A Generic and Efficient Framework for Estimating Lossy Compressibility of Scientific Data

Md Hasanur Rahman
University of Iowa
IA, USA
mdhasanur-rahman@uiowa.edu

Sheng Di
Argonne National Laboratory
IL, USA
sdi1@anl.gov

Guanpeng Li
University of Iowa
IA, USA
guanpeng-li@uiowa.edu

Franck Cappello
Argonne National Laboratory
IL, USA
cappello@mcs.anl.gov

Abstract—**Modern HPC applications produce extremely large amounts of scientific data, which cannot be stored and transferred efficiently. Error-bounded lossy compression has been effective in significantly reducing the volume of data. However, lossy compressors are mainly driven by user-specified error-bound, so the compression ratios would be unknown until compression is completed. In practice, however, many users need to forecast the compression ratio beforehand, such that related data storage, transfer and management could be orchestrated more efficiently and proactively. In this paper, we propose XTIMATE, a compressor-agnostic lossy compression framework capable of accurately and efficiently estimating compression ratio of scientific data. Our key observation is that the characteristics of data textures play an important role in understanding lossy compressibility. The key contributions are three-fold. (1) We provide an in-depth analysis on effective data features, especially analyzing the characteristics of varying data textures. (2) We carefully explore the best-fit kernel filter and develop a series of optimization strategies, in light of both estimation capability and performance. (3) We comprehensively evaluate XTIMATE based on three state-of-the-art lossy compressors using 10 real-world scientific datasets from 5 HPC applications and compare with state-of-the-art related works. Experiments show that average estimation error of XTIMATE is only 6.77%. Furthermore, XTIMATE exhibits a 30× speedup on average compared to the related works.**

Index Terms—**HPC Application, Scientific Data, Lossy Compressibility Estimation, Data Textures**

I. Introduction

Contemporary high-performance computing (HPC) applications generate vast volumes of scientific data across various domains such as astronomy, environment, and physics. For example, as stated by Gleckler et al. [1], the Coupled Model Intercomparison Project with Community Earth System Model (CESM) may produce about 2.5PB of data plus an additional 170TB of data during post-processing [2]. Moreover, Cosmological simulations using the Nyx [3] code can also generate around 2.8PB of data considering 5 simulation runs each with 200 snapshots. These vast amounts of simulation data present substantial challenges in terms of data storage, data transmission, and posthoc analysis for scientific research. Even modern supercomputers such as Summit [4] generally offer only a limited storage space to users (50TB storage space assigned to each Summit user [5]). Therefore, significantly reducing large volume of data to fit within limited memory capacity, storage space, and network I/O bandwidth is critical to users in practice.

While deduplication [6] and various lossless compressions [7]–[9] have been studied, they either cannot be used for scientific data or can offer only $\sim 2\times$ compression ratio [10]. To this end, error-bounded lossy compression [11]–[14] is arguably the most promising solution to reach high compression ratios (e.g., 10x or even higher [15]–[17]) with strict control on the data distortion to ensure high data fidelity for post-hoc analysis [2], [18].

The existing error-bounded lossy compressors, however, are mainly designed based on the error-bound constraint, so the compression ratio is unknown until the compression is finished, which may cause serious issues in practice. Here, we give three practical use cases (yet not limited) that require knowing the compression ratio in advance before running the compressor. Note that the discussion of these use cases pertains to the compressed data, rather than the original data, as XTIMATE primarily concentrates on estimating the size of compressed data. (1) **Control storage space on demand**. A supercomputer user is always allocated a limited storage space (e.g., 50TB per user on Summit). As such, when the user needs to run a large-scale simulation producing a sheer amount of data (such as molecular dynamics simulations [19], [20] and cosmological simulations [21], [22]), using an error-bounded lossy compressor is an effective method to reduce the storage space. However, the compressed data size still needs to be estimated beforehand to avoid a simulation crash caused by exceeding the user's allocated storage space limit. (2) **Control memory footprint to improve scalability**. In real-world applications such as quantum circuit simulation [23] and deep learning, strict memory constraints often limit execution scalability. While quite a few studies [17], [24], [25] leverage error-bounded lossy compression techniques to address this, the unknown memory requirements (only be known after the execution of lossy compression) lead to inefficient allocation, wasting resources and potentially hindering other tasks. To overcome this, forecasting compression ratios becomes crucial for optimizing memory allocation. (3) **Control data transfer time**: Scientific data is frequently shared through remote data storage [26], [27]. To expedite posthoc analysis, recipients must transfer this extensive data to local system. Using highly compressed data can significantly reduce transfer time and alleviate I/O bottlenecks. Accurately predicting compressed data size aids users in assessing required data transfer bandwidth without the need for actual compression. More details about these use cases are discussed in Section X. Considering these use cases, quite a few studies have been proposed to estimate lossy compressibility [28]–[30]. However, these studies lack either compressor-agnostic solutions or are not capable of providing accurate estimation under different compression configurations/models.

In this paper, we propose a *compressor-agnostic* lossy **compressibility estimation** framework, XTIMATE[1], to accurately and efficiently estimate compression ratio. The significant benefit of this framework would be that (1) the HPC applications are able to foresee the compressed data size such that the data can be orchestrated more efficiently at runtime, e.g., pre-allocating appropriate memory, preserving proper storage space, (2) the estimation would be accurate under any compression configurations/models. To achieve such highly accurate and efficient compressibility estimation, our key observations are: (1) Scientific datasets frequently display varied *data textures*, analogous to textures in images, with characteristics that significantly differ across different fields. Here, the term *field* refers to a specific metric/modeling in an application: e.g., density, pressure, speed, etc, and (2) This substantial variation in data textures across these fields have significantly influence on data smoothness. As state-of-the-art lossy compressors is largely designed based on the exploitation of the data smoothness, thereby the variation of smoothness markedly affect the compressibility of the data. These observations are very crucial in understanding diverse data compressibilities offered by scientific datasets. However, accurately and efficiently estimating the compression ratio of a dataset based on a particular error-bound presents significant challenges. (1) Even for the dataset fields from the same application, their data characteristics could be very diverse, which are dominating factors of data compressibility. Identifying effective data features to project compressibility among diverse dataset fields is very challenging. (2) Various compressors have their own design principles, which could project largely different compression quality even with the same dataset under same error bound. (3) Even with the same lossy compressor, the compression ratio could also be largely varying according to user-specified error bounds [11], [12]. (4) It is non-trivial to explore the characteristics of various data textures to the projection of data compressibility, *which is the first study of its kind*. Specifically, there exist many different types of kernel filters to characterize data textures, where each kernel filter exhibits distinct effect. Hence, identifying the best-qualified kernel filter demands a rigorous analysis of the correlation of those filter data to the compresibility result. (5) Furthermore, the kernel filter-based texture characterization are often quite expensive compared to the actual compression time, so how to substantially lower the characterization time for the online analysis is a significant challenge.

[1]Available at https://github.com/hasanur-rahman/XTIMATE.

Our key contributions are fourfold:

- We propose new observations to understand the data compressibility, especially the substantial impact of diverse characteristics of the data textures on the accurate and efficient estimation of lossy compressibility of the scientific datasets. To the best of our knowledge, *we are the first to leverage data textures to the projection of data compressibility in a compressor-agnostic manner across different application dataset fields.*
- We conduct rigorous analysis to determine the best-fit kernel filter (in the context of our study) to characterize the various inherent properties of data textures offered by scientific datasets.
- We propose two optimization strategies to make XTIMATE significantly faster than the actual compression time. We comprehensively analyze and rank the contribution of each convolution space to the estimation ability, which is followed by pruning the convolution space. Moreover, we significantly reduce the data space to further boost the XTIMATE performance without hurting accuracy.
- We comprehensively evaluate our framework, XTIMATE, with three state-of-the-art lossy compressors based on 10 testing scientific dataset fields from 5 real-world HPC applications and compare it with the state-of-the-art lossy compressibility estimators. Experiments show that the average estimation error by XTIMATE is at most 7.40% when compared with the measured compression ratios obtained from those three lossy compressors. Moreover, our XTIMATE outperforms the related works by 30x, on average.

The rest of the paper is organized as follows. Section II provides the background and introduces the scientific datasets. We discuss the related works in Section III. Moreover, we introduce the problem formulation and presents the motivation in Section IV. Section IV is followed by Section VI and VII, where we present the design details of our framework XTIMATE. Then, we present our experiment setup and results in Section VIII and IX, which is followed by the discussion of real-world use cases in Section X and the conclusion in Section XI.

II. Scientific Data & Research Background

In this section, we provide the details of scientific datasets and present the research background.

A. Scientific Datasets:

For this study, we choose 43 real-world scientific datasets on various fields from 5 different HPC simulations/applications of diverse domains. We pull the datasets from SDRBench benchmark [26], which is widely used in lossy compression studies [11], [31]–[35]. We present the dataset details as follows: (1) *Exaalt*: Exaalt [19] application dataset fields are generated by molecular dynamics simulation. We use 1D (2869440) Exaalt dataset fields. Exaalt has six fields: X, Y, Z, Vx, Vy, Vz. (2) CESM [36] is produced by climate simulation.

We use 2D (1800x3600) CESM fields. We run our experiments on dataset fields CLDHGH, CLDLOW, CLDMED, CLDTOT, FLDS, FLNT, FLNS, FREQSH, FREQZM, FSDS, PHIS from CESM. (3) *Hurricane*: Hurricane dataset (3D: 100x500x500) was produced by a Climate simulation [37]. We evaluate all the 13 fields such as QCLOUD, QGRAUP, QICE, QRAIN, QSNOW, QVAPOR, CLOUD, PRECIP, P, TC, U, V, W. (4) *Miranda*: Miranda [38] datasets can be obtained from large turbulence simulations;. We use all the 7 3D (256x384 x384) Miranda fields for our experiment: Density, Diffusivity, Pressure, Velocity-x, Velocity-y, Velocity-z, and Viscocity. (5) Nyx [39] is a cosmology simulation data. We use all the 6 3D (512x512x512) Nyx dataset fields for our experiments: Baryon density, Dark matter density, Temperature, Velocity-x, Velocity-y and Velocity-z.

B. Background

1) Kernel Filters:: Different types of kernel filters are frequently used in the area of image processing to catch diverse characteristics of images such as textures. In particular, *sobel* [40] and *canny* [41] are two most commonly used filters to detect image textures [42]–[45]. In general, a d-dimensional filter is applied on a d-dimensional image. Note that as discussed later in Section VII-A2c, *sobel* filter is the best fit in our framework XTIMATE. Therefore, we here discuss about sobel filter-based convolution. In the following, as an example, we demonstrate the steps of applying 2D sobel filter-based convolution, which can be easily extended to the other dimensional cases (such as 1D and 3D).

First, the 2D *sobel* filters along the two different directions (x and y) are outlined as follows, respectively:

$$F_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad F_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

We apply these filters on each 3×3 region of a given 2D dataset D to derive the convolution results C_x and C_y for x and y directions respectively. Finally, we measure the magnitude C of the convolution by aggregating the results with the Formula: $C = \sqrt{{C_x}^2 + {C_y}^2}$. Note that the 2D filter has 9 convolution points along each direction, whereas the 3D filter has 27 convolution points along each direction (x, y or z). Noticeably, the convolution space/complexity increases rapidly as the number of dimensions increases.

2) Peak Signal-to-Noise Ratio (PSNR): We use PSNR as a metric to compare the visualization quality of a scientific dataset (compared) with respect to another dataset (baseline). *PSNR* is widely used in the lossy compression community [26], [46]. Suppose, *D* and *D'* are the *baseline* and *compared* datasets. Formula (1) determines the *PSNR*, where *max(*D_i*)*, *min(*D_i*)*, and *rmse(D,D')* denote maximum data value in *D*, minimum data value in *D*, and root mean squared error between *D* and *D'* respectively. By observing Formula (1), we see that a lower value of *rmse(D,D')* reveals smaller error, which in turn leads to a higher value of *PSNR*.

$$PSNR = 20 \cdot log_{10} \frac{max(D_i) - min(D_i)}{rmse(D, D')} \tag{1}$$

III. Related Works

In this section, we discuss the related works in two facets: existing error-bounded compressors, and existing compression ratio estimation frameworks.

Existing state-of-the-art lossless compressors can only provide $\sim 2\times$ compression ratio [10], which is far from desired in practice, especially in the realm of very large amount of scientific simulation data to be fitted in the available HPC storage systems. To tackle the issue, there have been multiple error-bounded lossy compressors developed to significantly reduce the sheer amounts of scientific data while preserving the reconstructed data quality on demand. According to the prior studies [12], [35], [47]–[49], SZ, ZFP and MGARD(+) exhibit the best performance in both compression ratio and speed from among all the existing modern lossy compressors. SZ [12], [15] is a typical error-bounded lossy compressor developed based on a prediction-based compression model. ZFP [13] is another state-of-the-art error-bounded lossy compressor developed based on the orthogonal transform. MGARD [50], [51] (short for MultiGrid Adaptive Reduction of Data) is an error-controlled lossy compressor designed based on multilevel/multigrid methods. MGARD+ [14] is an accelerated version with the same compression quality. We use value range-based relative error bound mode offered by the above compressors for this study. Compared with the above three error-bounded lossy compressors, other compressors either suffer from much lower compression ratios or much lower compression speed. For instance, Bit-grooming [52] and Digitrounding [53] are designed based on the calculation of the significant number of bits required to satisfy the user-specified error bound. These compressors suffer from much lower compression ratios [47] than the aforementioned compressors which take advantage of the correlation of nearby data in space.

In general, the error-bounded lossy compressors are driven by error constraint, so the users cannot foresee the compressed data size or compression ratio until the full compression operation is completed. To mitigate the problem, there have been quite a few compression ratio estimation methods proposed. Wang et al. [30] proposed an extrapolation method to estimate compression ratios for SZ, and also proposed a ratio estimation method for ZFP by analyzing the weighted average of *BitsPerBitplane* for each block. However, their solution still suffers from large estimation errors as shown in their evaluation results. Jin et al. [28] developed an analytical model to efficiently estimate compression ratio based on the SZ3 [12] in particular. Lu et al. [29] leveraged sampling and trial-based approach to estimate compression ratio for SZ1.4 [11] and ZFP [13] (namely ZFPv0.5.0) respectively. *While achieving moderate efficiency and accuracy, these solutions come with significant drawbacks, which will be discussed in Section IV-B.*

IV. Research Formulation & Motivation

In this section, we formulate the research problem and discuss the motivation behind this study.

A. Problem Formulation

We formulate the research problem to describe our research objective. Given a N-dimensional dataset $D = \{d_1, d_2, ..., d_n\}$ (n is the number of data points), a lossy compressor *Comp*, and a user-specified error bound *e*, our research objective is to efficiently and accurately estimate lossy compression ratio *ECR* by analysing and extracting the effective data characteristics from *D* while applying a set of our proposed optimization strategies *Opt*. More specifically, our objective with XTIMATE is twofold: 1) Accuracy: the *ECR* should be very close to the measured compression ratio (*MCR*) obtained by running the compressor *Comp* under *e*. To determine accuracy under *e*, we use the *estimation error* denoted as $|MCR - ECR|$ / *MCR*. 2) Efficiency: the execution time of XTIMATE is supposed to be much faster than the compressor's actual *compression time*, as required by many online use-cases.

B. Motivation

Research Challenges: Exploring effective data features poses a considerable challenge, particularly in the light of two expected capabilities in XTIMATE: (1) being compressor-agnostic, and (2) accurately estimating compressibility even based on an unseen dataset field in an application. It is noteworthy that datasets in an application are from two different types: (1) Datasets may correspond to different fields. Recall that the term *field* refers to the name of a specific simulation metric in an application: e.g., density, pressure, speed, etc., and (2) Datasets may originate from the same field but are generated at different time steps. Generally, datasets across different time steps within the same field often manifest highly similar data characteristics. This assertion is exemplified by drawing datasets from time-steps 40, 45, and 48 within the field W of the Hurricane application, as depicted in Figure 1 (a), (b), and (c) respectively. As illustrated, these time-step datasets share strikingly similar data patterns and properties, resulting in compression ratios that are close to each other under the same error bound and same compressor, measuring 121.71, 124.44, and 128.19 respectively.

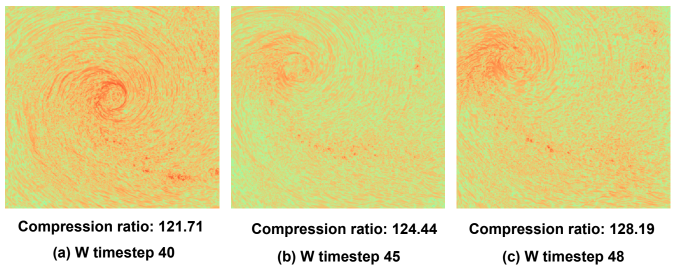


Fig. 1: Illustrating Similar Data Properties across Different Time-step Datasets of Same Field.

On the other hand, datasets from different fields within an application often exhibit notably distinct data characteristics, as illustrated in Figure 2 (a), (b), (c) and (d). Here, these fields V, QVAPOR, W and QICE of the Hurricane application demonstrate sharp differences in terms of visualization, data patterns, and properties, which have significant impact on the data compressibilities of those fields. Consequently, compression ratios obtained based these dataset fields under the same error bound and same compressor are distinct, measuring 39.64, 60.96, 128.19 and 321.26 respectively.

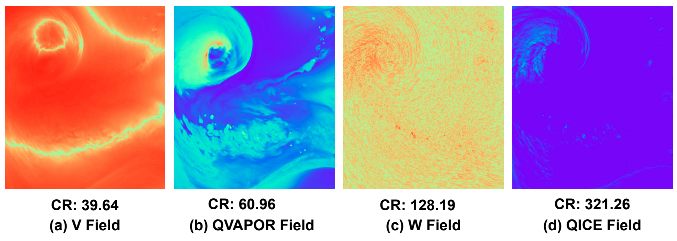


Fig. 2: Illustrating Distinct Data Properties across Different Field Datasets. Here, CR Denotes The Compression Ratios.

Thus, it is significantly challenging to estimate lossy compressibility across different fields without meticulous attention to nuanced details on data properties/patterns that render the compressibility of these dataset fields notably distinct. *The above comparison demonstrates that exploiting effective data features to project compressibility across different fields is rather challenging.* Owing to the above reasons, the focus of this paper is on the accurate estimation of data compressibility across different fields rather than across different time-steps, as the compressibility estimation across time-steps is relatively straightforward.

Key Advantages over Related Works: Compared with existing related works such as [28], [29], XTIMATE has two major advantages as follows: (1) XTIMATE provides better compression ratio estimation accuracy while also maintaining efficiency because XTIMATE is able to provide accurate compression ratio estimation even for compression configurations other than the default one (e.g., if we use a different prediction model/parameter in the SZ compressor). (2) While existing related works such as Jin et al. [28] provides compressor-specific modeling based on SZ3 (it is designed based on SZ3's design principle), our framework XTIMATE is compressor-agnostic, because of our generic data feature-driven analysis. Therefore, XTIMATE can be applied to any error-bounded lossy compressor for estimating the compression ratios efficiently. (3) The feature analysis in existing related works [28]–[30] is primarily based on intrinsic properties of either prediction-based or transformation-based compression models. In contrast, our framework, XTIMATE, examines data features – particularly those based on data texture – that are independent of the design principles of prediction-based and transformation-based compression models.

V. Our Observations

In this study, we observe that (1) scientific datasets across different fields obtained from HPC applications often exhibit various manifestations of data textures. Although existing related works [28]–[30] have studied various data features in understanding lossy compressibility, they do not particularly focus on this pivotal property of scientific data. We elucidate and analyze this observation with Figure 2. For illustrative purposes, we provide visualizations of four different dataset fields (V, QVAPOR, W, and QICE) from the Hurricane application. As depicted in the Figure, each dataset field manifests distinct forms of data textures. For instance, QICE field showcases data textures with less pronounced features, indicating relatively smoother transitions across regions. In contrast, V field demonstrates a more pronounced data texture with relatively abrupt transitions across regions.

Furthermore, (2) these distinct attributes of data textures also exert notable influence on data compressibility. The reason is twofold. (i) On one hand, as previously elucidated, various manifestations of data textures exert a significant influence on data smoothness. As such, in Figure 2, QICE exhibits data textures characterized by subdued features, thereby yielding a dataset field with comparatively smoother transitions, whereas the V field portrays data textures accentuated by prominent features, resulting in relatively abrupt transitions within the regions. (ii) On the other hand, existing state-of-the-art error bounded lossy compressors are mainly designed by exploiting the spatial correlation inherent within the scientific dataset. For example, the compression ratio provided by SZ [11], [12] compressor primarily hinges on the predictive accuracy derived from nearby data values, whereas ZFP [13] compressor transforms the original data domain to a new data space to decorrelate the data values for better compressibility. Therefore, as distinct characteristics of data textures influence data smoothness, they also affect the compressibility of data provided by lossy compression algorithms. For instance, as illustrated in the Figure, since QICE exhibits data textures leading to relatively smoother transitions compared to QVAPOR, the compression ratio based on the QICE field surpasses that based on the V field under the same error bound and same underlying compressor, measuring 321.26 and 39.64 respectively. Therefore, it is crucial to incorporate this special data characteristics as a feature in XTIMATE to comprehensively understand and accurately estimate data compressibility across different fields.

VI. Design Overview

In this section, we provide a design overview of our efficient and compressor-agnostic lossy compressibility estimation framework XTIMATE. Code is publicly available at https://github.com/hasanur-rahman/XTIMATE.

We present the design of XTIMATE in Figure 3. As shown in the Figure, XTIMATE takes user-specified error bound as the input along with the scientific dataset in interest. As for the design structure, XTIMATE contains three primary modules: (1) *Feature Extraction Module*, (2) *Optimization Module*, and

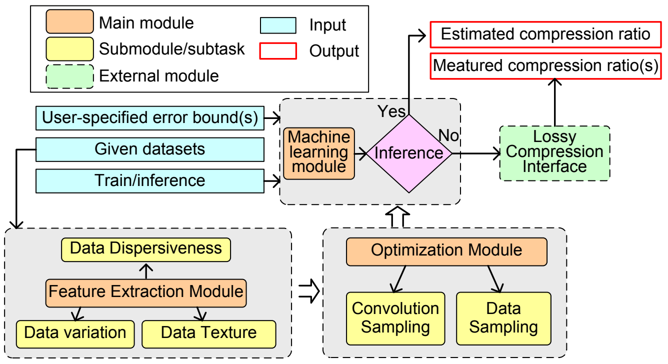


Fig. 3: XTIMATE Workflow

(3) *Machine learning (ML) Module*. Each primary module is divided into various submodules/subtasks. With the user-specified error bound and the given dataset, XTIMATE performs in-depth analysis to exploit diverse data characteristics, especially the various characteristics of data textures in the dataset. Analyzing and extracting these data characteristics help XTIMATE project data compressibility in a compressor-agnostic manner. Therefore, XTIMATE does not require to consider the specific design principles of the lossy compressor of interest at all. This *Feature Extraction Module* also depends on *Optimization Module* to further improve estimation accuracy and boost XTIMATE performance. The *Optimization Module* meticulously seeks for the improvement opportunities and optimizes various aspects of XTIMATE such as analyzing and reducing the broader convolution space. Finally, the *ML Module* becomes operational. More precisely, in the inference phase, utilizing both the extracted data features and the user-provided error bound, XTIMATE adeptly estimates the lossy compression ratio at runtime, eliminating the need for executing the associated lossy compressor.

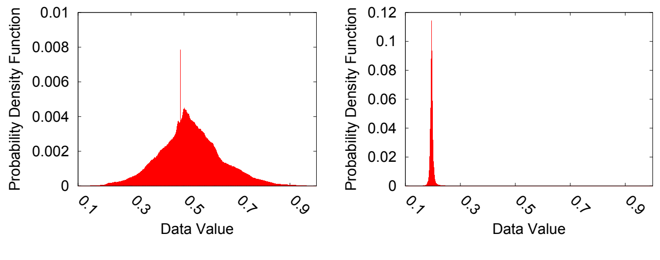
VII. Design Details

A. Feature Extraction Module

In this subsection, we discuss our first primary module and its submodules/subtasks. First, in Section VII-A1, we first focus on how our proposed data features can contribute to exploiting different aspects of data characteristics in a compressor-agnostic manner. Finally, in Section VII-A2, we describe how we can derive/quantify our data features.

1) ***Evaluating Diverse Facets of Data Traits:*** In this subsection, we perform a comprehensive investigation on how the data features we consider contribute to the data compressibility (both individually and collectively). For the purpose of in-depth understanding, we show the analysis results based on a set of *Hurricane* dataset fields among all our evaluated HPC applications. We use the compression results (error bound vs compression ratio) obtained from running SZ3 [12] compressor. Compression results with other dataset applications obtained from other compressors produce similar results. Note that the lossy compressors will be discussed in Section VIII-C respectively.

a) Impact of Data Dispersiveness: Data Dispersiveness expresses how much the data values are spread in a dataset over the value range of the dataset. On one hand, if most of the data values are clustered within a certain relatively small value interval, implying that the data values tend to be closer to each other, then they would be relatively easy to compress. On the other hand, if data values are scattered largely over the whole value range, then the dataset usually would be harder to compress.



(a) *V* (b) *W*

Fig. 4: Illustrating *Data Dispersiveness* with Data Distribution Based on Hurricane *V* and *W* fields.

Figure 4 validates the above point with the probability distribution function (PDF) of two dataset fields, *V* and *W*, from *Hurricane* application. Other dataset fields or other application datasets exhibit similar characteristics. As different dataset fields have different value ranges, we normalize the data values for each dataset to fit within [0,1] to have a fair comparison across different fields. Note that *V* and *W* have equal number of data points. In Figure 4, it is observed that *V* has a more dispersed data distribution than *W* has. Considering this observation, we expect *W* to be more compressible than *V*. To verify this point, we obtain the measured compression ratios (*MCR*s) for both datasets by running the *SZ3* compressor under same error bound. The *MCR* for *V* and *W* are 39.64 and 128.19 respectively, which matches our above observation.

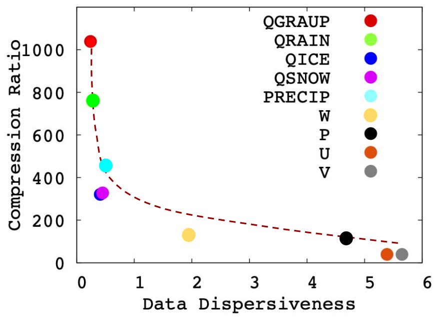


Fig. 5: Demonstrating The Impact of Data Dispersiveness on Compression Ratio among *Hurricane* Fields

Furthermore, we draw a pool of dataset fields from *Hurricane* application to demonstrate the following relationship: the more data are dispersed, the less the compression ratio will be, as shown in Figure 5. Note that we will discuss how we quantify the *Data Dispersiveness* feature in Section VII-A2. In Figure 5, each colored circle is a pair of values – *Data Dispersiveness* value (along x-axis) vs. *MCR* (along y-axis) under the same error bound – for each dataset field. Moreover, the red dotted curve shows the trendline of the relationship. As can be easily seen, dataset fields such as *QGRAUP* that have lower *Data Dispersiveness* generally are more compressible. In contrast, dataset fields such as *V* are less compressible because of the relatively higher value of *Data Dispersiveness*.

b) Impact of Data Variation: As scientific data are usually structured, spatial data locality would play an important factor to reveal the data compressibility. Specifically, when the data values correlate more to their nearby data values, the dataset turns out to be more compressible, and vice versa. The reason stems from the fact that lossy compressors often leverage various forms of compression models [11], [12], [14] as an intermediate phase in the compression pipeline. The effectiveness of those compression models is highly dependent on the smoothness/variation of the data.

We use Figure 6 to emphasize and quantify the above concept. The labels in the figure are similar to the ones in Figure 5 except that the x-axis now denotes the *Data Variation* feature values of each dataset field. It is worth noting that we quantify the variation by considering the difference between data values and mean of their neighboring data values. A lower value of the *Data Variation* indicates higher data smoothness, hence the higher data compressibility. The trendline is consistent with the analysis we discussed above. For example, a lower *Data Variation* value of the field *QGRAUP* exhibits a higher compression ratio.

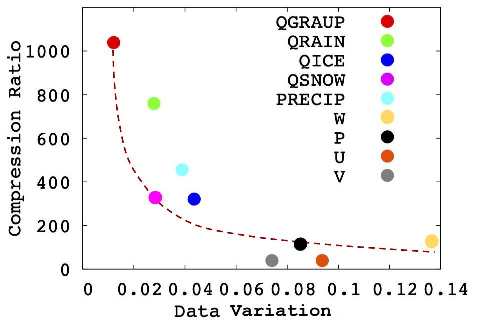


Fig. 6: Demonstrating The Impact of *Data Variation* on Compression Ratio among *Hurricane* Fields

c) Impact of Data Texture: Note that *Data Variation* feature may be limited to disclosing only the local characteristics as opposed to providing a broader view of the dataset because it only considers the data changes offered by nearby data points. Therefore, it is also crucial to understand data compressibility in a broader perspective. In Section V, we observe that data textures play an important role in understanding lossy compressibility. The reason is that data textures usually spread across the whole dataset (see our observations in Section V). Consequently, analyzing the data textures may reveal relatively broader aspects of data characteristics of a dataset. Therefore, in this study, we also characterize the different properties of data textures across dataset fields to comprehensively understand lossy compressibility.

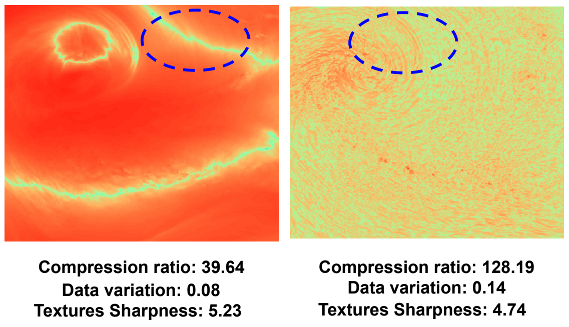


Fig. 7: Illustrating The Importance of Analyzing Textures

We use Figure 7 to verify the above point, which provides visualization of two dataset fields: V and W from *Hurricane* application. Recall that the lower the *Data Variation* value is, the higher the compression ratio tends to be. However, our measurement shows that W has higher data variation yet it is more compressible, which seems counter-intuitive at first glance. More specifically, the data variation values of V and W are 0.08 and 0.14 respectively while their compression ratios are 39.64 and 128.19 respectively under the same error bound. In fact, this seemingly counter-intuitiveness can be explained by the characteristics of data texture of a dataset. Here, we will give a brief intuitive explanation of different properties of data texture, especially the texture sharpness, with Figure 7. More details about how we quantify the characteristics of *Data Texture* can be found in Section VII-A2. Based on the visualization in the Figure, we can see that although V turns out to be more smooth (relatively less value of *Data Variation*) in the local regions than W does, it has more pronounced data texture. The above point can be validated by comparing the relationship among *MCR*s with the SZ3 compressor (under the same error bound), the value of *Data Variation* feature, and the value of sharpness of *Data Texture* feature based on dataset field V and W in Figure 7.

Therefore, to complement the inadequate capability of capturing broader data characteristics by the *Data Variation* feature, we include the characteristics of data texture, such as texture sharpness, in a dataset as they have ability to provide the broader view of the dataset. That is, when the data texture is more pronounced (texture sharpness is higher), signifying more abrupt changes in the broader data context, it leads to lower data compressibility. The reason is that more pronounced data textures foster more towards abrupt data changes in a relatively broader region. To verify this, we extract the sharpness of *Data Texture* feature values for both V and W, which are 5.23 and 4.74 respectively. Comparing regions marked with a blue dotted circle in each sub-figure of Figure 7, we observe that dataset V has more pronounced data textures than W does. Accordingly, V exhibits a lower *MCR* than W does. Therefore, we conclude that *Data Texture* feature acts as an effective addition to comprehensively understand data compressibility in such cases.

Similar to Figure 5 and 6, we also draw the trendline to find the relationship between the sharpness of *Data Texture* and compression ratio (under the same error bound) for each dataset field in Figure 8. According to the trendline, as the sharpness of data textures increases, the compressibility decreases, which is consistent with our above analysis.

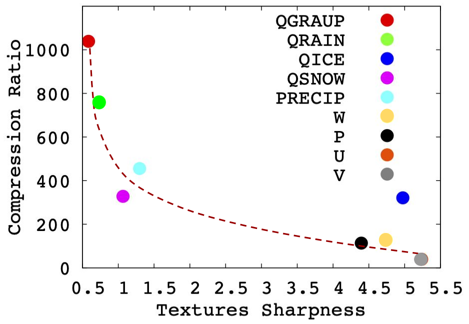


Fig. 8: Demonstrating The Impact of *Data Texture* Sharpness on Compression Ratio among *Hurricane* Fields

2) ***Quantifying Our Data Features:*** After providing a fine-grained analysis of the impacts of our proposed data features, we now present how to quantify those data features.

a) Data Dispersiveness: We use data distribution to quantify the data dispersiveness. Specifically, we first build a histogram based on the data distribution. The histogram/distribution of the data is constructed based on sampled data points for the purpose of high performance. We describe our sampling-based optimization strategy later in Section VII-B2. To build the histogram, we calculate the appropriate bin index of the histogram using Formula (2).

$$bin\ index = \frac{D_i - min}{bin\ size} \tag{2}$$

where D_i and *min* are referred to as the ith value and minimal value of all sampled data points, respectively. We set the bin size according to the user-defined error bound because a larger error bound leads to a more spiky distribution of histogram bins, which is consistent with the fact that a larger error bound tends to produce a higher compression ratio.

After building the histogram, we can now measure the entropy based on the frequency of different histogram bins. This measured entropy serves as a proxy to quantify the *Data Dispersiveness*. Entropy can be calculated by Formula (3), where $Freq_i$ is the frequency (number of data values) of bin i, B is the total number of bins, and S is the total number of sampled data points across all bins.

$$Entropy = \sum_{i=1}^{B} -\frac{Freq_i}{S} \times log_2 \frac{Freq_i}{S} \tag{3}$$

This entropy value represents the *Data Dispersiveness* feature essentially. Intuitively, a lower entropy signifies a greater concentration of data values within a reduced number of bins. Consequently, the dispersion of data values would be limited, resulting in a narrower distribution. As discussed, this phenomenon contributes to enhancing the compressibility of the dataset.

b) Data Variation: As *Data Variation* reveals the relationship between data points and their nearby data points, we can approximate it by comparing the currently processed data value with its neighboring values. Figure 9 shows the neighboring data points (red circles) with respect to a currently processed data point (blue circle) for 2D and 3D datasets respectively.

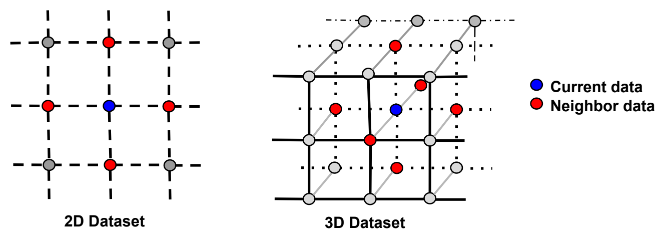


Fig. 9: Demonstrating *Data Variation* Feature Calculation

Data Variation feature is calculated as follows: We first gauge the difference between the current sampled data point and the mean of all its direct neighboring points. In Formula (4), $D_{i,j}$ refers to the currently processed data point in a 2D dataset while $D_{i-1,j}$, $D_{i,j-1}$, $D_{i+1,j}$ and $D_{i,j+1}$ are four neighboring data points. There are 6 neighboring data points in the 3D case. Finally, we compute the average of the difference values across all of the sampled data points. The intuition is that if the difference is lower, then the data values in a dataset tend to be more smooth or less variant in local regions, thus the overall dataset is expected to be more compressible, and vice versa.

$$variation_{i,j} = D_{i,j} - (D_{i-1,j} + D_{i,j-1} + D_{i+1,j} + D_{i,j+1})/4 \qquad (4)$$

c) Data Texture: In image processing, researchers often leverage different kernel filters to characterize different textures in an image [42]–[44]. As mentioned in Section V, scientific datasets also exhibit different data textures. Figure 7 serves as a good example of scientific datasets revealing data textures of varying sharpness. Analogous to image processing where kernel filters are applied to characterize image textures, we also apply kernel filters to characterize the data textures present in scientific datasets. As discussed in Section II-B, researchers in the image processing community frequently use two types of kernel filters: (1) *sobel* and (2) *canny* [40], [41], [45]. We infer that the *sobel* filter is the best choice in our problem context due to the following reason, Although both kernel filters-based characterization provide similar XTIMATE accuracy, as the number of dimensions and dataset size increases, *canny* filter-based texture characterization generally becomes very expensive. The key reason is that *canny* needs multiple passes/steps over the same data points as well as tuning its parameters (e.g., low and high threshold) for non-maximum suppression to characterize the textures. On the other hand, *sobel* filter-based characterization requires only one pass [54]. Therefore, *sobel* filter-based texture characterization has a good balance between accuracy and efficiency.

To verify the above point, we compare both the accuracy (i.e., compression ratio estimation ability) and efficiency (i.e., execution speed) of XTIMATE using *sobel* filter-based vs. *canny*-based data texture characterization. Note that during the comparison, we keep our other proposed features (e.g., *Data Dispersiveness*, *Data Variation*) in XTIMATE as it is. We use the 1D and 2D training datasets to derive the comparison results, as canny-based convolution would provide even slower performance for the cases with 3D datasets. Note that we do not include testing datasets used in the evaluation for fairness. Our experiments show that the average estimation errors are similar. Specifically, they are 6.20% and 5.84% using *sobel* filter and *canny* filter-based data texture characterization respectively in XTIMATE. Moreover, average execution time of XTIMATE using *sobel* is only 0.041 sec while it is 0.335 sec using *canny*. Based on our results, the *sobel* filter-based texture characterization in XTIMATE brings out two key conclusions: (1) similar accuracy (i.e., estimation error) as that of using *canny*-based convolution, and (2) much faster execution time as compared to *canny*-based convolution in XTIMATE. Consequently, we exclude *canny* filter-based convolution and only focus on applying *sobel* filter-based data texture characterization for *Data Texture* feature in XTIMATE.

We now quantify the sharpness of *Data Texture* feature based on *sobel* filter-based convolution strategy. First, we apply d-dimensional *sobel* filter on d-dimensional dataset D across all sampled data points, where each sample data point acts as a centered point in each convolution). Note that our data sampling-based optimization strategy is discussed in Section VII-B. These convolution results over the sampled regions correspond to the characteristics of the data texture. Next, to characterize the sharpness of *Data Texture* from these convolution results, we build a histogram to analyze the distribution of convolution results. Finally, we determine the entropy of the histogram in a similar way as we did for the *Data Dispersiveness* feature extraction by Formula (2) and (3). The entropy measurement follows the same process as that of the *Data Dispersiveness* feature extraction except that the texture-based histogram projects important insights on the characteristics of data texture as opposed to that of data values.

3) ***Correlation Analysis of Features:*** We now assess the average correlation between our features and compression ratio across different training dataset fields based on different error bounds. Table I shows the results. To compute the correlation for each feature within a specific compressor context, we evaluate feature values and compression ratios under diverse error bounds for different training datasets. For each error bound, we obtain two sets of results (feature value vs. compression ratio) linked with all training datasets. Finally, we calculate the Spearman Ranking Correlation Coefficient for these two result arrays and determine the average correlation across diverse error bounds. Recall that Spearman Ranking Correlation

Coefficient provides measurement between -1 and +1, where -1 implies strong negative correlation and +1 implies strong positive correlation. Consistent with the insights/observations from our feature analysis in Section VII-A1, all of our proposed features yield a strong correlation with compression ratio. Across all compressors, the average correlations are -0.69, -0.70, and -0.78 for the three features, respectively. These results confirm that our proposed features are expected to be effective in estimating compression ratios with XTIMATE.

TABLE I: Average Correlation Coefficient between Each Feature Value and Compression Ratio across Different Training Dataset Fields Based on Different Error Bounds

Compressor	***Data Dispersive-ness***	***Data Variation***	***Data Texture***
SZ3	-0.68	-0.69	-0.80
SZ1.4	-0.67	-0.70	-0.79
ZFP0.5.0	-0.73	-0.69	-0.77
MGARD+	-0.70	-0.73	-0.77

B. Optimization Module

Now, we present our optimization strategies for reducing kernel filter convolution space and data points during feature extraction to speed up the XTIMATE performance.

*1) **Optimization 1: Reduce Convolution Space for Sobel Filter**:* We expect our framework XTIMATE to be faster than the the actual compression time with a lossy compressor. Note that *compression time* refers to the time needed to execute a particular lossy compressor with an error bound on a dataset. One of the dominant factors in the execution time of XTIMATE is the number of convolved regions required during *Data Texture* feature extraction. Assuming we have N data points and need M convolutions for each data point (this point acting as the centered one during convolution) along each direction, the total convolution space is $N \times M \times d$, where d is the number of dimensions of the dataset. This $N \times M \times d$ would pose a serious challenge to maintain efficiency for higher-dimensional datasets. For example, as discussed in Section II-B for a 3D dataset, there are a total of 27 ($3 \times 3 \times 3$) convolution regions in 3D sobel filter along each axis direction. Consequently, the total number of convolution regions along all directions for all N data points is $N \times 27 \times 3$, which would be extremely large for a high-dimensional dataset even if we perform convolution only on sampled data points. Recall that the expected goal of XTIMATE is to reach a high accuracy yet without compromising performance.

To address the issue of large convolution space, we extensively analyze the relationship among convolution space, visual data texture quality and estimation accuracy, aiming to reduce the space effectively. Our approach involves identifying the regions in *sobel* filter that most significantly contribute to XTIMATE's accuracy (i.e., ratio estimation). By selecting these critical points, we achieve two objectives: (1) maintaining accuracy to the greatest extent possible compared to using all convolution points, and (2) sustaining high XTIMATE efficiency. Then the question becomes: *How can we rank the convolution filter regions to retain only the critical ones?* To answer this question, we must revisit the core motivation for incorporating kernel filters in our study, which is to exploit the characteristics of data textures inherent in scientific datasets. As such, the critical convolution points are the ones that enable the detection of data textures more accurately (the detection would fail without all of them).

We conduct an extensive offline analysis to pinpoint the critical convolution regions in the *sobel* filter. Our analysis starts with applying the Sobel filter to all data points in dataset D to obtain the full convolution result, denoted by FC. We then conduct multiple tests using modified Sobel filters on D, each forming by skipping a different convolution point for comparison. For the test iteration i, we denote the sampled convolution result as SC_i. We employ Peak Signal-to-Noise Ratio ($PSNR$) to assess convolution distortion between FC and SC_i. $PSNR$ is a common metric for comparing visualization quality in scientific datasets. In our context, higher $PSNR$ values indicate better-preserved data texture quality. Using these $PSNR$ measurements for all SC_i, we construct a ranking based on the normalized values. This ranking allows us to generate heatmaps for the regions in *sobel* filter along each axis. Higher values on these heatmaps indicate greater significance of the corresponding filter regions in preserving texture quality.

Fig. 10 shows the heat map of 3D filter convolution along each x, y, z direction based on the average ranking results across all 3D datasets fields used in the XTIMATE training only. We do not include the testing dataset fields for fairness purpose. In the Figure, the more darker the cells are, the more critical the convolution points would be. Accordingly, we select the top 10 ranked points for the 3D Sobel filter in XTIMATE. Likewise, we make similar selections for Sobel filters of other dimensions (1D and 2D).

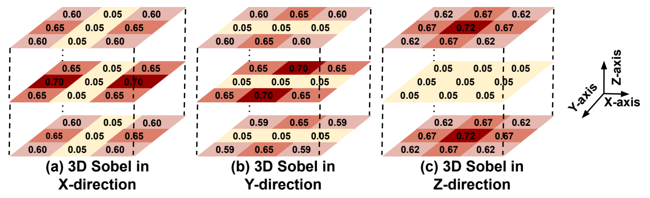


Fig. 10: Heat Map to Identify Significance Level of Each Convolution Space

TABLE II: Average XTIMATE Estimation Error and Running Time using Full and Reduced Convolution Space with 2D and 3D Datasets Across Different Error Bounds.

Metrics	CESM (2D)		Miranda (3D)		Nyx (3D)	
	Full	**Reduced**	Full	**Reduced**	Full	**Reduced**
Estimation Error	9.24%	9.34%	3.76%	5.10%	5.36%	4.55%
Running time(s)	12.80	8.84	43.24	23.94	269.33	141.32

Finally, we validate XTIMATE's accuracy and efficiency with and without this optimization. Table II shows the average estimation error and running time across various error bounds for scientific datasets from different applications. Due to space constraints, we present results for a selected few datasets, which closely resemble the other datasets. Regarding accuracy, the table demonstrates that reduced convolution space yields similar average estimation errors compared to full convolution space. However, XTIMATE with reduced convolution is approximately 2× faster that that with full convolution space. Moreover, as seen, with increasing dimensions and dataset size, XTIMATE's running time significantly increases when employing full convolution space. Based on the results, we verify that our reduced convolution space-based optimization is essential for XTIMATE to execute faster.

2) ***Optimization 2: Reduce Data Points in Feature Extraction:*** Now we discuss our second optimization, which aims at further improving the XTIMATE efficiency while still maintaining good accuracy.

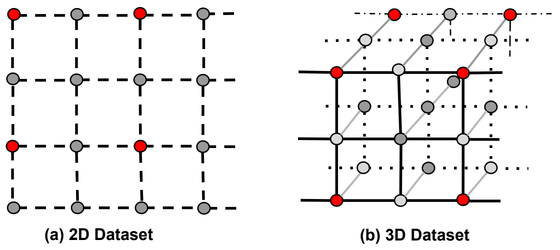


Fig. 11: Stride-2 Data Sampling

During the features extraction, we employ stride-k uniform sampling on the dataset. Figure 11 shows two examples of stride-2 uniform data sampling in 2D and 3D datasets respectively. Our sampling strategy maintains a low average sampling rate of just 3.02% across all 8 testing datasets. With 3.02% sampling, XTIMATE's average running time is approximately 90× faster compared to using all points. We further validate that this sampling strategy can still maintain high accuracy for XTIMATE. Based on our experiment results, XTIMATE average estimation errors across our evaluated datasets and all the compressors using all and 3.02% sampled data points are 5.90% and 6.77% respectively. As we can see, the difference between these estimation errors is negligible, which consolidates our sampling-based optimization strategy. Therefore, with this optimization, XTIMATE further improves the performance significantly while maintaining low estimation error. Note that the above accuracy and efficiency results are obtained with the *Optimization 1* is in place.

C. ML Module

In this section, we discuss how we train our framework XTIMATE. Finally, we detail the inference process for XTIMATE.

1) Advantage of ML Models over CNNs: In this paper, XTIMATE ties to traditional statistical ML models for training and inference instead of depending on CNNs. The reason is two-fold: (1) CNN-based modeling tends to operate at a high level of abstraction, lacking the granularity needed to discern the fine details of which data features play a crucial role in capturing the compressibility of the data. This limitation arises due to the automatic feature exploration performed by the inner algorithm of CNNs. In the realm of lossy compression studies, understanding the intricate interactions between data features and their impact on data compressibility is essential. For example, we would not know that data textures play an important role in understanding data compressibility (mentioned in Section V). Only through this comprehensive understanding can we propose more effective compression algorithms in the future, which can achieve higher compression ratio and minimized data distortion. (2) the training and inference processes involved in CNNs are time-consuming and resource-intensive. In the context of XTIMATE, when dealing with a scientific dataset, the necessity to partition the dataset into blocks and treat each block as individual entity to feed CNNs for training and inference introduces considerable time overhead. This is especially significant considering that the compression time taken to execute lossy compressor typically involves only a few full passes over the data values. On the other hand, CNNs often comprise numerous layers ranging from tens or hundreds of intermediate layers to a couple of fully connected layers, which require extensive fine-tuned parameterizing. In contrast, traditional statistical ML models are lightweight, enabling researchers to scrutinize the significance of each data feature with respect to compressibility very efficiently.

2) Choice of ML models: Recall that the goal of XTIMATE is to efficiently estimate lossy compression ratios based on user-provided error bound. As estimating compression ratios is a regression task, we focus on widely used ML model regressors such as decision tree regressor (*DTR*), random forest regressor (*RFR*) [55], support vector regressor (*SVR*) [56]. Based on our experimental results, we find that *DTR* and *RFR* are the best fit for our framework XTIMATE. Generally, *RFR* is more robust and superior to *DTR* because it aggregates many decision trees to make decisions and prevent overfitting. Hence, we adopt *RFR* in XTIMATE for the estimation.

SVR is not a good fit for our problem setting. The reason stems from the properties of *SVR* model. In particular, *SVR* requires feeding non-overlapping spaces of training samples to provide good accuracy. For our problem setting, we observe that measured compression ratios could sometimes be very close to each others so that there would be some overlapping regions in training samples.

To verify the above analysis, we perform a comparison of average compression ratio estimation errors of XTIMATE using *SVR* and *RFR* based on different testing datasets and compressors across different error bounds. Table III outlines the results. For each ML model and within a specific compressor context, we calculate the estimate errors under a set of various error bounds based on each testing dataset field. On average, we choose 10 uniformly distributed error bounds from the range 1E-10 to 1E-1. Finally, for each testing field, we calculate the average of such estimation errors across all four compressors. In the Table, compared to *RFR*, *SVR* incurs very high estimation errors for all testing fields. On average, while XTIMATE with *RFR* incurs only 6.77%, XTIMATE with *SVR* incurs 44.11% estimation error.

TABLE III: Comparison of average estimation errors with *SVR* and *RFR* across four lossy compressors for testing fields based on diverse error bounds

ML Model	Exaalt (2D) Vy	Exaalt (2D) Y	CESM (2D) CLDMED	CESM (2D) FLNT	Hurricane (3D) V	Hurricane (3D) PRECIP
SVR	18.31%	21.19%	21.89%	23.59%	75.72%	66.43%
RFR	4.49%	4.76%	10.07%	6.80%	9.29%	13.76%

ML Model	Miranda (3D) Viscocity	Miranda (3D) Velocityy	Nyx (3D) Temperature	Nyx (3D) Velocityy
SVR	114.39%	37.80%	19.95%	20.39%
RFR	6.57%	8.14%	8.16%	2.01%

3) XTIMATE *Training:* We now delve into the training of XTIMATE. To construct the training samples based on a given dataset, we first execute the corresponding compressor based on 25∼30 error bounds with a range encompassing from ∼1e-7 to ∼1e-1. After the executions, we collect the compression ratio results. Moreover, for each such error bound, we extract our three key data features by applying optimization strategies. Following the training data collection, we employ a 5-fold cross-validation on the training samples to fine-tune the parameters of *RFR* to further improve the XTIMATE accuracy.

4) XTIMATE *Inference:* We now introduce the inference phase of XTIMATE. This phase requires inputting a dataset, a required error bound and an associated lossy compressor for which the compression ratio would be estimated. With these inputs, XTIMATE engages its features extraction and optimization strategies to produce the testing sample. Finally, this testing sample is fed into XTIMATE to accurately estimate the compression ratio. We can see that XTIMATE does not require any execution of lossy compressors for the compression ratio estimation.

VIII. Experimental Methodology

In this section, we present the evaluation setup by discussing the system environment, datasets, lossy compressors and baselines.

A. System Setup & Baselines

We conduct our experiments on a server that is equipped with Intel Xeon E5-2695v4 nodes, where each node has up to 128GB DDR4 and 36 cores. The experiments regarding different compressors and estimation methods were executed in exactly the same environment for fairness. We compare our framework XTIMATE with two state-of-the-art related works: ICDE22 [28] and IPDPS18 [29]. ICDE22 proposes compression ratio modeling with the SZ3 compressor, which is customized for SZ3 in particular. For fairness, we use its default settings for the prediction model – lorenzo and regression. On the other hand, IPDPS18 aims to estimate the compression ratio with the SZ1.4 and ZFP0.5.0 compressors based on sampling-based gaussian modeling and trial-and-error approach respectively. As for IPDPS18 to estimate SZ1.4 compression ratio, it first reconstructs a new dataset based on sampling the original dataset, followed by extracting different parameters results (e.g, curve-fitting hit ratio, huffman tree size, etc.) from running SZ1.4 on the newly sampled dataset. After that, based on the parameters results, it applies a gaussian model to estimate the huffman node count on the original dataset. Finally, IPDPS18 uses a formula to estimate the compression ratio of the original dataset. Moreover, as for IPDPS18 to estimate ZFP0.5.0 compression ratio, it relies on a trial-and-error-based iterative approach on the new dataset based on the sampling. The default number of max-iterations of IPDPS18 is 20. We also perform IPDPS18 evaluation with ZFP0.5.0 under 50 max-iterations.

B. Dataset Evaluation Methodology

Evaluated datasets are presented earlier in Section II-A. Since our goal is to assess XTIMATE across different fields in an application, we randomly choose one dataset field (among all fields) as the testing dataset and all the rest of the fields as training datasets for each assessment. For each application, we show results with two dataset fields to balance the experiment time. For example, when we select the Temperature dataset field from *Nyx* as testing, we train XTIMATE using all the remaining fields from *Nyx* to assess XTIMATE. In a separate instance, the Velocity-y field from *Nyx* is chosen as the testing for XTIMATE, while the rest of the fields serve as the training datasets. This approach ensures that the evaluation process tests the XTIMATE's ability to generalize to the unseen dataset accurately, as it must perform well on dataset that it hasn't been explicitly trained on. Moreover, we also set up the evaluation process according to strength of the baselines. Specifically, as IPDPS18 works well with low dimensional data, we also include 1D and 2D datasets to have a fair comparison with it. *Note that in the evaluation results, labeling a figure with a dataset field from an application implies its use as a testing field for that application.*

C. Evaluated Lossy Compressors

We conduct the evaluation with three state-of-the-art lossy compressors such as SZ [11], [12], ZFP [13] and MGARD+ [50], [51], each with distinct compression design principles.

- *SZ*: SZ is a prediction-based error-bounded widely used lossy compressor. As two baselines in this study use two different versions of SZ, we conduct the evaluation with two versions of SZ: SZ1.4 [11] and SZ3 [12].
- *ZFP*: ZFP by Lindstrom et al. [13] is a tranform-based error-controlled lossy compressor. We use ZFP0.5.0 in our experiments as required for the comparison with the related works. While different transformation methods are available, Lindstrom et al. [13] indeed shows that block orthogonal transformation is more efficient on data correlation than other transformations such as discrete haar wavelet transform (HWT). Recall that the data

smoothness is one of the most important aspects when it comes to scientific data compression. Hence, we choose ZFP from the family of transform-based compression models.

- *MGARD+*: MGARD+ [14] is an accelerated version of MGARD [50], [51] with same compression quality. It leverages multigrid adaptive reduction approach.

IX. Evaluation

In this section, we provide the evaluation results. We consider two different metrics: (1) accuracy: the less the estimation error is, the more accurate the corresponding model is, (2) efficiency: how fast the execution of the corresponding model is.

A. Ablation Study

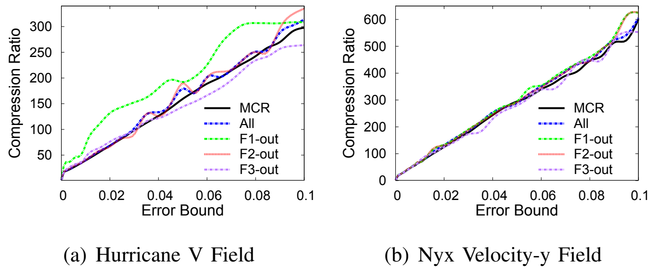


Fig. 12: Ablation Study for Data Features

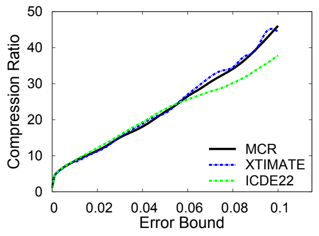
In this subsection, we validate the significance of each of our proposed data features with an ablation study based on Hurricane and Nyx application datasets. For the purpose of illustration, we show results under SZ3 compressor across different error bounds based on two testing fields *V* and *Velocity-y* from Hurricane and Nyx application respectively. Other application dataset fields show similar results. To verify and illustrate how each data feature takes effect and contribute to the overall compressibility estimation, we construct XTIMATE by excluding one feature each time, and observe the effect in estimating the compression ratio. Figure 12 demonstrates the results. In the Figure, *MCR*, *All*, *F1-out*, *F2-out* and *F3-out* denote the measured compression ratio (*MCR*), XTIMATE estimation, XTIMATE estimation without *Data Dispersiveness* feature, XTIMATE estimation without *Data Variation* feature and XTIMATE estimation without *Data Texture* feature respectively. As seen in the figure, the contribution of each feature towards the estimation accuracy varies across both different error bounds and different dataset fields. For example, the contribution of *Data Dispersiveness* feature would have the most significant contribution (highest estimation error of 38.86% among all other models) when using Hurricane *V* dataset. On the other hand, the *Data Texture* feature contributes mostly (highest estimation error of 5.08% among all other models) to overall compressibility when using Nyx *Velocity-y* field. It is also evident that the contribution is not uniform across different error bounds. However, if we use all of our proposed features (*All*), the estimated compression ratios would be the closest to the ground truth *MCR*, which are 3.87% and 2.36% with Hurricane *V* and Nyx *Velocity-y* respectively. These result signify the importance of considering all of our proposed features in XTIMATE to obtain accurate estimation of lossy compressibility.

B. Accuracy

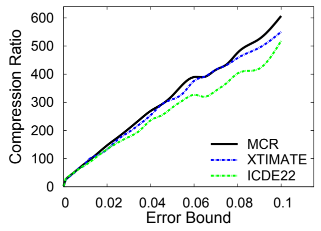
Here, we perform the accuracy analysis of our framework XTIMATE compared to the two related works: ICDE22 [28] and IPDPS18 [29] based on the estimation error.

1) Comparison between XTIMATE *and* ICDE22*:* We compare the accuracy between XTIMATE and ICDE22 by analyzing the estimation error across different error bounds. Figure 13 shows the estimation error across uniformly chosen 25$\sim$30 error bounds in the range from $\sim$1E-7 to $\sim$1E-1. For the sake of space, Figure 13 shows partial results with 4 testing dataset fields from 4 different HPC applications. Other testing dataset fields demonstrate similar results. On average, the estimation error for our framework XTIMATE is 7.40% while ICDE22 shows 12.91% estimation error. As seen in Figure 13, ICDE22 usually struggles to maintain good accuracy as the error bound gets larger. The key reason is that ICDE22 lacks enough quantization information to model the SZ3 execution under larger error bounds. Nevertheless, the average estimation is still reasonably good as ICDE22 model follows SZ3 design principle. The main disadvantage of ICDE22 is that it cannot be applied to estimate the compression ratios for other lossy compressors.

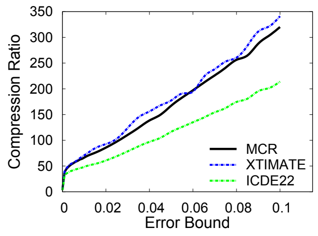
2) Comparison between XTIMATE *and* IPDPS18*:* We also compare the accuracy between XTIMATE and IPDPS18 by analyzing the estimation error across a set of error bounds. These error bounds are used in the IPDPS18 paper [29]. For the interest of space, in Figure 14, we provide partial results with two compressors: SZ1.4 and ZFP0.5.0 based on with 3 testing dataset fields (from 3 different applications). As seen in Figure 14, the compression ratio estimation by our framework XTIMATE is very close to the *MCR* for both SZ1.4 and ZFP0.5.0 results. However, IPDPS18, especially with SZ1.4, encounters higher estimation error under the majority of error bounds. We also observe that as the number of dimensions in the dataset become higher, the estimation error for IPDPS18 gets higher. The root cause behind the higher estimation error is the sampling approach adopted by IPDPS18 in the case with SZ1.4. After performing the sampling, IPDPS18 first reforms a new synthetic dataset by only considering those sampled data points and then performs the rest of its design steps. Contrasting to our approach in which we still consider retaining the characteristics of the original dataset, IPDPS18 loses the actual data charateristics from original dataset by constructing a completely new dataset with sampled data points. This contributes to the incorrect node count estimation in the guassian model used by IPDPS18, which in turn produces larger estimation errors. In the case of ZFPv0.5.0 compression ratio estimation, IPDPS18 follows trial-and-error-based approach after the sampling. We perform the evaluation under two max-iterations: 20 (default) and 50 for IPDPS18. As demonstrated in Figure 14, as we set larger



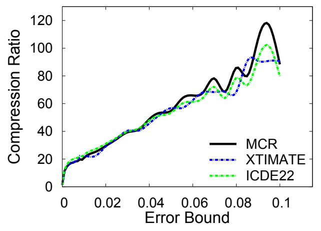
(a) Exaalt Y (with SZ3)



(b) CESM FLNT (with SZ3)

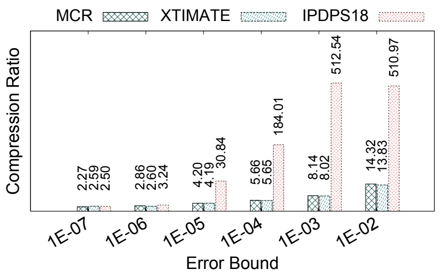


(c) Miranda Velocityy (with SZ3)

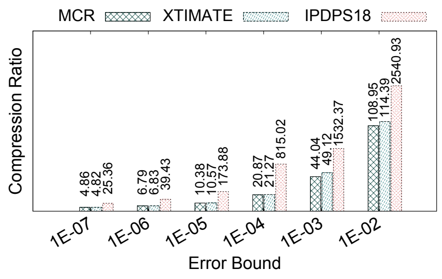


(d) Nyx Temperature (with SZ3)

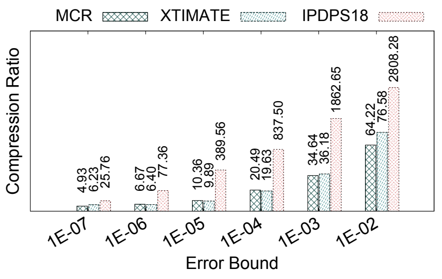
Fig. 13: Comparison of Estimation Error between XTIMATE and ICDE22 Models across Different Error Bounds.



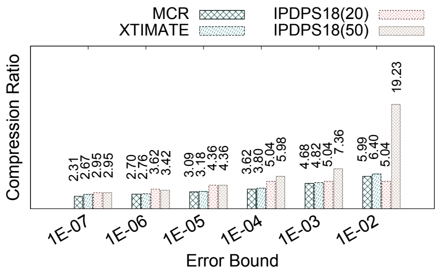
(a) Exaalt Y (with SZ1.4)



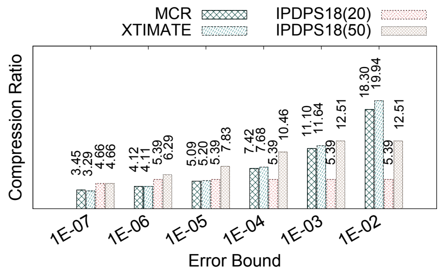
(b) CESM FLNT (with SZ1.4)



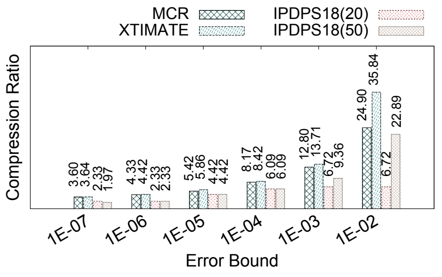
(c) Hurricane V (with SZ1.4)



(d) Exaalt Y (with ZFP0.5.0)



(e) CESM FLNT (with ZFP0.5.0)



(f) Hurricane V (with ZFP0.5.0)

Fig. 14: Comparison of Estimation Error between XTIMATE and IPDPS18 Models across Different Error Bounds. *IPDPS18(20)* and *IPDPS18(50)* denote the model execution under 20 and 50 max-iterations respectively.

max-iterations, the estimation error becomes lower. However, larger max-iterations also drastically reduces the performance because of the relatively larger number of executions of the corresponding compressor. On average, the estimation error for XTIMATE with SZ1.4 and ZFP0.5.0 are only 6.66% and 7.26%, while it is more than 20% for IPDPS18 with both SZ1.4 and ZFP0.5.0.

TABLE IV: Average Estimation Error of XTIMATE with MGARD+ across Different Error Bounds.

Comp.	Exaalt (2D)		CESM (2D)		Hurricane (3D)	
	Vy	Y	CLDMED	FLNT	V	PRECIP
MGARD+	2.10%	1.65%	6.31%	5.42%	6.82%	13.01%

Comp.	Miranda (3D)		Nyx (3D)		Average
	Viscocity	Velocityy	Temperature	Velocityy	
MGARD+	2.07%	2.79%	8.91%	2.42%	5.15%

3) Accuracy of XTIMATE *with MGARD+:* We claim that our framework is compressor-agnostic. To support the claim, we also evaluate XTIMATE accuracy by analyzing estimation error with the MGARD+ compressor. Table IV shows the average estimation error across different error bounds with MGARD+ based on all 10 testing datasets. On average across all dataset fields, XTIMATE only causes 5.15% estimation error. This result confirms that XTIMATE can be applied to accurately estimate compression ratios with any lossy compressor.

C. Efficiency

Now, we evaluate the execution time of XTIMATE, ICDE22 and IPDPS18 by comparing them with actual compression time.

Table V shows the experiment results. We perform comparisons with two related works. As for the comparison with ICDE22, we can see from the Table that, on average across all testing fields, XTIMATE is 10× faster than ICDE22. The main drawback of ICDE22 is that even though it leverages sampling and estimation approaches, the model still requires going through each stage of SZ3 compressor design. In contrast, we only need one pass to extract the data features.

TABLE V: Average Execution Time Compared to Corresponding Compressor's *Compression Time* for XTIMATE, ICDE22 and IPDPS18 (20 Iterations) Models across Different Error Bounds

Application	Testing Field	SZ3		SZ1.4		ZFP0.5.0		MGARD+
	Datasets	**XTIMATE**	ICDE22	**XTIMATE**	IPDPS18	**XTIMATE**	IPDPS18	**XTIMATE**
Exaalt(1D)	Vy	0.195x	0.301x	0.405x	7.668x	0.398x	5.416x	0.414x
	Y	0.181x	0.309x	0.340x	7.187x	0.338x	4.593x	0.405x
CESM(2D)	CLDMED	0.028x	0.390x	0.071x	10.404x	0.063x	3.692x	0.078x
	FLNT	0.025x	0.372x	0.064x	8.951x	0.059x	4.088x	0.071x
Hurricane(3D)	V	0.017x	0.663x	0.026x	6.377x	0.022x	1.359x	0.041x
	PRECIP	0.011x	0.666x	0.022x	8.367x	0.014x	0.943x	0.032x
Miranda(3D)	Viscocity	0.012x	0.757x	0.029x	6.782x	0.031x	1.134x	0.034x
	Velocityy	0.015x	0.672x	0.024x	5.508x	0.026x	1.058x	0.038x
Nyx(3D)	Temperature	0.025x	0.718x	0.056x	4.274x	0.058x	0.796x	0.045x
	Velocity-y	0.023x	0.723x	0.059x	5.153x	0.065x	0.914x	0.045x
Average	Across All Fields	0.053x	0.557x	0.126x	7.342x	0.122x	2.743x	0.120x

On the other hand, as for the comparison with IPDPS18, we perform two experiments with SZ1.4 and ZFP0.5.0. Considering both cases, on average, XTIMATE outperforms IPDPS18 by about 40x. The two key reasons are: (1) we observe that the gaussian model-based estimation stage reduces the overall performance of IPDPS18 with SZ1.4, and (2) the trial-and-error-based iterative approach largely contributes to the performance decline of IPDPS18 with ZFP0.5.0. As to verify the generality of our framework XTIMATE, we also perform experiments with MGARD+ compressor. As seen in the Table, on average across all testing datasets, the execution time of XTIMATE compared with the MGARD+ compression time is only $0.120\times$.

X. Real-world Use Cases

In this section, we elaborate several real-world use-cases for our framework XTIMATE.

a. Control storage space on demand: In the realm of processing scientific data or conducting posthoc analyses, HPC systems frequently grapple with the constraint of limited storage capacity. Employing an error-bounded lossy compressor emerges as a potent strategy within this context, adeptly curtailing the expanse of storage required. It is worth noting that the size of compressed data might still remain substantial due to the sheer magnitude of the raw simulation data and varying error control quality. This juxtaposes the situation where users are typically allotted a fixed storage space. For example, a user in supercomputing facilities is often assigned with a very limited storage capacity (e.g., $\sim$50TB for an ORNL summit regular user [57] and $\sim$10TB for an ANL Theta regular user [58]). Consequently, it becomes imperative to gauge the compressed data size, necessitating ratio estimation beforehand (before running compressor) to avert potential crashes arising from breaching the confines of the designated storage space threshold.

b. Control memory footprint to improve scalability: In some real-world applications such as quantum circuit simulation [23] and deep learning, the execution scalability is often limited by the strict memory capacity. To address this issue, there have been quite a few explorations [17], [24], [25] leveraging error-bounded lossy compression techniques to reduce the memory capacity requirement. However, memory needs to be pre-allocated to avail the compressed data. As the memory requirement cannot be forecast without the compression, each execution of those applications has to allocate a much larger memory space than actually needed, which results in waste of available memory resources. This may contribute to impeding the seamless execution of other pivotal tasks on the same platform. To mitigate the above issue, it becomes imperative to forecast the compression ratio to determine expected memory footprint. By gaining an insight about the projected memory footprint, such applications can optimize memory allocation strategies, preventing wastage and ensuring the smooth coexistence of diverse tasks.

c. Control data transfer time: Scientific data generated by HPC applications is frequently disseminated through remote data storage systems [26], [27]. When it comes to local post-hoc analysis, recipients, including database endpoints and users, face the challenge of remotely transferring vast amounts of data. Opting for highly compressed data, as opposed to the original data, is expected to reduce transfer time and alleviate I/O bottlenecks. However, to make informed decisions, accurate estimation of the compressed data size is crucial. This foresight enables users to assess the necessary data transfer bandwidth without the need for actual compression procedures, streamlining the planning process for efficient data handling.

XI. Conclusion

In this paper, we introduce a compressor-agnostic framework, XTIMATE, for accurately and efficiently estimating compression ratios that meets user-defined error bounds. XTIMATE leverages key data features, with a particular emphasis on data textures, to enhance data compressibility projections in a compressor-agnostic manner. We also propose two optimization strategies to further improve efficiency. As our framework is lightweight, accurate and efficient, it would help user enabling more efficient data orchestration during runtime, including memory allocation, storage space preservation, and remote data transfer. On average, XTIMATE incurs only a 6.77% estimation error, and it can achieve up to $50\times$ speedup in execution compared to related approaches.

Acknowledgments

This research was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research (ASCR), under contract DE-AC02-06CH11357, DE-SC0024559, and supported by the National Science Foundation under Grant OAC-2003709, OAC-2104023, OAC-2211538, OAC-2311875. This research used resources of the Argonne Leadership Computing Facility and resources of the Laboratory Computing Resource Center (LCRC) at Argonne, a U.S. Department of Energy (DOE) Office of Science user facility at Argonne National Laboratory and is based on research supported by the U.S. DOE Office of Science-Advanced Scientific Computing Research Program, under Contract No. DE-AC02-06CH11357.

References

[1] P. J. Gleckler, P. J. Durack, R. J. Stouffer, G. C. Johnson, and C. E. Forest, "Industrial-era global ocean heat uptake doubles in recent decades," *Nature Climate Change*, vol. 6, no. 4, pp. 394–398, 2016.

[2] A. H. Baker, H. Xu, J. M. Dennis, M. N. Levy, D. Nychka, S. A. Mickelson, J. Edwards, M. Vertenstein, and A. Wegener, "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*, 2014, pp. 203–214.

[3] A. S. Almgren, J. B. Bell, M. J. Lijewski, Z. Lukić, and E. V. Andel, "Nyx: A MASSIVELY PARALLEL AMR CODE FOR COMPUTATIONAL COSMOLOGY," *The Astrophysical Journal*, vol. 765, no. 1, p. 39, feb 2013. [Online]. Available: https://doi.org/10.1088/0004-637x/765/1/39

[4] ORNL, "Summit supercomputer," https://www.olcf.ornl.gov/summit/, 2022.

[5] "Ornl data storage and transfer," https://docs.olcf.ornl.gov/data/index.html, 2022.

[6] W. Xia, H. Jiang, D. Feng, F. Douglis, P. Shilane, Y. Hua, M. Fu, Y. Zhang, and Y. Zhou, "A comprehensive study of the past, present, and future of data deduplication," *Proceedings of the IEEE*, vol. 104, no. 9, pp. 1681–1710, 2016.

[7] Zlib, https://www.zlib.net/, online.

[8] Zstandard, http://facebook.github.io/zstd/, 2018, online.

[9] M. Burtscher and P. Ratanaworabhan, "Fpc: A high-speed compressor for double-precision floating-point data," *IEEE transactions on computers*, vol. 58, no. 1, pp. 18–31, 2008.

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

[11] D. Tao, S. Di, Z. Chen, and F. Cappello, "Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization," in *2017 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2017, pp. 1129–1139.

[12] K. Zhao, S. Di, M. Dmitriev, T.-L. D. Tonellot, Z. Chen, and F. Cappello, "Optimizing error-bounded lossy compression for scientific data by dynamic spline interpolation," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 1643–1654.

[13] P. Lindstrom, "Fixed-rate compressed floating-point arrays," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2674–2683, 2014.

[14] X. Liang, B. Whitney, J. Chen, L. Wan, Q. Liu, D. Tao, J. Kress, D. Pugmire, M. Wolf, N. Podhorszki *et al.*, "Mgard+: Optimizing multilevel methods for error-bounded scientific data reduction," *IEEE Transactions on Computers*, vol. 71, no. 7, pp. 1522–1536, 2021.

[15] SZ2.1, https://github.com/szcompressor/SZ, 2022.

[16] J. Liu, S. Di, K. Zhao, S. Jin, D. Tao, X. Liang, Z. Chen, and F. Cappello, "Exploring autoencoder-based error-bounded compression for scientific data," in *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, 2021, pp. 294–306.

[17] S. Jin, S. Di, X. Liang, J. Tian, D. Tao, and F. Cappello, "Deepsz: A novel framework to compress deep neural networks by using error-bounded lossy compression," in *Proceedings of the 28th international symposium on high-performance parallel and distributed computing*, 2019, pp. 159–170.

[18] N. Sasaki, K. Sato, T. Endo, and S. Matsuoka, "Exploration of lossy compression for application-level checkpoint/restart," in *2015 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2015, pp. 914–922.

[19] "Ecp exaalt project," https://www.exascaleproject.org/research-project/exaalt/.

[20] "Lammps molecular dynamics simulator," https://www.lammps.org/.

[21] Nyx, https://portal.nersc.gov/project/nyx/highz/512/, 2013, online.

[22] S. Habib, V. Morozov, N. Frontiere, H. Finkel, A. Pope, K. Heitmann, K. Kumaran, V. Vishwanath, T. Peterka, J. Insley *et al.*, "HACC: extreme scaling and performance across diverse architectures," *Communications of the ACM*, vol. 60, no. 1, pp. 97–104, 2016.

[23] M. Smelyanskiy, N. P. Sawaya, and A. Aspuru-Guzik, "qhipster: the quantum high performance software testing environment," *arXiv preprint arXiv:1601.07195*, 2016.

[24] X.-C. Wu, S. Di, E. M. Dasgupta, F. Cappello, H. Finkel, Y. Alexeev, and F. T. Chong, "Full-state quantum circuit simulation by using data compression," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3295500.3356155

[25] S. Jin, C. Zhang, X. Jiang, Y. Feng, H. Guan, G. Li, S. L. Song, and D. Tao, "Comet: A novel memory-efficient deep learning training framework by using error-bounded lossy compression," 2021.

[26] K. Zhao, S. Di, X. Lian, S. Li, D. Tao, J. Bessac, Z. Chen, and F. Cappello, "Sdrbench: Scientific data reduction benchmark for lossy compressors," in *2020 IEEE international conference on big data (Big Data)*. IEEE, 2020, pp. 2716–2724.

[27] Nyx Cosmology Simulation Data, https://portal.nersc.gov/project/nyx/, online.

[28] S. Jin, S. Di, J. Tian, S. Byna, D. Tao, and F. Cappello, "Improving prediction-based lossy compression dramatically via ratio-quality modeling," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2022, pp. 2494–2507.

[29] T. Lu, Q. Liu, X. He, H. Luo, E. Suchyta, J. Choi, N. Podhorszki, S. Klasky, M. Wolf, T. Liu *et al.*, "Understanding and modeling lossy compression schemes on HPC scientific data," in *2018 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2018, pp. 348–357.

[30] J. Wang, T. Liu, Q. Liu, X. He, H. Luo, and W. He, "Compression ratio modeling and estimation across error bounds for lossy compression," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 7, pp. 1621–1635, 2020.

[31] X. Liang, S. Di, S. Li, D. Tao, B. Nicolae, Z. Chen, and F. Cappello, "Significantly improving lossy compression quality based on an optimized hybrid prediction model," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–26.

[32] K. Zhao, S. Di, X. Liang, S. Li, D. Tao, Z. Chen, and F. Cappello, "Significantly improving lossy compression for hpc datasets with second-order prediction and parameter optimization," in *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, 2020, pp. 89–100.

[33] J. Liu, S. Di, K. Zhao, S. Jin, D. Tao, X. Liang, Z. Chen, and F. Cappello, "Exploring autoencoder-based error-bounded compression for scientific data," in *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2021, pp. 294–306.

[34] X. Yu, S. Di, K. Zhao, J. Tian, D. Tao, X. Liang, and F. Cappello, "Ultrafast error-bounded lossy compression for scientific datasets," in *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*, 2022, pp. 159–171.

[35] M. H. Rahman, S. Di, K. Zhao, R. Underwood, G. Li, and F. Cappello, "A feature-driven fixed-ratio lossy compression framework for real-world scientific datasets," in *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 2023, pp. 1461–1474.

[36] J. E. Kay and 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.

[37] Hurricane ISABEL Simulation Data, http://vis.computer.org/vis2004contest/data.html, 2004, online.

[38] "Miranda turbulence simulation," https://wci.llnl.gov/simulation/computer-codes/miranda.

[39] A. S. Almgren, J. B. Bell, M. J. Lijewski, Z. Lukić, and E. Van Andel, "Nyx: A massively parallel amr code for computational cosmology," *The Astrophysical Journal*, vol. 765, no. 1, p. 39, 2013.

[40] I. Sobel, G. Feldman *et al.*, "A 3x3 isotropic gradient operator for image processing," *a talk at the Stanford Artificial Project in*, pp. 271–272, 1968.

[41] J. Canny, "A computational approach to edge detection," *IEEE Transactions on pattern analysis and machine intelligence*, no. 6, pp. 679–698, 1986.

[42] V. A. Kotkar and S. S. Gharde, "Review of various image contrast enhancement techniques," *International journal of innovative research in Science, Engineering and Technology*, vol. 2, no. 7, 2013.

[43] B. R. Naidu, P. L. Rao, M. P. Babu, and K. Bhavani, "Efficient case study for image edge gradient based detectors-sobel, robert cross, prewitt and canny," *International Journal of Electronics Communication and Computer Engineering*, vol. 3, no. 3, pp. 561–571, 2012.

[44] B. Kaur and A. Garg, "Mathematical morphological edge detection for remote sensing images," in *2011 3rd International Conference on Electronics Computer Technology*, vol. 5. IEEE, 2011, pp. 324–327.

[45] H. Spontón and J. Cardelino, "A review of classic edge detectors," *Image Processing On Line*, vol. 5, pp. 90–123, 2015.

[46] D. Tao, S. Di, H. Guo, Z. Chen, and F. Cappello, “Z-checker: A framework for assessing lossy compression of scientific data,” *The International Journal of High Performance Computing Applications*, vol. 33, no. 2, pp. 285–303, 2019.

[47] R. Underwood, J. Bessac, S. Di, and F. Cappello, “Understanding the effects of modern compressors on the community earth science model,” in *2022 IEEE/ACM 8th International Workshop on Data Analysis and Reduction for Big Scientific Data (DRBSD)*. IEEE, 2022, pp. 1–10.

[48] J. Liu, S. Di, K. Zhao, X. Liang, Z. Chen, and F. Cappello, “Dynamic quality metric oriented error bounded lossy compression for scientific datasets,” in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2022, pp. 1–15.

[49] R. R. Underwood, S. Di, S. Jin, M. H. Rahman, A. Khan, and F. Cappello, “Libpressio-predict: Flexible and fast infrastructure for inferring compression performance,” in *Proceedings of the SC’23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, 2023, pp. 272–280.

[50] M. Ainsworth, O. Tugluk, B. Whitney, and S. Klasky, “Multilevel techniques for compression and reduction of scientific data—the univariate case,” *Computing and Visualization in Science*, vol. 19, no. 5-6, pp. 65–76, 2018.

[51] ——, “Multilevel techniques for compression and reduction of scientific data—the multivariate case,” *SIAM Journal on Scientific Computing*, vol. 41, no. 2, pp. A1278–A1303, 2019.

[52] C. S. Zender, “Bit Grooming: statistically accurate precision-preserving quantization with compression, evaluated in the netCDF Operators (NCO, v4.4.8+),” *Geoscientific Model Development*, vol. 9, no. 9, pp. 3199–3211, Sep. 2016.

[53] X. Delaunay, A. Courtois, and F. Gouillon, “Evaluation of lossless and lossy algorithms for the compression of scientific datasets in netcdf-4 or hdf5 files,” *Geoscientific Model Development*, vol. 12, pp. 4099–4113, 09 2019.

[54] P. Ganesan and G. Sajiv, “A comprehensive study of edge detection for image processing applications,” in *2017 international conference on innovations in information, embedded and communication systems (ICIIECS)*. IEEE, 2017, pp. 1–6.

[55] L. Breiman, “Random forests,” *Machine learning*, vol. 45, pp. 5–32, 2001.

[56] H. Drucker, C. J. Burges, L. Kaufman, A. Smola, and V. Vapnik, “Support vector regression machines,” *Advances in neural information processing systems*, vol. 9, 1996.

[57] “Data storage and transfers at oak ridge leadership computing facility (olcf),” https://docs.olcf.ornl.gov/data/index.html#data-storage-and-transfers.

[58] “Anl theta,” https://www.alcf.anl.gov/support-center/theta-and-thetagpu.