

Benchmarking Compression Algorithms for Scientific Data Arrays

Jeremy Magland, Center for Computational Mathematics, Flatiron Institute

January 2025

Early draft, work in progress.

Abstract

Benchcompress is a benchmarking framework designed to evaluate the performance of various compression algorithms on scientific data arrays, with a special focus on electrophysiology traces. The framework automates the benchmarking process, measuring compression ratio, encoding throughput, and decoding throughput for each algorithm-dataset pair. Results are verified through decompression and comparison with the original data, ensuring accuracy. The benchmark results are stored and visualized through an interactive web interface, allowing users to filter, sort, and explore the data. This paper presents the design, implementation, and preliminary results of *Benchcompress*, highlighting its utility in identifying optimal compression techniques for scientific datasets.

Introduction

Scientific research often generates large volumes of data that must be stored, shared, and analyzed efficiently. In fields ranging from electrophysiology to climate modeling, these datasets can be enormous, making data compression essential for reducing file sizes, accelerating data transfer, and facilitating both short- and long-term storage. While the performance of standard compression strategies for electrophysiology data has been studied [buccino2023compression], the specialized nature of numeric data arrays poses unique challenges that many general-purpose compression algorithms, often optimized for text, images, or video, do not address effectively.

Researchers are typically wary of introducing lossy compression to raw data collected from laboratory devices. Rather than experimenting with a wide range of lossy methods and analyzing their impact on scientific results, we adopt a more transparent strategy for lossy compression: apply well-defined, carefully documented transformations, such as quantization, filtering, or normalization, and then apply strictly lossless compression. For instance, floating-point data often contains bits beyond the precision actually needed, so it frequently compresses poorly unless a preliminary quantization step is used. By estimating the measurement resolution of the acquisition process, we can convert floating-point values to a suitably scaled integer representation, effectively removing these superfluous bits. This approach typically imposes negligible loss in subsequent analyses while significantly improving compressibility. Similarly, filtering (to focus on a specific band of frequencies) and normalization may further enhance compression while remaining straightforward to analyze.

Unlike these preprocessing transformations, some techniques are strictly reversible and should therefore be considered part of the lossless method, rather than part of the preprocessing. For example, delta encoding significantly reduces sample magnitudes in data drawn from continuous signals, improving compression ratios in most lossless compressors. Delta encoding plus a lossless method should then be viewed as a compound lossless technique. For datasets exhibiting even smoother behavior, a more powerful variant, which we will call linear Markov prediction, extends delta encoding to higher-order autoregressive models, often resulting in superior compression.

When assessing compression performance, it is important to consider not only final compression ratios but also how quickly data can be encoded and decoded. Unlike typical web delivery scenarios where data

are compressed once and then decompressed repeatedly, scientific workflows may require efficient encoding because data are often compressed at the point of acquisition and then decompressed only once prior to archival. Moreover, scientific requirements vary widely, from long-term preservation to on-demand, cloud-based visualization. In some cases, significant data loss may be acceptable for a high-level overview, whereas in others, minimal alteration of raw measurements is paramount.

To help researchers navigate these questions, we introduce Benchcompress, a benchmarking framework designed to systematically evaluate and compare compression algorithms for numeric data arrays. By automating the testing and providing tools for analysis, Benchcompress simplifies the process of identifying which methods work best for particular datasets. In the sections that follow, we describe the system in detail, outline both standard and specialized compression strategies, and present preliminary results from our benchmarks.

Traditional Compression Methods and Scientific Data

Traditional compression algorithms like LZW (Lempel-Ziv-Welch) [w Welch1984technique], LZ77 [ziv1977universal], BWT (Burrows-Wheeler Transform) [burrows1994block], and Huffman coding [huffman1952method] were originally designed for text data where patterns manifest as repeated sequences of characters or words. While these methods can compress numeric data from scientific instruments, they may not be optimal for this use case. Text compression excels at identifying exact matches of recurring patterns, whereas scientific measurements often exhibit more complex relationships between values.

For example, in text compression, finding repeated instances of common words or phrases leads to efficient encoding. In contrast, scientific data arrays often contain continuous variations where measurement noise and uncertainties mean exact repetition is rare. Even when numeric values are similar, their byte-level representations may share some common bits while differing in others. This characteristic of scientific data suggests that methods explicitly accounting for numerical relationships between values may achieve better compression ratios than general-purpose algorithms.

Independent identically distributed samples

Moving from text to numeric data, we begin with the fundamental case of independently and identically distributed (i.i.d.) discrete data. For such data, where each sample is drawn from a discrete probability distribution (e.g., Bernoulli sampling or quantized Gaussian noise), the theoretical compressed size in bits per sample is determined by Shannon’s entropy formula [shannon1948mathematical]:

$$H(X) = - \sum_i p(x_i) \log_2 p(x_i).$$

Here, $p(x_i)$ represents the probability of occurrence of the i -th symbol x_i in the discrete distribution. So for example, if we have a Bernoulli distribution with $p = 0.5$, the entropy is

$$H_{\text{Bernoulli, } p=0.5} = -0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$$

bits per sample. This means that the optimal compression ratio for such a dataset is 8, assuming the samples are stored as 8-bit integers. On the other hand, if $p \neq 0.5$, the entropy becomes lower and we can achieve compression at a rate of less than 1 bit per sample (e.g., for $p = 0.1$, the entropy is around 0.47 bits per sample, so the compression ratio would be around 17).

Asymmetric numeral systems

In practice, achieving this theoretical compression ratio requires sophisticated encoding techniques. Arithmetic coding [witten1987arithmetic] is one such method, but it is challenging to implement and can be computationally inefficient. A more modern and efficient alternative is asymmetric numeral systems (ANS) [duda2015use], which closely approaches the theoretical limit and is incorporated into state-of-the-art

compressors such as ZStandard [collet2018zstandard]. However, these large, general-use packages are primarily optimized for structured data types, such as text, rather than for numeric scientific data. In our benchmarks, we evaluate ANS using a simple, no-frills, implementation using a Python package we developed for this purpose called `simple_ans`. As anticipated, we show that ANS demonstrates superior performance when compressing i.i.d. samples from a discrete distribution.

Delta encoding

The efficiency of pure entropy coders (such as ANS) diminishes when handling more structured data, such as continuous signals (e.g., voltage traces in electrophysiology). Applying delta encoding partially mitigates this limitation by leveraging the continuity properties of the data through differencing. This reversible preprocessing step enhances ANS performance because the deltas are typically smaller than the original samples, leading to lower entropy when assuming independence between samples.

Linear Predictive Coding

Even with delta encoding for real or realistic datasets, ANS still can fall short of the compression achieved by methods like ZStandard. To improve its performance by further exploiting temporal correlations in the data, we consider a generalization of delta encoding which we call linear Markov predictive modeling. The Markov prediction scheme employs a linear autoregressive model where each sample is predicted as a linear combination of M previous samples. For a given integer time series $x[t]$, the prediction $\hat{x}[t]$ is computed as:

$$\hat{x}[t] = \text{round} \left(\sum_{i=1}^M c_i x[t-i] + b \right)$$

where the coefficients c_i and bias term b are determined through least squares regression on a subset of the data. The rounding operation ensures integer predictions. Rather than compressing the original signal directly, the algorithm compresses the integer residual error sequence $r[t] = x[t] - \hat{x}[t]$. This approach proves effective because the residuals typically have smaller magnitude compared with the original signal or with the deltas. The compression process stores three components: the floating-point model coefficients, a small set of initial integer values required to begin prediction, and the compressed integer residual sequence. During reconstruction, the original signal is recovered by computing predictions and adding back the residuals:

$$x[t] = \text{round} \left(\sum_{i=1}^M c_i x[t-i] + b \right) + r[t]$$

This predictive preprocessing step improves compression performance compared to applying ANS (or other algorithms) directly to either the raw signal or delta-encoded data.

Compression Algorithms

The benchmark suite evaluates a diverse set of compression algorithms, ranging from established general-purpose methods to specialized techniques.

Zstandard (zstd) is a fast real-time compression algorithm developed by Facebook that provides a broad spectrum of compression levels, from fast compression (level 4) to maximum compression (level 22).

LZMA (Lempel-Ziv-Markov chain Algorithm) employs a dictionary compression scheme similar to LZ77 but with sophisticated modeling of repeating patterns using Markov chains. It focuses on achieving high compression ratios through larger dictionary sizes and complex modeling, generally resulting in slower processing but smaller file sizes.

LZ4, designed for speed, is particularly well-suited for real-time compression scenarios. It belongs to the LZ77 family of byte-oriented compression schemes and offers various compression levels, from its fastest mode (level 0) to maximum compression (level 16).

Zlib, implementing the DEFLATE algorithm, combines LZ77 and Huffman coding to provide a balance between compression ratio and speed across its different compression levels (1-9).

Bzip2 employs the Burrows-Wheeler transform along with Huffman coding, typically achieving better compression than traditional LZ77-based algorithms but at reduced speed.

Brotli, developed by Google, combines LZ77, Huffman coding, and context modeling to provide effective general-purpose compression across multiple compression levels.

We also include a custom implementation of ANS, as described above. The implementation uses the `simple_ans` Python package developed specifically for this project.

To enhance the performance of these basic compression algorithms, we also test reversible preprocessing steps before compression. These include, where appropriate, delta encoding, linear Markov prediction, and zero run length encoding for sparse data.

Datasets

Our benchmark suite combines synthetic test cases and real scientific measurements to evaluate compression performance across diverse scenarios. The synthetic datasets include Bernoulli sequences with varying probabilities ($p=0.1$ to 0.5) providing well-defined theoretical entropy bounds, and Gaussian-distributed data in both quantized integer and floating-point formats with different standard deviations ($\sigma = 1$ to 8).

For real-world data, we draw from multiple sources in neuroscience and geophysics. Extracellular electrophysiology samples from the DANDI Archive [`@dandi_archive`] are provided in three forms: raw 30 kHz recordings, bandpass filtered (300-6000 Hz) and normalized traces, and sparse versions using activity-based suppression. We also include intracranial EEG data from OpenNeuro dataset ds005592 [`@markiewicz2021openneuro`], consisting of concatenated recordings from ten channels as 32-bit floating-point voltage measurements.

The marine seismic dataset, collected during the Roger Revelle voyage RR1508 [`@gorman_2023_8152964`], provides both original floating-point measurements and quantized integer versions from a gas hydrate system survey.

Finally, we include a sample of functional MRI data [`@ds005880`] consisting of 16-bit integer BOLD signal time series, representing neural activity patterns as 3D brain volumes over time.

System Architecture

The framework is implemented primarily in Python for the core benchmarking functionality and TypeScript/React for the web interface. Performance-critical components, such as the Markov prediction algorithm, are implemented in C++ with Python bindings.

The system is organized around two primary registries in an open-source repository: compression algorithms and test datasets. Both are defined through Python modules, enabling the scientific community to contribute through GitHub pull requests. Each algorithm specifies encoding and decoding functions along with tags indicating its capabilities, while datasets specify generation parameters and tags describing their characteristics like data type, dimensionality, and properties (continuous, sparse, etc.).

The tag system manages algorithm-dataset compatibility. For example, algorithms with delta encoding or Markov prediction tags require datasets marked as continuous integer time series, while those using zero run-length encoding are matched with sparse datasets. This ensures algorithms are only tested on appropriate dataset types.

The benchmarking process runs automatically through GitHub Actions when changes are pushed to the main branch. For compatible algorithm-dataset pairs, the system measures compression ratio and processing speeds, verifying results through decompression. A cloud-based caching system prevents redundant benchmark runs, only re-evaluating when components change.

Results are presented through an interactive web interface where users can explore dynamically generated visualizations comparing compression ratios and processing speeds across different algorithms and datasets.

The interface supports filtering by dataset characteristics and algorithm properties, enabling focused analysis of specific use cases.

Results

NOTE: There are no figures in this section because the reader is expected to browse the results interactively on the Benchcompress website. But eventually we can include some curated figures here.

We first report on compression ratio performance, and then we will move on to encoding and decoding speeds.

For the synthetic datasets that involve i.i.d. samples, ANS demonstrates superior compression performance compared to other algorithms, achieving compression ratios close to the theoretical entropy bounds. For example, in the Bernoulli sequence with $p = 0.1$ (<https://concept-collection.github.io/benchcompress/dataset/bernoulli-0.1>), ANS achieves a compression ratio of ~ 16.9 whereas the next best algorithms (bzip2, lzma, btrotli-11, zstd-19, and zstd-22) achieve ratios around 13.9. Zlib struggles for this dataset, with a compression ratio of only ~ 12.2 at the highest compression level (9), and much lower values for lower levels. Brotli and Zstandard at the lower compression levels also perform poorly (for example ~ 9.6 for zstd-7). The results show a similar trend for the other Bernoulli datasets with different probabilities, but the spread is less pronounced as the entropy increases. The story is similar for the quantized Gaussian datasets, with ANS outperforming the other algorithms, especially for lower standard deviations (e.g., $\sigma = 3$, <https://concept-collection.github.io/benchcompress/dataset/gaussian-q3>).

Discussion

[Discussion to be added]

Conclusion

[Conclusion to be added]

References