Using Neural Networks for Two Dimensional Scientific Data Compression

Lucas Hayne
University of Colorado Boulder
Boulder, CO

John Clyne
Nat'l Center for Atmospheric Research
Boulder, CO

Shaomeng Li
Nat'l Center for Atmospheric Research
Boulder, CO

***Abstract*—Continual advances in high-performance computing have enabled the development of higher resolution and more realistic simulations of a wide variety of scientific phenomena. As a result, many computational science communities are increasingly constrained by the massive volumes of data produced,for example, strict storage constraints often force reductions in the number of output variables, data output frequency, or simulation length. Accordingly, modelers across many scientific domains are beginning to adopt purpose-built scientific data compression techniques as an effective mitigation for these challenges. The origins of scientific data compression tools every so often lie in image and video compression. Recently, compression researchers have achieved state-of-the-art performance using neural networks for natural image compression, but this achievement has yet to be adapted to scientific data. This paper assesses the performance of an existing autoencoder neural network compression algorithm on two sets of two-dimensional floating-point scientific data. Compared to state-of-the-art scientific data compression algorithms SZ and ZFP, this out-of-the-box neural network achieves higher peak signal-to-noise ratios at low bit rates, and remains competitive in controlling maximum point-wise error. This preliminary assessment paves the way for future research into neural network compression on floating-point scientific data.**

I. Introduction

Modern high performance computing (HPC) simulations produce a staggering amount of data. For example, in 2019, the Computational and Information Systems Laboratory (CISL) at the National Center for Atmospheric Research (NCAR) provided access to more than 5.1 petabytes of observational and model data through its Research Data Archive [1]. Furthermore, the amount of data available to scientists grows at a rapid pace. As another example, a recent climate simulation generated 260 terabytes of data every 16 seconds [2]. The sheer data volume as well as fast pace of data generation puts significant pressure on simulation scientists—they need to consciously make choices on what data to preserve or discard.

One mitigation to this pressure is data compression. Data compression comes in two flavors, *lossless* and *lossy*. Lossless compression allows for storage of the original data without the loss of any information by compactly representing bit/byte patterns. However, lossless methods have limited effects on floating-point scientific data, often times reducing the size of data sets by no more than a factor of two [3], [4]. Alternatively, practitioners employ lossy compression methods. Lossy compression involves compressing data sets to the point where certain information is lost. In the context of scientific data compression, that lost information usually translates to small errors on individual data points. If the impacts of imprecise data points prove inconsequential in subsequent analyses, then lossy methods can be acceptable with the added benefit of significant reductions in data sizes.

Multiple lossy compression algorithms have been developed for scientific data; they generally fall into one of two categories: predictive methods and transform-based methods. Predictive methods attempt to predict individual data points from other points in the data set, and encode only the differences that exceed a certain threshold. These differences tend to be smaller in magnitude and follow patterns, thus easier to compactly represent. Transform-based methods transform the data into a different domain, such as frequency or wavelet, and then encode the transformed values, which also tend to be more easily compressed. In both cases—predictive and transform-based—the aim is to prevent the storage of redundant information.

Despite state-of-the-art performance in the domain of image compression, neural networks, which can be classified as transform-based compressors, have been understudied for scientific data compression. In this study, we will evaluate the viability of neural networks to compress floating-point scientific data. We begin with a brief introduction to lossy compression in general and neural network based compression methods (Section II), and a discussion on challenges of floating-point scientific data compression (Section III). Then, we study the effectiveness of a neural network compression model (Section IV) on two-dimensional scientific data (Section V). Results from this study show remarkable performance of this network model, especially its excellent control of average error. Finally, Section VI discusses the limitations of this neural network and suggests future directions for investigation into neural network based floating-point compressors.

II. Background

A. Lossy Compression

Lossy compression has a long history of use in areas with relatively high tolerance for data noise. Consumer images fit in this use case, and as a result, image compression was standardized very early on in the 1990's with the introduction of JPEG [5], which is still popular 30 years later. Today, newer image compression techniques achieve higher compression ratios while maintaining similar visual qualities, with notable

examples being JPEG2000 [6] which is an upgrade to JPEG; HEIF [7], and WebP [8] which is promoted by Google.

Simulation scientists traditionally have a stringent pursuit of data with the highest quality. However, under the growing pressure of humongous data volumes, lossy compression has started to see slow adoption [9]–[11], including a direct application of JPEG2000 on climate data [12]. Lossy compressors specifically designed for scientific data also emerged in the past decade with SZ [13], [14] and ZFP [15] being the most prominent ones. A noteworthy capability of these compressors is that they can provide a maximum point-wise error guarantee, which helps scientists to perform subsequent analyses with more confidence. Finally, a more comprehensive survey [16] provides a broader landscape of scientific data reduction techniques and their applications.

B. Neural Networks for Lossy Image Compression

When nonlinear artificial neural networks emerged, researchers suspected they could be used to provide better compression quality than traditional methods that use linear transforms. Indeed, in 2015, an early work showed that a neural network trained on a large data set of thumbnail images outperformed JPEG and other transform coders at a range of bit rates [17]. Since then, the field has published a deluge of neural network based image compression algorithms employing a variety of network architectures, including recurrent neural networks [18], convolutional neural networks [19], [20], generative adversarial networks [21], [22], and the very popular autoencoders [23], [24]. All of these networks make incremental improvements over previously published algorithms, largely due to their utilization of nonlinear transformations and end-to-end optimization strategies [19].

C. Neural Networks for Lossy Scientific Data Compression

Scientific data compression using neural networks only emerged in recent years. Liu et al. [25] use a generative adversarial network to compress computational fluid dynamics data. Despite their innovative network architecture, their proposed method does not show significant improvements over the discrete wavelet transform on root-mean-square error.

Chandak et al. [26] investigate the compression of multivariate time series from IoT devices (smartwatches and sensors, etc.). They use a prediction-quantization-entropy model that predicts the next points in a time series and encodes the prediction error. This neural network outperforms SZ and produces the best compression ratio in their study.

Glaws et al. [27] develop, train, and test a custom convolutional autoencoder network to compress data from computational fluid dynamic simulations. Compared to singular value decomposition at a compression level of 64:1, their neural networks show improved mean-squared-error performance. However, their neural network only compresses at a fixed compression level: 64:1, and they do not compare its performance with other lossy compression algorithms.

To the best of our knowledge, no research has yet investigated the characteristics of neural networks applied to the compression of floating-point scientific data in general, and this work is likely the first attempt to do so.

III. Challenges

Though sharing similarities with image compression, scientific data compression faces unique challenges due to the nature of the input data itself, and the intended use scenario, which is quantitative scientific analysis.

Scientific data produced by a numerical simulation is usually represented as 32-bit or 64-bit floating point values, which often exhibit vastly different dynamic ranges from variable to variable (see examples in Section V-B). Images, however, have a fixed dynamic range across the board. Another challenge is the number of discrete values—images have a fixed number of 256 discrete values per channel, allowing for effective use of certain techniques such as quantization. In contrast, scientific data has almost infinite possible discrete values,

Intended to be used for quantitative analyses, lossy scientific data compression has much stricter quality requirements than image compression. First, the tolerance for average error, which is often expressed by peak signal-to-noise ratio, is smaller. Second, because scientists often require a guarantee on the worst-cast scenario, it is considered equally if not more important to reduce the maximum point-wise error that could occur at any data point. Finally, any noise resulting from lossy compression should be highly random and unbiased so no artificial pattern is introduced to interfere with the science. With these challenges in mind, we rigorously test the performance of our neural network based compressor and present our findings in the remainder of this paper.

IV. Neural Network

For our experiments we adapted a neural network introduced by Ballé et al. [23]. We choose this network from a number of recently published works because 1) it shows competitive image compression performance, 2) has an open-sourced codebase, and 3) uses a mean-squared-error loss function instead of an image-centric measure (e.g., SSIM), which is likely less relevant on scientific analysis.

Though this network is designed for compressing natural images, we expect that its compression ability will largely translate to scientific data because both applications can share a set of learned functions. More specifically, modern convolutional neural networks consist of a hierarchy of layers and each layer learns some spatial information in the data. Layers lower in the network hierarchy learn primitive functions such as edges, corners, and gradients over small spatial windows. Layers higher in the hierarchy leverage learned functions from lower layers, and they are capable of learning more complex functions that incorporate features over larger spatial areas. While natural images and scientific data overlap less in terms of large-scale features, they share most small features. With four convolutional layers in the Encoder of this model, we hypothesize that a significant amount of learned functions, especially the ones concerning small features, will perform well in our application of scientific data compression.

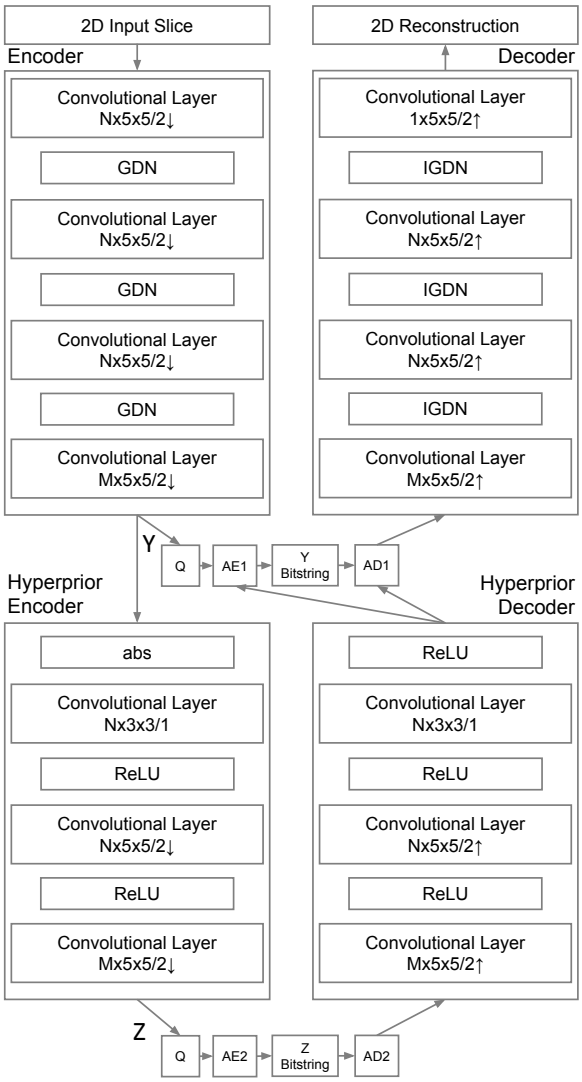


Fig. 1: Our compression neural network architecture. This figure is adapted from work by Ballé et al. [23]. The 2D data slice is input to the network starting at the top left. The *Encoder*, *Hyperprior Encoder*, *Hyperprior Decoder*, and *Decoder* are four major neural network modules; they are accompanied by Quantizers (Q), Arithmetic Encoders (AE1 and AE2), and Arithmetic Decoders (AD1 and AD2) to complete the data compression pipeline.

A. Network Architecture

The neural network in this study employs an autoencoder architecture and a fine-tuned arithmetic encoder (AE1) in its compression pipeline. Autoencoders are a class of neural networks that pass their input through bottleneck layers that have fewer dimensions than the input. The most important features of the input are then "learned" and preserved by the bottleneck layers in a compressed form. The rest of this subsection will explain this pipeline in more detail.

Figure 1 presents this network architecture. Data flows in this network as follows: input data is processed by the *Encoder* which produces a latent representation *Y*. A copy of *Y* is then passed into the *Hyperprior Encoder* which learns a subsequent latent representation (*Z*) that is used to fine tune the first arithmetic encoder (*AE1*). *AE1* then further de-correlates *Y* leveraging knowledge kept in *Z* to generate a highly compact representation: *Y Bitstring*. Because *Z* is required to correctly decode *Y Bitstring*, the final output of the compressor then consists of *Y Bitstring* concatenated with *Z Bitstring*, a compressed version of *Z* produced using a standard arithmetic encoder (AE2).

Model	1	2	3	4	5
λ	0.001	0.01	0.05	0.1	0.5
Model	6	7	8	9	10
λ	1	3	10	100	10000

TABLE I: Neural network models based on ten λ settings.

In practice, autoencoders utilize components from other classes of neural networks as building blocks. In the *Encoder* block of this network, alternating convolutional layers and non-linearities of generalized divisive normalization (GDN) layers are used to create the bottleneck representation and effectively downsample the input to produce Y.

The *Hyperprior Encoder* takes input of *Y* and learns a piece of side information called a *scale hyperprior Z*. *Z* encodes parameters of an entropy model conditioned on *Y*. The motivation for using a *Hyperprior Encoder* is that output of the *Encoder* (*Y*) still contains obvious spatial correlation, and an arithmetic encoder could remove this correlation with a proper entropy model (this removal of correlation is very well illustrated in the original paper [23]). Note that the storage required for this entropy model (*Z*) is negligible compared to the output of the *Encoder* (*Y*).

Finally, we made a modification to the out-of-the-box network so that it receives and reconstructs data in a single channel instead of three (red, green, and blue). This modification does not change the data flow and architecture of this network.

B. Training

Our neural network is trained on natural images from the ImageNet [28] data set which contains 1.28 million images with an average size of 400×350. These images are treated with two conditioning steps before training: first, they are converted to grayscale because we modified the network to accept data of a single channel rather than three. Second, the grayscale values are converted to floating-point values in the range of $[0.0, 1.0]$, which is what the network accepts.

The training process uses a configurable loss function:

$$\mathcal{L} = BPP + \lambda \times MSE, \tag{1}$$

which is a linear combination of the resulting bit per pixel (BPP) and mean-square-error (MSE). Here both low BPP and low MSE are desirable outcomes, but they cannot remain low simultaneously, so a hyperparameter λ is used to balance their relative weight, producing slightly different versions of the

loss function. Essentially, a smaller λ directs the network to use less bits for encoding, resulting in more aggressive compression but higher MSE. Similarly, a larger λ results in more bits per pixel and lower MSE. Note that λ fine tunes the training process to produce slightly different models that achieve different levels of compression; it is not, however, a parameter that the network takes as input when performing compression. At the end, we trained ten different models using ten different settings of λ as listed in Table I.

The ten resulting models compress at ten different levels, but those levels are not convenient for a human user because even for an individual model the BPP and MSE values can still vary based on the input data. Therefore, unlike SZ and ZFP, we cannot establish an error tolerance that we would like the compressor to achieve.

The actual training proceeded in two phases. First, a small model with a bottleneck layer of $M = 256$ and a large model with a bottleneck layer of $M = 512$ were trained using $\lambda = 1.0$. The models were updated using batches of eight randomly selected images for one million steps. Second, we duplicated each model into five separate models: Models 1–5 are each duplicates of the small model from phase one and Models 6–10 are each duplicates of phase one's large model. After duplication, we trained each with a different setting of λ (as shown in Table I) for an additional one million steps using batches of eight randomly selected images. This two phase training scheme reduced the total training hours.

V. Study

A. Methodology

Our study aims to better understand the viability and characteristics of neural networks for scientific data compression. There are two phases of this study: 1) *comparison* and 2) *error bias evaluation*.

In the comparison phase, we compare the neural network against two leading scientific data compressors, *SZ* and *ZFP*, aiming to test if neural networks can achieve similar compression qualities with similar storage budgets. This comparison uses "rate-distortion" curves, which depict the level of distortion occuring at each compression rate. There are two distortion measures used here: peak signal-to-noise ratio (PSNR) and maximum point-wise error. These two measures are both necessary because while PSNR provides an overall trend between compression quality and bit rates, maximum point-wise error captures the worst-case scenario noise that could risk the soundness of subsequent quantitative analyses.

In our experiment, PSNR is calculated using TensorFlow's built-in function which implements the following equation:

$$\text{PSNR}(X, \hat{X}) = 10 \log_{10} \left(\frac{R^2}{\text{MSE}} \right), \tag{2}$$

where X is the original data, $\hat{X}$ is the reconstruction, and R is the maximum fluctuation in the data. MSE can be calculated as $\sum_1^N (X_i - \hat{X}_i)^2 / N$. In most image processing applications, acceptable PSNR values range from 20.0 to 100.0. For maximum point-wise error, we normalize it to the range of $[0.0, 1.0]$ (noted as NPME) so results from different variables are easily comparable:

$$\text{NPME}(X, \hat{X}) = \frac{|\max(X - \hat{X})|}{\max(X) - \min(X)}. \tag{3}$$

The comparison is between rate-distortion curves from the neural network and curves from traditional compressors. Section V-D presents findings of this evaluation.

In the error bias evaluation phase, we plot histograms of error introduced by the neural network, and test whether they follow roughly zero-meaned Gaussian or uniform distributions, which are the two most common forms of random distributions [29]. The findings are presented in Section V-E.

Variable	Minimum	Maximum	Mean	Std. Dev.
Velocity	-5.71e+0	6.88e+0	-1.13e-1	1.51e+0
Enstrophy	3.45e-3	1.91e+6	6.29e+3	1.69e+4

TABLE II: Summary statistics for two variables from the Isotropic Turbulence data sets.

B. Test Data

We use two data sets for our evaluation, and they are chosen for two reasons. Firstly, both data sets are publicly available, which facilitates the reproducibility of our study. Secondly, both data sets are representative of the types of data commonly produced by HPC simulations. These two data sets also have distinct characteristics which make them nice benchmarks for compression in general. We note that each variable from these two data sets are normalized to the range of $[0.0, 1.0]$ to accommodate the neural network.

The first data set concerns a turbulance simulation. It includes two-dimensional slices of velocity and enstrophy from a Forced Isotropic Turbulence Simulation from the Johns Hopkins Turbulence Database (JHTDB) [30]. We selected data set #5 which solves Navier–Stokes equations using pseudo-spectral method on a 4096^3 grid.

The velocity field of this simulation is homogeneous in nature, and we extracted 2D slices from the 3D volumes for our evaluation. More specifically, twenty random two-dimensional slices of size 1024^2 were selected from the 4096^3 three-dimensional snapshot by first selecting a random X coordinate and then extracting a 2D slice centered in the Y and Z directions. All three components of velocity were used, so there are sixty velocity slices in total.

To further investigate whether the neural network based compressor is sensitive to data sizes, as some of the early neural networks achieve much higher compression efficiency on smaller images [17], we also test on the same slices but in smaller subdomains. More specifically, we performed domain decomposition on every 1024^2 slice to generate four subdomains at size 512^2 and sixteen subdomains at size 256^2. These subdomains are compressed individually and then later concatenated together to restore slices of 1024^2 for evaluation purposes. Overall, this process provided us an additional 240 slices at 512^2 and 960 slices at 256^2.

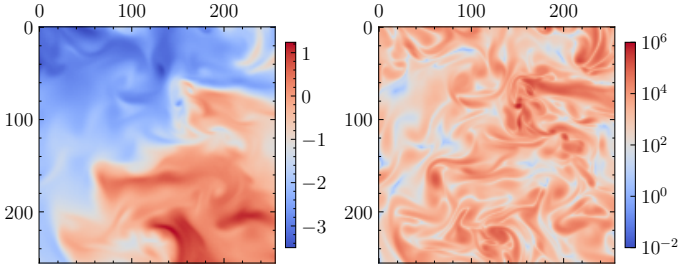


Fig. 2: Example 2D visualization from the Isotropic Turbulence data sets: velocity on the left using a linear scale, and enstrophy on the right using a logarithmic scale.

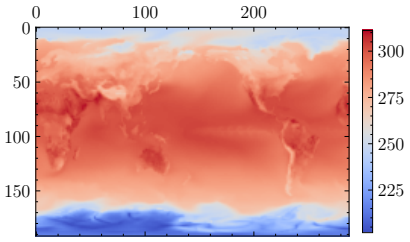


Fig. 3: Example 2D visualization of `TS` (surface temperature), a rather smooth variable, from CESM LENS data set.

Based on velocity, we derive enstrophy for evaluation as well, since enstrophy is a quantity with many small-scale features that often exhibit sudden value changes, and thus is considered hard to compress. Given a velocity field $\boldsymbol{v}$ over a domain D, enstrophy is defined by Equation 4.

$$\mathcal{E}(v) = \int_D |\text{curl}(\boldsymbol{v})|^2 dD. \tag{4}$$

We approximated enstrophy using a sixth-order finite difference method for twenty 2D slices at the same locations where we extracted the 1024^2 velocity slices, and these enstrophy slices are also of size 1024^2. Each enstrophy slice also goes through domain decomposition to produce four 512^2 slices and sixteen 256^2 slices, which left us with twenty 1024^2 slices, eighty 512^2 slices and 320 256^2 slices in total.

Combining velocity and enstrophy, the first data set has eighty independent slices at 1024^2 and their domain decomposed versions at smaller sizes. Table II summarizes basic statistics associated with the selected velocity and enstrophy slices, and Figure 2 provides sample visualizations of both variables. Note that even though velocity is color mapped on a linear scale and enstrophy is on a log scale, the enstrophy plot still shows finer structures and more rapid value changes.

The second data set concerns a climate simulation: it is from the Community Earth System Model (CESM) Large Ensemble (LENS) Community Project [31]. For our experiment we used ten time steps representing ten days from October 1922, all from the first ensemble member. Among available 2D variables, we removed ones that have missing values and ones that are all zeros—variables of this kind are usually treated with special care in a climate analysis and are not of our major concern with regard to compression. At the end of variable selection, we have 51 variables from both the model's dynamical core and adiabatic/chemical processes. With ten time steps for each variable, there are 510 members in this data set, and all members have a resolution of 192×288. As an example, Figure 3 visualizes variable `TS`, surface temperature.

It is worth noting that this CESM-LENS data set is highly diverse with different variables having different characteristics that could greatly impact how "friendly" they respond to compression. To quantify these differences, Table IV in the Appendix provides statistics on each one of these 51 variables. For example, `so4_a1_SRF` and `PRECSC` have a range in the order of 10^{-8}, while `Z050` and `z500` have a range in the order of 10^3, demonstrating the big difference in range. In addition to traditional statistics, we also measure how self-similar a variable is by calculating their auto-correlation scores. Auto-correlation is defined by shifting a variable by one or more grid points (lags) and then computing the correlation between the original and shifted versions of it [29]. An autocorrelation score then ranges between zero and one, with zero meaning no self-similarity at all and one meaning the highest degree of self-similarity. A higher autocorrelation score often means that a variable is easier to compress. Table IV also provides one-lag autocorrelation in two directions (latitude and longitude) of each variable. It shows that most autocorrelation scores range between 0.8 and 1.0, and a few noticable exceptions are `PRECSL` at 0.58, `PRECSC` at 0.18, and `ICEFRAC` having drastically different autocorrelations in two directions (0.28 and 0.93). This difference in autocorrelation scores again demonstrates the diverse nature of this CESM-LENS data set.

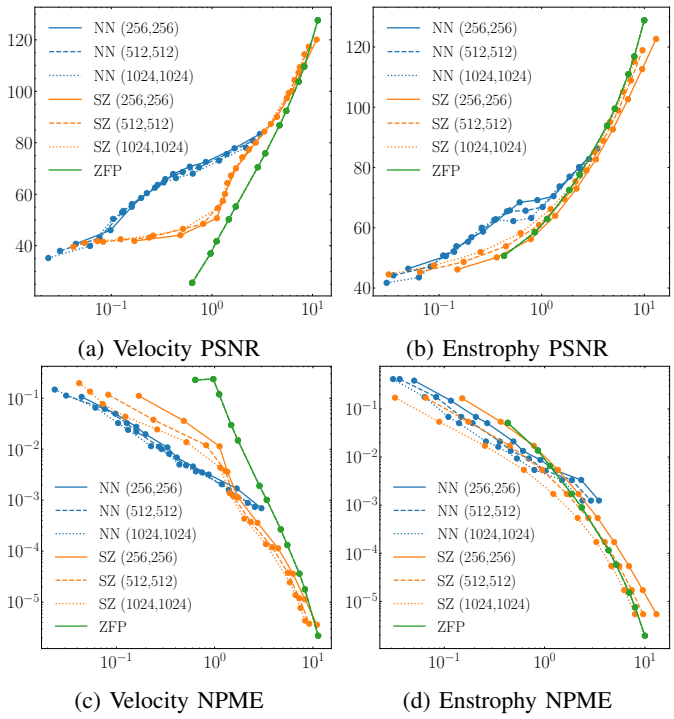
Finally, in our experiment, each variable has multiple samples (twenty in Turbulence data set and ten in CESM-LENS data set). To calculate PSNR and NPME over all samples, we concatenate all samples together and calculate the statistics over the concatenated domain, though each sample is individually compressed and reconstructed.

C. Comparison Compressors

We selected two leading scientific data compressors, *SZ* and *ZFP*, to compare against the neural network algorithm.

SZ was originally designed in 2016 and has gone through a few iterations of improvements [13], [14]. SZ is a predictive method with a selection of prediction schemes, and it features very rich controls of compression qualities expressed in a number of metrics, including PSNR and maximum point-wise error among others. In our experiment, we used its PSNR control to achieve different compression levels. This study uses SZ version `2.1.11.1`.

ZFP was introduced in 2014 specifically for scientific data compression [15]. It partitions any d-dimensional input data into 4^d blocks, and then individually applies an orthogonal transform to decorrelate each block, making it a transform-based method. ZFP also supports multiple controls of its compression quality, and in our experiment, we used the "fixed-accuracy" control which guarantees the maximum point-wise error. As its documentation noted, this mode "gives the highest quality (in terms of absolute error) for a given compression rate" [32]. The software version was `0.5.5` in this study.



(a) Velocity PSNR (b) Enstrophy PSNR

(c) Velocity NPME (d) Enstrophy NPME

Fig. 4: Rate-distortion cuves that compare different compressors and different slice dimensions. In each plot, the X-axis is compression rate measured by bit per pixel, and is plotted on a logarithmic scale. The Y-axis is distortion: Subfigures 4a and 4b use PSNR and Subfigures 4c and 4d use NPME. Note that there is only one curve for ZFP in each plot because ZFP produces the exactly same results in all three slice dimensions.

D. Results—Comparison with SZ and ZFP

1) Comparisons on Turbulence Data Set: Using the Turbulence data set described in Subsection V-B, we plot aggregated rate-distortion curves for velocity and enstrophy variables.

Subfigures 4a and 4b present rate-distortion curves with PSNR, which compile results from multiple compression levels of all three compressors. They show that neural networks often achieve higher PSNR in low bit rate ranges (i.e., less than one bit per pixel), but their lead is less significant in higher rate ranges. Neural network curves also do not extend beyond around three bits per pixel on this data set because the network was trained on greyscale images that do not contain more than eight bits of useful information. So, in our case, the network did not learn functions necessary to compress using higher bit rates. Finally, different slice dimensions show a very slight effect on the compression efficiency of neural networks. This effect is no more than what SZ exhibits, but is more than what ZFP exhibits, which is essentially zero, because ZFP operates on 4^2 blocks anyway in all three slice dimensions.

Subfigures 4c and 4d present rate-distortion curves with NPME; they show different comparison results between two test variables. With enstrophy, three compressors are in a virtual tie. With velocity, neural networks show a clear advantage below one bit per pixel, but this advantage starts to disappear in higher bit rate ranges. With regard to sensitivity to slice dimensions, neural networks exhibit a modest fluctuation where bigger slices tend to yield smaller NPME. This fluctuation is clearly less than what SZ exhibits, but more than what ZFP exhibits, which is essentially zero.

	Velocity		Enstrophy	
Compression Level	**Mean**	**Standard Deviation**	**Mean**	**Standard Deviation**
1	-9.3e-2	1.97e-1	7.05e2	1.56e4
3	-4.6e-2	1.49e-1	-1.59e2	1.17e4
7	-3.1e-2	1.23e-1	-4.32e2	9.59e3
10	-2.3e-2	1.08e-1	-3.28e2	8.31e3

TABLE III: The mean and standard deviation of each error distribution shown in Figure 6. This table is laid out in the same way as Figure 6. Each row represents a different compression level, as indicated in the first column. The compression level numbers are defined in Table I.

2) Comparisons on CESM-LENS Data Set: Using the CESM-LENS data set described in Section V-B, Figure 7 in the Appendix plots rate-distortion curves with PSNR for all 51 variables. These curves vary greatly from one to another, indicating the diverse nature of this data set. Also, no variable or compression level uses more than seven bits per pixel, a consequence of the limit of neural networks trained on images that contain no more than eight bits per pixel of useful information. Among all 51 variables, we choose three to examine more closely, which are `T010` (temperature at 10 mbar pressure surface) representing "easy to compress" variables; `FLUT` (upwelling longwave flux at top of model) representing "hard to compress" variables; and `TS` (radiative surface temperature) in between. These three variables are marked in Figure 7 with dots on their respective curves.

Figure 5 provides detailed comparison results on the three selected variables. When comparing PSNR, all three plots show that neural networks are able to achieve a higher PSNR than SZ and ZFP with the same bit rate, especially when bit rates are low. When comparing NPME, neural networks yield a little lower, higher, and similar NPME scores with `T010`, `TS`, and `FLUT` respectively, but in all cases the amount of difference is small. These findings are consistent with comparisons using the Turbulence data set that neural networks often achieve higher PSNRs while still staying competitive in controlling NPME.

E. Results—Error Bias Evaluation

In phase two of our study, we determine if error introduced by lossy compression exhibit obvious biases. Figure 6 plots error distributions of the two variables from the Turbulence data set, which are representative of outcome from this evaluation. Here errors are calculated by subtracting the reconstructed values from the original values. The four subplots within each variable represent four different compression levels, from more compression (top) to less compression (bottom), and this is reflected in the histograms—the standard deviation shrinks from top to bottom (note the scales on the X axis). Table III provides mean and standard deviation of these distributions.

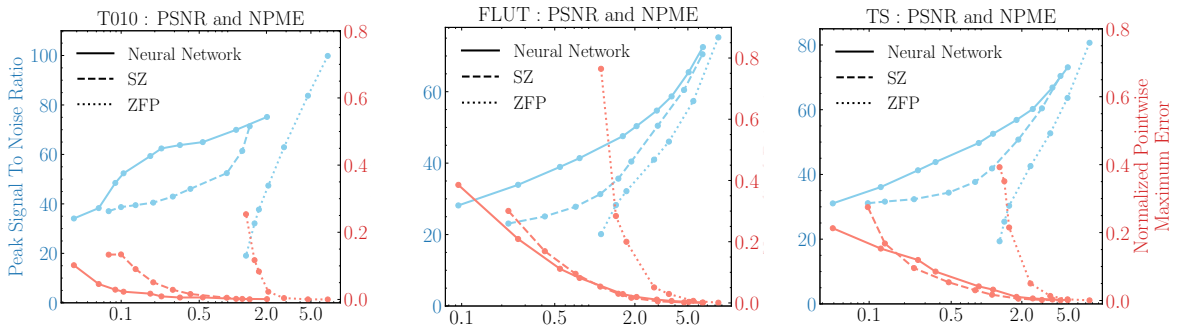


Fig. 5: Rate-distortion curves on three variables from the CESM-LENS data set. In each plot, PSNR curves are plotted against the left axis in blue and NPME curves are plotted against the right axis in red. The X-axis is bit per pixel.

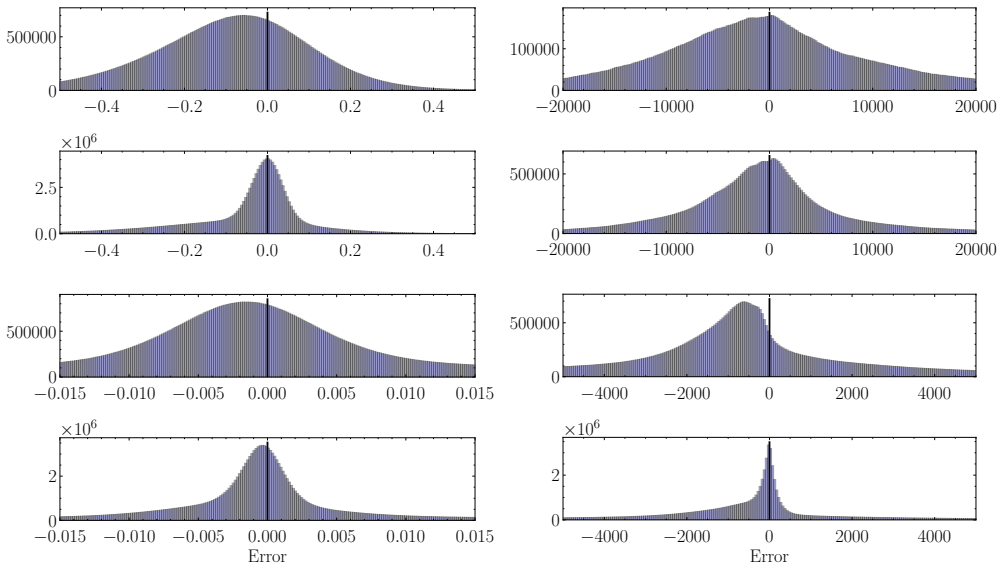


Fig. 6: Histograms showing error distribution for the neural network compressor on variables velocity (left) and enstrophy (right) from the Turbulence data set. Each histogram represents a different compression level: from more compression at the top to less compression at the bottom, or levels 1, 3, 7, and 10 defined in Table I. The vertical black line marks the point on the X-axis where error is zero. Mean and standard deviations of these distributions are presented in Table III.

These histograms show that errors approximately follow Gaussian distributions. However, the left-side tails often appear bigger and the means are often skewed towards the negative side. This indicates that the neural network tends to overestimate reconstructed values. An interesting observation is that for enstrophy, compression level 3 actually achieves the closest-to-zero mean, even though levels 7 and 10 produce less error overall. This hard-to-explain phenomenon might suggest directions of how to reduce the skewness with other compression levels. Overall, our tested neural network shows less-than-satisfactory performance in maintaining bias free, which warrants further investigation and research.

VI. Limitations and Future Work

The current neural network compression model has a few limitations, including the ones discussed in previous sections: non-intuitive compression quality controls, limited bit rate ranges, and skewed error distributions. This section discusses two other limitations and potential future work.

The first limitation is with regard to compression artifacts. During the course of this study we discovered that the neural networks sometimes inject obvious compression artifacts at the edges of some variables with small or uneven dimensions. Despite looking obvious when plotted, these artifacts do not seem to have a big impact on statistics such as PSNR, which we suspect is why the neural networks did not prevent them from happening. Unfortunately, even with some rudimentary investigations, we were not able to determine the cause of these artifacts nor a set of conditions to reliably trigger them, which suggests that more investigattion is needed to eliminate these unexpected artifacts.

The second limitation is computational time. Currently the neural networks takes around five seconds to compress or decompress a slice at 1024^2 on a CPU, which is approximately one order of magnitude slower than traditional compressors. This performance could potentially be improved by moving the calculations to a GPU, but this approach might incur rounding errors as pointed out by [33]. After all, more research is needed to improve the inference speed and mitigate rounding errors so GPU compression/decompression can be possible.

With regard to future work the most prominent is training the model on relevant data sets, namely floating point scientific data. More ideally, they can be trained on generic data sets and then be fine tuned using data specifically from their intended compression task. The loss function during training can also be adjusted in an attempt to address some of the limitations discussed here. Finally, the network can be improved to

support 3D volumetric data as well, since it poses greater pressure to HPC data storage and management systems.

VII. Conclusion

Our study shows the promising potential of using neural network based lossy compressors on floating-point scientific data, and supports the need for their further investigation. The neural network in this study, despite that it was trained and optimized exclusively on natural images, performs well on two representative scientific data sets. Using measures of peak signal-to-noise ratio and maximum point-wise error, it often outperforms two purpose-built scientific data compressors in lower bit rates, and remains competitive in higher bit rates. The two test data sets we chose are from HPC application domains that are some of the most challenged by data deluge, and are highly varied in their characteristics, providing a strong indicator that other HPC applications might also benefit from neural-network-based compression.

Acknowledgment

This material is based upon work supported by the National Center for Atmospheric Research, which is a major facility sponsored by the National Science Foundation under Cooperative Agreement No. 1852977. We also acknowledge the SIParCS internship program funded by the National Science Foundation (Award No. AGS-1852977).

References

[1] Computational and N. C. f. A. R. Information Systems Laboratory, "Advance earth system science through hpc and data services, part of 2019 cisl annual report," 2019. [Online]. Available: https://nar.ucar.edu/2019/cisl/advance-earth-system-science-through-hpc-and-data-services

[2] I. Foster, M. Ainsworth, B. Allen, J. Bessac, F. Cappello, J. Y. Choi, E. Constantinescu, P. E. Davis, S. Di, W. Di *et al.*, "Computing just what you need: Online data analysis and reduction at extreme scales," in *European conference on parallel processing*. Springer, 2017.

[3] T. Bicer, J. Yin, D. Chiu, G. Agrawal, and K. Schuchardt, "Integrating online compression to accelerate large-scale data analytics applications," in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. IEEE, 2013, pp. 1205–1216.

[4] P. Lindstrom and M. Isenburg, "Fast and efficient compression of floating-point data," *IEEE transactions on visualization and computer graphics*, vol. 12, no. 5, pp. 1245–1250, 2006.

[5] G. K. Wallace, "The jpeg still picture compression standard," *IEEE transactions on consumer electronics*, vol. 38, no. 1, 1992.

[6] A. Skodras, C. Christopoulos, and T. Ebrahimi, "The jpeg 2000 still image compression standard," *IEEE Signal processing magazine*, vol. 18, no. 5, pp. 36–58, 2001.

[7] J. Lainema, M. M. Hannuksela, V. K. M. Vadakital, and E. B. Aksu, "Hevc still image coding and high efficiency image file format," in *2016 IEEE International Conference on Image Processing (ICIP)*, 2016.

[8] Google, "WebP, a new image format for the web," https://blog.chromium.org/2010/09/webp-new-image-format-for-web.html, 2010, (Online, accessed March 11 2021).

[9] A. Baker, H. Xu, D. Hammerling, S. Li, and J. Clyne, "Toward a multi-method approach: Lossy data compression for climate simulation data," in *The 1st International Workshop on Data Reduction for Big Scientific Data (DRBSD-1)*. Springer, 2017.

[10] S. Li, S. Sane, L. Orf, P. Mininni, J. Clyne, and H. Childs, "Spatiotemporal wavelet compression for visualization of scientific simulation data," in *2017 IEEE international conference on cluster computing (Cluster)*. IEEE, 2017, pp. 216–227.

[11] S. Li, N. Marsaglia, C. Garth, J. Woodring, J. Clyne, and H. Childs, "Data reduction techniques for simulation, visualization, and data analysis," in *Computer Graphics Forum*, no. 0. Wiley Online Library, 2018.

[12] J. Woodring, S. Mniszewski, C. Brislawn, D. DeMarle, and J. Ahrens, "Revisiting wavelet compression for large-scale climate data using jpeg 2000 and ensuring data precision," in *2011 IEEE Symposium on Large Data Analysis and Visualization*. IEEE, 2011, pp. 31–38.

[13] S. Di and F. Cappello, "Fast error-bounded lossy hpc data compression with sz," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2016, pp. 730–739.

[14] X. Liang, S. Di, D. Tao, S. Li, S. Li, H. Guo, Z. Chen, and F. Cappello, "Error-controlled lossy compression optimized for high compression ratios of scientific datasets," in *2018 IEEE International Conference on Big Data (Big Data)*, 2018, pp. 438–447.

[15] P. Lindstrom, "Fixed-rate compressed floating-point arrays," *IEEE transactions on visualization and computer graphics*, vol. 20, no. 12, 2014.

[16] S. Li, N. Marsaglia, C. Garth, J. Woodring, J. Clyne, and H. Childs, "Data reduction techniques for simulation, visualization and data analysis," *Computer Graphics Forum*, vol. 37, no. 6, pp. 422–447, 2018.

[17] G. Toderici, S. M. O'Malley, S. J. Hwang, D. Vincent, D. Minnen, S. Baluja, M. Covell, and R. Sukthankar, "Variable rate image compression with recurrent neural networks," *arXiv preprint arXiv:1511.06085*, 2015.

[18] G. Toderici, D. Vincent, N. Johnston, S. Jin Hwang, D. Minnen, J. Shor, and M. Covell, "Full resolution image compression with recurrent neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5306–5314.

[19] J. Ballé, V. Laparra, and E. P. Simoncelli, "End-to-end optimized image compression," *arXiv preprint arXiv:1611.01704*, 2016.

[20] X. Zhang and X. Wu, "Ultra high fidelity deep image decompression with l_∞-constrained compression," *IEEE Transactions on Image Processing*, vol. 30, pp. 963–975, 2020.

[21] E. Agustsson, M. Tschannen, F. Mentzer, R. Timofte, and L. V. Gool, "Generative adversarial networks for extreme learned image compression," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 221–231.

[22] F. Mentzer, G. Toderici, M. Tschannen, and E. Agustsson, "High-fidelity generative image compression," *arXiv preprint arXiv:2006.09965*, 2020.

[23] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston, "Variational image compression with a scale hyperprior," *arXiv preprint arXiv:1802.01436*, 2018.

[24] D. Minnen and S. Singh, "Channel-wise autoregressive entropy models for learned image compression," in *2020 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2020, pp. 3339–3343.

[25] Y. Liu, Y. Wang, L. Deng, F. Wang, F. Liu, Y. Lu, and S. Li, "A novel in situ compression method for cfd data based on generative adversarial network," *Journal of Visualization*, vol. 22, no. 1, pp. 95–108, 2019.

[26] S. Chandak, K. Tatwawadi, C. Wen, L. Wang, J. A. Ojea, and T. Weissman, "Lfzip: Lossy compression of multivariate floating-point time series data via improved prediction," in *2020 Data Compression Conference (DCC)*. IEEE, 2020, pp. 342–351.

[27] A. Glaws, R. King, and M. Sprague, "Deep learning for in situ data compression of large turbulent flow simulations," *Physical Review Fluids*, vol. 5, no. 11, p. 114602, 2020.

[28] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[29] P. Lindstrom, "Error distributions of lossy floating-point compressors," Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), Tech. Rep., 2017.

[30] Y. Li, E. Perlman, M. Wan, Y. Yang, C. Meneveau, R. Burns, S. Chen, A. Szalay, and G. Eyink, "A public turbulence database cluster and applications to study lagrangian evolution of velocity increments in turbulence," *Journal of Turbulence*, no. 9, p. N31, 2008.

[31] J. E. Kay, C. Deser, A. Phillips, A. Mai, C. Hannay, G. Strand, J. M. Arblaster, S. Bates, G. Danabasoglu, J. Edwards *et al.*, "The community earth system model (cesm) large ensemble project: A community resource for studying climate change in the presence of internal climate variability," *Bulletin of the American Meteorological Society*, vol. 96, no. 8, pp. 1333–1349, 2015.

[32] P. Lindstrom, "ZFP documentation: Compression modes," https://zfp.readthedocs.io/en/release0.5.5/modes.html, 2014, (Online, accessed March 17 2021).

[33] J. Ballé, N. Johnston, and D. Minnen, "Integer networks for data compression with latent-variable models," in *International Conference on Learning Representations*, 2018.

VIII. Appendix

Variable	Range	Minimum	Maximum	Mean	Standard Deviation	Auto-corr (Latitude)	Auto-corr (Longitude)
Q200	1.89e-04	6.63e-07	1.89e-04	1.64e-05	1.67e-05	0.969	0.976
FSNSC	3.21e+02	0.00e+00	3.21e+02	1.68e+02	1.10e+02	0.870	0.985
so4_a1_SRF	2.22e-08	7.29e-18	2.22e-08	3.54e-10	8.14e-10	0.915	0.935
FSNS	3.21e+02	0.00e+00	3.21e+02	1.47e+02	1.04e+02	0.869	0.973
U010	1.79e+02	-5.08e+01	1.28e+02	1.09e+01	3.43e+01	0.995	0.996
V850	6.53e+01	-3.13e+01	3.40e+01	2.76e-01	5.76e+00	0.970	0.963
UBOT	5.54e+01	-3.04e+01	2.50e+01	3.00e-01	6.26e+00	0.982	0.967
so4_a3_SRF	7.24e-10	2.12e-19	7.24e-10	6.72e-12	1.87e-11	0.922	0.948
FLNSC	1.90e+02	-1.05e+01	1.79e+02	7.97e+01	2.31e+01	0.945	0.951
SHFLX	4.57e+02	-8.49e+01	3.72e+02	1.35e+01	2.92e+01	0.905	0.899
dst_a1_SRF	1.04e-06	2.12e-20	1.04e-06	1.24e-09	9.84e-09	0.878	0.884
TAUY	2.69e+00	-1.25e+00	1.44e+00	-2.22e-03	1.02e-01	0.955	0.946
PRECT	1.92e-06	-4.53e-22	1.92e-06	2.82e-08	6.22e-08	0.917	0.872
TREFHTMX	1.13e+02	2.02e+02	3.15e+02	2.79e+02	2.32e+01	0.950	0.972
VBOT	5.71e+01	-2.90e+01	2.81e+01	4.04e-01	5.39e+00	0.972	0.962
TMQ	7.18e+01	6.16e-02	7.18e+01	1.78e+01	1.53e+01	0.973	0.987
PSL	1.06e+04	9.37e+04	1.04e+05	1.01e+05	1.47e+03	0.975	0.987
Q500	6.32e-03	8.39e-06	6.32e-03	8.80e-04	1.06e-03	0.966	0.968
Z050	2.75e+03	1.81e+04	2.09e+04	2.02e+04	7.76e+02	0.985	0.982
PRECSL	5.11e-07	-2.24e-20	5.11e-07	4.91e-09	1.72e-08	0.584	0.840
FLNS	2.36e+02	-5.63e+01	1.79e+02	6.64e+01	2.91e+01	0.936	0.934
TREFHT	1.21e+02	1.94e+02	3.15e+02	2.76e+02	2.36e+01	0.948	0.974
TAUX	2.53e+00	-1.30e+00	1.23e+00	-1.67e-02	1.22e-01	0.965	0.956
pom_a1_SRF	1.64e-07	1.16e-19	1.64e-07	4.39e-10	2.05e-09	0.921	0.931
bc_a1_SRF	1.54e-08	1.19e-20	1.54e-08	6.86e-11	3.38e-10	0.916	0.928
soa_a1_SRF	7.07e-08	2.44e-17	7.07e-08	9.36e-10	3.54e-09	0.929	0.916
TS	1.13e+02	2.00e+02	3.13e+02	2.77e+02	2.46e+01	0.932	0.974
Q850	1.65e-02	1.59e-06	1.65e-02	4.51e-03	3.75e-03	0.958	0.982
T200	4.51e+01	1.89e+02	2.34e+02	2.13e+02	7.95e+00	0.977	0.978
T500	4.93e+01	2.22e+02	2.72e+02	2.51e+02	1.30e+01	0.974	0.986
PRECL	1.17e-06	-4.53e-22	1.17e-06	1.37e-08	4.18e-08	0.866	0.840
PRECSC	7.00e-08	0.00e+00	7.00e-08	1.30e-10	1.37e-09	0.176	0.472
dst_a3_SRF	2.78e-05	6.31e-18	2.78e-05	2.09e-08	2.14e-07	0.860	0.858
WSPDSRFAV	3.33e+01	3.07e-01	3.36e+01	7.61e+00	4.33e+00	0.956	0.952
T850	1.04e+02	1.98e+02	3.02e+02	2.71e+02	1.90e+01	0.968	0.972
so4_a2_SRF	2.48e-09	4.69e-18	2.48e-09	1.01e-11	3.66e-11	0.892	0.908
QBOT	2.33e-02	1.59e-06	2.33e-02	6.58e-03	5.66e-03	0.961	0.986
ICEFRAC	1.00e+00	0.00e+00	1.00e+00	1.29e-01	3.18e-01	0.280	0.935
Z500	1.34e+03	4.61e+03	5.94e+03	5.50e+03	3.62e+02	0.976	0.988
U200	1.39e+02	-3.95e+01	9.93e+01	1.46e+01	1.74e+01	0.990	0.984
U500	8.14e+01	-2.77e+01	5.37e+01	6.52e+00	1.10e+01	0.989	0.983
OMEGA500	1.99e+00	-1.09e+00	9.02e-01	-1.38e-03	9.36e-02	0.924	0.885
T010	6.19e+01	1.87e+02	2.49e+02	2.25e+02	1.01e+01	0.985	0.968
TREFHTMN	1.11e+02	1.94e+02	3.05e+02	2.74e+02	2.40e+01	0.945	0.975
FSNTOA	4.00e+02	0.00e+00	4.00e+02	1.87e+02	1.21e+02	0.862	0.974
FLUT	2.47e+02	9.75e+01	3.44e+02	2.23e+02	5.05e+01	0.956	0.966
soa_a2_SRF	2.67e-10	1.05e-22	2.67e-10	1.62e-12	6.62e-12	0.864	0.866
U850	7.31e+01	-3.96e+01	3.35e+01	1.83e+00	7.63e+00	0.983	0.971
V200	1.14e+02	-5.97e+01	5.48e+01	-2.42e-01	1.21e+01	0.985	0.988
LHFLX	8.16e+02	-7.20e+01	7.44e+02	6.23e+01	6.82e+01	0.941	0.958
V500	7.55e+01	-3.48e+01	4.07e+01	-1.26e-02	7.51e+00	0.978	0.982

TABLE IV: Summary Statistics for CESM LENS Variables used in our experiment. Autocorrelation (noted as Auto-corr) was calculated with a 1-lag shift in either latitude or longitude directions. Autocorrelation values were averaged for each variable across 10 timesteps.

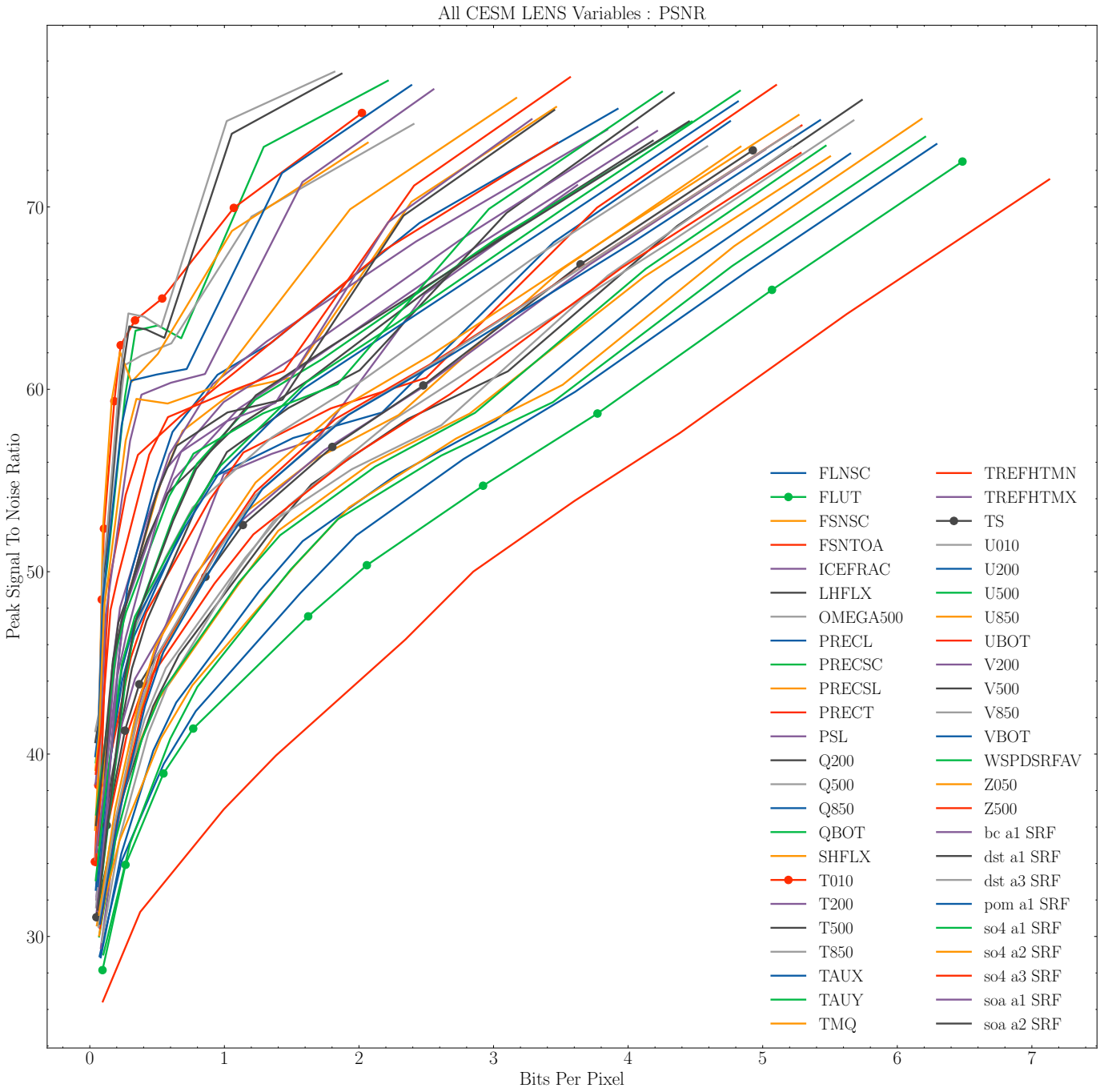


Fig. 7: Rate-distortion curves with PSNR for every variable in the CESM-LENS dataset. The different colors represent different variables in the dataset averaged across 10 timesteps. Lines with dots along them mark the variables which are studied in the main text and compared with other compressors in Figure 5.