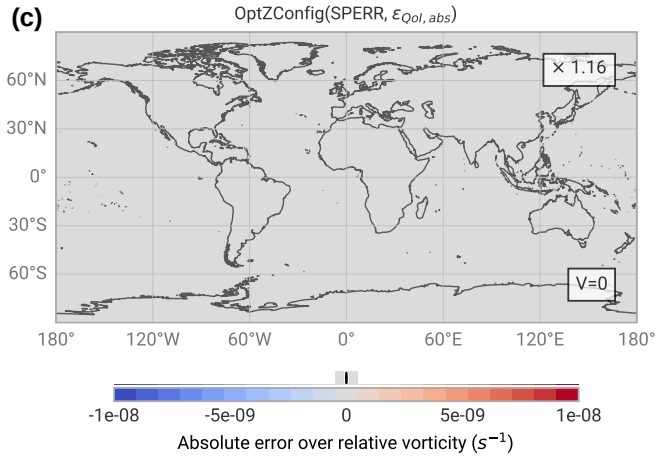


Figure B8. (a)-(c) Error maps for relative vorticity derived from compressed wind components from section 5.6. The left column shows in the first four rows the maps for the original data and the errors over the relative vorticity of the decompressed data from `ZFP`, `SZ3`, and `SPERR`. In the right column, the `compression-safeguarded` equivalents are given, which have been configured to bound the absolute error over the vorticity. The last two rows present the maps from the `OptZConfig`-wrapped compressors, which are tuned to similarly bound the error. For the safeguarded compressors, the compression ratios from iteratively-reduced corrections are given in parentheses. For compressors with violations, a map of the violations in red is added. For the safeguarded compressors, a map of the corrections is overlaid. The map is light green where the there are both the one-shot and iterative corrections and dark green where only the one-shot corrections are needed, i.e. where the iterative method was able to remove corrections.

Compressor	Safeguarded	Corrections	$L_\infty(\hat{u})$	$L_\infty(\hat{v})$	$L_\infty(\hat{\zeta})$	$L_2(\hat{\zeta})$	V	C	CR
0	$\epsilon_{QoI,abs}$	lossless	0.0	0.0396	9.16e-09	2.2e-11	0	100.0%	× 2.66
		one-shot	0.00795	0.00728	9.59e-09	2.71e-09	0	99.9%	× 3.97
		iterative	0.00795	0.0396	9.59e-09	2.71e-09	0	99.9%	× 3.97
ZFP(ϵ_{abs})	-		0.00197	0.00232	0.0127	0.000142	1.1%		× 1.9
	$\epsilon_{QoI,abs}$	lossless	0.00197	0.00232	1e-08	9.14e-10	0	1.7%	× 1.88
		one-shot	0.00197	0.00232	7.45e-09	6.91e-10	0	1.7%	× 1.89
		iterative	0.00197	0.00232	1e-08	9.42e-10	0	1.0%	× 1.89
OptZConfig(ZFP)	$\epsilon_{QoI,abs}$		0.0	0.0	0.0	0.0	0		× 1.11
SZ3(ϵ_{abs})	-		0.01	0.01	0.0815	0.00016	12.8%		× 6.43
	$\epsilon_{QoI,abs}$	lossless	0.01	0.01	1e-08	3.63e-09	0	22.9%	× 3.34
		one-shot	0.00797	0.00796	9.73e-09	2.66e-09	0	30.3%	× 4.5
		iterative	0.01	0.01	1e-08	3.85e-09	0	10.9%	× 5.38
OptZConfig(SZ3)	$\epsilon_{QoI,abs}$		0.0	0.0	0.0	0.0	0		× 1.66
SPERR(ϵ_{abs})	-		0.01	0.01	0.112	0.00132	6.2%		× 9.33
	$\epsilon_{QoI,abs}$	lossless	0.01	0.01	1e-08	3.07e-09	0	11.2%	× 5.61
		one-shot	0.00796	0.00796	9.57e-09	2.33e-09	0	14.8%	× 6.72
		iterative	0.01	0.01	1e-08	3.16e-09	0	4.9%	× **8.06**
OptZConfig(SPERR)	$\epsilon_{QoI,abs}$		4.66e-10	4.66e-10	7.45e-09	3.96e-11	0		× 1.16
ZSTD(22)	-		0.0	0.0	0.0	0.0	0		× 2.12

Table B8. Compression metrics for the vorticity example in section 5.6. For each compressor, the L_∞ errors over u and v and the relative vorticity ζ as well as the L_2 error over ζ, the percentage of violations V, and the resulting compression ratio CR are given. For each safeguarded compressor, the percentage of corrections C are provided as well. The highest *safe* compression ratio is highlighted in bold.

B8 Example: data structural similarity index measure

This appendix showcases how the expressive quantity of interest support in the `compression-safeguards` implementation can be used to preserve a complex metric, here the structural similarity index for floating-point data, dSSIM (Baker et al., 2024). We compress a dataset of u wind component. The pointwise dSSIM between the original u and compressed $\hat{u}$ is then defined as:

$$\text{dSSIM}(u, \hat{u}) = S_1(q(u), q(\hat{u})) \cdot S_2(q(u), q(\hat{u})), \text{ where} \tag{B1}$$

$$S_1(a, b) = \frac{2\mu_a\mu_b + C_1}{\mu_a^2 + \mu_b^2 + C_1}, \tag{B2}$$

$$S_2(a, b) = \frac{2\sigma_{a,b} + C_2}{\sigma_a^2 + \sigma_b^2 + C_2}, \tag{B3}$$

where further $C_1 = C_2 = 10^{-8}$. $q(x)$ first rescales x to $[0; 1]$ based on the combined minimum and maximum of u and $\hat{u}$, and then quantises the rescaled x into 256 bins, which was inspired by the rescaling and quantisation during plotting. The dSSIM captures changes in local features in the means μ_a and μ_b, variances σ_a and σ_b, and the covariance $\sigma_{a,b}$, which are all computed over a local neighbourhood using a Gaussian kernel which is typically symmetric and of size 11×11 (Baker et al., 2024). The dSSIM ranges between -1 and 1, with more similar data achieving scores closer to 1. It is usually reported as the mean over all points, excluding NaN points and boundary values where the neighbourhood is not defined.

We configure each lossy compressor with an absolute error bound of $0.1\text{m}\,\text{s}^{-1}$ with the aim of achieving a dSSIM score of at least 0.995, as suggested by (Baker et al., 2024). The safeguards are configured to guarantee the dSSIM to be ≥ 0.995 by translating Equation (B1) into a quantity of interest. We make use of a few tricks and simplifications in this translation:

1. We conservatively bound the pointwise dSSIM, not the mean dSSIM, and reduce the neighbourhood size to 3×3.
2. The rescaling range is kept constant by preserving the global minimum and maximum using two sign safeguards.
3. The quantity of interest computes the dSSIM with respect to the original uncompressed data. The absolute error with respect to $\text{dSSIM}(u, u) = 1$ is bounded to $\epsilon_{abs(\text{dSSIM})} = 1 - 0.995$ to guarantee that $\text{dSSIM}(u, \hat{u}) \geq 0.995$.
4. The dSSIM reference implementation (Pinard et al., 2025) avoids NaN values bleeding over in Gaussian smoothing by replacing them with Gaussian-interpolated values before the smoothing. We implement a simpler variant that excludes NaN values from the convolution, sets the corresponding per-point kernel weights to zero, and then rescales the kernel weights. Like Pinard et al. (2025), the quantity of interest returns dSSIM $=$ NaN for data points that are NaN in the input.

Figure B9 shows that all absolute-data-error-bounded compressors violate the data structural similarity index (dSSIM) bound. Even `ZFP`, which usually produces lower errors for the same absolute error bound (e.g., Reichelt et al., 2026), still exceeds the pointwise dSSIM bound sufficiently often to violate the mean dSSIM bound. We also check how the `SZ3` compressor performs with a pointwise normalised absolute error bound of $\epsilon_{noa} = 1 \,/\, 255$, which corresponds to the bin width of

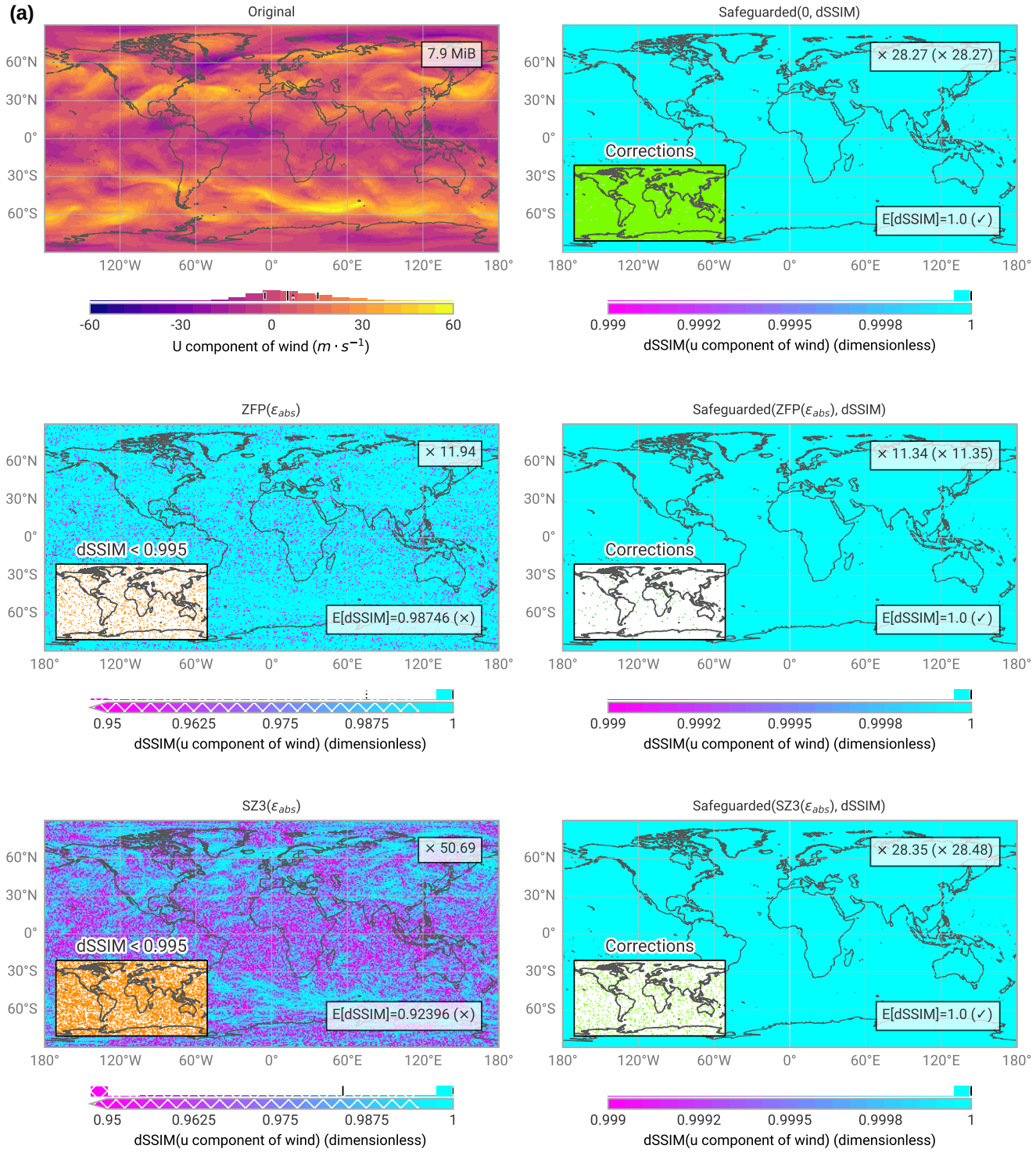
the quantisation in dSSIM. Importantly, Table B9 shows that such a bound does *not* guarantee a high dSSIM score, because even a tiny non-zero error bound still allows values to cross quantisation bin boundaries.

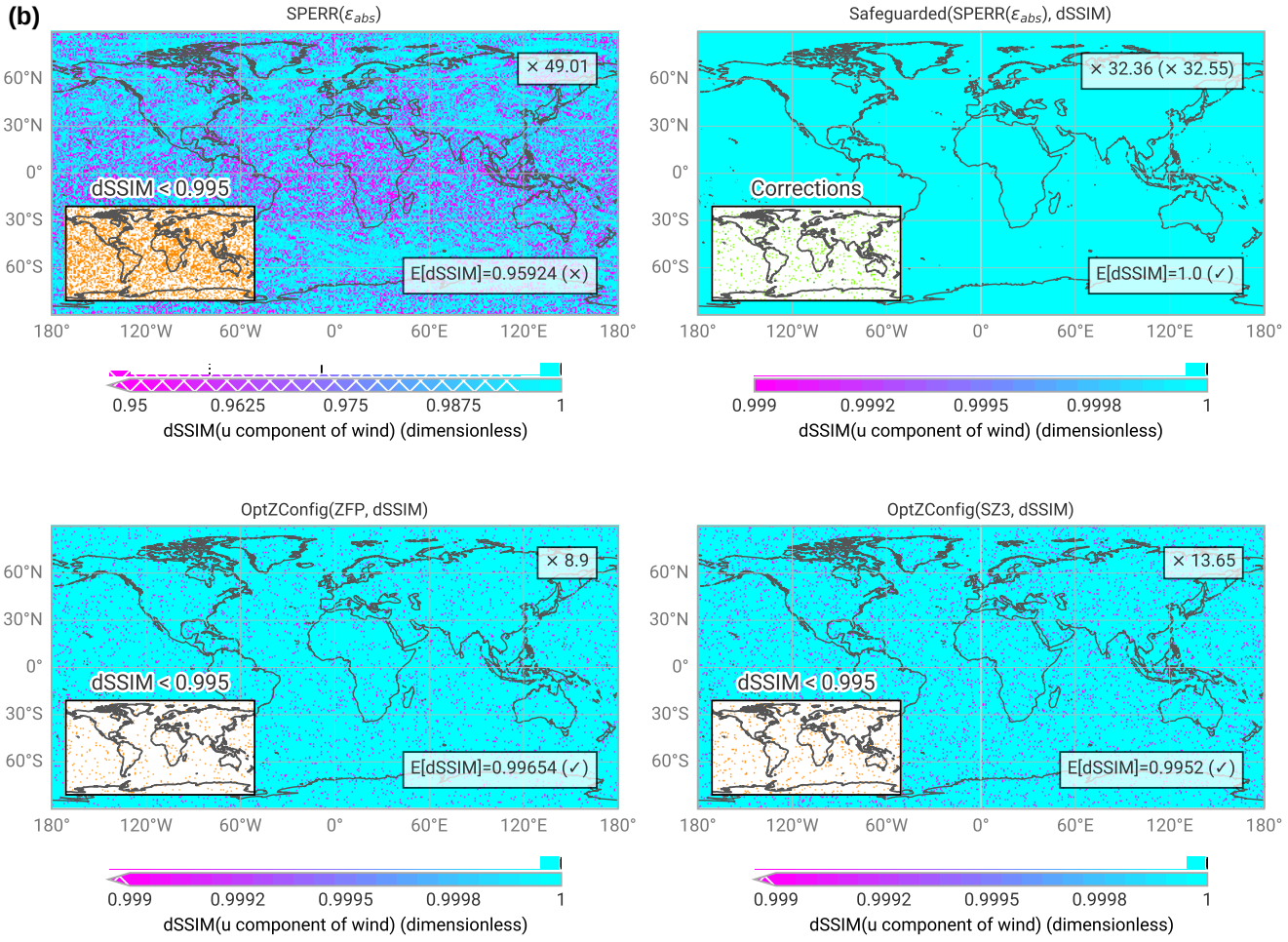
The safeguards eliminate the mean dSSIM bound violation by producing overly conservative corrections everywhere that significantly reduce compression ratios for `SZ3` and `SPERR`, which are not significantly improved by iteratively-reduced corrections. If the `compression-safeguards` implementation was able to preserve a mean error bound, the compression ratio would likely be higher.

To further investigate the origin of the overly conservative corrections, we also test a simplified safeguard, which we refer to as `bins(256)`, that guarantees that the rescaled and *quantised* data values remain the same (see listing C9). In effect, this method guarantees a perfect dSSIM score of 1 without requiring zero compression error. Table B9 shows that this method currently produces the same results as the more complex dSSIM quantity of interest. Since Equation (B1) contains variances and co-variances that are calculated as non-linear sums and products, where the safeguards need to account for the worst-case error propagation, the propagated error bound is likely reduced to near-zero before propagating through the quantisation, thus arriving at the same result as the simplified safeguard. However, future improvements in the quantity of interest safeguards, e.g. incremental corrections (see appendix B9), could allow the dSSIM quantity of interest to produce less conservative corrections.

We also examine in Table B9 how using a larger absolute data error bound $\epsilon_{abs} = 1 \mathrm{m\,s^{-1}}$ impacts the safeguarded compression ratios. For all three compressors, the percentage of violations increases drastically, resulting in many more corrections. These additional corrections reduce the overall safeguarded compression ratios for `SZ3` and `SPERR` even though the wrapped inner compressors provide better compression. For `ZFP`, however, the safeguarded compression ratio increases for the higher error bound, highlighting the possible benefits of compressor-specific error bound tuning.

`OptZConfig`, which is configured to maximise compression while bounding the mean dSSIM, also safely keeps above the mean bound. To our surprise, one-shot safeguard corrections still provide better compression than tuning, even though `OptZConfig` directly bounds the mean whereas the safeguards conservatively bound the pointwise dSSIM scores. This again highlights the power of the Compression Safeguards, which often only need to correct a small number of outliers.


(a)
Original
7.9 MiB
60°N
30°N
0°
30°S
60°S
120°W
60°W
0°
60°E
120°E
180°
-60
-30
0
30
60
U component of wind ($m \cdot s^{-1}$)
Safeguarded(0, dSSIM)
× 28.27 (× 28.27)
Corrections
E[dSSIM]=1.0 (✓)
0.999
0.9992
0.9995
0.9998
1
dSSIM(u component of wind) (dimensionless)
ZFP(ε_{abs})
× 11.94
dSSIM < 0.995
E[dSSIM]=0.98746 (×)
0.95
0.9625
0.975
0.9875
1
dSSIM(u component of wind) (dimensionless)
Safeguarded(ZFP(ε_{abs}), dSSIM)
× 11.34 (× 11.35)
Corrections
E[dSSIM]=1.0 (✓)
0.999
0.9992
0.9995
0.9998
1
dSSIM(u component of wind) (dimensionless)
SZ3(ε_{abs})
× 50.69
dSSIM < 0.995
E[dSSIM]=0.92396 (×)
0.95
0.9625
0.975
0.9875
1
dSSIM(u component of wind) (dimensionless)
Safeguarded(SZ3(ε_{abs}), dSSIM)
× 28.35 (× 28.48)
Corrections
E[dSSIM]=1.0 (✓)
0.999
0.9992
0.9995
0.9998
1
dSSIM(u component of wind) (dimensionless)


(b)
SPERR(ε_{abs})
× 49.01
dSSIM < 0.995
E[dSSIM]=0.95924 (×)
Safeguarded(SPERR(ε_{abs}), dSSIM)
× 32.36 (× 32.55)
Corrections
E[dSSIM]=1.0 (✓)
OptZConfig(ZFP, dSSIM)
× 8.9
dSSIM < 0.995
E[dSSIM]=0.99654 (✓)
OptZConfig(SZ3, dSSIM)
× 13.65
dSSIM < 0.995
E[dSSIM]=0.9952 (✓)
60°N
30°N
0°
30°S
60°S
180°
120°W
60°W
0°
60°E
120°E
180°
0.95
0.9625
0.975
0.9875
1
0.999
0.9992
0.9995
0.9998
1
dSSIM(u component of wind) (dimensionless)

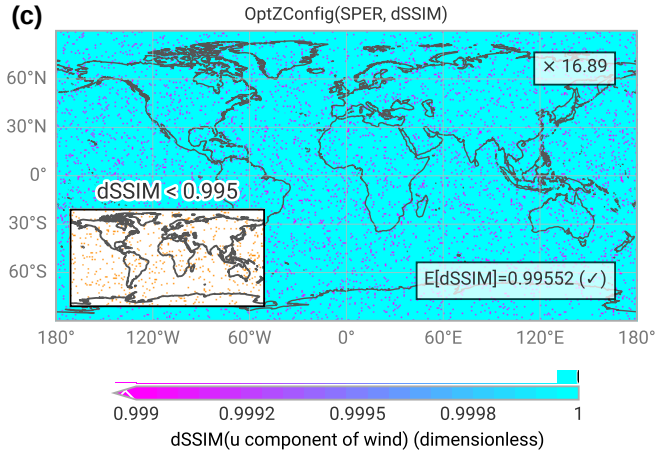


Figure B9. (a)-(c) Pointwise dSSIM maps for compressed u wind data. The left column shows in the first four rows the maps for the original data and the pointwise dSSIM scores for the decompressed data from `ZFP`, `SZ3`, and `SPERR`. In the right column, the `compression-safeguarded` equivalents are given, which have been configured to guarantee that the pointwise dSSIM at or above 0.995. The last two rows present the maps from the `OptZConfig`-wrapped compressors, which are tuned to bound the mean dSSIM. For the safeguarded compressors, the compression ratios from the iteratively-reduced corrections are given in parentheses. For unsafeguarded compressors, a map showing in orange where the pointwise dSSIM is below 0.995 is added. For the safeguarded compressors, a map of the corrections is overlaid. The map is light green where the there are both the one-shot and iterative corrections and dark green where only the one-shot corrections are needed, i.e. where the iterative method was able to remove corrections.

Compressor	ϵ_{abs}	Safeguarded	Corrections	$L_\infty(\hat{u})$	$L_2(\hat{u})$	E[D]	min(D)	max(D)	D < 0.995	S	C	CR
0		dSSIM ≥ 0.995	lossless	0.207	0.00151	1.0	1.0	1.0	0	✓	99.7%	× 5.51
			one-shot	0.356	0.137	1.0	1.0	1.0	0	✓	98.6%	× 28.27
			iterative	0.356	0.137	1.0	1.0	1.0	0	✓	98.6%	× 28.27
		bin(256)	lossless	0.226	0.0123	1.0	1.0	1.0	0	✓	98.6%	× 5.64
			one-shot	0.356	0.137	1.0	1.0	1.0	0	✓	98.6%	× 28.27
			iterative	0.356	0.137	1.0	1.0	1.0	0	✓	98.6%	× 28.27
SZ3(ϵ_{noa})		-		0.358	0.173	0.82942	-0.997	1.0	92.3%	×		× 110.98
ZFP(ϵ_{abs})	1.000000	-		0.543	0.0987	0.88125	-0.997	1.0	80.4%	×		× 24.35
		dSSIM ≥ 0.995	lossless	0.469	0.033	0.99995	0.995	1.0	0	✓	81.3%	× 5.05
			one-shot	0.355	0.0789	1.0	1.0	1.0	0	✓	21.5%	× 15.73
			iterative	0.469	0.0792	0.99998	0.995	1.0	0	✓	21.2%	× 15.77
	0.100000	-		0.0392	0.00825	0.98746	-0.183	1.0	13.6%	×		× 11.94
		dSSIM ≥ 0.995	lossless	0.0392	0.00756	0.99999	0.995	1.0	0	✓	14.2%	× 8.9
			one-shot	0.227	0.0178	1.0	1.0	1.0	0	✓	1.8%	× 11.34
			iterative	0.227	0.0175	0.99999	0.995	1.0	0	✓	1.8%	× 11.35
OptZConfig(ZFP)		E[dSSIM] ≥ 0.995		0.00975	0.0022	0.99654	-0.149	1.0	3.9%	✓		× 8.9
SZ3(ϵ_{abs})	1.000000	-		1.0	0.328	0.78367	-0.999	1.0	94.9%	×		× 262.5
		dSSIM ≥ 0.995	lossless	0.973	0.0269	0.99997	0.995	1.0	0	✓	97.4%	× 0.94
			one-shot	0.358	0.135	1.0	1.0	1.0	0	✓	62.4%	× 17.29
			iterative	0.973	0.137	0.99998	0.995	1.0	0	✓	62.2%	× 17.33
	0.100000	-		0.1	0.0515	0.92396	-0.997	1.0	61.8%	×		× 50.69
		dSSIM ≥ 0.995	lossless	0.1	0.0287	0.99996	0.995	1.0	0	✓	62.9%	× 1.42
			one-shot	0.226	0.0527	1.0	1.0	1.0	0	✓	12.1%	× 28.35
			iterative	0.226	0.0527	0.99998	0.995	1.0	0	✓	11.9%	× 28.48
OptZConfig(SZ3)		E[dSSIM] ≥ 0.995		0.00487	0.00275	0.9952	-0.149	1.0	5.3%	✓		× 13.65
SPERR(ϵ_{abs})	1.000000	-		0.998	0.176	0.84181	-0.997	1.0	89.2%	×		× 195.53
		dSSIM ≥ 0.995	lossless	0.857	0.0333	0.99995	0.995	1.0	0	✓	90.1%	× 1.01
			one-shot	0.357	0.107	1.0	1.0	1.0	0	✓	36.4%	× 24.74
			iterative	0.857	0.108	0.99998	0.995	1.0	0	✓	36.0%	× 24.86
	0.100000	-		0.1	0.0293	0.95924	-0.442	1.0	40.0%	×		× 49.01
		dSSIM ≥ 0.995	lossless	0.1	0.021	0.99997	0.995	1.0	0	✓	41.0%	× 2.14
			one-shot	0.227	0.0362	1.0	1.0	1.0	0	✓	6.4%	× 32.36
			iterative	0.227	0.0361	0.99998	0.995	1.0	0	✓	6.2%	× **32.55**
OptZConfig(SPERR)		E[dSSIM] ≥ 0.995		0.00752	0.00281	0.99552	-0.149	1.0	5.1%	✓		× 16.89
ZSTD(22)		-		0.0	0.0	1.0	1.0	1.0	0	✓		× 4.51

Table B9. Compression metrics for u wind in the dSSIM example. For each compressor and for two absolute data error bounds, the L_∞ and L_2 errors over u, the mean E[D], minimum min(D), and maximum max(D) of the pointwise dSSIM metric D, the percentage of points where D < 0.995, the success S of whether the mean dSSIM lower bound was met, and the resulting compression ratio CR are given. For each safeguarded compressor and the three correction methods, the percentage of corrections C is given as well. The table also includes results for the `SZ3(`ϵ_{noa}`)` and `Safeguarded(0, bin(256))` compressors. The highest *safe* compression ratio is highlighted in bold.

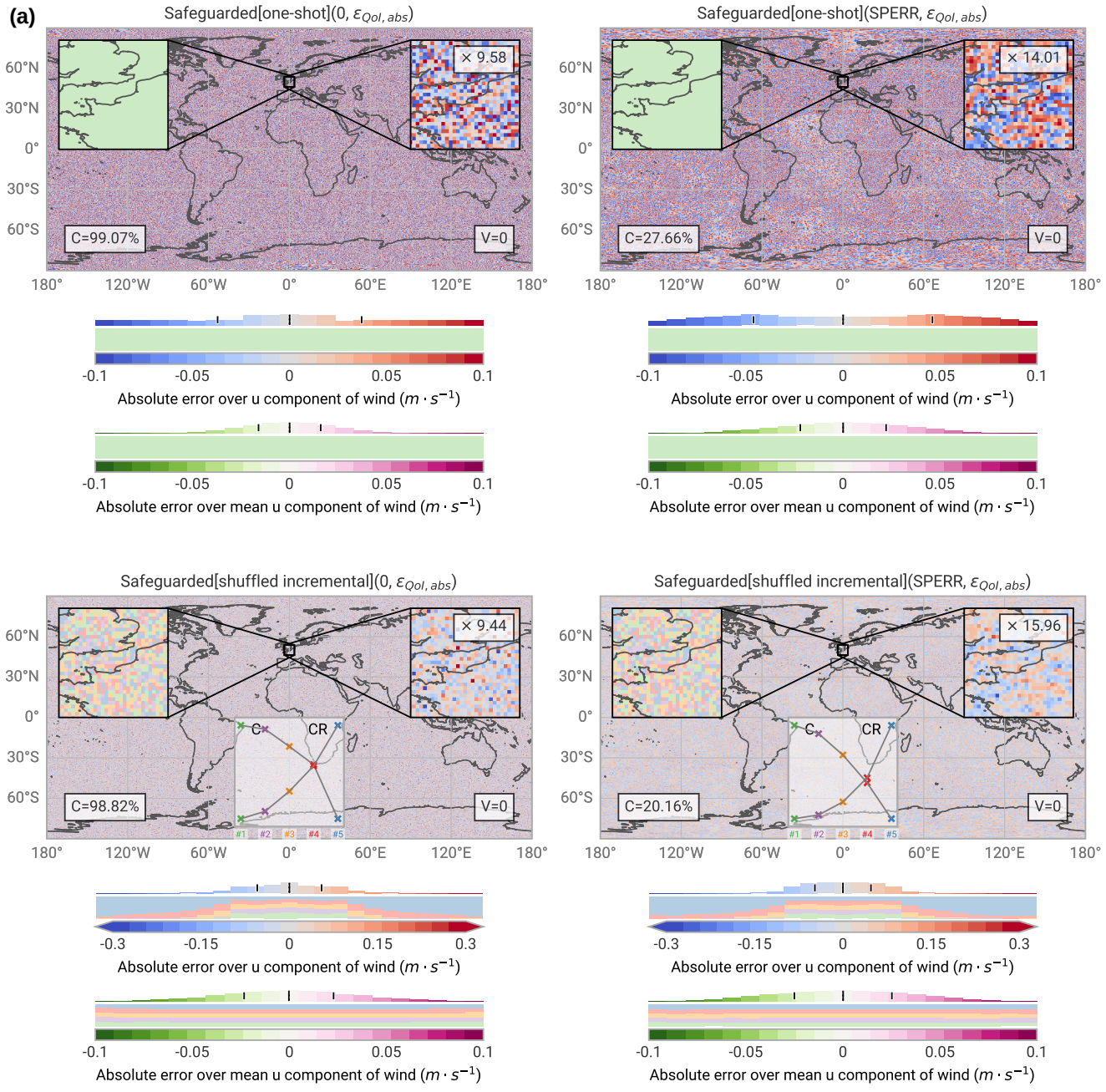
B9 Preview: incremental corrections

This appendix investigates the potential benefits of replacing the one-shot corrections in the `compression-safeguards` implementation with incrementally computed corrections. Incremental corrections can be less conservative for non-pointwise safeguards by conditioning the corrections for next data points on the already corrected previous points, potentially improving compression ratios. For this example, we bound the box mean u-wind to $0.1\mathrm{m\,s^{-1}}$. The unweighted box mean is computed over a five-element neighbourhood along the latitude axis (see listing C10). With one-shot corrections, preserving a mean error has to be equivalent to preserving a pointwise error, since we need to account for the worst-case error accumulation. However, some elements could tolerate a higher error because others compressed with a lower error.

We test an ad hoc implementation for computing incremental corrections, where all data points are partitioned into K disjoint subsets. The safeguards are applied K times and given access to the corrections from prior rounds, but only the corrections for this round's subset of data points are kept. We compare two variants of this approach: the shuffled method randomly partitions the data into $K = 5$ groups, while the step-by method partitions based on the latitude axis index modulo five. We test both with `SPERR` and the all-zero predictor. Here, `SPERR` has been configured with the peak signal-to-noise ratio bound $PSNR = 49.875$ that effectively bounds the root mean square error to roughly $0.1\mathrm{m\,s^{-1}}$; it will exceed the conservative pointwise absolute error bound for some but not all points.

Figure B10 shows that the incremental corrections become less conservative with every round, i.e. have larger data errors that still preserve the error bound over the block mean quantity of interest. Increasingly less conservative corrections also compress better, partly because they have lower entropy, which is more prevalent for the all-zero predictor, but mainly since fewer corrections will be needed, particularly when producing sparse corrections for `SPERR`. Even though the step-by incremental corrections produce a clear step pattern along the longitude axis, fewer corrections are needed than for the random partitioning, as Table B10 shows.

This brief investigation highlights that further improvements in compression ratios can be achieved by computing corrections in a different order instead of all at once. However, the overall positive impact of incremental corrections on compression ratios appears to be minor, which might be improved by iteratively refining the incremental corrections in future work.


(a)
Safeguarded[one-shot](0, $\epsilon_{QoI,abs}$)
× 9.58
C=99.07%
V=0
Safeguarded[one-shot](SPERR, $\epsilon_{QoI,abs}$)
× 14.01
C=27.66%
V=0
60°N
30°N
0°
30°S
60°S
180°
120°W
60°W
0°
60°E
120°E
180°
-0.1
-0.05
0
0.05
0.1
Absolute error over u component of wind ($m \cdot s^{-1}$)
Absolute error over mean u component of wind ($m \cdot s^{-1}$)
Safeguarded[shuffled incremental](0, $\epsilon_{QoI,abs}$)
× 9.44
C=98.82%
V=0
Safeguarded[shuffled incremental](SPERR, $\epsilon_{QoI,abs}$)
× 15.96
C=20.16%
V=0
C
CR
#1
#2
#3
#4
#5
-0.3
-0.15
0
0.15
0.3

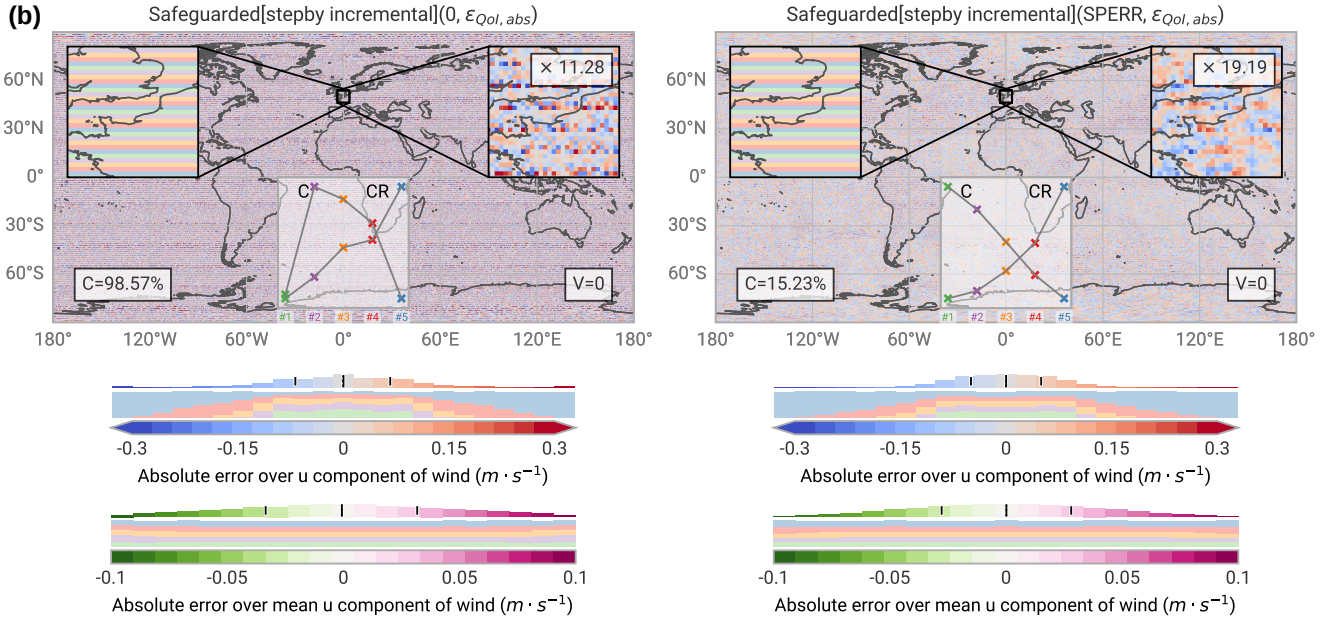


Figure B10. (a)-(b) Data error maps for compressed u-wind that has been safeguarded to bound the absolute error over the five-element block mean along the latitude axis. Three correction methods are compared, one per row: one-short, incremental with shuffled rounds, and incremental with step-by partitions. The left column shows the results for the safeguarded all-zero predictor, the right for safeguarded `SPERR`. Below the error distribution histogram and above the colour bars, the relative contributions of each incremental correction round towards the overall errors are shown. The change in the per-round percentage of corrections and compression ratios is plotted in the bottom centre of each map; the final compression ratio and percentage of corrections are given in the lower left and upper right corners. The smaller sub-maps in the top left and right highlight the partition pattern and data error details around the English Channel. While the colour bar and distribution for the error over the box mean are given as well, this error is not shown in the maps.

Compressor	Safeguarded	Corrections	$L_\infty(\hat{u})$	$L_2(\hat{u})$	$L_\infty(\overline{\hat{u}})$	$L_2(\overline{\hat{u}})$	V	C	CR
0	-		5.9e+01	1.5e+01	5.8e+01	1.5e+01	99.4%		× 415296.0
	$\epsilon_{QoI,abs}$	one-shot	0.1	0.052	0.095	0.024	0	99.1%	× 9.58
		shuffled incremental	0.71	0.087	0.1	0.034	0	98.8%	× 9.44
		stepby incremental	0.76	0.12	0.1	0.045	0	98.6%	× 11.28
SPERR(ϵ_{abs})	-		0.96	0.1	0.53	0.059	8.9%		× 58.86
	$\epsilon_{QoI,abs}$	one-shot	0.1	0.054	0.096	0.031	0	27.7%	× 14.01
		shuffled incremental	0.47	0.063	0.1	0.035	0	20.2%	× 15.96
		stepby incremental	0.65	0.071	0.1	0.039	0	15.2%	× **19.19**

Table B10. Compression metrics for the incremental corrections example. For each compressor, the L_∞ and L_2 errors over u and $\overline{u}$, the percentage of violations V, and the resulting compression ratio CR are given. For each safeguarded compressor and for the three correction methods, the percentage of corrections C are provided as well. The highest *safe* compression ratio is highlighted in bold.

B10 Performance evaluation

This appendix contains the throughput for the non-native lossy compressors in WASM.

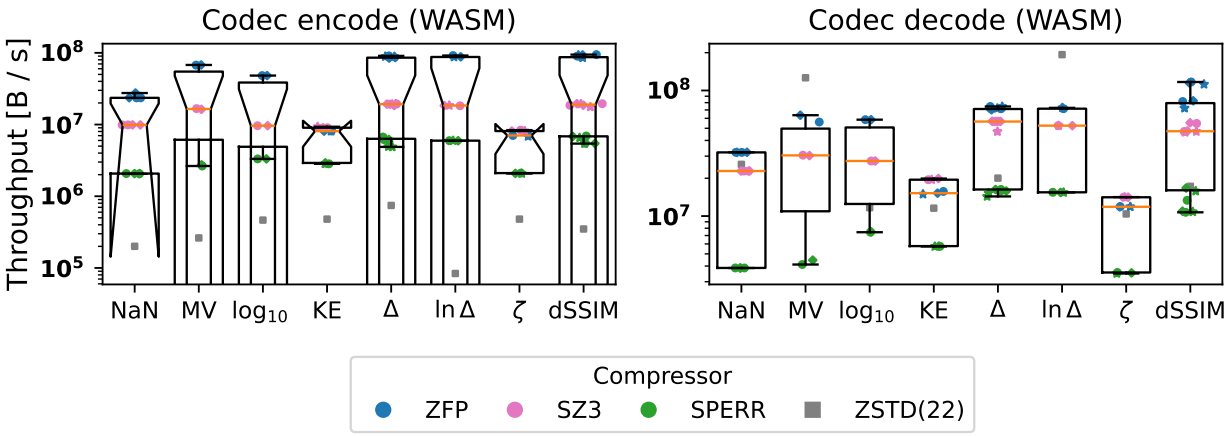


Figure B11. Throughput of compression and decompression, measured in B s^{-1}, with `ZFP`, `SZ3`, `SPERR`, and maximum-level `Zstandard` across all examples in section 5 and appendix B. The throughput is measured for the non-native single-threaded `numcodecs-wasm-*` compressors (see section 5.2), which 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).

Our evaluation uses the `numcodecs-wasm-*` compressors, which provide reproducible results at the cost of significantly reduced runtime performance. Therefore, the throughput measurements of the WASM codecs should not be directly compared with the safeguarding throughputs in Figure 5.

Figure B11 shows that `ZFP` has the highest compression throughput across all examples, while `SPERR` has the lowest. While there are some differences across examples, no clear pattern emerges. The kinetic energy and relative vorticity examples are exceptional, since they stack the wind u and v vectors into one dimension, which `ZFP` has to pad to size four, thus doubling its compression workload.

All three compressors have higher decompression throughputs, with `ZFP` again performing the best and `SPERR` performing the worst, with the same exception for kinetic energy and relative vorticity.

Figure B11 also compares the throughput of the lossless `ZStandard` compressor at its maximum level. While it has significantly lower compression throughput than `SPERR`, its decompression throughput typically lies between `SZ3`'s and `SPERR`'s, with the exception of the non-NaN missing value and logarithm of the Laplacian examples where `ZStandard` decompresses the fastest.

B11 ClimateBenchPress

This appendix contains additional results from ClimateBenchPress in section 5.8. The safeguards eliminate all violations.

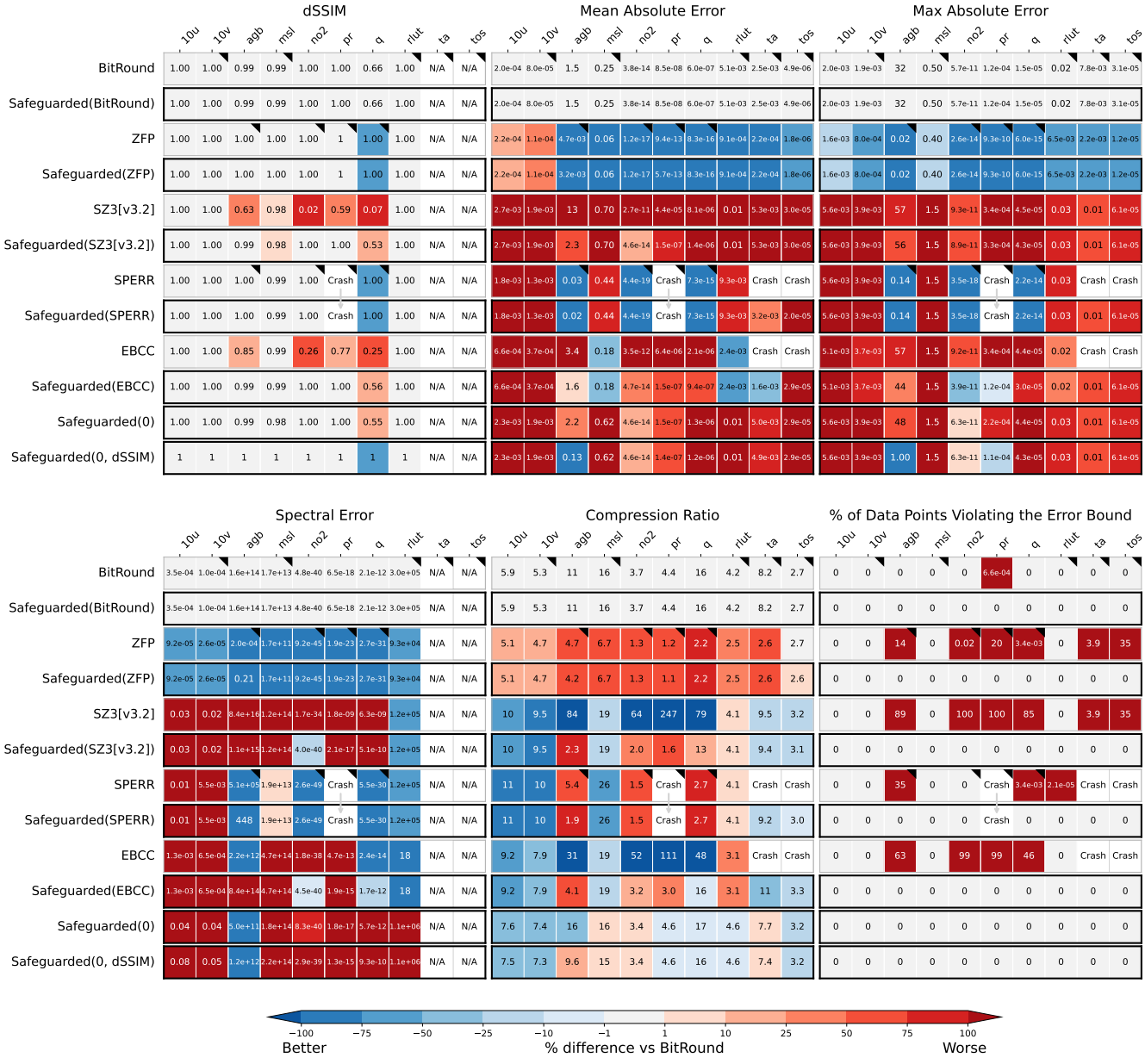


Figure B12. Scorecard of four distortion metrics, compression ratios, and error bound violations for several error-bounded lossy compressors and their `compression-safeguard`ed equivalents for the *low* level error bound. The dSSIM and spectral error are not computed for the CMIP6 "ta" and "tos" variables, which contain large regions of NaN values (Reichelt et al., 2026). Please see Figure 7 for the full description.

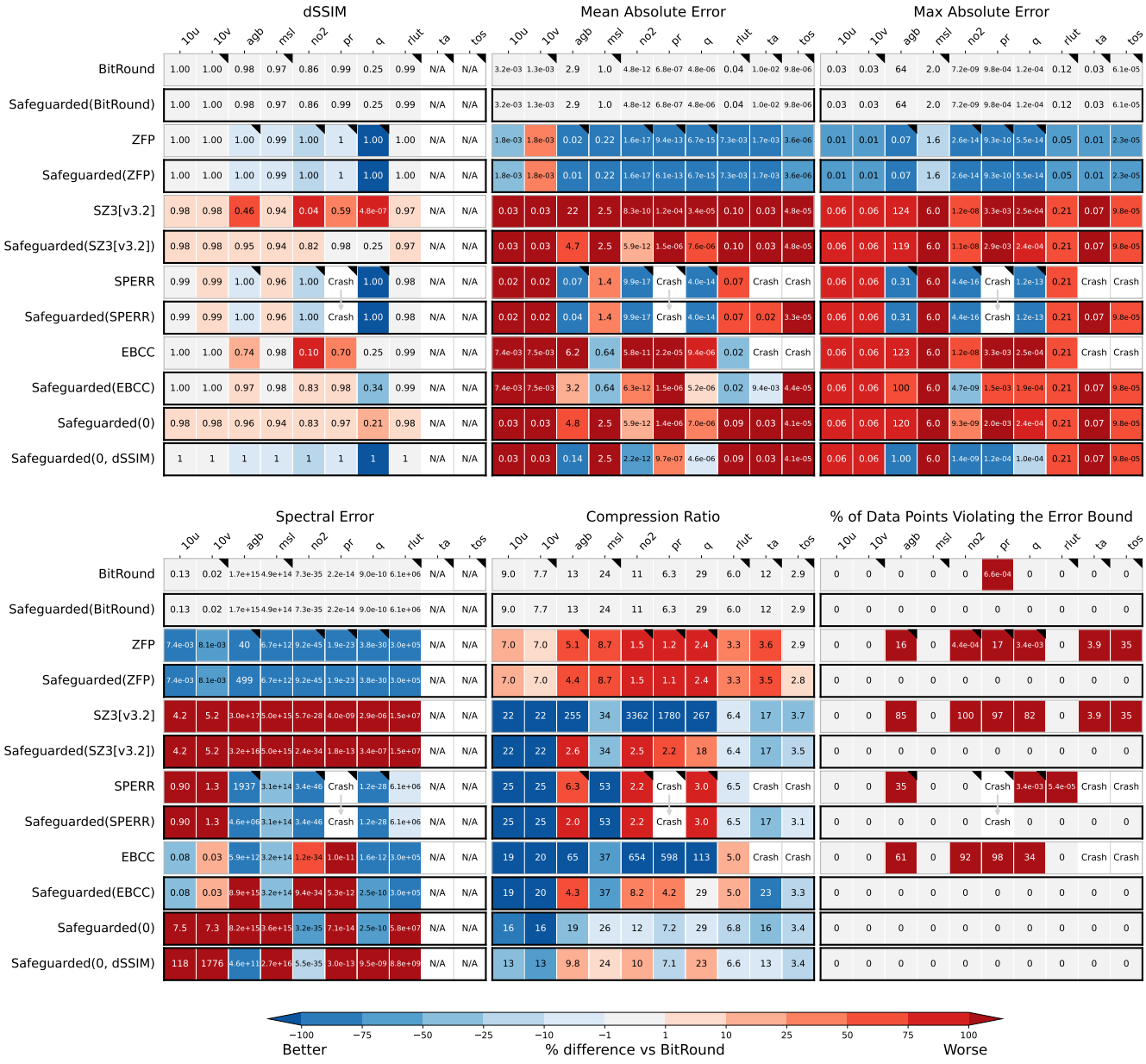


Figure B13. Scorecard of four distortion metrics, compression ratios, and error bound violations for several error-bounded lossy compressors and their `compression-safeguard`ed equivalents for the *mid* level error bound. Please see Figure 7 for the complete figure description.

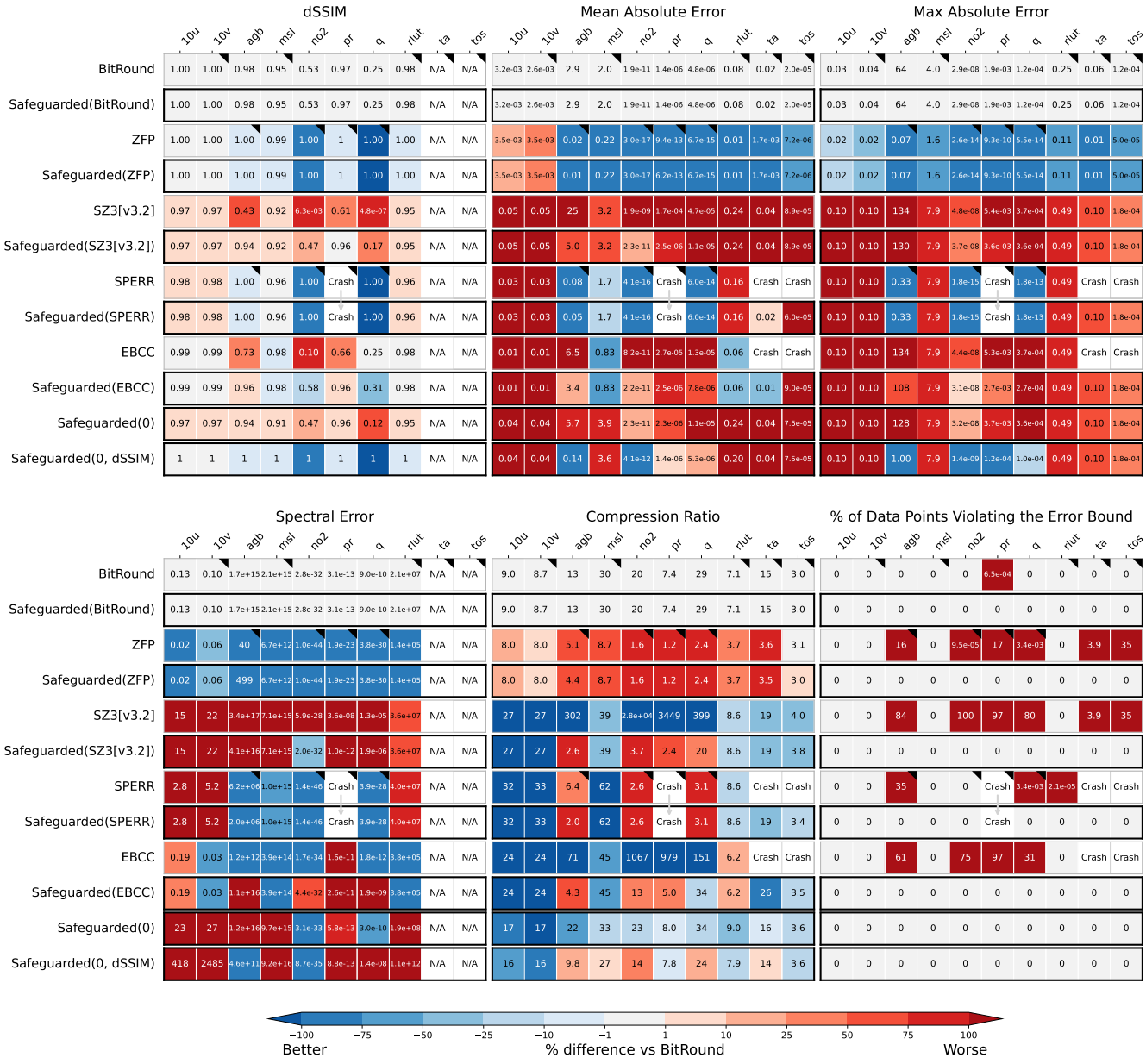


Figure B14. Scorecard of four distortion metrics, compression ratios, and error bound violations for several error-bounded lossy compressors and their `compression-safeguarded` equivalents for the *high* level error bound. Please see Figure 7 for the complete figure description.

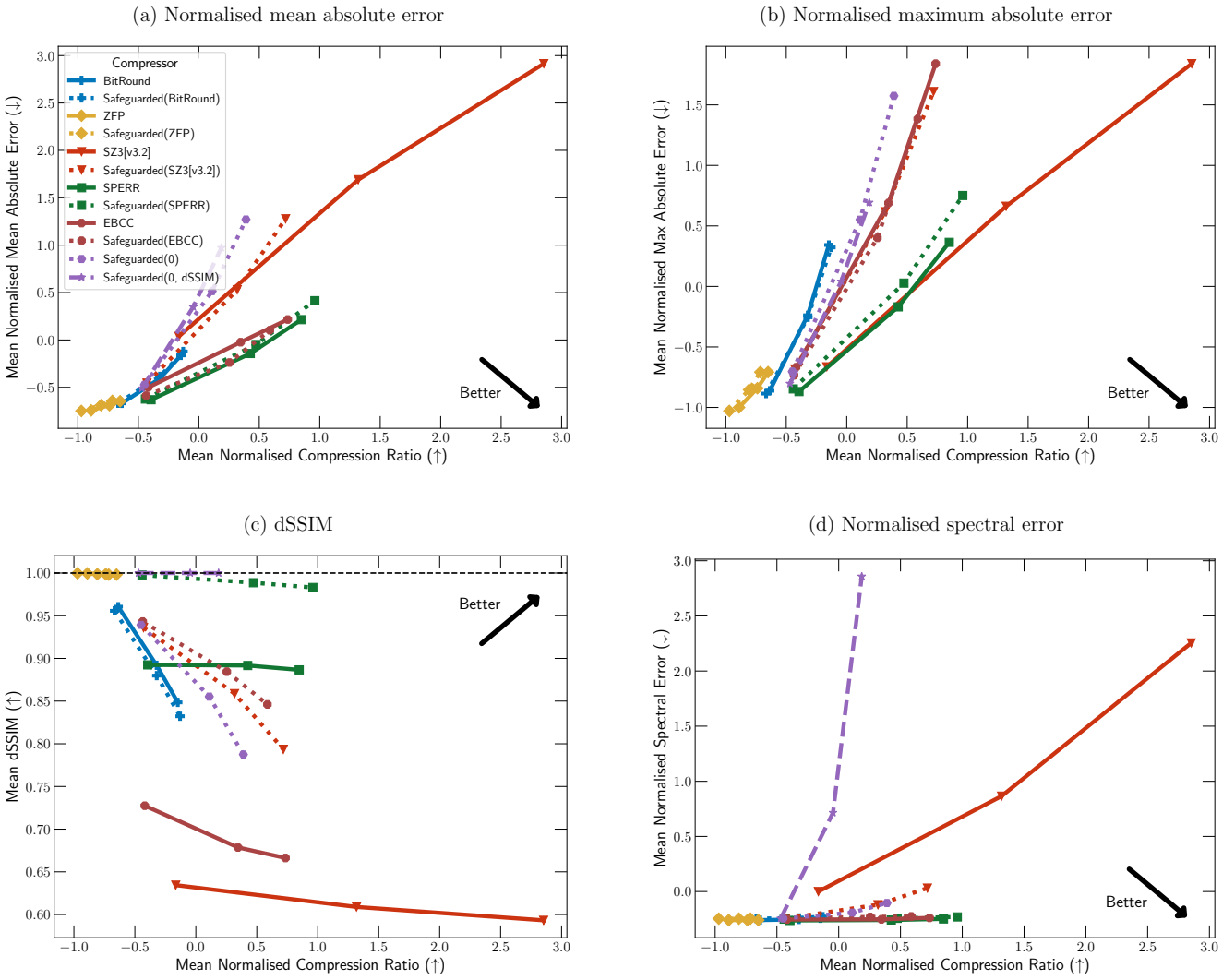


Figure B15. Comparison of the full rate-distortion plots, for four distortion metrics, for several error-bounded lossy compressors, shown as solid lines, and their `compression-safeguarded` equivalents, shown as dotted lines. Each plot shows the compression ratio on the x-axis and a different distortion metric on the y-axis. The compression ratios and rate-distortions for each compressor are normalised across all variables as in Figure 3 of Reichelt et al. (2026), except for dSSIM. In the dSSIM scores panel, the optimal dSSIM $= 1$ is highlighted as a dashed line for comparison. This figure is the uncropped version of Figure 8.

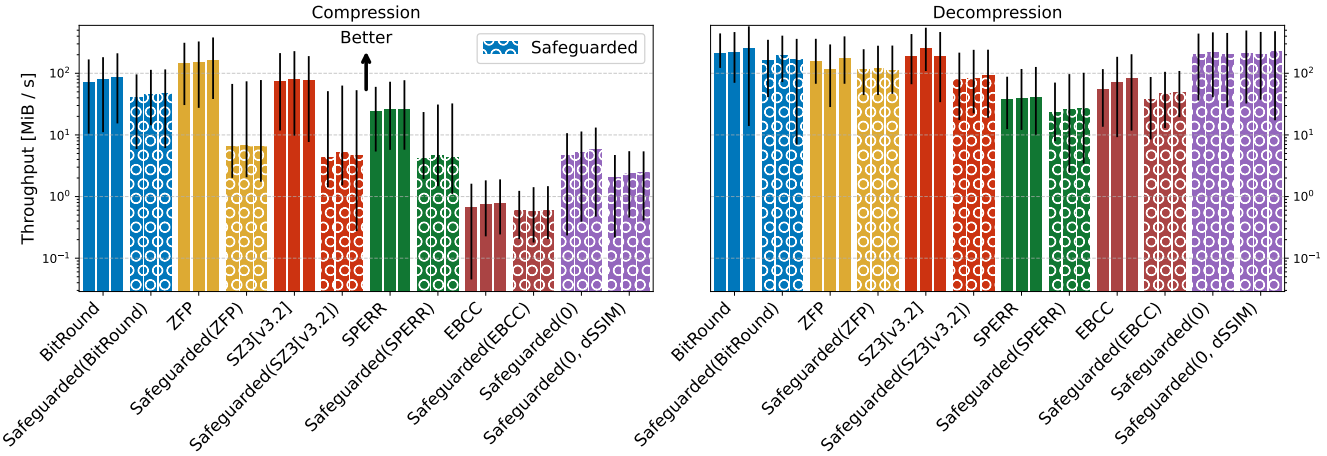


Figure B16. Comparison of the compression and decompression throughput, measured in MiB s^{-1}, across all variables in Figure 7, for several error-bounded lossy compressors, in solid bars, and their `compression-safeguard`ed equivalents, in hatched bars, for the low, mid, and high error bound levels from left to right. Only the absolute or relative error bounds are safeguarded, except for `Safeguarded(0, dSSIM)` which also guarantees a perfect dSSIM score. Each bar height represents the median across all variables, the uncertainty lines indicate the 25% and 75% quantiles. The throughput is measured with the single-threaded `numcodecs-wasm-*` compressors (see section 5.2), which have significant overhead to guarantee reproducible execution. While native compressors would have significantly higher throughput, the throughput of their safeguarded counterparts would remain bottlenecked by the low throughput of computing corrections during safeguarded compression. Therefore, the throughput measurements should only be used for relative comparison between only safeguarded, or between only non-safeguarded compressors. The measurements are run on the same setup used by Reichelt et al. (2026), an AMD EPYC 9654 CPU.

We also evaluate the runtime performance, i.e. the time taken to compress and decompress, using ClimateBenchPress. ClimateBenchPress also uses the `numcodecs-wasm-*` compressors, which provide reproducible results at the cost of significantly reduced runtime performance, which only permits relative comparisons between only safeguarded, and between only non-safeguarded compressors. While these compressors are single-threaded, they can be externally parallelised over data chunks. ClimateBenchPress can use WebAssembly-bytecode-based instruction counters to produce cross-platform reproducible instructions-per-compressed-byte measurements. Since the `compression-safeguards` are implemented in pure Python and are not supported by this metric, we only report the wall-clock-based throughput for each compressor, even though it is platform-dependent and contains measurement noise.

Figure B16 shows that `ZFP` has the highest compression throughput of the lossy error-bounded compressors and `EBCC` the lowest. For most compressors, excluding `ZFP`, decompression has significantly higher throughput than compression, with `BitRound` and `SZ3` performing the best and `SPERR` the worst. Most compressors, excluding `SZ3`, have slightly higher compression throughput for higher error bounds, while only `EBCC` displays this clearly at decompression time. Applying safeguards to the compressors reduces the throughput significantly during compression, especially for `ZFP`, `SZ3`, and `SPERR`.

At decompression time, the overhead is typically only minor, except for `SZ3`, where many corrections have to be decompressed. The compression throughput of the safeguarded zero-predictor is low, outperforming only safeguarded `EBCC`, even more so when preserving the dSSIM metric, showcasing that the compression time cost increases with the number of corrections and the complexity of the safety requirements that are being preserved. At decompression time, they have a high throughput that is significantly better than that of the other safeguarded compressors. The results mirror the outcomes from appendix B10.

B12 Performance scaling of chunked safeguarding with the `xarray-safeguards` frontend

This appendix investigates the scaling of the runtime performance and memory footprint of safeguarding chunked data using the `xarray-safeguards` frontend (see section 3.6). For this investigation, we reuse the relative vorticity example from section 5.6, since relative vorticity is a non-pointwise and non-linear quantity of interest that computes spatial derivatives across the longitude and latitude axes and makes use of boundary conditions. To support cross-chunk corrections for non-pointwise safeguards, the `xarray-safeguards` and `compression-safeguards` internally extend each chunk with some overlap (see section 3.6, which increases computational load. We only test the `SZ3` compressor since its violations produce one-shot corrections that are spread throughout (see Figure B8). We measure the real time, also called wall-clock time, and the allocated memory footprint across different chunkings of the data. The measurements were performed on an Apple M2 CPU running Python 3.11 with `dask` 2025.12.0, `numpy` 2.4.6, and `xarray` 2025.12.0.

To establish a baseline, we first measure the performance of compressing and safeguarding all data at once using the non-chunked `compression-safeguards` API. Next, we compare using the low-level chunked APIs in the `compression-safeguards` directly, again to safeguard a single chunk. Finally, we use the chunking-aware `xarray-safeguards` frontend to safeguard the relative vorticity across an increasing number of chunks and check that it produces equivalent corrections. Between the steps of compressing and decompressing the data, computing the corrections, compressing and decompressing the corrections, and applying the corrections, we force computation and materialise the chunked results into memory. Importantly, as this investigation is focused on chunking overheads but not on the general scalability of multi-thread or multi-node parallelism, we compute across the chunks without any multithreading, which also makes non-chunked and chunked performance measurements directly comparable.

Figure B17 shows that the costs of compressing and decompressing and applying corrections both grow as more chunks are used and the overlap between chunks increases. The highly-performant application of corrections (see section 5.7) has only negligible per-chunk overhead itself, so the difference between the non-chunked baseline and single-chunk case as well as the increase in runtime likely come entirely from `xarray` and `dask`. Compressing and decompressing the corrections initially displays a similar increase, but with a higher baseline, which is likely caused by the increasing per-compression overhead. Applying corrections eventually overtakes compressing corrections in overhead; this is likely caused by `xarray` having to combine two arrays in application but only map over one in compression, leading to the worse scaling.

Computing the corrections, which is the most expensive operation, at first shows a very minor decrease in runtime as the number of chunks increase, before also adopting a similar runtime increase. The initial decrease likely comes from the quantity of interest implementation in the `compression-safeguards`, which has to check for and correct rounding errors after every propagation of the safe intervals throughout the expression tree (see section 3.4). When chunks are smaller, sparse rounding error corrections only require changing smaller arrays, which reduces the total runtime, which could be employed as an internal performance optimisation in the future. As the number of chunks further increases, the per-compute-corrections and chunk-overlap overheads outweigh these small benefits.

The peak memory usage, which is primarily impacted by computing the corrections, also first declines before increasing again. The initial decrease likely comes from scheduling the chunked computations serially, such that each computation now only runs over a smaller data subset. The later decrease likely comes from the increasing memory overhead of the higher number of chunks, its associated task graph, and the required overlap between chunks. The final memory usage also increases, though its cause is less clear.

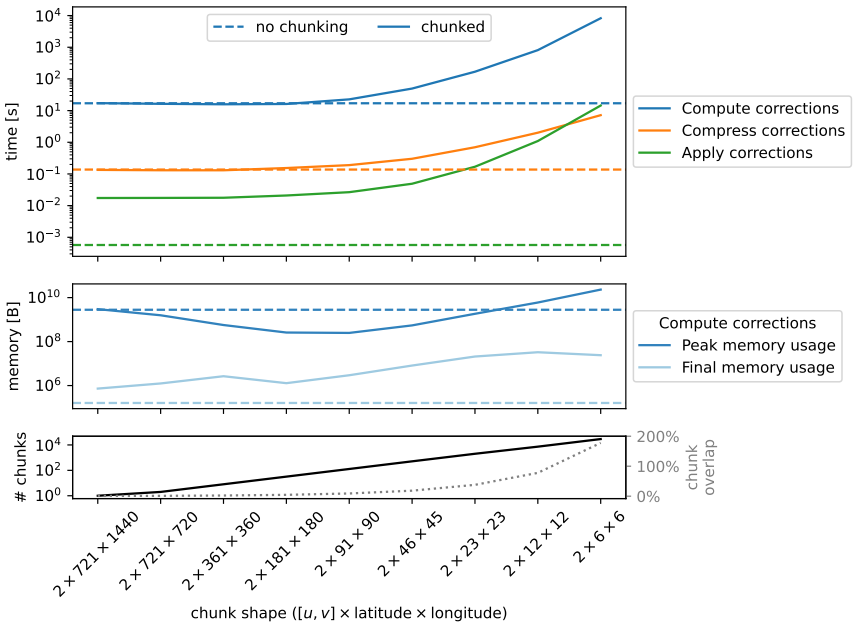


Figure B17. Scaling of single-threaded wall-clock runtime, memory footprint, and chunk overlap with the `xarray-safeguards` frontend when safeguarding the relative vorticity quantity of interest (see appendix B7) across an increasing number of chunks. The upper plot shows the wall-clock time, the middle 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.

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.

Appendix C: Experimental Evaluation: Safeguard Configurations

C1 Example: NaN missing values

```
SafeguardedCodec(
    codec=codec,  # any numcodecs codec
    safeguards=[
        # preserve isnan(x), but NaN payload bit patterns may change
        dict(kind="eb", type="abs", eb=0.1, equal_nan=True),
    ],
)
```

Listing C1. Configuration for bounding the absolute error and preserving NaN missing values using `numcodecs-safeguards` in section 5.4 and appendix B1.

C2 Example: non-NaN missing values

```
SafeguardedCodec(
    codec=codec,  # any numcodecs codec
    safeguards=[
        dict(kind="eb", type="abs", eb=1.0),
        dict(kind="same", value=9999, exclusive=True),
    ],
)
```

Listing C2. Configuration for bounding the absolute error and preserving the special non-NaN missing value 9999 using `numcodecs-safeguards` in appendix B2.

C3 Example: logarithm as a pointwise quantity of interest

```
SafeguardedCodec(
    codec=codec,  # any numcodecs codec
    safeguards=[
        dict(kind="qoi_eb_pw", qoi="log10(x)", type="abs", eb=0.25)
    ],
)
```

Listing C3. Configuration for bounding the the absolute error over the pointwise $\log_{10}(x)$ quantity of interest using `numcodecs-safeguards` in appendix B4.

C4 Example: kinetic energy as a multi-variable pointwise quantity of interest

```
SafeguardedCodec(
    codec=codec,  # any numcodecs codec
    safeguards=[
        dict(
            kind="qoi_eb_stencil",
            # u and v are extracted from the stacked dimension
            qoi="0.5 * (square(X[0]) + square(X[1]))",
            type="abs",
            eb=1.0,
            neighbourhood=[
                # [u, v]: stacked variables
                dict(axis=0, before=0, after=1, boundary="valid"),
            ],
        )
    ],
)
```

Listing C4. Configuration for bounding the the absolute error over the multi-variable pointwise kinetic energy quantity of interest using `numcodecs-safeguards` in section 5.5 and appendix B5.

C5 Example: laplacian as a stencil quantity of interest

```
SafeguardedCodec(
    codec=codec,  # any numcodecs codec
    safeguards=[
        # we use a constant grid spacing and the valid boundary condition
        # to preserve an absolute error bound over the Laplacian, but not
        # on the data boundaries where data points do not have a left,
        # right, upper, or lower neighbour
        dict(
            kind="qoi_eb_stencil",
            qoi="""
            (
                finite_difference(
                    # type=0 refers to a central difference
                    x, order=2, accuracy=2, type=0,
                    axis=0, grid_spacing=c["dy"]
                ) + finite_difference(
                    x, order=2, accuracy=2, type=0,
                    axis=1, grid_spacing=c["dx"]
                )
            )
            """,
            neighbourhood=[
                dict(axis=0, before=1, after=1, boundary="valid"),
                dict(axis=1, before=1, after=1, boundary="valid"),
            ],
            type="abs",
            eb=0.01,
        ),
    ],
    fixed_constants=dict(dx=dx, dy=dy),
)
```

Listing C5. Configuration for bounding the absolute error of the Laplacian over a 2D grid with fixed grid point spacing using `numcodecs-safeguards` in appendix B6. The Laplacian is approximated using the 2nd order central finite difference on the inner points, but not on the boundary points.

```
SafeguardsCodec(
    codec=codec,  # any numcodecs codec
    safeguards=[
        # we allow for arbitrary grid point spacing,
        # and switch from 2nd order central finite differences to 2nd
        #  order forward/backwards finite differences at the data
        #  boundaries, where data points do not have a left, right, upper,
        #  and lower neighbour
        # this enables us to compute the Laplacian at all data points,
        #  including at the boundaries
        # we pad the data with NaNs to denote the missing values
        dict(
            kind="qoi_eb_stencil",
            qoi="""
                # declare a new variable d2Udy2
                V["d2Udy2"] = where(
                    c["Y"] > -1,
                    where(
                        c["Y"] < 1,
                        # use 2nd order central finite difference,
                        # where possible
                        finite_difference(
                            x, order=2, accuracy=2, type=0, axis=0,
                            grid_centre=c["Y"]
                        ),
                        # use 2nd order backwards finite difference
                        # at the upper boundary
                        finite_difference(
                            x, order=2, accuracy=1, type=-1, axis=0,
                            grid_centre=c["Y"]
                        ),
                    ),
                    # use 2nd order forward finite difference at the
                    # lower boundary
                    finite_difference(
                        x, order=2, accuracy=1, type=1, axis=0,
                        grid_centre=c["Y"]
```

```
                    ),
                );

                V["d2Udx2"] = where(
                    c["X"] > -1,
                    where(
                        c["X"] < 1,
                        # use 2nd order central finite difference
                        # where possible
                        finite_difference(
                            x, order=2, accuracy=2, type=0, axis=1,
                            grid_centre=c["X"]
                        ),
                        # use 2nd order backwards finite difference
                        # at the right boundary
                        finite_difference(
                            x, order=2, accuracy=1, type=-1, axis=1,
                            grid_centre=c["X"]
                        ),
                    ),
                    # use 2nd order forward finite difference at the
                    # left boundary
                    finite_difference(
                        x, order=2, accuracy=1, type=1, axis=1,
                        grid_centre=c["X"]
                    ),
                );

                # compute the natural logarithm of the Laplacian
                return ln(V["d2Udy2"] + V["d2Udx2"]);
            """,
            neighbourhood=[
                dict(
                    axis=0, before=1, after=1, boundary="constant",
                    constant_boundary=numpy.nan
                ),
                dict(
                    axis=1, before=1, after=1, boundary="constant",
```

```
                        constant_boundary=numpy.nan
                    ),
                ],
                type="abs",
                eb=0.1,
            ),
        ],
        fixed_constants=dict(X=X, Y=Y),
)
```

Listing C6. Configuration for bounding the absolute error over the natural logarithm of the Laplacian over a 2D grid with arbitrary grid point locations using `numcodecs-safeguards` in appendix B6. The Laplacian is approximated using the 2nd order central finite difference on inner points and the 2nd order forward/backwards finite difference on boundary points.

C6 Example: relative vorticity as a stencil quantity of interest

```
SafeguardsCodec(
    codec=codec,  # any numcodecs codec
    safeguards=[
        dict(
            kind="qoi_eb_stencil",
            qoi="""
                V["earth_radius"] = 6371000;  # [m], globally averaged

                # extract u and v from the stacked (first) dimension
                V["u"] = X[0, I[1], I[2]];
                V["v"] = X[1, I[1], I[2]];
                # convert latitude in degrees to radians
                V["latRad"] = c["lat"] * pi / 180;

                # (1) along the longitude axis, we handle the 360 degree periodic
                #     boundary
                V["dVdPhi"] = finite_difference(
                    V["v"],
                    order=1, accuracy=2, type=0, axis=2,
                    grid_centre=c["lon"], grid_period=360,
                );

                # approximate the spatial derivatives with finite differences
                # (2) along the latitude axis, we need to handle the polar boundary
                V["dUdTheta"] = where(
                    abs(c["lat"]) < 90,
                    # use central difference where possible
                    finite_difference(
                        V["u"] * cos(V["latRad"]),
                        order=1, accuracy=2, type=0, axis=1,
                        grid_centre=c["lat"],
                    ),
                    # at the poles, use forward/backwards difference
                    where(
                        c["lat"] == -90,
                        finite_difference(
```

```
                                V["u"] * cos(V["latRad"]),
                                order=1, accuracy=1, type=-1, axis=1,
                                grid_centre=c["lat"],
                            ),
                            finite_difference(
                                V["u"] * cos(V["latRad"]),
                                order=1, accuracy=1, type=+1, axis=1,
                                grid_centre=c["lat"],
                            ),
                        ),
                    );

                    # compute the relative vorticity
                    return (V["dVdPhi"] - V["dUdTheta"]) / (
                        V["earth_radius"] * cos(V["latRad"])
                    );
                """,
                type="abs",
                eb=1e-8,
                neighbourhood=[
                    # [u, v]: stacked variables
                    dict(axis=0, before=0, after=1, boundary="valid"),
                    # latitude: edge boundary is good enough since we ensure in the
                    #           QoI that the edge-extended values are not used
                    dict(axis=1, before=1, after=1, boundary="edge"),
                    # longitude: wrapping boundary
                    dict(axis=2, before=1, after=1, boundary="wrap"),
                ],
            )
        ],
        # provide the latitude and longitude dimensions as late-bound fixed constants
        # and ensure that they can be broadcast to the data shape
        fixed_constants=dict(
            lat=latitude.reshape(1, -1, 1), lon=longitude.reshape(1, 1, -1),
        ),
    )
```

Listing C7. Configuration for bounding the absolute error of the relative vorticity over a latitude-longitude grid with arbitrary grid point locations using `numcodecs-safeguards` in section 5.6 and appendix B7.

C7 Example: data structural similarity index measure

```
SafeguardsCodec(
    codec=codec,  # any numcodecs codec
    safeguards=[
        # guarantee that the global minimum and maximum are preserved,
        #  which simplifies the rescaling in the quantity of interest
        dict(kind="sign", offset="$x_min"),
        dict(kind="sign", offset="$x_max"),
        dict(
            kind="qoi_eb_stencil",
            qoi="""
                # dssim constants
                V["sigma"] = 1.5;
                V["C1"] = 1e-8;
                V["C2"] = 1e-8;

                # we guarantee that
                #  min(data) = min(corrected) and
                #  max(data) = max(corrected)
                # with the sign safeguards above
                # dSSIM assumes that there are no infinite values
                V["smin"] = c["$x_min"];
                V["smax"] = c["$x_max"];
                V["r"] = V["smax"] - V["smin"];

                # re-scale to [0-1] and quantise to 256 bins
                V["sc_a1"] = round_ties_even(
                    ((C["$X"] - V["smin"]) / V["r"]) * 255
                ) / 255;
                V["sc_a2"] = round_ties_even(
                    ((X - V["smin"]) / V["r"]) * 255
                ) / 255;

                # create a 2D 3x3 Gaussian kernel
                V["i"] = [-1, 0, 1];
                V["k"] = 1/(sqrt(2*pi)*V["sigma"]) * exp(
                    (-square(V["i"])) / (2*square(V["sigma"]))
```

```
        );
        V["kern"] = matmul([V["k"]].T, [V["k"]]);

        # handle sparse NaN values by excluding them from the
        # convolution, then renormalise the kernel
        V["kern_no_NaN1"] = where(isnan(V["sc_a1"]), 0, V["kern"]);
        V["kernel1"] = V["kern_no_NaN1"] / sum(V["kern_no_NaN1"]);
        V["kern_no_NaN2"] = where(isnan(V["sc_a2"]), 0, V["kern"]);
        V["kernel2"] = V["kern_no_NaN2"] / sum(V["kern_no_NaN2"]);
        V["kern_no_NaN12"] = where(
            isnan(V["sc_a1"] * V["sc_a2"]), 0, V["kern"]
        );
        V["kernel12"] = (
            V["kern_no_NaN12"] / sum(V["kern_no_NaN12"])
        );

        # apply the Gaussian mean filter via convolution
        V["a1_mu_noNaN"] = sum(V["sc_a1"] * V["kernel1"]);
        V["a2_mu_noNaN"] = sum(V["sc_a2"] * V["kernel2"]);
        V["a1a1_noNaN"] = sum(square(V["sc_a1"]) * V["kernel1"]);
        V["a2a2_noNaN"] = sum(square(V["sc_a2"]) * V["kernel2"]);
        V["a1a2_noNaN"] = sum(
            V["sc_a1"] * V["sc_a2"] * V["kernel12"]
        );

        # keep NaN for elements that are originally NaN
        V["a1_mu"] = where(
            isnan(V["sc_a1"])[I], NaN, V["a1_mu_noNaN"]
        );
        V["a2_mu"] = where(
            isnan(V["sc_a2"])[I], NaN, V["a2_mu_noNaN"]
        );
        V["a1a1"] = where(
            isnan(V["sc_a1"])[I], NaN, V["a1a1_noNaN"]
        );
        V["a2a2"] = where(
            isnan(V["sc_a2"])[I], NaN, V["a2a2_noNaN"]
        );
```

```
                V["a1a2"] = where(
                    isnan(V["sc_a1"] * V["sc_a2"])[I], NaN, V["a1a2_noNaN"]
                );

                # compute the variances and covariance
                V["var_a1"] = V["a1a1"] - square(V["a1_mu"]);
                V["var_a2"] = V["a2a2"] - square(V["a2_mu"]);
                V["cov_a1a2"] = V["a1a2"] - V["a1_mu"] * V["a2_mu"];

                # compute the SSIM components
                V["ssim_t1"] = 2 * V["a1_mu"] * V["a2_mu"] + V["C1"];
                V["ssim_t2"] = 2 * V["cov_a1a2"] + V["C2"];
                V["ssim_b1"] = (
                    square(V["a1_mu"]) + square(V["a2_mu"]) + V["C1"]
                );
                V["ssim_b2"] = V["var_a1"] + V["var_a2"] + V["C2"];

                V["ssim_1"] = V["ssim_t1"] / V["ssim_b1"];
                V["ssim_2"] = V["ssim_t2"] / V["ssim_b2"];

                # compute the pointwise dSSIM
                return V["ssim_1"] * V["ssim_2"];
            """,
            type="abs",
            # guarantee dSSIM >= 0.995
            eb=1 - 0.995,
            # 3x3 neighbourhood
            neighbourhood=[
                # latitude
                dict(axis=0, before=1, after=1, boundary="valid"),
                # longitude
                dict(axis=1, before=1, after=1, boundary="valid"),
            ],
        ),
    ],
)
```

Listing C8. Configuration for ensuring a minimum pointwise 3x3 dSSIM score over a 2D grid using `numcodecs-safeguards` in appendix B8.

```
SafeguardedCodec(
    codec=codec,  # any numcodecs codec
    safeguards=[
        # guarantee that the global minimum and maximum are preserved
        dict(kind="sign", offset="$x_min"),
        dict(kind="sign", offset="$x_max"),
        dict(
            kind="qoi_eb_pw",
            qoi="""
                # the above sign safeguards guarantee that
                #  min(data) = min(corrected) and
                #  max(data) = max(corrected)
                v["smin"] = c["$x_min"];
                v["smax"] = c["$x_max"];
                v["r"] = v["smax"] - v["smin"];

                # re-scale to [0-1] and quantise to 256 bins
                v["sc_a2"] = round_ties_even(
                    ((x - v["smin"]) / v["r"]) * 255
                ) / 255;

                # force the quantised value to stay the same
                return v["sc_a2"];
            """,
            type="abs",
            eb=0,
        ),
    ],
)
```

Listing C9. Configuration for ensuring a perfect pointwise dSSIM score using `numcodecs-safeguards` in appendices B8 and B11 and section 5.8.

C8 Preview: incremental corrections

```
SafeguardsCodec(
    codec=codec,  # any numcodecs codec
    safeguards=[
        dict(
            kind="qoi_eb_stencil",
            qoi="sum(X) / size(X)",  # box mean
            neighbourhood=[
                # neighbourhood along the latitude axis
                dict(axis=1, before=2, after=2, boundary="reflect"),
            ],
            type="abs",
            eb=0.1,
        ),
    ],
)
```

Listing C10. Configuration for bounding the absolute error over the five-element box mean along the latitude axis using `numcodecs-safeguards` in appendix B9.

Author contributions. Juniper Tyree: Conceptualisation, Methodology, Software, Investigation, Writing - Original Draft, Visualisation
Robert Underwood: Methodology, Investigation, Writing - Review & Editing
Clément Bouvier: Investigation, Writing - Review & Editing, Supervision
Daniel Köhler: Investigation, Writing - Review & Editing, Visualisation
Tim Reichelt: Investigation, Writing - Review & Editing, Visualisation
Peter Dueben: Writing - Review & Editing
Sara Faghih-Naini: Writing - Review & Editing
Heikki J. Järvinen: Writing - Review & Editing, Supervision, Funding acquisition
Milan Klöwer: Methodology, Writing - Review & Editing, Supervision

Competing interests. The contact author has declared that none of the authors has any competing interests.

Acknowledgements. Juniper Tyree wants to thank her colleagues in the Dynamic Meteorology group at the University of Helsinki and the Climate Modelling Research Group at the University of Oxford for their valuable conversations during the development of the Compression Safeguards, in particular Clément and Victoria who always found time to give feedback and help out. Juniper wants to further thank Swayam Singh and Nathan Goldbaum for developing Numpy-QuadDType and accommodating the needs and bug fixes for this project. Juniper also wishes to acknowledge Ulla Eronen, Janika Lähdes, Niina Lehtimäki, Maria Ruuska, and Liisa Mayow from Kaskas for their fruitful work and collaboration on the infographics and pictograms for the Compression Safeguards. Most importantly, Juniper wants to express her gratitude to Arden, Keagan, and Rhea for enduring months of her struggle with floating-point arithmetic edge cases.

Juniper Tyree, Peter Dueben, Sara Faghih-Naini, and Heikki J. Järvinen are funded by the ESiWACE3 Centre of Excellence. Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) under grant agreement No 101093054. This work received 50% complementary funding from the Research Council of Finland under grant No 359787.

Robert Underwood is funded by the National Science Foundation (NSF) CSSI "FZ" project with Grant #2311875.

Clément Bouvier is funded by the European Union's Destination Earth Initiative and the Research Council of Finland (grant nos. 338615 and 337549).

Daniel Köhler is funded by the University of Helsinki Doctoral School.

Tim Reichelt received funding from ARIA, and DSIT, and Pillar VC under the Encode: AI for Science Fellowship.

Peter Dueben acknowledges funding from the WeatherGenerator Horizon Europe project (grant No 101187947).

Milan Klöwer acknowledges funding from the Natural Environment Research Council under grant number UKRI191.

References

Agarwal, T., Di, S., Liang, X., Su, Z., Li, Y., Gopalakrishnan, G., Guo, H., and Cappello, F.: TopoSZp: Lightweight Topology-Aware Error-controlled Compression for Scientific Data, https://doi.org/10.48550/ARXIV.2602.17552, 2026.

Ainsworth, M., Tugluk, O., Whitney, B., and Klasky, S.: Multilevel Techniques for Compression and Reduction of Scientific Data-Quantitative Control of Accuracy in Derived Quantities, SIAM Journal on Scientific Computing, 41, A2146–A2171, https://doi.org/10.1137/18M1208885, 2019.

Alted, F.: c-blosc: Meta-compresion and advantages over other existing compressors, GitHub repository, https://github.com/Blosc/c-blosc/commit/f445706cda788b5ac52a9b5a334368cc69a06c53, 2010.

Amici, A.: CFGRIB: easy and efficient GRIB file access in xarray, https://www.ecmwf.int/en/elibrary/80908-cfgrib-easy-and-efficient-grib-file-access-xarray, 2018.

Andersson, A., Graw, K., Schröder, M., Fennig, K., Liman, J., Bakan, S., Raphoe, N., Hollmann, R., and Klepp, C.: Hamburg Ocean Atmosphere Parameters and Fluxes from Satellite Data - HOAPS4.0, https://doi.org/10.24381/cds.92db7fef, accessed: 16th April 2025, 2021.

Baker, A. H.: On Preserving Scientific Integrity for Climate Model Data in the HPC Era, Computing in Science & Engineering, 23, 16–24, https://doi.org/10.1109/MCSE.2021.3119509, 2021.

Baker, A. H., Xu, H., Dennis, J. M., Levy, M. N., Nychka, D., Mickelson, S. A., Edwards, J., Vertenstein, M., and Wegener, A.: A methodology for evaluating the impact of data compression on climate simulation data, in: Proceedings of the 23rd International Symposium on High-Performance Parallel and Distributed Computing, HPDC '14, p. 203–214, Association for Computing Machinery, New York, NY, USA, ISBN 9781450327497, https://doi.org/10.1145/2600212.2600217, 2014.

Baker, A. H., Hammerling, D. M., Mickelson, S. A., Xu, H., Stolpe, M. B., Naveau, P., Sanderson, B., Ebert-Uphoff, I., Samarasinghe, S., De Simone, F., Carbone, F., Gencarelli, C. N., Dennis, J. M., Kay, J. E., and Lindstrom, P.: Evaluating lossy data compression on climate simulation data within a large ensemble, Geoscientific Model Development, 9, 4381–4403, https://doi.org/10.5194/gmd-9-4381-2016, 2016.

Baker, A. H., Xu, H., Hammerling, D. M., Li, S., and Clyne, J. P.: Toward a Multi-method Approach: Lossy Data Compression for Climate Simulation Data, in: High Performance Computing, edited by Kunkel, J. M., Yokota, R., Taufer, M., and Shalf, J., pp. 30–42, Springer International Publishing, Cham, ISBN 978-3-319-67630-2, https://doi.org/10.1007/978-3-319-67630-2_3, 2017.

Baker, A. H., Pinard, A., and Hammerling, D. M.: On a Structural Similarity Index Approach for Floating-Point Data, IEEE Transactions on Visualization and Computer Graphics, 30, 6261–6274, https://doi.org/10.1109/TVCG.2023.3332843, 2024.

Ballester-Ripoll, R., Lindstrom, P., and Pajarola, R.: TTHRESH: Tensor Compression for Multidimensional Visual Data, IEEE Transactions on Visualization and Computer Graphics, 26, 2891–2903, https://doi.org/10.1109/TVCG.2019.2904063, 2020.

Banerjee, T., Choi, J., Lee, J., Gong, Q., Wang, R., Klasky, S., Rangarajan, A., and Ranka, S.: An Algorithmic and Software Pipeline for Very Large Scale Scientific Data Compression with Error Guarantees, in: 2022 IEEE 29th International Conference on High Performance Computing, Data, and Analytics (HiPC), pp. 226–235, https://doi.org/10.1109/HiPC56025.2022.00039, 2022.

Cappello, F., Underwood, R., Alexeev, Y., Baker, A., Bozdağ, E., Burtscher, M., Chard, K., Di, S., Felker, K. G., O'Grady, P. C., Guo, H., Huang, Y., Jiang, P., Jin, S., Johansson, P., Li, S., Liang, X., Lindahl, E., Lindstrom, P., Lukić, Z., Lundborg, M., Lykov, D., Nagaso, M., Sato, K., Singh, A., Son, S. W., Song, S., Tang, W., Tao, D., Tian, J., Yoshii, K., and Zhao, K.: What to Support When You're Compressing: The State of Practice Gaps and Opportunities for Scientific Data Compression, in: Proceedings of the International Conference for High

Performance Computing, Networking, Storage and Analysis, SC '25, p. 1966–1979, Association for Computing Machinery, New York, NY, USA, ISBN 9798400714665, https://doi.org/10.1145/3712285.3759856, 2025.

Christopoulos, C. A., Ebrahimi, T., and Skodras, A. N.: JPEG2000: the new still picture compression standard, in: Proceedings of the 2000 ACM Workshops on Multimedia, MULTIMEDIA '00, p. 45–49, Association for Computing Machinery, New York, NY, USA, ISBN 1581133111, https://doi.org/10.1145/357744.357757, 2000.

Collet, Y.: Zstandard Compression and the "application/zstd" Media Type, https://doi.org/10.17487/rfc8878, 2021.

Copernicus Climate Change Service: Monthly and 6-hourly total column water vapour over ocean from 1988 to 2020 derived from satellite observations, https://doi.org/10.24381/cds.92db7fef, accessed: 16th April 2025, 2022.

Copernicus Climate Change Service, Climate Data Store: Fire danger indices historical data from the Copernicus Emergency Management Service, https://doi.org/10.24381/cds.0e89c522, accessed: 24th June 2025, 2019.

Delaunay, X., Courtois, A., and Gouillon, F.: Evaluation of lossless and lossy algorithms for the compression of scientific datasets in NetCDF-4 or HDF5 formatted files, Geoscientific Model Development, 12, 4099–4113, https://doi.org/10.5194/gmd-2018-250, 2019.

Di, S. and Cappello, F.: Fast Error-Bounded Lossy HPC Data Compression with SZ, in: 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 730–739, https://doi.org/10.1109/IPDPS.2016.11, 2016.

Di, S., Liu, J., Zhao, K., Liang, X., Underwood, R., Zhang, Z., Shah, M., Huang, Y., Huang, J., Yu, X., Ren, C., Guo, H., Wilkins, G., Tao, D., Tian, J., Jin, S., Jian, Z., Wang, D., Rahman, M. H., Zhang, B., Song, S., Calhoun, J. C., Li, G., Yoshii, K., Alharthi, K. A., and Cappello, F.: A Survey on Error-Bounded Lossy Compression for Scientific Datasets, arXiv, https://doi.org/10.48550/ARXIV.2404.02840, 2024.

Diffenderfer, J., Fox, A. L., Hittinger, J. A., Sanders, G., and Lindstrom, P. G.: Error Analysis of ZFP Compression for Floating-Point Data, SIAM Journal on Scientific Computing, 41, A1867–A1898, https://doi.org/10.1137/18M1168832, 2019.

Eaton, B., Gregory, J., Drach, B., Taylor, K., Hankin, S., Caron, J., Signell, R., Bentley, P., Rappa, G., Höck, H., Pamment, A., Juckes, M., Raspaud, M., Blower, J., Horne, R., Whiteaker, T., Blodgett, D., Zender, C., Lee, D., Hassell, D., Snow, A. D., Kölling, T., Allured, D., Jelenak, A., Soerensen, A. M., Gaultier, L., Herlédan, S., Manzano, F., Bärring, L., Barker, C., and Bartholomew, S. L.: NetCDF Climate and Forecast (CF) Metadata Conventions, https://doi.org/10.5281/ZENODO.14274886, 2024.

ECMWF: ECMWF's vision for Big Data, AI and cloud computing, https://climate.copernicus.eu/sites/default/files/2019-11/Martin%20Palkovic.pdf, accessed: 23rd December 2025, 2019.

Fallin, A. and Burtscher, M.: Lessons Learned on the Path to Guaranteeing the Error Bound in Lossy Quantizers, arXiv, https://doi.org/10.48550/ARXIV.2407.15037, 2024.

Fallin, A., Gorski, N., Agarwal, T., Wang, B., Gopalakrishnan, G., and Burtscher, M.: Fast Topology-Aware Lossy Data Compression with Full Preservation of Critical Points and Local Order, https://doi.org/10.48550/arXiv.2603.26968, 2026.

Flajolet, P., Éric Fusy, Gandouet, O., and Meunier, F.: HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm, Discrete Mathematics & Theoretical Computer Science, DMTCS Proceedings vol. AH, 2007 Conference on Analysis of Algorithms (AofA 07), 10, https://doi.org/10.46298/dmtcs.3545, 2007.

Fornberg, B.: Generation of finite difference formulas on arbitrarily spaced grids, Mathematics of Computation, 51, 699–706, https://doi.org/10.1090/s0025-5718-1988-0935077-0, 1988.

Fox, A., Diffenderfer, J., Hittinger, J., Sanders, G., and Lindstrom, P.: Stability Analysis of Inline ZFP Compression for Floating-Point Data in Iterative Methods, SIAM Journal on Scientific Computing, 42, A2701–A2730, https://doi.org/10.1137/19M126904X, 2020.

Gorski, N., Liang, X., Guo, H., Yan, L., and Wang, B.: A General Framework for Augmenting Lossy Compressors With Topological Guarantees, IEEE Transactions on Visualization & Computer Graphics, 31, 3693–3705, https://doi.org/10.1109/TVCG.2025.3567054, 2025.

Hadade, I., Klocke, D., Enkovaara, J., Lunttila, T., Rackow, T., Engels, J. F., Frauen, C., Redler, R., Kontkanen, J., Jung, T., Sein, D., Sandu, I., Reuter, B., Wedi, N., Milinski, S., Doblas-Reyes, F., Castrillo, M., Acosta, M., Girona, S., and Manninen, P.: Destination Earth: The Climate Change Adaptation Digital Twin, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '25, p. 99–110, Association for Computing Machinery, New York, NY, USA, ISBN 9798400714665, https://doi.org/10.1145/3712285.3771790, 2025.

Hammerling, D. M., Baker, A. H., Pinard, A., and Lindstrom, P.: A Collaborative Effort to Improve Lossy Compression Methods for Climate Data, in: 2019 IEEE/ACM 5th International Workshop on Data Analysis and Reduction for Big Scientific Data (DRBSD-5), pp. 16–22, https://doi.org/10.1109/DRBSD-549595.2019.00008, 2019.

Han, T., Du, D., Chen, Z., Guo, S., Ouyang, W., and Bai, L.: CRA5 a high-fidelity compressed reanalysis atmospheric dataset for weather and climate research, Scientific Data, https://doi.org/10.1038/s41597-026-07381-2, 2026.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E.: Array programming with NumPy, Nature, 585, 357–362, https://doi.org/10.1038/s41586-020-2649-2, 2020.

Hart, D.: Cheyenne and Beyond: NCAR's Research Computing and Storage Roadmap to 2022, https://www.cisl.ucar.edu/sites/default/files/2021-10/Hart.pdf, accessed: 14th July 2026, 2017.

Hassell, D. and Cimadevilla Alvarez, E.: cmip7repack: Repack CMIP7 netCDF-4 datasets, https://doi.org/10.5281/zenodo.18849756, 2026.

Hersbach, H., Bell, B., Berrisford, P., Biavati, G., Horányi, A., Muñoz Sabater, J., Nicolas, J., Peubey, C., Radu, R., Rozum, I., Schepers, D., Simmons, A., Soci, C., Dee, D., and Thépaut, J.-N.: ERA5 hourly data on pressure levels from 1940 to present, https://doi.org/10.24381/cds.bd0915c6, accessed: 8th August 2025, 2018a.

Hersbach, H., Bell, B., Berrisford, P., Biavati, G., Horányi, A., Muñoz Sabater, J., Nicolas, J., Peubey, C., Radu, R., Rozum, I., Schepers, D., Simmons, A., Soci, C., Dee, D., and Thépaut, J.-N.: ERA5 hourly data on single levels from 1940 to present, https://doi.org/10.24381/cds.adbb2d47, accessed: 8th August 2025, 2018b.

Hersbach, H., Bell, B., Berrisford, P., Hirahara, S., Horányi, A., Muñoz-Sabater, J., Nicolas, J., Peubey, C., Radu, R., Schepers, D., Simmons, A., Soci, C., Abdalla, S., Abellan, X., Balsamo, G., Bechtold, P., Biavati, G., Bidlot, J., Bonavita, M., De Chiara, G., Dahlgren, P., Dee, D., Diamantakis, M., Dragani, R., Flemming, J., Forbes, R., Fuentes, M., Geer, A., Haimberger, L., Healy, S., Hogan, R. J., Hólm, E., Janisková, M., Keeley, S., Laloyaux, P., Lopez, P., Lupu, C., Radnoti, G., de Rosnay, P., Rozum, I., Vamborg, F., Villaume, S., and Thépaut, J.-N.: The ERA5 global reanalysis, Quarterly Journal of the Royal Meteorological Society, 146, 1999–2049, https://doi.org/https://doi.org/10.1002/qj.3803, 2020.

Hoyer, S. and Hamman, J.: xarray: N-D labeled arrays and datasets in Python, Journal of Open Research Software, 5, https://doi.org/10.5334/jors.148, 2017.

Huang, H., Castruccio, S., Baker, A. H., and Genton, M. G.: Rejoinder on 'Saving Storage in Climate Ensembles: A Model-Based Stochastic Approach', Journal of Agricultural, Biological and Environmental Statistics, 28, 370–374, https://doi.org/10.1007/s13253-023-00542-5, 2023.

Huang, L. and Hoefler, T.: Compressing Multidimensional Weather and Climate Data into Neural Networks, https://doi.org/10.48550/arXiv.2210.12538, 2023.

Huang, L., Fusco, L., Scheidl, F., Zibell, J., Sprenger, M. A., Schemm, S., and Hoefler, T.: Error bounded compression for weather and climate applications, arXiv, https://doi.org/10.48550/arXiv.2510.22265, 2025.

IEEE: IEEE Standard for Floating-Point Arithmetic, IEEE Std 754-2008, pp. 1–70, https://doi.org/10.1109/IEEESTD.2008.4610935, 2008.

Intergovernmental Panel on Climate Change (IPCC): Summary for Policymakers, p. 3–32, Cambridge University Press, https://doi.org/10.1017/9781009157896, 2023.

Jian, Z., Di, S., Liu, J., Zhao, K., Liang, X., Xu, H., Underwood, R., Wu, S., Huang, J., Chen, Z., and Cappello, F.: CliZ: Optimizing Lossy Compression for Climate Datasets with Adaptive Fine-tuned Data Prediction, in: 2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 417–429, https://doi.org/10.1109/IPDPS57955.2024.00044, 2024.

Jiao, P., Di, S., Guo, H., Zhao, K., Tian, J., Tao, D., Liang, X., and Cappello, F.: Toward Quantity-of-Interest Preserving Lossy Compression for Scientific Data, Proc. VLDB Endow., 16, 697–710, https://doi.org/10.14778/3574245.3574255, 2022.

Klocke, D., Frauen, C., Engels, J. F., Alexeev, D., Redler, R., Schnur, R., Haak, H., Kornblueh, L., Brüggemann, N., Chegini, F., Römmer, M., Hoffmann, L., Griessbach, S., Bode, M., Coles, J., Gila, M., Sawyer, W., Calotoiu, A., Budanaz, Y., Mazumder, P., Copik, M., Weber, B., Herten, A., Bockelmann, H., Hoefler, T., Hohenegger, C., and Stevens, B.: Computing the Full Earth System at 1km Resolution, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '25, p. 125–136, Association for Computing Machinery, New York, NY, USA, ISBN 9798400714665, https://doi.org/10.1145/3712285.3771789, 2025.

Klöwer, M., Razinger, M., Dominguez, J. J., Düben, P. D., and Palmer, T. N.: Compressing atmospheric data into its real information content, Nature Computational Science, 1, 713–724, https://doi.org/10.1038/s43588-021-00156-2, 2021.

Kouznetsov, R.: A note on precision-preserving compression of scientific data, Geoscientific Model Development, 14, 377–389, https://doi.org/10.5194/gmd-14-377-2021, 2021.

Kunkel, J. M., Kuhn, M., and Ludwig, T.: Exascale storage systems: an analytical study of expenses, Supercomputing Frontiers and Innovations, 1, 116–134, https://doi.org/10.14529/jsfi140106, 2014.

Lawrence Livermore National Laboratory: zfp and Derivatives, https://computing.llnl.gov/projects/zfp/zfp-and-derivatives, accessed: 16th June 2026, 2024.

Li, S.: H5Z-SPERR [GitHub Repository], https://github.com/NCAR/H5Z-SPERR, accessed: 16th June 2026, 2026.

Li, S., Lindstrom, P., and Clyne, J.: Lossy Scientific Data Compression With SPERR, in: 2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 1007–1017, https://doi.org/10.1109/IPDPS54959.2023.00104, 2023.

Li, Y., Xia, M., Liang, X., Wang, B., Underwood, R., Di, S., Sharma, H., Beniwal, D., Cappello, F., and Guo, H.: pMSz: A Distributed Parallel Algorithm for Correcting Extrema and Morse Smale Segmentations in Lossy Compression, https://doi.org/10.48550/arXiv.2601.01787, 2026.

Liang, X., Di, S., Tao, D., Chen, Z., and Cappello, F.: An Efficient Transformation Scheme for Lossy Data Compression with Point-Wise Relative Error Bound, in: 2018 IEEE International Conference on Cluster Computing (CLUSTER), pp. 179–189, https://doi.org/10.1109/CLUSTER.2018.00036, 2018.

Liang, X., Whitney, B., Chen, J., Wan, L., Liu, Q., Tao, D., Kress, J., Pugmire, D., Wolf, M., Podhorszki, N., and Klasky, S.: MGARD+: Optimizing Multilevel Methods for Error-Bounded Scientific Data Reduction, IEEE Transactions on Computers, 71, 1522–1536, https://doi.org/10.1109/TC.2021.3092201, 2022.

Liang, X., Zhao, K., Di, S., Li, S., Underwood, R., Gok, A. M., Tian, J., Deng, J., Calhoun, J. C., Tao, D., Chen, Z., and Cappello, F.: SZ3: A Modular Framework for Composing Prediction-Based Error-Bounded Lossy Compressors, IEEE Transactions on Big Data, 9, 485–498, https://doi.org/10.1109/TBDATA.2022.3201176, 2023.

Lindstrom, P.: Fixed-Rate Compressed Floating-Point Arrays, IEEE Transactions on Visualization and Computer Graphics, 20, 2674–2683, https://doi.org/10.1109/TVCG.2014.2346458, 2014.

Lindstrom, P.: Supporting Special Values in ZFP, Tech. rep., Office of Scientific and Technical Information (OSTI), https://doi.org/10.2172/2998448, 2025.

Lindstrom, P., Hittinger, J., Diffenderfer, J., Fox, A., Osei-Kuffuor, D., and Banks, J.: ZFP: A compressed array representation for numerical computations, The International Journal of High Performance Computing Applications, 39, 104–122, https://doi.org/10.1177/10943420241284023, 2025.

Liu, J., Di, S., Zhao, K., Liang, X., Jin, S., Jian, Z., Huang, J., Wu, S., Chen, Z., and Cappello, F.: High-performance Effective Scientific Error-bounded Lossy Compression with Auto-tuned Multi-component Interpolation, Proc. ACM Manag. Data, 2, https://doi.org/10.1145/3639259, 2024.

Liu, J., Jiao, P., Zhao, K., Liang, X., Di, S., and Cappello, F.: QPET: A Versatile and Portable Quantity-of-Interest-Preservation Framework for Error-Bounded Lossy Compression, Proc. VLDB Endow., 18, 2440–2453, https://doi.org/10.14778/3742728.3742739, 2025.

Magri, V. A. P. and Lindstrom, P.: A General Framework for Progressive Data Compression and Retrieval, IEEE Transactions on Visualization and Computer Graphics, 30, 1358–1368, https://doi.org/10.1109/TVCG.2023.3327186, 2024.

Meurer, A., Reines, A., Gommers, R., Fang, Y.-L. L., Kirkham, J., Barber, M., Hoyer, S., Müller, A., Zha, S., Shanabrook, S., Gacha, S. J., Lezcano-Casado, M., Fan, T. J., Reddy, T., Passos, A., Kwon, H., Oliphant, T., and Standards, C. f. P. D. A.: Python Array API Standard: Toward Array Interoperability in the Scientific Python Ecosystem, Scipy 2023, https://doi.org/10.25080/gerudo-f2bc6f59-001, 2023.

Miles, A., jakirkham, Hamman, J., Orfanos, D. P., Bussonnier, M., Stansby, D., Moore, J., Bennett, D., Augspurger, T., Rzepka, N., Verma, S., Bourbeau, J., Fulton, A., Cherian, D., Abernathey, R., Lee, G., Kristensen, M. R. B., Spitz, H., Patel, Z., Chopra, S., Rocklin, M., Jones, M., AWA, A. B., Nunez-Iglesias, J., Durant, M., de Andrade, E. S., Schut, V., and raphael dussin: zarr-developers/zarr-python: v3.0.0, Zenodo, https://doi.org/10.5281/ZENODO.14622720, 2025.

Mirowski, P., Warde-Farley, D., Rosca, M., Grimes, M. K., Hasson, Y., Kim, H., Rey, M., Osindero, S., Ravuri, S., and Mohamed, S.: Neural Compression of Atmospheric States, https://doi.org/10.48550/arXiv.2407.11666, 2024.

Moran, P. A. P.: Notes on Continuous Stochastic Phenomena, Biometrika, 37, 17, https://doi.org/10.2307/2332142, 1950.

Okuta, R., Unno, Y., Nishino, D., Hido, S., and Loomis, C.: CuPy: A NumPy-Compatible Library for NVIDIA GPU Calculations, in: Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS), http://learningsys.org/nips17/assets/papers/paper_16.pdf, 2017.

Peters-von Gehlen, K., Tyree, J., Faghih-Naini, S., Dueben, P., Squar, J., and Fuchs, A.: Lossy Data Compression Exploration in an Online Laboratory and the Link to HPC Design Decisions, https://doi.org/10.5281/ZENODO.15401734, 2025.

Pinard, A., Baker, A., and Hammerling, D.: Examining Variations in the Optimal Compression Level of Spatiotemporal Datasets Determined Using the Data Structural Similarity Index Measure (DSSIM), Tech. rep., NSF National Center for Atmospheric Research, https://doi.org/10.5065/4Q49-T141, 2021a.

Pinard, A., Baker, A., and Hammerling, D.: A Statistical Approach to Obtaining a Data Structural Similarity Index Cutoff Threshold, Tech. rep., NSF National Center for Atmospheric Research, https://doi.org/10.5065/JC2R-5289, 2021b.

Pinard, A., Baker, A., Banihirwe, A., and Levy, M.: NCAR/ldcpy: v0.20, https://doi.org/10.5281/zenodo.15133604, 2025.

Poppick, A., Nardi, J., Feldman, N., Baker, A. H., Pinard, A., and Hammerling, D. M.: A statistical analysis of lossily compressed climate model data, Computers & Geosciences, 145, 104 599, https://doi.org/10.1016/j.cageo.2020.104599, 2020.

Rackow, T., Pedruzo-Bagazgoitia, X., Becker, T., Milinski, S., Sandu, I., Aguridan, R., Bechtold, P., Beyer, S., Bidlot, J., Boussetta, S., Deconinck, W., Diamantakis, M., Dueben, P., Dutra, E., Forbes, R., Ghosh, R., Goessling, H. F., Hadade, I., Hegewald, J., Jung, T., Keeley, S., Kluft, L., Koldunov, N., Koldunov, A., Kölling, T., Kousal, J., Kühnlein, C., Maciel, P., Mogensen, K., Quintino, T., Polichtchouk, I.,

Reuter, B., Sármány, D., Scholz, P., Sidorenko, D., Streffing, J., Sützl, B., Takasuka, D., Tietsche, S., Valentini, M., Vannière, B., Wedi, N., Zampieri, L., and Ziemen, F.: Multi-year simulations at kilometre scale with the Integrated Forecasting System coupled to FESOM2.5 and NEMOv3.4, Geoscientific Model Development, 18, 33–69, https://doi.org/10.5194/gmd-18-33-2025, 2025.

Reich, G. and Heathcote, J.: Analysis of Generation Loss for Wavelet-Based Compression, SMPTE Motion Imaging Journal, 131, 34–42, https://doi.org/10.5594/JMI.2022.3173625, 2022.

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

Ren, C., Underwood, R., Di, S., Kutay, E., Lukic, Z., Yener, A., Cappello, F., and Guo, H.: FFCz: Fast Fourier Correction for Spectrum-Preserving Lossy Compression of Scientific Data, https://doi.org/10.48550/arXiv.2601.01596, 2026.

Santoalla, D. V. and Najm, S.: ecCodes, https://www.ecmwf.int/en/elibrary/80526-eccodes, 2017.

Sather, H., Pinard, A., Baker, A. H., and Hammerling, D. M.: What can real information content tell us about compressing climate model data?, in: 2022 IEEE/ACM 8th International Workshop on Data Analysis and Reduction for Big Scientific Data (DRBSD), pp. 29–36, https://doi.org/10.1109/DRBSD56682.2022.00009, 2022.

Segura, H., Pedruzo-Bagazgoitia, X., Weiss, P., Müller, S. K., Rackow, T., Lee, J., Dolores-Tesillos, E., Benedict, I., Aengenheyster, M., Aguridan, R., Arduini, G., Baker, A. J., Bao, J., Bastin, S., Baulenas, E., Becker, T., Beyer, S., Bockelmann, H., Brüggemann, N., Brunner, L., Cheedela, S. K., Das, S., Denissen, J., Dragaud, I., Dziekan, P., Ekblom, M., Engels, J. F., Esch, M., Forbes, R., Frauen, C., Freischem, L., García-Maroto, D., Geier, P., Gierz, P., González-Cervera, A., Grayson, K., Griffith, M., Gutjahr, O., Haak, H., Hadade, I., Haslehner, K., ul Hasson, S., Hegewald, J., Kluft, L., Koldunov, A., Koldunov, N., Kölling, T., Koseki, S., Kosukhin, S., Kousal, J., Kuma, P., Kumar, A. U., Li, R., Maury, N., Meindl, M., Milinski, S., Mogensen, K., Niraula, B., Nowak, J., Praturi, D. S., Proske, U., Putrasahan, D., Redler, R., Santuy, D., Sármány, D., Schnur, R., Scholz, P., Sidorenko, D., Spät, D., Sützl, B., Takasuka, D., Tompkins, A., Uribe, A., Valentini, M., Veerman, M., Voigt, A., Warnau, S., Wachsmann, F., Wacławczyk, M., Wedi, N., Wieners, K.-H., Wille, J., Winkler, M., Wu, Y., Ziemen, F., Zimmermann, J., Bender, F. A.-M., Bojovic, D., Bony, S., Bordoni, S., Brehmer, P., Dengler, M., Dutra, E., Faye, S., Fischer, E., van Heerwaarden, C., Hohenegger, C., Järvinen, H., Jochum, M., Jung, T., Jungclaus, J. H., Keenlyside, N. S., Klocke, D., Konow, H., Klose, M., Malinowski, S., Martius, O., Mauritsen, T., Mellado, J. P., Mieslinger, T., Mohino, E., Pawłowska, H., Peters-von Gehlen, K., Sarré, A., Sobhani, P., Stier, P., Tuppi, L., Vidale, P. L., Sandu, I., and Stevens, B.: nextGEMS: entering the era of kilometer-scale Earth system modeling, Geoscientific Model Development, 18, 7735–7761, https://doi.org/10.5194/gmd-18-7735-2025, 2025.

Seitola, T., Mikkola, V., Silen, J., and Järvinen, H.: Random projections in reducing the dimensionality of climate simulation data, Tellus A: Dynamic Meteorology and Oceanography, 66, 25 274, https://doi.org/10.3402/tellusa.v66.25274, 2014.

Soler, M., Plainchault, M., Conche, B., and Tierny, J.: Topologically Controlled Lossy Compression, in: 2018 IEEE Pacific Visualization Symposium (PacificVis), pp. 46–55, https://doi.org/10.1109/PacificVis.2018.00015, 2018.

Stevens, B., Satoh, M., Auger, L., Biercamp, J., Bretherton, C. S., Chen, X., Düben, P., Judt, F., Khairoutdinov, M., Klocke, D., Kodama, C., Kornblueh, L., Lin, S.-J., Neumann, P., Putman, W. M., Röber, N., Shibuya, R., Vanniere, B., Vidale, P. L., Wedi, N., and Zhou, L.: DYAMOND: the DYnamics of the Atmospheric general circulation Modeled On Non-hydrostatic Domains, Progress in Earth and Planetary Science, 6, https://doi.org/10.1186/s40645-019-0304-z, 2019.

Tao, D., Di, S., Chen, Z., and Cappello, F.: 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), pp. 1129–1139, https://doi.org/10.1109/IPDPS.2017.115, 2017.

The HDF Group: Hierarchical Data Format, version 5, https://doi.org/10.5281/zenodo.17808614, 2025.

Tintó Prims, O., Redl, R., Rautenhaus, M., Selz, T., Matsunobu, T., Modali, K. R., and Craig, G.: The effect of lossy compression of numerical weather prediction data on data analysis: a case study using enstools-compression 2023.11, Geoscientific Model Development, 17, 8909–8925, https://doi.org/10.5194/gmd-17-8909-2024, 2024.

Tyree, J.: #91: Fix NaN handling in the linear quantizer [GitHub Pull Request], https://github.com/szcompressor/SZ3/pull/91, accessed: 16th June 2026, 2025.

Tyree, J.: numcodecs-wasm, GitHub repository, https://github.com/juntyr/numcodecs-rs/tree/main/py/numcodecs-wasm, 2026.

Tyree, J., Underwood, R., Bouvier, C., Köhler, D., Reichelt, T., Dueben, P., Faghih-Naini, S., Järvinen, H., and Klöwer, M.: Safe, Trusted, and Fearless lossy compression with Compression Safeguards [ReadTheDocs], https://compression-safeguards.readthedocs.io/, accessed: 16th June 2026, 2026a.

Tyree, J., Underwood, R., Bouvier, C., Köhler, D., Reichelt, T., Dueben, P., Faghih-Naini, S., Järvinen, H., and Klöwer, M.: Safe, Trusted, and Fearless lossy compression with Compression Safeguards – How to safeguard ...? [ReadTheDocs], https://compression-safeguards.readthedocs.io/en/1.0.0rc1.post1/#how-to-safeguard, accessed: 29th October 2026, 2026b.

Tyree, J., Underwood, R., Bouvier, C., Köhler, D., Reichelt, T., Dueben, P., Faghih-Naini, S., Järvinen, H., and Klöwer, M.: compression-safeguards [GitHub Repository], https://github.com/juntyr/compression-safeguards, accessed: 16th June 2026, 2026c.

Tyree, J., Underwood, R., Bouvier, C., Köhler, D., Reichelt, T., Dueben, P., Faghih-Naini, S., Järvinen, H., and Klöwer, M.: compression-safeguards /examples/data [GitHub Repository], https://github.com/juntyr/compression-safeguards/tree/1.0.0rc1.post1/examples/data, accessed: 29th October 2026, 2026d.

Tyree, J., Underwood, R., Bouvier, C., Köhler, D., Reichelt, T., Dueben, P., Faghih-Naini, S., Järvinen, H., and Klöwer, M.: compression-safeguards /examples [GitHub Repository], https://github.com/juntyr/compression-safeguards/tree/1.0.0rc1.post1/examples, accessed: 29th October 2026, 2026e.

Tyree, J., Underwood, R., Bouvier, C., Köhler, D., Reichelt, T., Dueben, P., Faghih-Naini, S., Järvinen, H., and Klöwer, M.: compression-safeguards [PyPi], https://pypi.org/project/compression-safeguards/, accessed: 16th June 2026, 2026f.

Tyree, J., Underwood, R., Bouvier, C., Köhler, D., Reichelt, T., Dueben, P., Faghih-Naini, S., Järvinen, H., and Klöwer, M.: compression-safeguards, https://doi.org/10.5281/zenodo.23776642, 2026g.

Tyree, J., Underwood, R., Bouvier, C., Köhler, D., Reichelt, T., Dueben, P., Faghih-Naini, S., Järvinen, H., and Klöwer, M.: numcodecs-safeguards [PyPi], https://pypi.org/project/numcodecs-safeguards/, accessed: 16th June 2026, 2026h.

Tyree, J., Underwood, R., Bouvier, C., Köhler, D., Reichelt, T., Dueben, P., Faghih-Naini, S., Järvinen, H., and Klöwer, M.: xarray-safeguards [PyPi], https://pypi.org/project/xarray-safeguards/, accessed: 16th June 2026, 2026i.

Underwood, R., Di, S., Calhoun, J. C., and Cappello, F.: FRaZ: A Generic High-Fidelity Fixed-Ratio Lossy Compression Framework for Scientific Floating-point Data, in: 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 567–577, https://doi.org/10.1109/IPDPS47924.2020.00065, 2020.

Underwood, R., Malvoso, V., Calhoun, J. C., Di, S., and Cappello, F.: 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), pp. 1–10, IEEE, https://doi.org/10.1109/DRBSD754563.2021.00005, 2021.

Underwood, R., Calhoun, J. C., Di, S., Apon, A., and Cappello, F.: OptZConfig: Efficient Parallel Optimization of Lossy Compression Configuration, IEEE Transactions on Parallel and Distributed Systems, 33, 3505–3519, https://doi.org/10.1109/TPDS.2022.3154096, 2022.

Yan, L., Liang, X., Guo, H., and Wang, B.: TopoSZ: Preserving Topology in Error-Bounded Lossy Compression, IEEE Transactions on Visualization and Computer Graphics, 30, 1302–1312, https://doi.org/10.1109/TVCG.2023.3326920, 2024.

Zarr developers: numcodecs, GitHub repository, https://github.com/zarr-developers/numcodecs, 2025.

Zhao, K., Di, S., Lian, X., Li, S., Tao, D., Bessac, J., Chen, Z., and Cappello, F.: SDRBench: Scientific Data Reduction Benchmark for Lossy Compressors, in: 2020 IEEE International Conference on Big Data (Big Data), pp. 2716–2724, https://doi.org/10.1109/BigData50022.2020.9378449, 2020.

Zhao, K., Di, S., Dmitriev, M., Tonellot, T.-L. D., Chen, Z., and Cappello, F.: Optimizing Error-Bounded Lossy Compression for Scientific Data by Dynamic Spline Interpolation, in: 2021 IEEE 37th International Conference on Data Engineering (ICDE), pp. 1643–1654, https://doi.org/10.1109/ICDE51399.2021.00145, 2021.