SZ3: A Modular Framework for Composing Prediction-Based Error-Bounded Lossy Compressors

Xin Liang*, Kai Zhao*, Sheng Di, *Senior Member, IEEE,* Sihuan Li, Robert Underwood, Ali M. Gok, Jiannan Tian, Junjing Deng, Jon C. Calhoun, *Senior Member, IEEE,* Dingwen Tao, Zizhong Chen, *Senior Member, IEEE,* and Franck Cappello, *Fellow, IEEE*

Abstract—Today's scientific simulations require a significant reduction of data volume because of extremely large amounts of data they produce and the limited I/O bandwidth and storage space. Error-bounded lossy compression has been considered one of the most effective solutions to the above problem. In practice, however, the best-fit compression method often needs to be customized or optimized in particular because of diverse characteristics in different datasets and various user requirements on the compression quality and performance. In this paper, we address this issue with a novel modular, composable compression framework named SZ3. Our contributions are four-folds. (1) We develop SZ3 which features an innovative modular abstraction for the prediction-based compression framework, such that compression modules can be plugged in easily to create new compressors based on characteristics of data and user requirements. (2) We create a new compression pipeline by SZ3 for GAMESS data, which significantly improves the compression ratios over state-of-the-art compressors. (3) We develop an adaptive compression pipeline by SZ3 for APS data with minimal efforts, which leads to the best rate-distortion among all existing error-bounded lossy compressors for any bit-rate. (4) We compare the sustainability of SZ3 with leading error-bounded prediction-based compressors, and then demonstrate the necessity of diverse pipelines by integrating and evaluating several compression pipelines on diverse scientific datasets from multiple disciplines. Experiments show that SZ3 incurs very limited overhead in compressor integration and our customized compression pipelines lead to up to 20% improvement in compression ratios under the same data distortion, when compared with the best existing approach.

Index Terms—Big Data, Error-Bounded Lossy Compression, Data Reduction, Large-Scale Scientific Simulation

1 Introduction

DATA reduction is becoming increasingly important to scientific research because of the large amount of data produced by simulations running on exascale computing systems and experiments conducted on advanced instruments. For instance, recent climate research, which performs climate simulation in 1 km×1 km resolution, generates 260 TB of floating-point data every 16 seconds [1]. When the generated data are dumped into parallel file systems or secondary storage systems to ensure long-term access, the limited storage capacity and/or I/O bandwidth will impose great challenges. While scientists aim to significantly reduce the size of their data to mitigate this problem, they are also concerned about the quality of data reduction. General data reduction approaches, including traditional wavelet-based methods [2], [3] and emerging neural-network-based methods [4], [5] widely used in the image processing community, may lead to loss of important scientific insights as they do not enforce quantifiable error bounds on reconstructed data.

Over the past decade, error-bounded lossy compression [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18] has been proposed and employed to reduce scientific data while controlling the distortion. Depending on how the original data are decorrelated, existing compressors can be classified into prediction-based and transform-based. These compressors all allow users to specify an error bound during compression and ensure that the error between original and decompressed data is strictly lower than the bound. In this paper we focus mainly on prediction-based approaches because transformed-based approaches can be formulated to prediction-based ones by using the corresponding transforms as predictors (at the cost of certain speed degrada-

- *X. Liang is with the Department of Computer Science, University of Kentucky, Lexington, KY 40506. E-mail: xliang@cs.uky.edu*
- *K. Zhao is with the Department of Computer Science, University of Alabama at Birmingham, Birmingham, AL 35294. E-mail: kzhao@uab.edu*
- *S. Li is with Facebook Inc., Menlo Park, CA 94025. E-mail: sli049@ucr.edu*
- *S. Di, R. Underwood, and F. Cappello are with the Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL 60439. E-mail: sdi1@anl.gov, runderwood@anl.gov, cappello@mcs.anl.gov*
- *J. Deng is with the Advanced Photon Source, Argonne National Laboratory, Lemont, IL 60439. E-mail: junjingdeng@anl.gov*
- *A. M. Gok is with Cerebras Systems, Sunnyvale, CA 94085. E-mail: ali.gok@cerebras.net*
- *J. Tian and D. Tao are with the Department of Computer Science, Indiana University, Bloomington, IN 47405. E-mail: {jiannan.tian, dingwen.tao}@wsu.edu*
- *J.C. Calhoun is with the Holcombe Department of Electrical and Computer Engineering, Clemson University, Clemson, SC 29634. E-mail: jonccal@clemson.edu*
- *Z. Chen is with the Department of Computer Science & Engineering, University of California, Riverside, Riverside, CA 92521. E-mail: chen@cs.ucr.edu*
- ** The two authors contributed equally to this paper.*
- *Sheng Di is the corresponding author.*

tion), as suggested by prior works [19].

Although existing prediction-based approaches such as SZ [6], [20], [21] are general and can be applied to various scenarios, they may not lead to the best quality and performance given a specific dataset or error bound requirement. The best-fit compression method is never universal, which is true even for the same dataset because the compression efficiency would be affected by the required error bounds as well. For instance, SZ-1.4 [21] with a Lorenzo predictor shows very good compression ratios with low error bounds, but it suffers from low quality and artifacts with high error bounds, where approaches with a regression-based predictor [6] or an interpolation-based predictor [8] have been proved to be much more efficient. Likewise, data generated by the GAMESS quantum chemistry package [22] exhibits periodic scaled patterns, where a pattern-based predictor demonstrates obvious improvements in both compression speed and ratios [23]. Thus, a loosely coupled compression framework that allows for customization of the prediction-based error-bounded lossy compression model is critical to optimizing the compression quality and performance for users in practice.

Developing a modular error-controlled compression framework that can adapt to diverse scientific datasets is very challenging. First, existing compressors have diverse designs that could be hard to unify, thus designing such a framework requires in-depth understanding and analysis of both their algorithms and the underlying implementations. Second, stages in the unified compression framework should provide different variations to allow for generalization to different compression methods, which poses challenges on designing the corresponding interfaces. Third, it is very difficult to retain the performance while providing generality and modularity, because optimizations usually come from specialities. Last but not the least, composing efficient compressors for applications is challenging, because existing compressors have already been well-optimized, and some of them are specifically designed (e.g., SZ-Pastri for GAMESS data).

In this paper, we present a modular and composable framework—SZ3—which can be used to easily create new error-bounded lossy compressors on demand. SZ3 features a modular abstraction for the prediction-based compression pipelines such that modules can be developed and adopted independently. Specifically, users can customize any stages in the compression pipeline, including preprocessing, prediction, quantization, encoding, and lossless compression, via carefully designed modules. Based on these customized modules, SZ3 allows users to compose their own compressors (or compression pipelines) to adapt to diverse data characteristics and requirements, thus achieving high compression quality and performance with minimal effort. Such a composable design is able to provide a variety of useful supports, including point-wise relative error bounds (logarithmic transform-based preprocessor [24]), feature-preserving compression (element-wise quantizer [25]), and speed-ratio tradeoffs (module bypass). Although designed for data in Cartesian grids, SZ3 can also work with data in unstructured grids by applying a linearization which rearranges data to a one-dimensional array.

We summarize our contributions as follows.

- We design and develop the first general prediction-based error-bounded lossy compression framework — SZ3. With modularity and adaptivity in mind, SZ3 allows for easy creation and customization of various prediction-based error-bounded lossy compressors that can obtain high compression quality in diverse use cases. This is critical for the developers and users of scientific data compressors, because these compressors are expected to handle scientific data with diverse characteristics and meet varying user requirements in practice.
- We develop a new compressor using SZ3 for data generated from GAMESS quantum chemistry package. Specifically, we propose and implement a quantizer which significantly reduces the storage overhead of unpreditable data. Composed by this quantizer and a newly added lossless compression stage, the developed compressor leads to up to 18% improvement in compression ratios compared with the best existing approach.
- We develop an efficient compressor using SZ3 for data collected from Advanced Photon Source instruments. Noticing that the best-fit compression pipelines vary with the target error requirements, we compose an adaptive compression pipeline that leads to the best rate-distortion under any bit rate.
- We implement SZ3 in a way that delivers high usability with low overhead. By leveraging advanced template programming and compile-time polymorphism, SZ3 significantly shrinks the codebase of SZ2 while improving the sustainability and functionality. We also propose and develop a pipeline mapping technique to map the composed pipeline to optimized implementations. The effectiveness of SZ3 is validated by extensive experiments on six diverse scientific datasets, which demonstrate that compressors composed by SZ3 achieve comparable compression ratios and throughput compared to state of the arts.

The target end users of SZ3 include both compression experts and non-experts. For the former, SZ3 allows for the reuse of compression modules (e.g., Huffman encoder) and utility functions (e.g., the multi-dimensional iterator) that are tricky to implement, such that compression experts only need to implement a subset of modules to develop new compressors. For instance, we develop the new compressor for GAMESS data by implementing only a new quantizer, with the majority of the implementation reusing existing codes. For the latter, SZ3 could be either used as references to multiple built-in compression pipelines that are widely used, or paired with high level tools such as OptZConfig [26] to automatically select or suggest the appropriate sets of module instances. As such, we envision SZ3 as a very useful and practical tool with a wide range of users.

The rest of the paper is organized as follows. In Section 2 we discuss related work. In Section 3 we present the design and modules of SZ3 framework. In Section 4 and Section 5 we describe how we leverage the proposed framework to create efficient compressors for GAMESS and APS data in details. In Section 6 we present the comparison on sustainability and evaluation for diverse pipelines. In Section 7 we conclude with a vision of future work.

2 Related Work

With more powerful high-performance computing (HPC) systems and high-resolution instruments, the volume and generation speed of scientific data have been experiencing an unprecedented increase in recent years, causing problems in data storage, transmission, and analysis. Compared with the fast evolution of computing resources, the I/O systems are heavily underdeveloped, remaining a bottleneck in most scenarios. Data compression is regarded as a direct way to mitigate such a bottleneck, and many approaches have been presented in the literature to address this issue.

Lossless compressors [27], [28], [29], [30], [31] ensure that no information is lost during the compression. Despite their success in many fields, lossless compressors suffer from low compression ratios on floating-point scientific data due to the almost randomly distributed mantissas. Previous work [32] has shown that state-of-the-art lossless compressors can lead to a compression ratio of only 2 when directly applied to most floating-point scientific datasets, whereas scientific applications usually require over $10\times$ reduction on their data [33].

Lossy compressors [2], [3], [4], [5], [34], [35] offer the flexibility to trade off data quality for high compression ratios, but they may result in a higher distortion than users' expectation. The unbounded distortion may result in unexpected behaviors in post hoc data analytics and even false discoveries, leaving risks in trusting the analysis results on the decompressed data.

In comparison with traditional lossy compression, error-bounded lossy compression has been rapidly developed to fill the gap by reducing the size of scientific data while guaranteeing quantifiable error bounds. Prediction-based and transform-based models are the most popular models for designing error-bounded lossy compressors. One of the most well-known transform-based error-bounded lossy compressors is ZFP [13], which decorrelates the data using a near-orthogonal transform and encodes the transformed coefficients using embedded encoding. MGARD [15], [16] is another compressor relying on the transform-based model. It leverages wavelet theories and L^2 projection for data decorrelation, followed by linear-scaling quantization, variable-length encoding, and lossless compression. It is further used to compress the data while preserving certain derived quantities [17], [36], [37]. In a recent compressor [38], High-Order Singular Value Decomposition (HOSVD) is leveraged to improve the compression ratios for low accuracy cases, but it suffers from orders of higher computational overhead due to the expensive decomposition.

According to recent studies [39], SZ [6], [20], [21] is regarded as one of the leading prediction-based lossy compressor in the scientific computing community. SZ follows a 4-step pipeline to perform the compression, namely data prediction, quantization, Huffman encoding, and lossless compression. Significant efforts have been made to enable new features or functionalities based on this pipeline. For instance, in [24], a logarithmic transform was used in a preprocessing step to change a pointwise-relative-error-bound compression problem to an absolute-error-bound compression problem, which is then solved by the SZ compression pipeline. In [25], the authors derived the element-wise error bounds based on how critical points are extracted, and they leveraged the SZ compression pipeline along with element-wise quantization to ensure that those critical points are preserved in the decompressed data. In [23], the authors adjusted the pipeline by using a pattern-based predictor to better exploit the correlation in data and a predefined fixed Huffman tree for faster encoding. Attempts were also made to use the near-orthogonal transform in ZFP as a predictor in the pipeline [19]. All the above works, however, are developed within a tightly-coupled design, so that the compression pipelines cannot be adjusted on demand, which thus cannot adapt to user's diverse requirements or different use-cases in turn. By contrast, the proposed SZ3 framework offers a breakthrough, flexible, modular framework, which can be leveraged to adapt to diverse use-cases very efficiently. Furthermore, SZ3 could serve as a building block and benefit the development of new compressors with more complex workflows including online selection [40], parameter tuning [7], and configuration search [26].

Although many efforts have been spent on abstracting lossy compression, most of them are focused on enabling an adaptive selection of existing compressors. For instance, SCIL [41] attempts to abstract across compressors and acts as a metacompressor that provides backends to various existing algorithms. LibPressio [42] provides a common API for different compressors to allow for easy integration of lossy compression in an extensible fashion. Instead, SZ3 separates and abstracts stages in the prediction-based compression model, allowing for easy creation of new compressors in fine granularity rather than selection of existing ones. To the best of our knowledge, this is the first attempt to build a generic framework that allows users to easily customize their own compressors based on their actual needs.

3 SZ3: A Modular Compression Framework

In this section we introduce the design and implementation of SZ3. With modularity in mind, SZ3 enables easy customization of prediction-based compression pipelines with minimal overhead.

3.1 Design overview

Figure 1 illustrates the design overview of SZ3. The compression process is abstracted into five stages (displayed as the dotted boxes), each of which serves as an individual module. Orange boxes depict the key functionalities of each module and green boxes illustrate several corresponding instances. A compressor is realized by identifying a compression pipeline which is composed by instances from each module. This figure demonstrates how five leading compressors designed for different purposes, namely FPZIP [14], SZ1.4 [21], SZ2 [6], SZ-Pastri [23], and cpSZ [25], are composed using this abstraction (see the solid lines), which shows the generality of the abstraction. For instance, the FPZIP compression pipeline bypasses the precessor and leverages Lorenzo predictor for data decorrelation, followed by residual encoding to ensure error control and arithmetic

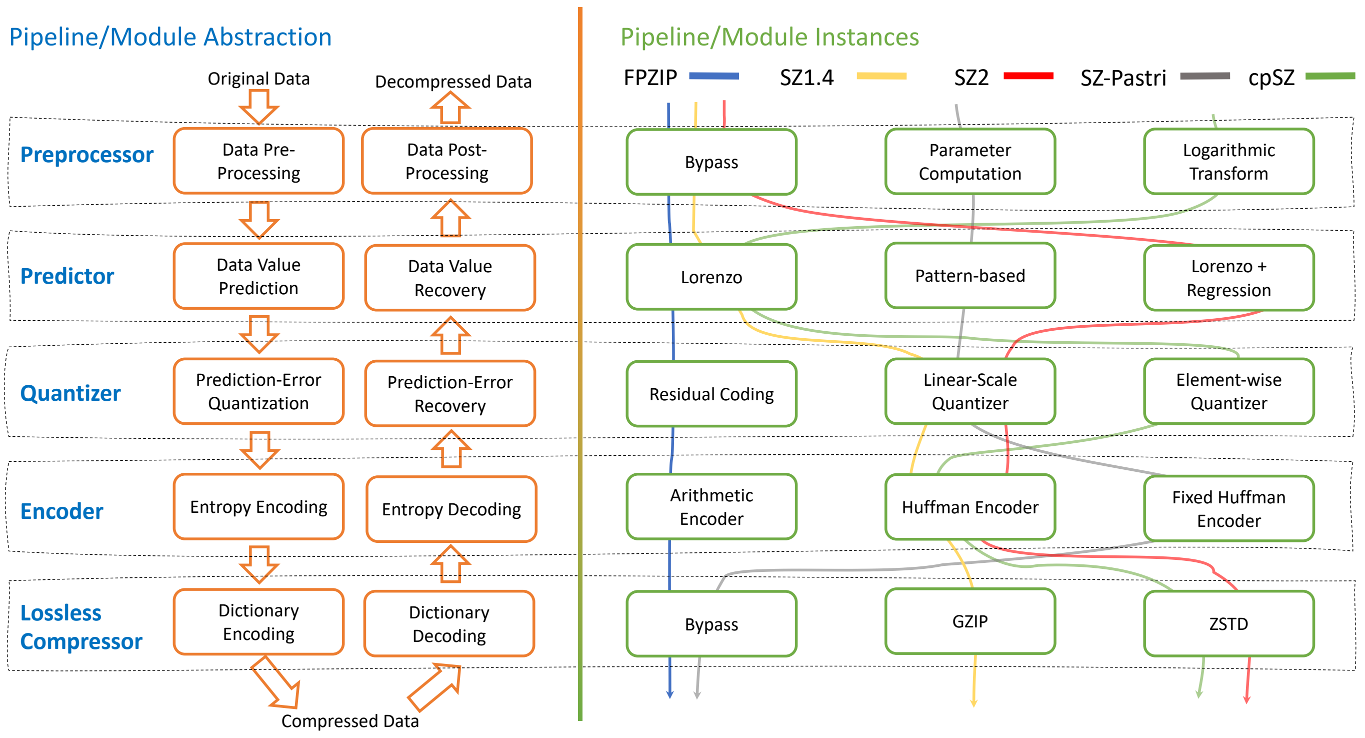


Fig. 1. SZ3 design overview: left part of the figure shows the abstraction and key functionalities of prediction-based compression pipeline with SZ3 modules; right part of the figure displays common instances of these modules and how five leading compressors are composed by these instances.

encoding for size reduction. In the following text we will detail the modular design in SZ3, along with example instances of the modules.

3.2 Modularity

In this section we discuss the five modules in SZ3, namely preprocessor, predictor, quantizer, encoder, and lossless compressor, with module instances that have proven to be effective for scientific datasets. Developers can write their own module instances and plug them in the compression pipeline to design prediction-based error-bounded lossy compression for their dataset. Due to space limitation, we present only the most important functions and several representative instances for each module. Detailed interfaces for each module are listed in Appendix A.

Preprocessor (see Appendix A.1): The preprocessor is used to process the input dataset for achieving high efficiency or diverse requirements before performing the actual compression. The key function in the preprocessor, namely `preprocess`, takes in original data and compression configuration as input, and then transforms the data in an in-place fashion and change the compression configuration accordingly. If users want to keep original data while the preprocessor needs to alter the data, a separate buffer is required to perform the preprocessing. Based on the actual design, the `postprocess` function either reverses the preprocessing procedure or is omitted.

Instances: A typical preprocessor for error-bounded lossy compressors is the logarithmic transform used to enable point-wise relative error bounds [24], where data are transformed to the logarithmic domain and compressed with an absolute error bound transformed from the point-wise relative one. Besides, SZ-Pastri [23] may require a preprocessing step to identify the proper parameters, such as block size and pattern size if not provided, for the pattern-based predictor. In Section 5, we further leverage a preprocessor to alter the layout of data for better compression ratio. This is based on our observations that some 3D datasets will have a better compression ratio when treated as a 2D or 1D dataset (as will be detailed later).

Predictor (see Appendix A.2): Predictors are the key components of prediction-based compressors, which perform value prediction based on diverse patterns for data decorrelation. There are two important functions in the predictor interface, namely `predict` and `save/load`. The `predict` function outputs the predicted value based on the characteristics of the underlying predictor using the multidimensional iterator (to be detailed in Section 6.1). Necessary information about the predictor, for instance the coefficients of the regression predictor [6], [7], will be recorded in the `save` function. During decompression, `load` function will be invoked to reconstruct the predictor.

Instances: Lorenzo predictor [43] and its high order variations [21], which perform multidimensional prediction for each data point based on its neighbor data points, are classic and powerful prediction methods used in lossy compressors such as SZ [21] and FPZIP [14]. In [6], a regression-based predictor is proposed to construct a hyperplane and uses points on the hyperplane as predicted values, which significantly improves the prediction efficiency when user-specified error bound is high. We further implement a composite predictor instance inherited from this interface, which may consist of multiple predictors using different prediction algorithms. This requires an error estimation function for each predictor, which will be used to determine the best-fit predictor for a given data chunk. The statistical approach in [6] and [18] is generalized as the estimation criterion in SZ3. With the composite predictor, multialgorithm designs with more than one predictors can be implemented very easily.

Quantizer (see Appendix A.3): The quantizer is used to approximate prediction errors generated by the predictors with a smaller countable set to reduce their entropy while respecting the error bound. As the only module that introduces errors in the compression pipeline, quantizer determines how the final errors in the decompressed data are controlled. The `quantize` function is the most important function in a quantizer, where the prediction error is quantized based on the original data value and its predicted value from the predictor. During decompression, the decompressed data value is computed by the `recover` function, which reverses the steps in the `quantize` function. The quantizer module is also responsible for encoding/decoding the unpredictable data, i.e., data fall out of the countable set. This is realized in the `save/load` function.

Instances: Linear-scaling quantizer [21] is a widely used quantizer to enable absolute error control in lossy compression. In particular, this quantizer constructs a set of equal-sized consecutive bins each with twice the error bound in length. Then, the prediction error will be translated into the index of the bin containing it. Prediction errors that fall out of range are regarded as unpredictable and will be encoded and stored separately. Besides, log-scale quantizer [44] is used to adjust the size of bins for a more centralized error distribution and element-wise quantizer [25] is used to provide fine-granularity error control for each data point.

Encoder (see Appendix A.4): Encoder is a lossless scheme to reduce the storage of integer indices (or symbols) generated by quantizers. The encoder module involves two essential functions—`encode` and `save/load`. The `encode` function transforms the quantized integers from the quantizer to compressed binary formats; similar to other modules, the encoder module has a `decode` function which performs the reverse process during decompression. This module also has `save/load` functions for storing/recovering metadata such as the Huffman tree.

Instances: Huffman encoder [45] is a classic variable-length encoding algorithm that uses fewer bits to represent more common symbols. This encoder first constructs a Huffman tree based on the frequency of input data using a greedy algorithm, generates codebook according to the tree, and then compress the data using the codebook. The fixed Huffman encoder used in SZ-Pastri [23] is a variation of the Huffman encoder, which uses a predefined Huffman tree instead of constructing one on the fly to eliminate the cost for both construction and storage of the tree. Arithmetic encoder is another type of encoder widely used in data compression, which represents current information as range and encodes the entire data into a single number.

Lossless Compressor (see Appendix A.5): Lossless compressors are used to further shrink the size of compressed binary formats produced by the encoders, because the entropy-based encoders may overlook repeated patterns in the data thus lead to suboptimal compression ratios. The lossless compressor module in SZ3 acts mainly as a proxy of state-of-the-art lossless compression libraries. This module invokes external libraries to compress the output from the encoder module with `compress` and `decompress` interfaces.

Instances: We provide portable interfaces in SZ3 to integrate with state-of-the-art lossless compressors including ZSTD [28], GZIP [27], and BLOSC [31]. Because lossless compressor is a standlone module attached to the previous stages, it would be fairly easy to include and integrate new lossless compression routines as well.

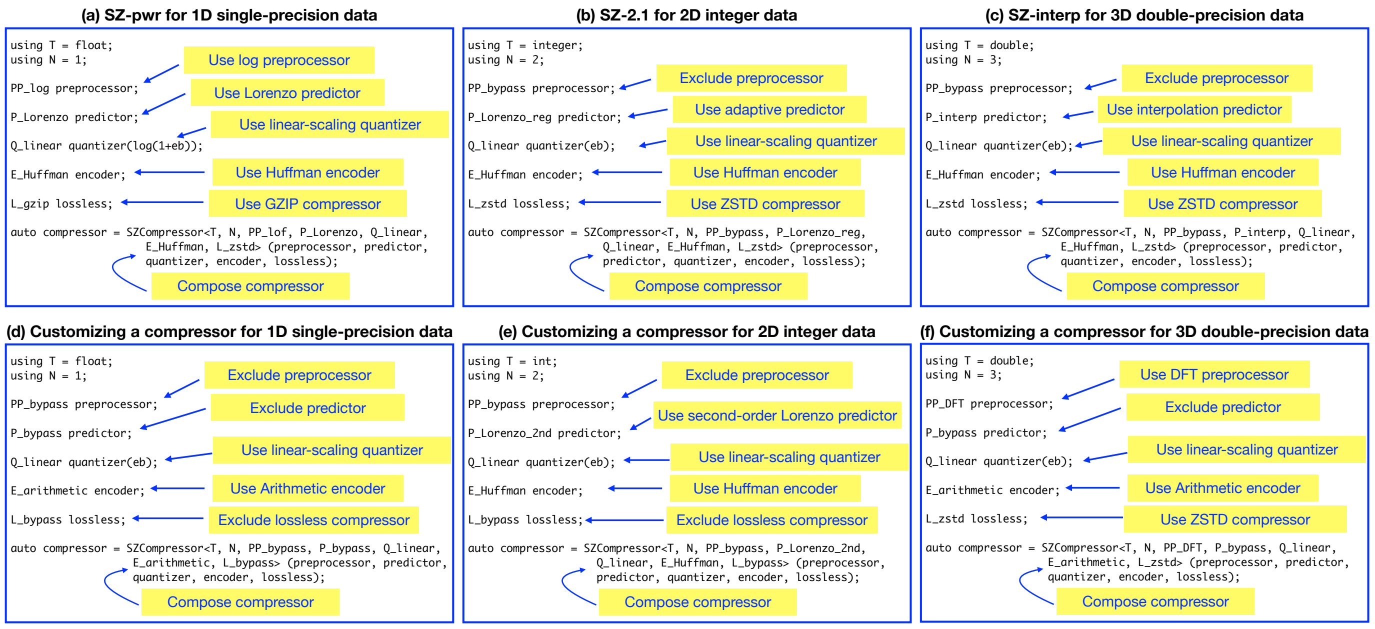
3.3 Compression pipeline composition

In SZ3, a compression pipeline can be composed by identifying the instances of modules and putting them together. Algorithm 1 shows how the algorithm of a general error-bounded lossy compressor using the given preprocessor, predictor, quantizer, encoder, and lossless compressor. In addition, SZ3 employs compile-time polymorphism (see Section 6.1) such that users can switch the instances without bothering to modify the compression functions. This makes SZ3 highly adaptive to diverse use cases, with significantly reduced efforts on compressor development.

Algorithm 1 A GENERAL COMPRESSOR IN SZ3

```
Input: input data $d$ of size $n$, compression configuration $conf$
Output: compressed data $cc$
1: $preprocessor$.process($d, conf$) /*perform preprocessing*/
2: for $i = 1 \rightarrow n$ do
3:     $p \leftarrow predictor$.predict($d[i]$) /*perform prediction*/
4:     $q[i] \leftarrow quantizer$.quantize($d[i], p$) /*perform quantization*/
5: end for
6: $c \leftarrow$ allocate_memory()
7: $predictor$.save($c$) /*save predictor*/
8: $quantizer$.save($c$) /*save quantizer*/
9: $encoder$.encode($q, c$) /*perform encoding*/
10: $encoder$.save($c$) /*save encoder*/
11: $cc \leftarrow lossless\_compressor$.compress($c$) /*perform lossless compression*/
12: return $cc$
```

With the algorithm of the general compressor, compression developers or users can easily customize their own compressor by setting each component in the pipeline. We illustrate how to leverage SZ3 to implement three existing compressors, namely SZ-pwr [24], SZ-2.1 [6], and SZ-interp [8] in Figure 2(a)-(c). We further show how SZ3 can be used to implement customized compressors with different specialities. Figure 2(d) composes a customized compressor with only linear-scaling quantizer and arithmetic encoder. By skipping the prediction and lossless compression stages that are computationally expensive when the target bitrate is medium or high, this compressor can yield higher compression throughput at the cost of compression ratios. The compressor in Figure 2(e) leverages a second-order Lorenzo predictor [21] which significantly improves the prediction accuracy (while introducing more computation) when the target bitrate is relatively high. It also excludes the lossless compressor which provides minor size reduction while bringing high computational overhead. Thus, it generally leads to better rate-distortion with lower compression throughput. A transform-based compressor is composed in Figure 2(f) to demonstrate the generality of our framework. This kind of compressors is widely used in image processing



(a) SZ-pwr for 1D single-precision data

```
using T = float;
using N = 1;

PP_log preprocessor;

P_Lorenzo predictor;

Q_linear quantizer(log(1+eb));

E_Huffman encoder;

L_gzip lossless;

auto compressor = SZCompressor<T, N, PP_lof, P_Lorenzo, Q_linear,
                      E_Huffman, L_zstd> (preprocessor, predictor,
                      quantizer, encoder, lossless);
```

(b) SZ-2.1 for 2D integer data

```
using T = integer;
using N = 2;

PP_bypass preprocessor;

P_Lorenzo_reg predictor;

Q_linear quantizer(eb);

E_Huffman encoder;

L_zstd lossless;

auto compressor = SZCompressor<T, N, PP_bypass, P_Lorenzo_reg,
                      Q_linear, E_Huffman, L_zstd> (preprocessor,
                      predictor, quantizer, encoder, lossless);
```

(c) SZ-interp for 3D double-precision data

```
using T = double;
using N = 3;

PP_bypass preprocessor;

P_interp predictor;

Q_linear quantizer(eb);

E_Huffman encoder;

L_zstd lossless;

auto compressor = SZCompressor<T, N, PP_bypass, P_interp, Q_linear,
                      E_Huffman, L_zstd> (preprocessor, predictor,
                      quantizer, encoder, lossless);
```

(d) Customizing a compressor for 1D single-precision data

```
using T = float;
using N = 1;

PP_bypass preprocessor;

P_bypass predictor;

Q_linear quantizer(eb);

E_arithmetic encoder;

L_bypass lossless;

auto compressor = SZCompressor<T, N, PP_bypass, P_bypass, Q_linear,
                    E_arithmetic, L_bypass> (preprocessor, predictor,
                    quantizer, encoder, lossless);
```

(e) Customizing a compressor for 2D integer data

```
using T = int;
using N = 2;

PP_bypass preprocessor;

P_Lorenzo_2nd predictor;

Q_linear quantizer(eb);

E_Huffman encoder;

L_bypass lossless;

auto compressor = SZCompressor<T, N, PP_bypass, P_Lorenzo_2nd,
                     Q_linear, E_Huffman, L_bypass> (preprocessor,
                     predictor, quantizer, encoder, lossless);
```

Exclude preprocessor
Use second-order Lorenzo predictor
Use linear-scaling quantizer
Use Huffman encoder
Exclude lossless compressor
Compose compressor

(f) Customizing a compressor for 3D double-precision data

```
using T = double;
using N = 3;

PP_DFT preprocessor;

P_bypass predictor;

Q_linear quantizer(eb);

E_arithmetic encoder;

L_zstd lossless;

auto compressor = SZCompressor<T, N, PP_DFT, P_bypass, Q_linear,
                    E_arithmetic, L_zstd> (preprocessor, predictor,
                    quantizer, encoder, lossless);
```

Fig. 2. Examples of using SZ3 to compose existing or customized compressors.

and some specific scientific domains, as it preserves the transformed coefficients which are important in certain post hoc data analysis such as power spectrum. As shown in these examples, users are allowed to reuse or replace any stages in SZ3 by simply defining the desired components and using them while composing the compressor.

4 Developing an Efficient Compressor for GAMESS Data using SZ3

In this section, we present how we create a new compressor using SZ3, which can improve the compression ratios for the data generated from the real-world scientific simulation GAMESS [22]. In the following text, we first introduce the GAMESS data and its current compressor — SZ-Pastri [23], and then present our characterization on the quanzation integers and the new customization method. At last, we evaluate the compression ratios and speed based on three representative data fields in GAMESS.

4.1 GAMESS data and SZ-Pastri Compressor

Quantum chemistry researchers often need to obtain a wavefunction by solving the Schrödinger differential equation, which involves all the chemical system's information. The wavefunction needs to be constructed by two-electron repulsion integrals (ERI), which requires a too large memory capacity to hold at runtime during the simulation. A straightforward solution is reproducing the ERI dataset whenever needed during the simulation, although this would significantly delay the simulation because of the fairly expensive cost in generating the ERI data. In our prior work, we developed an efficient error-bounded lossy compressor called SZ-Pastri [23], which can compress the ERI data in memory and decompress it in the beginning of each iteration of the simulation. Such a method can effectively avoid the ERI recalculation cost, so as to improve the overall performance. SZ-Pastri takes advantages of the periodic patterns that exist in the GAMESS dataset, because the ERI values are calculated in order and are dependent on shape and distance of electron clouds. Specifically, SZ-Pastri identifies a periodic pattern and uses it along with a scaling coefficient for each block to enable accurate data prediction. Using Figure 3, we give an example to further illustrate the scaled-pattern data feature of the GAMESS ERI dataset. Note that the chemistry shells have specific names, e.g., s, p, d, f, depending on the total angular momentum, so $(dd|dd)$ in the example indicates the ERI block formed by d, d, d, d shells. As shown in the figure, the adjacent data values are not smooth at all throughout the whole dataset. However, after splitting the entire dataset into many small blocks (each with 36 data points), we note that different blocks can overlap very well based on a scaling coefficient. That is, we just need to record one block of data as they appear exactly, and then use them to predict all other data blocks with a calculated scaling coefficient. This leads to substantial performance gain compared to existing general compressors [6], [13] that strongly relies on the smoothness of adjacent data points. In absolute terms, SZ-Pastri achieves a compression ratio that is 2.3$\times$ that of the best existing compressors, while maintaining the precision required by the scientists [23].

4.2 Data characterization and pipeline customization

We first characterize the quantization integers for SZ-Pastri, which are the most impactful factors for the final compression ratios. To enable correct decompression, SZ-Pastri needs to quantize and store the information for both the periodic patterns and block-wise scales. Thus, the quantization integers in SZ-Pastri consist of three components, which are computed from data, patterns, and scales, respectively. As displayed in Figure 5(a), the distribution of quantization integers for the pattern-based predictor is centered in 0,

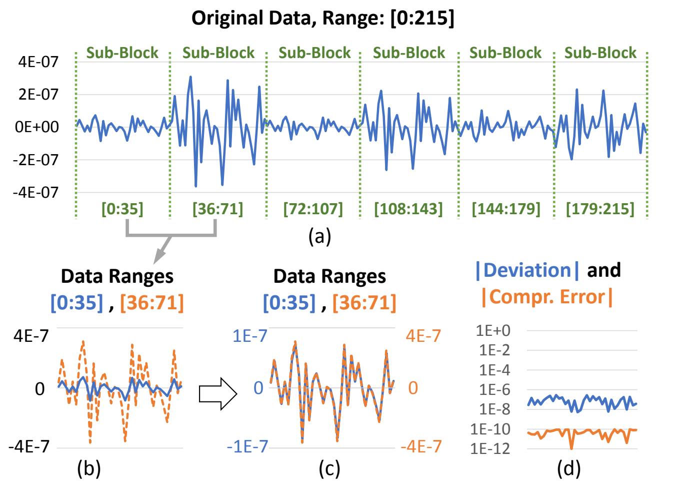


Fig. 3. ERIs can be presented as a 1D array where x-axis is the BF index and y-axis is the integral value. A part of the generated $(dd|dd)$ ERI block is shown in (a) [23].

which indicates very high prediction accuracy and thus high compression ratios. However, a significant percentage (20% for data) of the quantization integers fall out of the quantization range (64 in this setting), and these data are called 'unpredictable'. These data require additional mechanisms for storage in order to be correctly recovered during decompression. In the original design of SZ-Pastri, they are directly truncated and stored based on the user-specified error, which fails to exploit the correlation in the data to achieve high compression, though relatively fast compression speed is obtained.

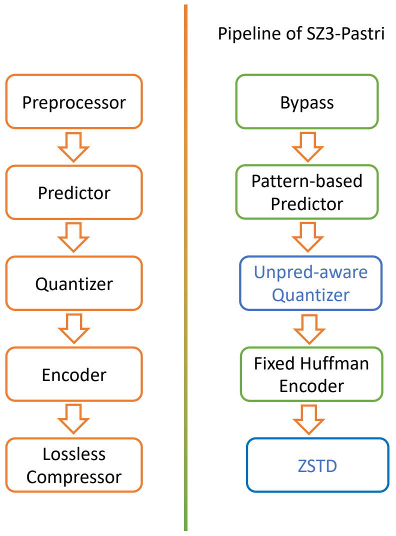


Fig. 4. Compression pipelines for GAMESS data. Blue boxes indicate optimized/added modules in SZ3-Pastri over SZ-Pastri.

Based on these observations, we improve the compression efficiency of SZ-Pastri by leveraging a specialized quantizer to deal with the unpredictable data. Inspired by the embedded encoding approaches widely used in transform-based compressors [13], [46], we store data in the order of bitplane instead of applying the truncation directly. A bitplane represents a set of bits corresponding to a given bit position in the binary representations of the data. Because small data values have meaningful bits only in less significant bitplanes, the relatively significant bitplanes will yield good compression ratios because of consecutive 0s. Similar to [13], we first align the exponents of the prediction difference on unpredictable data to that of the error bound to convert the floating-point data into integers. These integers are then recorded in the order of bitplanes, namely, from the most significant bitplane to the least significant bitplane. Compared with direct truncation, this encoding method will not change the encoded size at this stage; however, its compressive encoded format will promise better compression ratios when lossless compression is adopted. Since this quantizer takes special care of unpredictable data storage, we name it *Unpred-aware Quantizer* throughout the paper. To take advantage of this method, we also add a lossless stage to the composed compression pipeline, as displayed in Figure 4. This new compressor is called SZ3-Pastri, as it optimizes SZ-Pastri using the SZ3 framework.

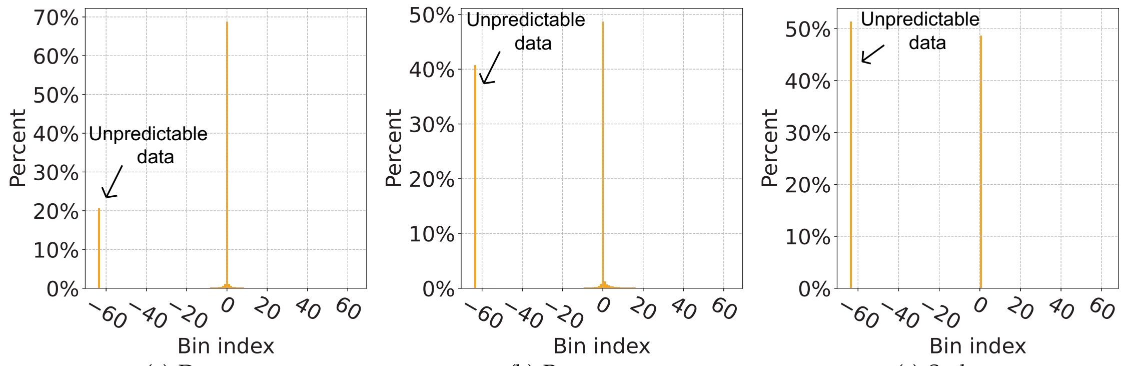
4.3 Evaluation results

We evaluate our method and compare it with SZ-Pastri and its variation (SZ-Pastri equipped with lossless compression) using three representative fields in GAMESS. Unless otherwise noted, all the experiments in this paper are conducted on the Bebop supercomputer [47] at Argonne National Laboratory. Bebop has 664 Broadwell nodes, each of which is equipped with two Intel Xeon E5-2695v4 processors containing 36 physical cores in total and 128 GB of DDL4 memory.

The rate-distortion graphs of the evaluation are displayed in Figure 6. This graph entails the correlation between bit rate and Peak Signal-to-Noise Ratio (PSNR). The bit rate equals $bits/cr$ where $bits$ is the number of bits in original data representation (e.g., 32 for single-precision and 64 for double-precision floating-point data) and cr is the compression ratio. PSNR is inversely proportional to the mean square error of decompress data and original data in logarithmic scale. Lower bit rate and higher PSNR indicate better compression quality. According to this figure, SZ3-Pastri leads to the best rate-distortion along almost all bit rates. For example, the improvements of compression ratios on the $(ff|ff)$ dataset are generally 40% and 20%, respectively, compared with SZ-Pastri and its lossless variation. We also show the exact compression ratio and speed of the three approaches under the desired absolute error tolerance (1E-10 according to the domain scientists) in Table 1. Compared with original SZ-Pastri, SZ3-Pastri significantly improves the compression ratios under the requirements. However, it has a degradation in performance, which is caused by the embedded encoding on unpredictable data (i.e., Unpred-aware Quantizer, which improves the compression ratio) and the final lossless compression.

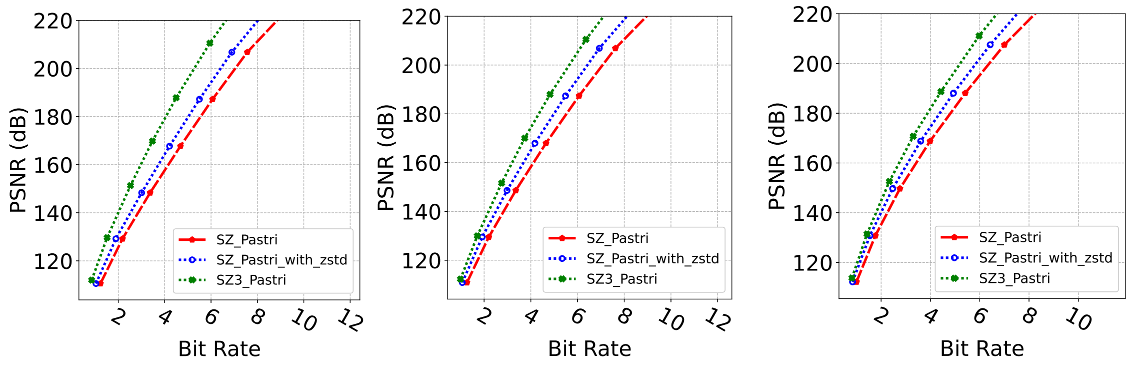
5 Composing an Efficient Compressor for APS Data using SZ3

We then leverage our SZ3 framework to create an adaptive compression pipeline for the X-ray ptychographic data acquired at the Advanced Photon Source (APS). Similar to the previous section, we first introduce APS data, followed by the data characterization and compression pipeline customization along with the evaluation.



(a) Data (b) Pattern (c) Scale

Fig. 5. Distribution of quantization integers in SZ3-Pastri.



(a) $(ff|ff)$ (b) $(ff|dd)$ (c) $(dd|dd)$

Fig. 6. Rate-distortion on GAMESS data. $(ff|ff)$, $(ff|dd)$, and $(dd|dd)$ indicate different electron repulsion integrals (ERI) blocks.

TABLE 1
Result on GAMESS data when absolute error bound is 1E-10

Dataset	Compressor	Ratios	Compression Speed
$(ff\|ff)$	SZ-Pastri	8.46	662.01 MB/s
	SZ-Pastri-with-zstd	9.27	377.17 MB/s
	SZ3-Pastri	10.76	244.43 MB/s
$(ff\|dd)$	SZ-Pastri	8.40	643.58 MB/s
	SZ-Pastri-with-zstd	9.23	370.88 MB/s
	SZ3-Pastri	10.06	221.03 MB/s
$(dd\|dd)$	SZ-Pastri	9.14	613.12 MB/s
	SZ-Pastri-with-zstd	9.96	364.51 MB/s
	SZ3-Pastri	10.71	226.80 MB/s

5.1 APS data

X-ray ptychography is a main high-resolution imaging technique that takes advantage of the coherence provided by the synchrotron source. However, this computational method of microscopic imaging requires much larger data volumes and computational resources compared with conventional microscopic techniques. A revolutionary increase of about 3 orders of magnitude in the coherent flux provided by the coming APS upgrade will aggravate the burden of the data transfer and storage. Therefore, a new compression strategy with high compression ratios is being highly pursued in ptychography. In order to represent most sample scenarios, two ptychographic datasets were acquired from a computer chip pillar (isolated sample) and a subregion of an entire flat chip (extended sample), respectively. In both cases, a Dectris Eiger detector (514×1030 pixels) was used to acquire diffraction patterns as X-ray beam scanned across the sample, and the 2D diffraction images were saved along the time dimension to form a 3D matrix array (19500×514×1030 for chip pillar and 16800×514×1030 for flat chip). In the data analysis, domain experts usually cropped only the central region of the diffraction pattern that contains X-ray signals (lots of zeros outside this region). To fairly assess our compression strategy without giving an overestimated compression ratio, we cropped only central 256×256 pixels.

5.2 Data characterization and pipeline customization

We design an adaptive compression pipeline for APS data based on the following analysis. First, multidimensional Lorenzo predictor introduces higher noise because more decompressed data values are used for prediction [6], even though it is usually superior to the one-dimensional one by exploiting the multidimensional correlation. Second, although APS data has three dimensions (e.g., 19500 × 256 × 256 for the chip pillar sample), it is actually a stack of 2D images along the time dimension with relatively low spatial correlation. When the spatial correlation is not strong, the benefit of using the multidimension Lorenzo predictor may not be able to make up the cost for the higher noise. In addition, considering the usually high correlation in time compared with that in spatial region, it might be more effective to compress the data along the time dimension, namely, treating the data as 256 × 256 1D time series. On the other hand, the multidimensional regression-based predictor should be included because it leverages the multidimensional correlation without being affected by the decompression noise [6], which yields good performance when error bound is relatively high. This requires switching predictors based on the error bound: using a traditional multialgorithm predictor that involves regression for high error bounds and a customized 1D Lorenzo predictor with a transposition preprocessor that reorganizes the data along the time dimension for low error bounds. In our implementation, we switch to the latter along with quantization bin width 2 when the user-specified absolute error bound is less than 0.5 since this setting generates lossless compression. Under such circumstance, the noise introduced by using decompressed data is reduced to 0 when the Unpred-aware Quantizer is leveraged, thanks to the restricted quantization bin and the principle of embedded encoding. We further employ a fixed Huffman encoder for fast encoding with comparable compression ratios. The corresponding compression pipeline for APS data is depicted in Figure 7.

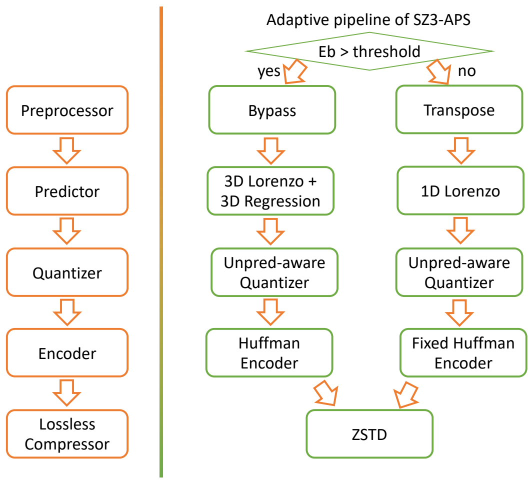


Fig. 7. Adaptive compression pipeline for APS data.

5.3 Evaluation results

We evaluate the customized APS compressor and compare it with 3 baselines: the generic SZ-2.1 compressor for 1D, 3D, and transposed 1D data. As illustrated in Figure 8, a 3D compressor leads to higher PSNR under low bit-rate (high compression ratios), but it suffers when the bit-rate increases to a certain level, where there is a sharp increase in the compression quality for 1D compressors. This is caused by the fact that the noise introduced by decompressed data is mitigated with such an error setting in this dataset. SZ-2.1 is not aware of this information and incorrectly estimates the Lorenzo prediction noise, leading to the selection of regression predictor even when Lorenzo predictor is better. SZ3-APS adaptively chooses the compression pipeline based on the error bound, which leads to comparable performance to that of SZ-2.1 for 3D data when error bound is high. Furthermore, the adopted Unpred-aware Quantizer exhibits higher compression ratios in low error bound, since it provides near-lossless decompressed data that improves the prediction efficiency of the Lorenzo predictor. In absolute terms, when the decompression data is near lossless (i.e., error bound less than 0.5), the compression ratio gain of the proposed compression pipeline is 18% on chip pillar and 12% on flat chip compared with the second best one. Note that SZ3-APS turns out to be lossless in this case, which leads to infinity PSNR in the figure. This experiment demonstrates that better rate-distortion can be achieved

TABLE 2
SZ2 contains more than 120 functions to support different data types, data dimensions, and data-processing methods

Data Type	FP32, FP64 INT8, INT16, INT32, INT64 UINT8, UINT16, UINT32, UINT64
Data Dimension	1D, 2D, 3D, 4D
Functionality	Compression Decompression Parameter Optimization

if the compression pipeline can be chosen based on the characteristics of the data along with the specific error requirements. By providing the flexibility to compose and explore adaptive compression pipelines, SZ3 is expected to deliver high compression quality for diverse datasets and use cases.

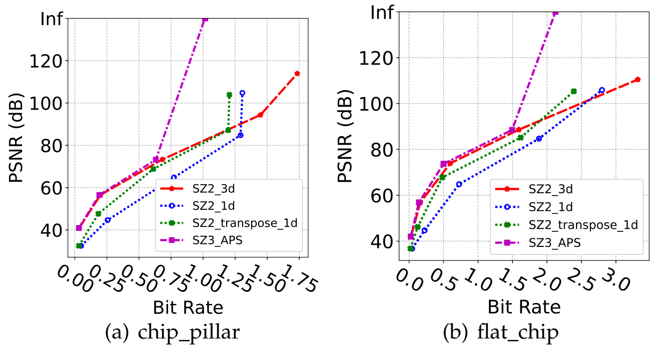


Fig. 8. Rate-distortion on APS data.

6 Sustainability, Quality, and Performance Investigation of SZ3

In this section we first discuss the sustainability of SZ3, and then leverage SZ3 to characterize the quality and performance of diverse compression pipelines.

6.1 Sustainability

We design SZ3 with modularity in mind to allow for a composable framework with high sustainability. Specially, we compare the design of SZ3 with that of SZ2 [48], one of the leading error-bounded lossy compressors with prediction-based pipeline, to demonstrate its superiority.

6.1.1 The codebase of SZ2

SZ2 has a large codebase including more than 120 functions with little code reuse, as shown in Table 2. For example, SZ2 has separate functions to handle the compression or decompression on a dataset with a specific data type, although the logic to compress and decompress different data types is similar. As a result, SZ2 needs to maintain separate code for each data type.

The lack of software architecture design makes it difficult and time-consuming to modify and extend the functionality of SZ2. With more than 120 functions to update, some of them are likely to be missed when adding new features to SZ2. Furthermore, the complexity of SZ2 brings challenges to fully validate the correctness of newly added features, because it is time-consuming to write test code that achieves high code coverage for so many functions of SZ2.

6.1.2 The Codebase of SZ3

We implement SZ3 using three key techniques widely used in software development , namely compile-time polymorphism that avoids the overhead of virtual functions calls, datatype abstraction that avoids separate implementations for different data types, and multidimensional iterator that unifies the interface for accessing data of arbitrary dimensions. With these effort, SZ3 features high sustainability, which allows for easy update and maintenance of the codebase with high efficiency.

Compile Time Polymorphism: SZ constructs the composed compression pipelines at compile time, because compile time polymorphism provides an efficient way to switch different implementations of modules to avoid runtime performance downgrade. For implementation, the module instances are placed as the template parameters of the compressor (see Appendix A.6). A static assert is executed during the construction of the compressor to ensure that only classes that inherent from specific module interfaces are allowed to be used to initialize template parameters.

Datatype Abstraction: We adopt datatype abstraction to simplify the codebase of SZ3 significantly. Most module interfaces, implementations, and compressor pipelines in SZ3 are designed with datatypes as template parameters for efficient code reuse. By comparison, SZ2 has separate implementations for each datatype, which result in a large code base without code reuse.

Multidimensional Iterator: A multidimensional iterator is designed in SZ3 to support data access patterns of different dimensions. This is totally different from SZ2, where independent implementations are required for each dimensionality. The multidimensional iterator in SZ3 provides a simple API to access the current and nearby data points and move to another position. The boundary situations are handled inside iterators. The iterator design eliminates the need to write separate code based on the data dimensions. The pseudocode of prediction and quantization using the multidimensional iterator is presented in Appendix A.7.

With the multidimensional iterator, the complex nested-loop to iterator through the data and the boundary condition checking are hidden from the users. The multidimensional iterator also supports arbitrary movement. For example, to change a 3D iterator to its upper left neighbor, developers can simply use iterator.move(-1, -1, -1) instead of calculating the offset for three dimensions.

6.2 Pipeline integration, mapping, and evaluation

In this section, We integrate three compression pipelines using SZ3, and propose a pipeline mapping technique to improve their performance. Moreover, we reveal the suitable cases of the three pipelines in terms of quality and performance.

6.2.1 Pipeline integration

We integrate three most important pipelines in SZ3, and the details are described as follows.

Compression pipeline SZ3-LR: SZ3-LR is the implementation of the classic compressor SZ2 [6] using SZ3's modular mechanism, which relies on a multialgorithm predictor for better data correlation. This predictor consists of a Lorenzo predictor and a regression-based predictor and predicts data using the better result in between based on blockwise error estimation. As depicted in Figure 1, it uses a linear-scaling quantizer and a Huffman encoder and the ZSTD lossless compressor in the other stages.

Compression pipeline SZ3-Truncation: SZ3-Truncation is a very fast compression pipeline designed for cases where speed is more important than compression ratio. Given the target bytes k as input parameter, it keeps k most-significant bytes of each floating-point data while discarding the rest of the bytes. To achieve high compression speed, it bypasses the other stages, which in turn leads to low compression ratios in general cases.

Compression pipeline SZ3-Interp: SZ3-Interp has interpolation-based predictors [8] in its pipeline. Both linear interpolation and cubic spline interpolation are included, and they are better than Lorenzo and regression predictors in many cases for the following reasons. On the one hand, interpolation-based predictors suffer less from the error accumulation effect that is normal in Lorenzo predictor because of smaller coefficients in the prediction formula. On the other hand, unlike linear regression, which has an overhead to store coefficients, SZ3-Interp has constant coefficients and therefore does not have storage overhead. Similar to SZ3-LR, it uses a linear-scaling quantizer for respecting error bounds, as well as a Huffman encoder and the ZSTD lossless compressor for high compression ratios.

6.2.2 Pipeline mapping

The multidimensional iterator we proposed in Section 6.1.2 allows users to access the input data without writing any dimension-specific logic, such that it dramatically improves the code simplicity of the predictor module. However, one drawback is that it does not have as good performance as the dimension-specific implementation. To give users the simplicity to develop new pipelines while guaranteeing the pipeline performance, the data access part of the pipeline can be replaced from multidimensional iterator to dimension-specific code when the development of a new pipeline is completed. As a result, the predictor in the pipeline will contain several codecs, each of which handles data in a specific dimension. Note that the pipeline still has a modular design and can be customized with different quantizer, encoder, and lossless modules.

6.2.3 Pipeline evaluation

We use datasets from five scientific domains: cosmology, climate, quantum structure, seismic wave, and turbulence. The detailed information is shown in Table 3.

We demonstrate the compression quality of the three pipelines using rate-distortion graph in Figure 9. Note that the rate distortion of SZ2.1 is identical to that of SZ3-LR; thus we do not show SZ2.1 in this figure. We observe from Figure 9 that SZ3-Truncation has the lowest compression quality, and this is consistent with its simple byte-truncation design. SZ3-Interp is better than SZ3-LR on most of the datasets, especially on cases with a high compression ratio

TABLE 3
Dataset Information

Application	Domain	#Fields	Dimensions	Total Size
Hurricane	Climate	13	100 × 500 × 500	1.2GB
NYX	Cosmology	6	512 × 512 × 512	3GB
SCALE-LETKF	Climate	6	98 × 1200 × 1200	3.2GB
QMCPack	Quantum Structure	1	288 × 115 × 69 × 69	0.6GB
RTM	Seismic Wave	3600	449 × 449 × 235	635GB
Miranda	Turbulence	7	256 × 384 × 384	1GB

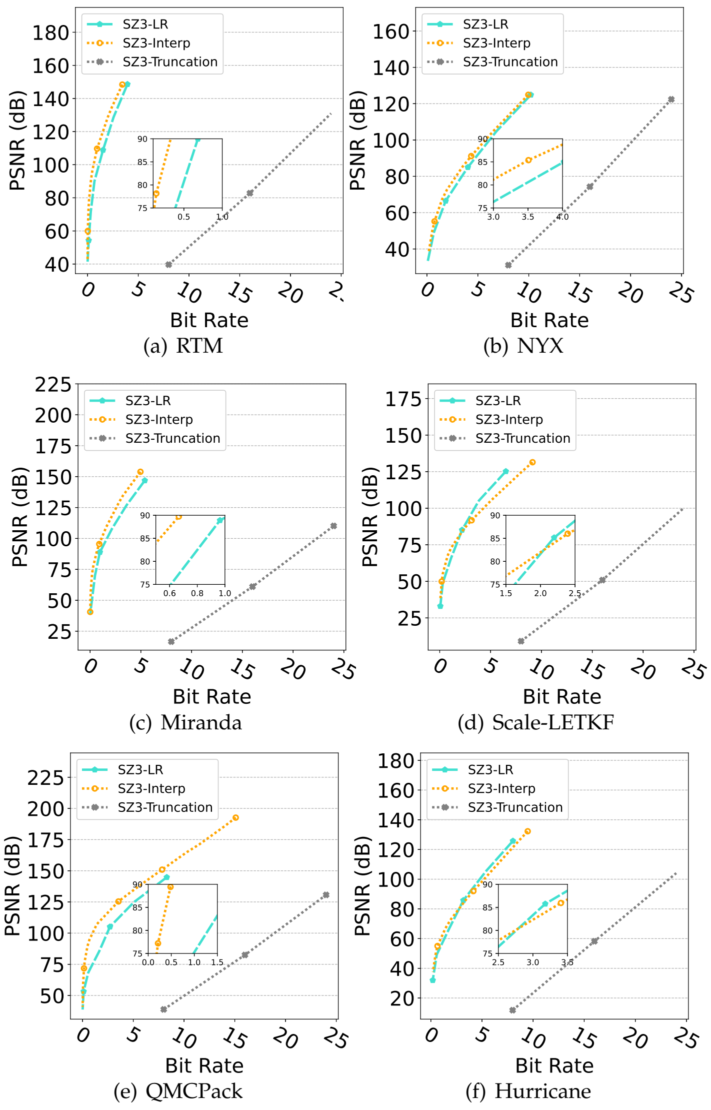


Fig. 9. Compression quality evaluation (lower bit rate & higher PSNR → better quality). Result for SZ2.1 is omitted since it is very similar to that of SZ3-LR.

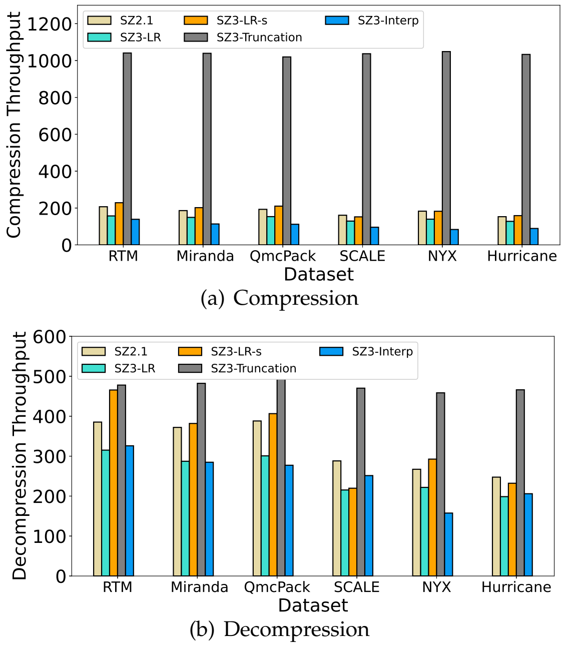


Fig. 10. Compression/decompression throughput (MB/s) when relative error bound (error bound normalized to value range) is 1E-3.

with a bit rate lower than 3. For example, on the Miranda dataset, under the same PSNR of 90, the compression ratio of SZ3-Interp is 47, and it is 56% higher than the compression ratio of SZ3-LR, which is 30. On the other hand, SZ3-LR is still the best choice on the Scale and Hurricane datasets when high compression accuracy is needed.

The performance evaluation is shown in Figure 10. We include SZ2.1 as the baseline. SZ-LR-s is the pipeline mapping optimized version of SZ-LR-s. We can see from Figure 10 that SZ3-LR-s has comparable performance with SZ2.1 on all datasets. SZ3-Truncation has the best performance among all compressors including SZ2.1. Its 1GB/s compression throughput is 4X higher than that of the second-best compressor. SZ3-Interp is not as fast as others, but its throughput is still higher than 100 MB/s in all cases.

The quality and performance evaluations reveal the suitable cases for the three built-in pipelines. Specifically, SZ3-Trunction, as a high-speed compressor, is the best choice when there are strict requirements on the compression time, as with some in situ applications. SZ3-Interp would be the first preference in cases where high compression ratio is wanted under relaxed time constraints, such as scientific applications that run for a long time and generate large amounts of data. SZ3-LR has balanced quality and speed; users could choose it as the default compressor in general situations where both high compression ratio and short compression time are needed.

7 Conclusion and Future Work

In this paper, we propose a modular, composable compression framework –SZ3– which allows users to customize on-demand error-bounded lossy compressors in an adaptive and extensible fashion with minimal effort. Using SZ3, we develop efficient error-bounded lossy compressors for two real-word application datasets based on the data characteristic and user requirements, which improve the compression ratios by nearly 20% when compared with other state-of-the-art compressors with the same data distortion. We also compare the sustainability of SZ3 with SZ2, and leverage SZ3 to integrate and evaluate different compression pipelines. In the future, we will integrate more instances to the framework for diverse use cases and provide support for various hardware including GPUs and FPGAs.

ACKNOWLEDGMENTS

This research was supported by the Exascale Computing Project (ECP), Project Number: 17-SC-20-SC, a collaborative effort of two DOE organizations – the Office of Science and the National Nuclear Security Administration, responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering and early testbed platforms, to support the nation's exascale computing imperative. The material was supported by the U.S. Department of Energy (DOE), Office of Science and DOE Advanced Scientific Computing Research (ASCR) office, under contract DE-AC02-06CH11357, and supported by the National Science Foundation under Grant OAC-2003709, OAC-2003624/2042084, SHF-1910197, SHF-1943114, OAC-2104023, and OAC-2153451. This research used resources of the Advanced Photon Source, a U.S. Department of Energy (DOE) Office of Science User Facility, operated for the DOE Office of Science by Argonne National Laboratory under Contract No. DE-AC02-06CH11357. We acknowledge the computing resources provided on Bebop, which is operated by the Laboratory Computing Resource Center at Argonne National Laboratory.

REFERENCES

[1] I. T. Foster *et al.*, "Computing just what you need: Online data analysis and reduction at extreme scales," in *European Conference on Parallel Processing*, Springer. Cham: Springer International Publishing, 2017, pp. 3–19.

[2] G. K. Wallace, "The JPEG still picture compression standard," *IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, pp. xviii–xxxiv, 1992.

[3] D. Taubman and M. Marcellin, *JPEG2000 Image Compression Fundamentals, Standards and Practice*. New York, NY, USA: Springer Publishing Company, Incorporated, 2013.

[4] L. Theis, W. Shi, A. Cunningham, and F. Huszár, "Lossy image compression with compressive autoencoders," *arXiv preprint arXiv:1703.00395*, 2017.

[5] 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.

[6] 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*, IEEE. New York, NY, USA: IEEE, 2018.

[7] 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*, ser. HPDC '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 89—-100.

[8] 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.

[9] K. Zhao, S. Di, D. Perez, X. Liang, Z. Chen, and F. Cappello, "Mdz: An efficient error-bounded lossy compressor for molecular dynamics," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, 2022, pp. 27–40.

[10] S. Li, S. Di, K. Zhao, X. Liang, Z. Chen, and F. Cappello, "Resilient error-bounded lossy compressor for data transfer," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21, 2021, pp. 1–14.

[11] 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.

[12] X. Yu, S. Di, K. Zhao, j. Tian, D. Tao, X. Liang, and F. Cappello, "Szx: an ultra-fast error-bounded lossy compressor for scientific datasets," https://arxiv.org/abs/2201.13020, 2022, online.

[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] 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.

[15] M. Ainsworth, S. Klasky, and B. Whitney, "Compression using lossless decimation: analysis and application," *SIAM Journal on Scientific Computing*, vol. 39, no. 4, pp. B732–B757, 2017.

[16] 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.

[17] Ainsworth, Mark and Tugluk, Ozan and Whitney, Ben and Klasky, Scott, "Multilevel techniques for compression and reduction of scientific data-quantitative control of accuracy in derived quantities," *SIAM Journal on Scientific Computing*, vol. 41, no. 4, pp. A2146–A2171, 2019.

[18] X. Liang, B. Whitney, J. Chen, L. Wan, Q. Liu, D. Tao, J. Kress, D. R. Pugmire, M. Wolf, N. Podhorszki *et al.*, "Mgard+: Optimizing multilevel methods for error-bounded scientific data reduction," *IEEE Transactions on Computers*, 2021.

[19] 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.

[20] S. Di and F. Cappello, "Fast error-bounded lossy HPC data compression with SZ," in *2016 IEEE International Parallel and Distributed Processing Symposium*. New York, NY, USA: IEEE, 2016, pp. 730–739.

[21] 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*. New York, NY, USA: IEEE, 2017, pp. 1129–1139.

[22] Y. Alexeev, M. P Mazanetz, O. Ichihara, and D. G Fedorov, "GAMESS as a free quantum-mechanical platform for drug research," *Current topics in medicinal chemistry*, vol. 12, no. 18, pp. 2013–2033, 2012.

[23] A. M. Gok, S. Di, A. Yuri, D. Tao, V. Mironov, X. Liang, and F. Cappello, "PaSTRI: A novel data compression algorithm for two-electron integrals in quantum chemistry," in *IEEE International Conference on Cluster Computing (CLUSTER)*. New York, NY, USA: IEEE, 2018, pp. 1–11.

[24] X. Liang, S. Di, D. Tao, Z. Chen, and F. Cappello, "An efficient transformation scheme for lossy data compression with point-wise relative error bound," in *IEEE International Conference on Cluster Computing (CLUSTER)*. New York, NY, USA: IEEE, 2018, pp. 179–189.

[25] X. Liang, H. Guo, S. Di, F. Cappello, M. Raj, C. Liu, K. Ono, Z. Chen, and T. Peterka, "Toward feature-preserving 2D and 3D vector field compression." in *PacificVis*, 2020, pp. 81–90.

[26] R. Underwood, J. C. Calhoun, S. Di, A. Apon, and F. Cappello, "Optzconfig: Efficient parallel optimization of lossy compression configuration," *IEEE Transactions on Parallel and Distributed Systems*, 2022.

[27] L. P. Deutsch, "GZIP file format specification version 4.3," 1996.

[28] zstd, https://github.com/facebook/zstd/releases, 2019, online.

[29] 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, Jan 2009.

[30] S. Claggett, S. Azimi, and M. Burtscher, "SPDP: An automatically synthesized lossless compression algorithm for floating-point data," in *2018 Data Compression Conference*. New York, NY, USA: IEEE, March 2018, pp. 335–344.

[31] F. Alted, "Blosc, an extremely fast, multi-threaded, meta-compressor library," https://www.blosc.org, 2017, online.

[32] P. Lindstrom, "Error distributions of lossy floating-point compressors," *Joint Statistical Meetings*, vol. 1, no. 1, pp. 2574–2589, 2017.

[33] F. Cappello, S. Di, S. Li, X. Liang, A. M. Gok, D. Tao, C. H. Yoon, X.-C. Wu, Y. Alexeev, and F. T. Chong, "Use cases of lossy compression for floating-point data in scientific data sets," *The*

International Journal of High Performance Computing Applications, vol. 33, no. 6, pp. 1201–1220, 2019.

[34] S. Li, S. Jaroszynski, S. Pearse, L. Orf, and J. Clyne, "Vapor: A visualization package tailored to analyze simulation data in earth system science," *Atmosphere*, vol. 10, no. 9, p. 488, 2019.

[35] H. Ma, D. Liu, N. Yan, H. Li, and F. Wu, "End-to-end optimized versatile image compression with wavelet-like transform," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[36] Q. Gong, X. Liang, B. Whitney, J. Y. Choi, J. Chen, L. Wan, S. Ethier, S.-H. Ku, R. M. Churchill, C.-S. Chang *et al.*, "Maintaining trust in reduction: Preserving the accuracy of quantities of interest for lossy compression," in *Smoky Mountains Computational Sciences and Engineering Conference*. Springer, 2021, pp. 22–39.

[37] J. Lee, Q. Gong, J. Choi, T. Banerjee, S. Klasky, S. Ranka, and A. Rangarajan, "Error-bounded learned scientific data compression with preservation of derived quantities," *Applied Sciences*, vol. 12, no. 13, p. 6718, 2022.

[38] R. Ballester-Ripoll, P. Lindstrom, and R. Pajarola, "TTHRESH: Tensor compression for multidimensional visual data," *IEEE transactions on visualization and computer graphics*, 2019.

[39] 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.

[40] D. Tao, S. Di, X. Liang, Z. Chen, and F. Cappello, "Optimizing lossy compression rate-distortion from automatic online selection between sz and zfp," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 8, pp. 1857–1871, 2019.

[41] J. Kunkel, A. Novikova, E. Betke, and A. Schaare, "Toward decoupling the selection of compression algorithms from quality constraints," in *High Performance Computing*, J. M. Kunkel, R. Yokota, M. Taufer, and J. Shalf, Eds. Cham: Springer International Publishing, 2017, vol. 10524, pp. 3–14.

[42] R. Underwood, S. Di, J. C. Calhoun, and F. Cappello, "FRaZ: A generic high-fidelity fixed-ratio lossy compression framework for scientific floating-point data," https://arxiv.org/abs/2001.06139, 2020, online.

[43] L. Ibarria, P. Lindstrom, J. Rossignac, and A. Szymczak, "Out-of-core compression and decompression of large n-dimensional scalar fields," in *Computer Graphics Forum*, vol. 22, no. 3. Wiley Online Library, 2003, pp. 343–348.

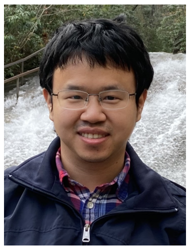
[44] Z. Chen, S. W. Son, W. Hendrix, A. Agrawal, W.-k. Liao, and A. Choudhary, "NUMARCK: machine learning algorithm for resiliency and checkpointing," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE Press. New York, NY, USA: IEEE, 2014, pp. 733–744.

[45] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[46] X. Liang, Q. Gong, J. Chen, B. Whitney, L. Wan, Q. Liu, D. Pugmire, R. Archibald, N. Podhorszki, and S. Klasky, "Error-controlled, progressive, and adaptable retrieval of scientific data with multilevel decomposition," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–13.

[47] Bebop_supercomputer, Available at https://www.lcrc.anl.gov/systems/resources/bebop , 2019, online.

[48] "Repository of sz2 compresssor," https://github.com/szcompressor/SZ, 2021, online.



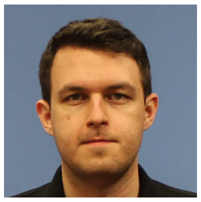
Xin Liang is an assistant professor with the Department of Computer Science at University of Kentucky. Prior to that, he worked as a Computer/Data Scientist in the Workflow Systems Group at Oak Ridge National Laboratory. He received his Ph.D. degree from University of California, Riverside in 2019 and his bachelor's degree from Peking University in 2014. His research interests include high-performance computing, parallel and distributed systems, scientific data management and reduction, big data analytic, and scientific visualization. He is a member of the IEEE. Email: xliang@cs.uky.edu.

Kai Zhao is an assistant professor of computer science at University of Alabama at Birmingham. He received his Ph.D. in computer science from University of California, Riverside in 2022. He received his bachelor's degree from Peking University in 2014. His research interests include high-performance computing, scientific data management and reduction, and resilient machine learning. Email: kzhao@uab.edu.

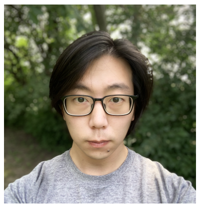
Sheng Di (Senior Member, IEEE) received his master's degree from Huazhong University of Science and Technology in 2007 and Ph.D. degree from the University of Hong Kong in 2011. He is currently a computer scientist at Argonne National Laboratory. Dr. Di's research interest involves resilience on high-performance computing (such as silent data corruption, optimization checkpoint model, and in-situ data compression) and broad research topics on cloud computing (including optimization of resource allocation, cloud network topology, and prediction of cloud workload/hostload). He is working on multiple HPC projects, such as detection of silent data corruption, characterization of failures and faults for HPC systems, and optimization of multilevel checkpoint models. He is the recipient of DOE 2021 Early Career Research Program Award, and 2019&2021 R&D 100 award. Email: sdi1@anl.gov.

Sihuan Li is a research scientist at Facebook. Before that, he received his Ph.D. degree in computer science at University of California, Riverside. He obtained his bachelor's degree in math from Huazhong University of Science and Technology, China. He did a long-term internship at Argonne National Laboratory. Broadly speaking, his research interests fall into High Performance Computing. Specifically, he mainly studies Algorithm Based Fault Tolerance (ABFT), lossy compression and their applications in large scale scientific simulations. Email: sli049@ucr.edu.

Robert Underwood is a Post Doctoral Appointee at Argonne National Laboratory. He received his Ph.D. in Computer Science from Clemson University. His research interests involve using approximate computing methods such as lossy data compression to accelerate parallel and distributed computing while ensuring that scientific data integrity is preserved. He is currently working on using optimization based approaches to configure lossy compression. Email: runderwood@anl.gov



Ali Murat Gok is a computer engineer at Cerebras Systems. He received his PhD in computer engineering from Northwestern University in 2018 and his B.S. in electronics engineering / mathematics double major program from Bogazici University, Turkey in 2012. He had long term internships followed by a postdoctoral position at Argonne National Laboratory. His research interest include energy efficient computer architecture, high performance computing, and scientific lossy data compression. Email: ali.gok@cerebras.net



Jiannan Tian is current PhD Candidate in Intelligent Systems Engineering at Indiana University Bloomington. His research interests include lossy compression for scientific data and error analysis, and GPU-centric computing. His ongoing project including developing GPU-accelerated compression algorithm and system design optmization of lossy compression framework. Email: jti1@iu.edu

Junjing Deng is a physicist at the Advanced Photon Source, Argonne National Laboratory. He received the Ph.D. degree in applied physics from Northwestern University in 2016. His research interests center on high-resolution synchrotron X-ray microcopy, lensless computational imaging, and their applications on a variety of scientific problems. Email: junjingdeng@anl.gov.

Jon C. Calhoun (Senior Member, IEEE) Jon is an assistant professor in the Holcombe Department of Electrical and Computer Engineering at Clemson University. He received a B.S. in Computer Science from Arkansas State University in 2012, a B.S. in Mathematics from Arkansas State University in 2012, and a Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign in 2017. He was awarded a prestigious NSF CAREER award in 2020. His research interests lie in fault tolerance and resilience for high-performance computing (HPC) systems and applications, lossy and lossless data compression, scalable numerical algorithms, power-aware computing, and approximate computing. Email: jonccal@clemson.edu

Dingwen Tao is an associate professor in the Luddy School of Informatics, Computing, and Engineering at Indiana University Bloomington. He received his Ph.D. in Computer Science from University of California, Riverside in 2018 and B.S. in Mathematics from University of Science and Technology of China in 2013. He works at the intersection of HPC and big data analytics, focusing on scientific data management, HPC storage and I/O, fault tolerance at extreme scale, and systems for machine learning. He was the receipt of the 2020 IEEE Computer Society TCHPC Early Career Researchers Award for Excellence in HPC and 2020 NSF CRII Award in 2020. Email: ditao@iu.edu.

Zizhong Chen (Senior Member, IEEE) received a bachelor's degree in mathematics from Beijing Normal University, a master's degree degree in economics from the Renmin University of China, and a Ph.D. degree in computer science from the University of Tennessee, Knoxville. He is a professor of computer science at the University of California, Riverside. His research interests include high-performance computing, parallel and distributed systems, big data analytics, cluster and cloud computing, algorithm-based fault tolerance, power and energy efficient computing, numerical algorithms and software, and large-scale computer simulations. His research has been supported by National Science Foundation, Department of Energy, CMG Reservoir Simulation Foundation, Abu Dhabi National Oil Company, Nvidia, and Microsoft Corporation. He received a CAREER Award from the US National Science Foundation and a Best Paper Award from the International Supercomputing Conference. He is a Senior Member of the IEEE and a Life Member of the ACM. He currently serves as a subject area editor for *Elsevier Parallel Computing* journal and an associate editor for the *IEEE Transactions on Parallel and Distributed Systems*. Email: chen@cs.ucr.edu.

Franck Cappello (Fellow, IEEE) is the director of the Joint-Laboratory on Extreme Scale Computing gathering six of the leading high-performance computing institutions in the world: Argonne National Laboratory, National Center for Scientific Applications, Inria, Barcelona Supercomputing Center, Julich Supercomputing Center, and Riken AICS. He is a senior computer scientist at Argonne National Laboratory and an adjunct associate professor in the Department of Computer Science at the University of Illinois at Urbana-Champaign. He is an expert in resilience, fault tolerance, and lossy compression for scientific computing and data analytics. Recently he started investigating lossy compression for scientific data sets to respond to the pressing needs of scientist performing large-scale simulations and experiments. His contribution to this domain is one of the best lossy compressors for scientific data set respecting user-set error bounds. He is a member of the editorial board of the *IEEE Transactions on Parallel and Distributed Computing* and of the *ACM HPDC* and *IEEE CCGRID* steering committees. He is a fellow of the IEEE. Email: cappello@mcs.anl.gov.

APPENDIX A

We demonstrate some representative interfaces and functions in this appendix. Note that `T` is the template for data type, `N` is the template for dimensionality, and `X` is the template for quantized data type.

A.1 Snippet of Preprocess Interface

```
template<class T, uint N>
class PreprocessInterface {
  virtual void preprocess(T * data, SZ::Config<T, N
      >& conf);
  virtual void postprocess(T * data, SZ::Config<T, N
      >& conf);
};
```

A.2 Snippet of Predictor Interface

```
template<class T, uint N>
class PredictorInterface {
  virtual T predict(const iterator &iter);
  virtual T estimate_error(const iterator &iter);
  virtual uint save(uchar *&c);
  virtual void load(uchar *&c);
};
```

A.3 Snippet of Quantizer Interface

```
template<class T, class X, uint N>
class QuantizerInterface {
  virtual X quantize(T data, T pred);
  virtual T recover(T pred, X quant_value);
  virtual uint save(uchar *&c);
  virtual void load(uchar *&c);
};
```

A.4 Snippet of Encoder Interface

```
 template<class T>
 class EncoderInterface {
  virtual size_t encode(vector<T> &bins, uchar *&
      bytes);
  virtual vector<T> decode(uchar *&bytes, size_t
      length);
  virtual uint save(uchar *&c);
  virtual void load(uchar *&c);
};
```

A.5 Snippet of Lossless Interface

```
class LosslessInterface {
    virtual uchar *compress(uchar *data, size_t
        inSize, size_t &outSize);
     virtual uchar *decompress(uchar *data, size_t&
         outSize);
};
```

A.6 Snippet of Compressor Class

```
template<class T, size_t N, class Preprocessor,
    class Predictor, class Quantizer, class Encoder,
     class Lossless>
class SZ_Compressor {..}
```

A.7 Snippet of Prediction and Quantization

```
vector<int> predict_quantize(T *data) {
    multidimensional_iter blocks(data)
    for (auto block = blocks->begin(); block!=blocks
        ->end(); ++block) {
        for (auto element = block->begin(); element
            != block->end(); ++element) {
            pred=predictor.predict(element);
            quan=quantizer.quantize(*element, pred);
            quantization_results.push_back(quan);
        }
    }
    return quantization_results;
}
```