CAROL: Significantly Improving Fixed-Ratio Compression Framework for Resource-limited Applications

Tri Nguyen
North Carolina State University
Raleigh, NC, USA
tmnguye7@ncsu.edu

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

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

Michela Becchi
North Carolina State University
Raleigh, NC, USA
mbecchi@ncsu.edu

ABSTRACT

Scientific simulations running on HPC facilities generate massive amount of data, putting significant pressure onto supercomputers' storage capacity and network bandwidth. To alleviate this problem, there has been a rich body of work on reducing data volumes via error-controlled lossy compression. However, fixed-ratio compression is not very well-supported, not allowing users to appropriately allocate memory/storage space or know the data transfer time over the network in advance. To address this problem, recent ratio-controlled frameworks, such as FXRZ, have incorporated methods to predict required error bound settings to reach a user-specified compression ratio. However, these approaches fail to achieve fixed-ratio compression in an accurate, efficient and scalable fashion on diverse datasets and compression algorithms.

This work proposes an efficient, scalable, ratio-controlled lossy compression framework (CAROL). At the core of CAROL are four optimization strategies that allow for improving the prediction accuracy and runtime efficiency over state-of-the-art solutions. First, CAROL uses surrogate-based compression ratio estimation to generate training data. Second, it includes a novel calibration method to improve prediction accuracy across a variety of compressors. Third, it leverages Bayesian optimization to allow for efficient training and incremental model refinement. Forth, it uses GPU acceleration to speed up prediction. We evaluate CAROL on four compression algorithms and six scientific datasets. On average, when compared to the state-of-the-art FXRZ framework, CAROL achieves 4× speedup in setup time and 36× speedup in inference time, while maintaining less than 1% difference in estimation accuracy.

ACM Reference Format:
Tri Nguyen, Md Hasanur Rahman, Sheng Di, and Michela Becchi. 2024. CAROL: Significantly Improving Fixed-Ratio Compression Framework for Resource-limited Applications. In *The 53rd International Conference on Parallel Processing (ICPP '24), August 12–15, 2024, Gotland, Sweden.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3673038.3673092

ICPP '24, August 12–15, 2024, Gotland, Sweden

ACM ISBN 979-8-4007-1793-2/24/08
https://doi.org/10.1145/3673038.3673092

1 INTRODUCTION

The simulation of real-world phenomena is at the core of modern research across a variety of scientific domains, such as particle physics, climate modeling, weather prediction, drug discovery and hydrodynamics, among others. The widespread adoption of AI has led to an increased use of simulation also in commercial applications. For example, in the gaming industry, data generated through simulation are used to aid in the training of video games for image enhancing and video generation tasks. Compute-intensive simulations have been well-supported by the continuous improvement in computational capabilities of modern CPUs and GPUs. However, modern storage capacity, network and memory bandwidth are ill-equipped to handle the massive amount of data produced by many scientific simulations [6, 8, 10, 16]. For example, simulations of baryonic gas and N-body treatment of dark matter (NYX [5]) can generate petabytes of data.

As a result, data compression has been adopted as a solution that trades certain computational throughput for better storage and bandwidth utilization [9, 12, 18, 23]. Modern high throughput compressors such as SZx [25] and cuSZp [11] can achieve 120GB/s end-to-end compression and up to 200 GB/s kernel compression throughput. High compression ratio solutions such as SPERR [17] and SZ3 [20] can achieve compression ratios on the order of thousands. In spite of their high performance, these compressors are error-bounded and their compression ratio is not known in advance. Users would provide a tolerance level for the accuracy of reconstructed data (error bound) and the compressor would try to achieve the highest possible compression ratio that is within this bound.

However, in practice, there are many situations where it is not only preferable but also required for the compression ratio to be determined ahead of time. Here, we present three common use cases. *Use case 1*: In research labs, scientists often share compute and storage resources on supercomputers. If each scientist is assigned a maximum amount of storage, the ability to control the amount of data each experiment produces becomes crucial to managing shared storage. *Use case 2*: Need for preserving data quality over a bandwidth-limited system. For example, black hole images generated by the Event Horizon Telescope [1] are the result of a collaborative effort among scientists all over the world. These images were constructed by analyzing telescope data from multiple locations. The first black hole image was generated from 5 petabytes

of data. Controlling storage and network requirements while preserving data quality is crucial in scientific collaboration. *Use case 3*: Large software pipelines require compressors to be compression ratio-controlled. For example, the compression of activation data in deep neural networks has proven an effective method to fit larger models on GPU or training with larger batch sizes [7, 13]. Knowing the compression ratio ahead of time enables adjusting the batch size to the available GPU memory.

Due to the practical demands for fixed-ratio compressors, there have been different works aimed to support fixed-ratio compression or compression ratio estimation. They can be machine learning based [22], surrogate based [15], or fixed-precision based [21]. These approaches come with their own trade-offs. A machine learning based model can offer high precision but incur large setup time. A surrogate based model can have low setup time but high execution time, and can be challenging to extend support for multiple compressors. A fixed-precision model can achieve high accuracy and low execution time but suffer from significantly degraded compression ratio or reconstructed data quality [24].

Amongst these approaches, FXRZ [22] is the first machine learning based framework to solve the fixed-ratio compression problem for error-bounded compressors. FXRZ is a straightforward solution that allows for achieving a desired compression ratio without compromising on data fidelity, and is easily extendable to support new compressors. Users simply need to collect execution data from the new compressor and retrain the model. In contrast, surrogate based frameworks (e.g., SECRE [15]) require designing a new lightweight compressor to mimic the new compressor's behavior. However, FXRZ has significant scaling issues that prevent it from large scale deployment, including: (1) large data collection time, (2) large and not scalable training time, and (3) large inference time. In addition, FXRZ does not adapt well to applications such that the characteristics of the data change over time. For example, consider the weather simulation of hurricane Isabel [2]. An FXRZ model trained on the initial time steps of the simulation can accurately perform prediction for a few subsequent time steps. However, as the simulation advances through time steps with diverse behaviors, FXRZ needs to continually update itself to maintain accuracy. Unfortunately, the current training approach used by FXRZ is not scalable.

In this work, we develop a highly efficient, scalable framework – namely CAROL, that addresses FXRZ's shortcomings while preserving data quality. CAROL is a machine learning based compression ratio-controlled framework with low setup time, low estimation cost and high prediction accuracy. It features a significantly higher performance than FXRZ in both training and inference stages. In particular, we make the following contributions:

- Through rigorous analysis, we identify scalability issues of the existing work ([22]) with regard to data collection time, model training time and inference/prediction time.
- We propose four novel optimizations to improve both setup time (data collection, model training and inference time) and prediction accuracy over state-of-the-art solutions. Specifically, CAROL: (1) uses surrogate-based compression ratio estimation to generate training data, (2) includes a novel calibration method to improve prediction accuracy across a variety of compressors, (3) leverages Bayesian optimization to allow for efficient training and incremental model refinement, and (4) uses GPU acceleration to speed up inference.
- We evaluate CAROL's performance and estimation accuracy on four state-of-the-art scientific lossy compressors: SZ3, SZx, ZFP and SPERR, using simulation datasets from six scientific applications. We compare CAROL against state-of-the-art machine learning and surrogate based solutions. Our experiments show that, on average, CAROL achieves a 4× speedup in setup time and a 36× speedup in inference time over FXRZ, while maintaining estimation accuracy within 1%.

The remainder of the paper is organized as follows. Section 2 provides background on scientific lossy compression and state-of-the-art compression ratio-controlled solutions. Section 3 formulates our research problem and objectives. Section 4 presents CAROL's overall design. Section 5 details CAROL's core contributions. Section 6 presents the results and key takeaways of our experimental evaluation of CAROL. Finally, Section 7 concludes our discussion.

2 BACKGROUND AND RELATED WORK

In this section, we provide background on state-of-the-art scientific lossy compression and fixed-ratio frameworks.

2.1 Lossy Compression

Unlike lossless compression, which requires original and decompressed data to be identical, lossy compression allows for the reconstructed data to *approximate* the original ones. Most compressors allow users to specify the acceptable data loss by providing a parameter called *error bound*. The error bound indicates the maximum acceptable difference (in absolute value or as a percentage of the value range) between any original data point and its reconstructed value. The compressor uses this information to efficiently encode the input data. There are multiple styles of lossy compression. In general, lossy compression can be classified into three categories:

- **Prediction-based compressors** use prior input data points to make prediction about later data points, and encode the differences between predicted and real data. The SZ family of compressors [19, 20] fall under this category and use various prediction techniques, such as Lorenzo predictor and spline interpolation.
- **Transformation-based compressors** leverage data transformation techniques from the image and signal processing domains to group similar data points into a common location. Notable compressors in this category include ZFP [21], which uses the decorrelating linear transform, and SPERR [17], which relies on the wavelet transform.
- **Delta-based compressors** primarily focus on byte-level differences between neighboring data points to quickly encode similar data points. State-of-the-art delta-based compressors include cuSZx [25] and cuSZp [11].

2.2 Compression Ratio-controlled Frameworks

The simplest method to control compression ratio ahead of compression is **fixed-rate compression**, which sets the target number of bits for each compressed data point. This technique controls the compression ratio by changing the compression paradigm from exploiting data similarities to setting the target compressed size. ZFP [21] uses this technique in its GPU implementation. However,

fixed-rate compression suffers from very low compression ratio compared with the error-bounding mode (as shown in prior work [24]). Moreover, it cannot guarantee reconstructed data quality since it does not take into account the values of the data points.

SECRE [15] is a framework to efficiently *predict the compression ratio* by mimicking the compression operation/behavior on sampled datasets. Currently SECRE supports four compressors: SZx, ZFP, SZ3 and SPERR. The critical drawback of SECRE is that its design is closely-coupled with the specific compression design, so that it cannot be extended to other compressors easily. Moreover, its estimation accuracy could be very low in some cases, in that it is non-trivial to infer the compression ratios accurately based on only sampled dataset especially because of complicated compression modules (such as Huffman encoder and Zstd used in SZ3).

To simultaneously preserve data quality and control compression ratio, the **FXRZ** [22] framework *predicts the error bound* that will result in a given compression ratio, by leveraging machine learning (ML) techniques. At a high level, FXRZ's design is motivated by two critical observations: (1) In order to get a high compression ratio, scientific lossy compression often relies on the regional smoothness of data, which can be expressed by a set of key features. FXRZ identifies five important features: value range, mean value, mean neighbor difference, mean lorenzo difference and mean spline difference. (2) Scientific lossy compressors exhibit monotonic behavior. In particular, when increasing the error bound, the compression ratio will increase or stay the same. These two observations allow estimating the relationship between compression ratio and error bound. In particular, FXRZ consists of three major steps:

- *Data collection*: This step extracts the key features of the input dataset and runs the dataset through a compressor multiple times to establish its range of compression ratios and corresponding error bounds.
- *Model training*: This step uses a random forest model to connect the data features to the error bound/compression ratio relationship.
- *Model prediction*: At inference time, the user provides the data set and the desired compression ratio. FXRZ extracts the features from the dataset, based on which it estimates the appropriate error bound that will result in the target compression ratio.

As mentioned in the last section, the key issue of FXRZ is its low scalability due to its limited parallel design and inferior estimation accuracy across different scientific datasets.

3 PROBLEM FORMULATION

Here, we formulate the problem to address and research objectives.

3.1 Problem Formulation

FXRZ's design suffers from three performance and scalability problems that prevent it from wide adoption.

- **Expensive data collection**: FXRZ's training data are collected in two steps. First, the features of the training datasets are extracted; second, each dataset is compressed over a range of error bounds and the corresponding compression ratios are collected. On high ratio compressors (SZ3 and SPERR), data collection can take up to 80% of the overall training time (i.e., multiple hours).

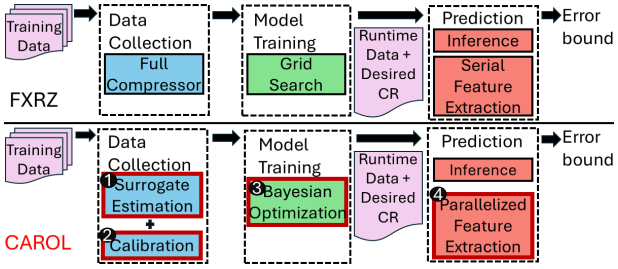


Figure 1: Comparison between FXRZ's and CAROL's overall design. CAROL's core contributions are numbered 1 to 4.

- **Large and non-scalable training time**: FXRZ uses a naïve grid search algorithm to find the hyper-parameters for the underlying random forest model. Due to the large hyper-parameter space, FXRZ avoids an exhaustive search and performs a randomized grid search aided by cross-validation. Not only does this method provide an unstable solution (the hyper-parameters are selected from a randomized grid search), but it also incurs large model fitting times. Anytime new training data are generated, the search must be performed again, invalidating any past training efforts.
- **Large execution time**: FXRZ's execution consists of two steps: (1) extracting the features from the input data, and (2) feeding the features vector and the desired compression ratio into the trained model for inference. While inference takes a relatively short amount of time (a few milliseconds), features extraction can be slower than the underlying compressor. This is especially problematic when applying FXRZ to GPU compressors such as SZx and cuSZp, where throughput is the primary objective.

3.2 Research Objectives

To achieve scalability and efficiency, CAROL's design aims to satisfy three non-trivial constraints:

- *The framework should not compromise accuracy for throughput*. One could simply use a surrogate compressor or similar estimation methods to quickly generate the error bound/compression ratio data pairs required to train the model, achieving speedup factors on the order of hundreds. However, doing so would significantly reduce overall accuracy to the point of impracticality.
- *The framework should not require more compute resources than the original method*. The data collection and model training processes can be accelerated naïvely by running multiple instances of the compressor or by fitting multiple hyperparameter configurations in parallel. However, doing so will cause a significant increase in the amount of compute resources required.
- *The framework should run no slower than its underlying compressor*. If the compressor runs faster than the prediction model, one could run the compressor multiple times and find the configuration achieving the desired compression ratio using a linear search and trial-and-error approach. The proposed framework must be practical and beneficial also for high throughput compressors such as SZx and cuSZp.

4 CAROL'S DESIGN OVERVIEW

Here, we present the design of CAROL, a scalable, ultra-fast ratio-controlled compression framework. As illustrated in Figure 1, CAROL makes four core contributions over FXRZ:

- **Core contribution 1**: We use the surrogate-based compression ratio estimation method (SECRE) to generate training data. Recall that SECRE includes lightweight versions of several state-of-the-art lossy compressors (SZx, ZFP, SZ3 and SPERR) that can be used to estimate the compression ratio achievable on an input under a given error bound (see Section 2). Leveraging SECRE allows us to significantly reduce the training data collection time (compared with running the full compressor).
- **Core contribution 2**: Relying on lightweight compressors and sampling of input data, SECRE may suffer from very high estimation errors (in excess of 100%). We develop a *calibration method* to correct the estimation error (to less than 5% in most cases). Our calibration technique allows us to retain FXRZ's training data accuracy while enjoying SECRE's high estimation throughput.
- **Core contribution 3**: We replace the randomized grid search of hyper-parameters with a more targeted search leveraging Bayesian optimization. Not only does this method reduce the amount of fitting needed to create a new model, but it also allows for scalable and incremental refinement of an existing model.
- **Core contribution 4**: We accelerate the feature extraction process on GPU, allowing for faster inference time even when comparing to GPU compressors.

5 CORE CONTRIBUTIONS

5.1 Surrogate Estimation

FXRZ requires running each input dataset through the compressor multiple times to estimate the relation between error bound and compression ratio. We call this relation $f(e)$, which expresses the compression ratio f as a function of the error bound e. Given a sample of error bounds $\{e_1,..,e_N\}$, FXRZ will estimate $f(e)$ by running the compressor N times, each time with a different error bound e_i, and then interpolating the compression ratios $f(e_i)$ achieved on the sample.

SECRE allows quickly estimating $f(e)$ without running the full compressor. It does so by using a combination of sampling and a lightweight compression pipeline that performs only a subset of the steps of the original compressor. In addition, since the goal of SECRE is to provide an estimate of the compression ratio, it does not save compressed data. Table 1 summarizes the estimation method used by SECRE for four compressors: SZx, SZP, SZ3 and SPERR.

- *SZx* [25] is a delta-based compressor with implementations for both CPU and GPU. It splits data into blocks of 128 elements each, and performs a byte-wise delta encoding customized for the IEEE 754 binary format. The lightweight version of SZx samples the data blocks, selecting one block every 128. It then performs IEEE 754 delta encoding on the sampled data, and uses the results to extrapolate the compression ratio for the entire dataset.
- *ZFP* [21] is a transform-based compressor. ZFP splits the data into multi-dimensional blocks of 32-elements per dimension. It then performs an orthogonal transformation to move data along the block diagonals, and then performs bit-wise embedded encoding to reduce the block size. The lightweight version of ZFP samples the multidimensional blocks, selecting one block every 32 along all dimensions. It then performs full compression on the sampled data to extrapolate the compression ratio for the entire dataset.
- *SZ3* [20] is a predictor-based compressor. SZ3 looks at each individual data point and performs spline interpolation along each data dimension iteratively. The final result is passed through a Huffman encoder to further improve compression ratio. The lightweight version of SZ3 samples the input data, selecting one data point every 5 along each dimension. It performs spline interpolation at the last iteration (the most compute intensive one) on the sampled data, but skips Huffman encoding.
- *SPERR* [17] is a transform-based compressor. SPERR splits the data into multi-dimensional blocks of 128-element per dimension. It first performs the CDF 9/7 transform to group high amplitude values together, and then performs recursive SPECK encoding to identify all the outlier data points. The outliers are stored in compressed-sparse-row (CSR) format and compressed separately. The result is passed through the lossless compressor zstd to maximize compression ratio. The lightweight version of SPERR samples the data by selecting one block every 128 along each dimension. It performs Wavelet transform and SPECK encoding on the sampled data, but skips the outliers encoding and zstd compression passes.

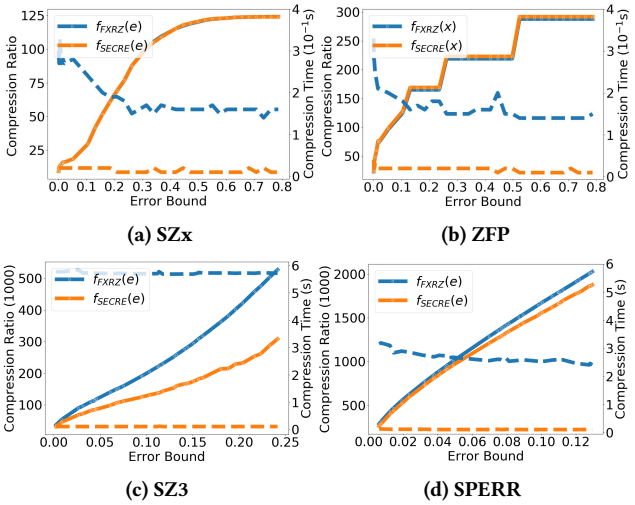


Figure 2: Estimated compression ratio (solid lines with left axis) and runtime (dotted lines with right axis) of FXRZ and SECRE on Miranda data (*viscosity*) with various compressors.

Figure 2 compares the estimates of the compression function $f(e)$ obtained by running the full compressor ($f_{FXRZ}(e)$) and SECRE ($f_{SECRE}(e)$) on Viscosity data from the Miranda dataset. In both cases, the estimations are done by performing linear interpolation of the compression ratios obtained on 35 error bound values e_i. The figure also reports the estimation time using FXRZ and SECRE (dashed line). As can be seen, SECRE can quickly estimate the compression function. In addition, SECRE's execution time is only a fraction of FXRZ execution time . As explained, SECRE achieves this speedup by: (1) using a lightweight compressor, (2) only performing compression on a small fraction of the input dataset (between 5% to 10%), and (3) not saving the compressed data.

Table 1: SECRE data sampling techniques and compression ratio estimation methods

	Original		SECRE	
	Compression Window	**Compression Technique**	**Sampling**	**Estimation Technique**
SZx	Block-wise	Delta Encoding	Block-wise	Delta Encoding
ZFP	Block-wise	Orthogonal Transform + Embedded Encd	Block-wise	Orthogonal Transform + Embedded Encd
SZ3	Point-wise	Spline Interpolation+ Huffman	Point-wise	Spline Interpolation
SPERR	Large Chunk	Wavelet Transform + ZSTD	Large Chunk	Wavelet Transform

We also observe that, for compressors such as SZx and ZFP (Figures 2a and 2b), SECRE's estimation is close to FXRZ's estimation. This is because, in these cases, the lightweight surrogate compressor resembles the core encoding technique of the full compressor. On the other hand, SECRE's estimation of SZ3 and SPERR compression is inaccurate. SPERR and SZ3 apply their core compression scheme in multiple iterations, while SECRE performs only the last algorithm's iteration.

5.2 Calibration

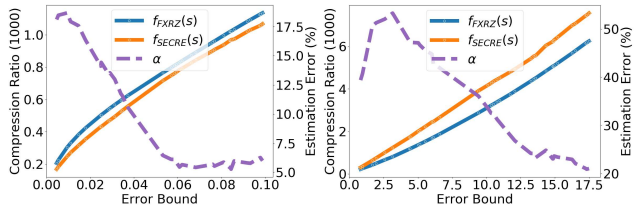
Given a sample of N error bounds e_i, we define the *estimation error* as:

$$\alpha = \frac{\sum_i \alpha_i}{N} \tag{1}$$

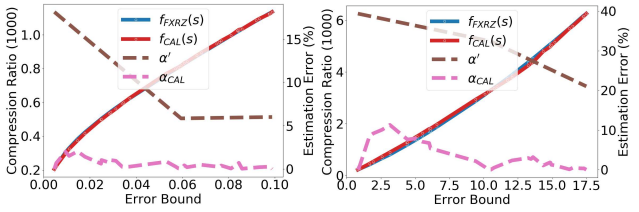
where α_i represents the percentage estimation error for the i^{th} element in the sample:

$$\alpha_i = 100 * \frac{|f_{SECRE}(e_i) - f(e_i)|}{f(e_i)} \tag{2}$$

SECRE reports low estimation error on ZFP and SZx (1.7% and 0.16%, respectively), while incurring significant estimation error (7% and 35%, respectively) on SPERR and SZ3. This large estimation error introduces significant noise in the training data, making them unsuitable for model construction. However, we note that, since the runtimes of SZ3 and SPERR (about 6 sec and 2.5 sec per data point) are larger than those of SZx and ZFP (about 0.2 sec per data point), SECRE would offer the most performance benefits for the two compressors incurring the highest estimation errors. We develop a low overhead calibration method to correct the estimation error.



(a) Miranda-Density **(b) Klacansky-duct**



(c) Miranda-Density Calibrated **(d) Klacansky-duct Calibrated**

Figure 3: Compression ratio (left axis) and estimation error (right axis) on *density* (from Miranda) and *duct* (from Klacansky) data using SPERR.

Our calibration method attempts to predict the estimation error from a few comparison data points, and then uses the predicted estimation error curve to shift and scale the estimated compression ratio to approximate the real compression ratio as accurately as possible. This method is based on the following observations:

- For a given dataset, SECRE always underestimates or always overestimates the true compression ratio.
- SECRE's estimation is typically bi-modal. Each estimation error curve has either a slow changing region and a fast changing one, or an increasing region and a decreasing one.

Figures 3a and 3b show the estimation error curves of SECRE on two datasets: *Density* from Miranda simulation and *DUCT* from fluid simulation. As can be seen, *Density*'s estimation error exhibits a fast decreasing region between error bounds 0 and 0.06, followed by a slowly increasing region between error bounds 0.06 and 0.1, while *duct*'s estimation error presents an increasing region between error bounds 0 and ~3, followed by a decreasing region starting at that error bound.

From these two observations, we conclude that, in order to correct SECRE's estimation error, we need to identify: (1) if SECRE is overestimating or underestimating the compression ratio, and (2) the bi-modal regions. Our calibration method consists of three steps:

- **Step 1**: Run the full compressor on additional data points (as discussed in Section 6.3, we found three to four additional data points to be sufficient in practice);
- **Step 2**: Compare the true and estimated compression ratio for these additional data points to identify underestimation or overestimation;
- **Step 3**: Correct the compression ratio estimation using the following formulas:

$$Overestimation: f_{CAL}(e_i) = \frac{f_{SECRE}(e_i)}{100 - \alpha_i} \tag{3}$$

$$Underestimation: f_{CAL}(e_i) = \frac{f_{SECRE}(e_i)}{100 + \alpha_i} \tag{4}$$

where $f_{CAL}(e_i)$ represents the calibrated compression ratio estimation for error bound e_i.

Figures 3c and 3d show that our calibration method correctly identifies whether SECRE overestimates or underestimates the compression ratio and significantly reduces the estimation error. The constructed estimation error (α' lines) resemble the bi-modal regions. Specifically, it reduces *density*'s estimation error from 9.4% to 0.5%, and *duct*'s estimation error from 34.2% to 3.4%

5.3 Model Training

FXRZ uses a random forest regression model to correlate the error bound/compression ratio function with the features of the training data (*average value*, *value range*, *mean Neighbor difference*, *mean*

Lorenzo difference, and *mean Spline difference*). The result is a forest of decision trees. Inference is performed in two steps: first, the features of the input data are computed; second, the random forest is traversed using those features and the desired compression ratio to retrieve the corresponding error bound. Figure 4 is an example of one such decision trees. For each node: the first line shows the data feature being evaluated (e.g., mean spline difference, mean Lorenzo difference, etc.); *mse* is the mean square error of the node; *samples* are the number of training data points associated to the node, and *value* is the output error bound (on a leaf node, the predicted error bound). The construction of this model is influenced by a set of user-defined hyperparameters that determine the structure of the random forest. The most notable hyper-parameters and the corresponding value ranges are the following:

- *n_estimators [90:1200]*: number of decision trees;
- *max_features [auto/sqrt]*: number of features at every splits;
- *max_depth [10:110]*: maximum depth of a decision tree;
- *min_sample_split [2,5,10]*: minimum number of samples required to split a node;
- *min_sample_leaf [1,2,4]*: minimum number of samples required at each leaf node;
- *bootstrapping [true/false]*: whether to perform resampling with replacement.

This hyperparameter space results in 396000 unique random forest configurations. An exhaustive search of all these configurations to find the optimal random forest model is impractical. Instead, FXRZ creates a small randomized set of unique configurations (currently 10). It then compares this set of configurations to find the best performing one. To this end, FXRZ uses the "k-fold" cross-validation method, which splits the training data into k groups (called "folds"), and bundles them into different combinations of training/testing data. The random forest model that performs the best across the considered training/testing data combinations is selected. This cross-validation method eliminates bias in the selection of training/testing data.

At a high level, the randomized grid search with cross-validation used is motivated by two observations: (1) in a hyper-parameter search space, there exists an optimal configuration (absolute maximum/minimum), and (2) configurations that are close to each other tend to perform similarly. FXRZ creates a set of configurations that are randomly distributed across the hyper-parameter space and performs training and validation to find the best among these configurations. Empirically, this method has shown to result in a

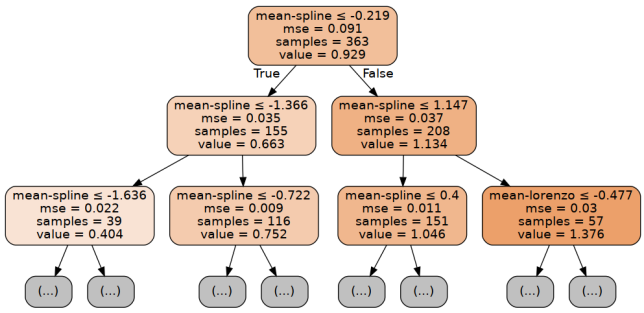
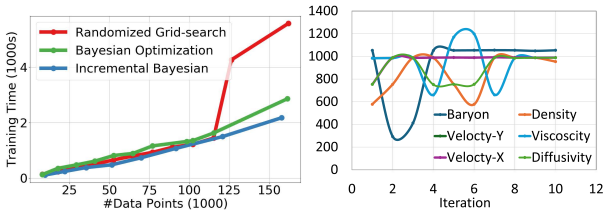


Figure 4: Example decision tree



(a) Training time when varying the training data size.

(b) Changes in the number of decision trees (*n_estimator*) over 10 search iterations.

Figure 5: Analysis of training time and convergence of hyperparameters using bayesian optimization.

reasonably accurate model (estimation error below 10% for SZ2 and 20% for ZFP). However, it suffers from two major scalability issues:

- Some applications (e.g., Hurricane Isabel simulation) exhibit behaviors and data properties that change over time, requiring the model to be progressively updated to maintain good prediction accuracy as the features of the input data change. The randomized grid search method, however, is unsuitable for incremental refinement of an existing model. As new data are generated, FXRZ would require refitting the model from scratch to adapt to changes in compression behavior, since it would need to generate a new set of randomized configurations and perform cross validation on them. Training would essentially start over, discarding any progress made by past searches.
- The compute and memory requirements of the randomized grid search method increase with the training data volume. As more training data are collected, not only does the compute time of each configuration increase, but its memory footprint also increases. Typically, FXRZ maps the construction of a random forest instance to a CPU core. As each instance's memory footprint increases, memory capacity can become a bottleneck, requiring serial training of random forest instances. Figure 5a shows the increase in training time (red line) as more data are added.

In order to reduce overall training time and allow for incremental model refinement, we perform hyperparameter search using Bayesian optimization (rather than randomized grid search). Bayesian optimization is an iterative method that performs targeted searches in the hyperparameter space. It starts with a randomized set of initial hyperparameter configurations, and constructs and tests the corresponding random forests. Each configuration is evaluated by a user-defined scoring function. After having computed the initial score, the optimizer fits a curve to model the score behavior and identify potential high performing regions. In each subsequent iteration, it tries out new configurations and updates the scoring curve. The set of configurations evaluated in each iteration is determined by a combination of "exploitation" (configurations from a high performing region according to the scoring curve) and "exploration" (configuration from a new region). Figure 5b shows how the *n_estimators* hyperparameter (i.e., the number of decision trees) changes over ten search iterations for six datasets. Initially, ten configurations of this parameter are chosen. From this initial set of configurations, the optimizer tries out values in a couple of regions, leading to a wide range of configurations in iteration 1 to 5. From this "exploration", it identifies a region as high performing and tries out multiple configurations from this region ("exploitation") until

it settles to a final value. In figure 5b, this step happens between iteration 5 to 10. The balance between exploration and exploitation is determined by the size of the hyper-parameter space and the number of search iterations.

Even though the Bayesian optimization process is inherently sequential, its targeted search allows for convergence while evaluating fewer random forest configurations. Moreover, as new training data are generated, the optimization process can start from the current model's hyperparameters and thus significantly reduce the training time of any subsequent model retrain. In essence, Bayesian optimization allows for "checkpointing" of the training process, enabling scalable incremental model refinement. Figure 5a compares the training time using randomized grid search with the training time using Bayesian optimization. While the training time increases with the training data volume in all cases, the randomized grid search time spikes at 120 thousand data points, whereas the Bayesian optimization's training time (with and without checkpointing) continues to increase linearly. For randomized grid search, at 120 thousand data points system memory cannot house all random forest configurations, forcing some jobs to run sequentially. Incremental model update through Bayesian optimization outperforms the other training methods, especially as the size of the training data increases.

5.4 Parallel Feature Extraction

One of the key contributions of FXRZ is the identification of five data features that can help predict a dataset's compressibility, namely: *mean value*, *value range*, *mean neighbor difference* (MND), *mean Lorenzo difference* (MLD) and *mean spline difference* (MSD). Mean value and value range represent a dataset's overall amplitude and value spread, while MND, MLD and MSD are related to its local and spatial smoothness. Below are the formulas for calculating MND, MLD and MSD for a data point $d_{i,j,k}$ in a 3D dataset.

$$MND_{i,j,k} = d_{i,j,k} - (d_{i-1,j,k} + d_{i,j-1,k} + d_{i,j,k-1} + d_{i+1,j,k} + d_{i,j+1,k} + d_{i,j,k+1})/6 \quad (5)$$

$$MLD_{i,j,k} = d_{i-1,j,k} + d_{i,j-1,k} + d_{i,j,k-1} + d_{i-1,j-1,k-1} - d_{i-1,j-1,k} - d_{i-1,j,k-1} - d_{i,j-1,k-1} \quad (6)$$

$$spline_i = -\frac{1}{16}d_{i-3} + \frac{9}{16}d_{i-1} + \frac{9}{16}d_{i+1} - \frac{1}{16}d_{i+3} \quad (7)$$

$$MSD_{i,j,k} = |d_i - spline_i| + |d_j - spline_j| + |d_k - spline_k| \quad (8)$$

While these features are effective in modeling data compressibility, they are expensive to collect. For example, Figure 6 shows that extracting the features on a portion of the NYX dataset with dimensions 512×512×512 on CPU takes 15 seconds, 2.5× to 3.75× the execution time of SZ3 (6 seconds) and SPERR (4 seconds). When compared to high throughput compressors such as SZx (0.15 seconds) and ZFP (0.15 seconds), feature extraction is about 100× slower. While this cost is low when compared to the training and data collection times, it is significant for the inference process. FXRZ mitigates this issue by sampling the data using a stride of 4 (resulting in 1.5% of the data being sampled on a 3D dataset), and computing the features on the sampled data alone.

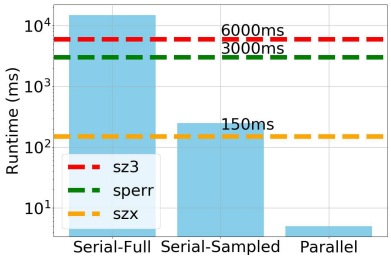


Figure 6: Feature extraction time on a 512MB portion of the Nyx dataset using CPU implementation without and with sampling (*Serial-Full* and *Serial-Sampled*), and GPU implementation (*Parallel*). For comparison, we show SZx, SZ3 and SPERR compression time. The experiments are run on the system of Section 6.

Feature extraction on sampled data for the same NYX dataset takes about 0.25 seconds on CPU. This is significantly lower than the original cost and 20-60× faster than the considered CPU compressors (SZ3 and SPERR). However, compared to GPU compressors such as SZx and cuSZp, the feature extraction cost is still significant. To tackle this problem, we perform GPU parallelization of feature extraction. To make the extraction process amenable for GPU execution, we make the following implementation choices.

- We do not perform feature extraction on the data points on the "surface" of the data set. We note that the MND, MLD and MSD of a data point are calculated based on its neighboring values, which vary in number depending on whether the data point is located on the surface region. In practice, discarding these data points saves conditional statements in the code, avoiding branch divergence on GPU.
- To facilitate memory coalescing on GPU, we perform block-wise sampling instead of point-wise sampling. Specifically, we use D-dimensional blocks (D being the dimensionality of the dataset) with 32 elements per dimension, we sample 1 block every 4, and we perform feature extraction based on all the data points in the sampled blocks.
- We use shared memory for intermediate results (i.e., running feature values).

As shown in Figure 6, for NYX dataset, GPU parallelization (rightmost bar) brings feature extraction time down to about 5 ms, about 50× faster than FXRZ's serial feature extraction and about 30× faster than the execution of SZx, a high throughput GPU compressor.

6 EXPERIMENTAL EVALUATION

6.1 Experimental Setup

Hardware Setup. We perform our experiments on the Bebop and Swing supercomputers at Argonne National Laboratory. The Bebop system has 284 nodes, each comprising an Intel Xeon Phi 7230 CPU with 64 cores, 96 GB DDR4 and 16GB MCDRAM. The Swing system has 6 compute nodes, each equipped with 8 Nvidia A100 GPUs.

Datasets. We evaluate CAROL on four widely used datasets from the SDRBench benchmark [26] and two from the Klacansky data collection [3]. Specifically, we use the following datasets:

Table 2: Dataset summary

	#Fields	Dimension	Size	Domain
Miranda	7	256×84×384	1GB	Turbulence
NYX	6	512×512×512	3.1GB	Cosmology
CESM	77	1800×3600	1.9GB	Climate
Hurricane	48×13	100×500×500	58.1GB	Weather
HCCI	1	560×560×560	0.7GB	Autoignition
MRS	1	512×512×512	512MB	Magnetic

- Miranda [4]: 3D hydrodynamics data obtained from turbulence simulations. The data contain 7 fields: density, diffusivity, pressure, velocity (X,Y,Z) and viscosity;
- NYX [5]: Data from 3D cosmological hydrodynamics simulations. Each snapshot contains 4 fields: baryon density, dark matter density, temperature, and velocity-X;
- Hurricane ISABEL [2]: The data contain 48 snapshots, each including 13 fields;
- CESM [14]: Data from climate simulations generated by the Community Atmosphere Model;
- Klacansky [3]: Data from homogeneous charge compression ignition (HCCI), isotropic turbulence (IT), jet in crossflow (JIC), and magnetic re-connection simulations (MRS).

Table 2 summarizes the main characteristics of these datasets.

Baseline and target Compressors . We use the state-of-the-art compression ratio controlled framework FXRZ [22] as our baseline and four lossy compressors: SZx, SZ3, SZP and SPERR as reference.

6.2 Overall Performance

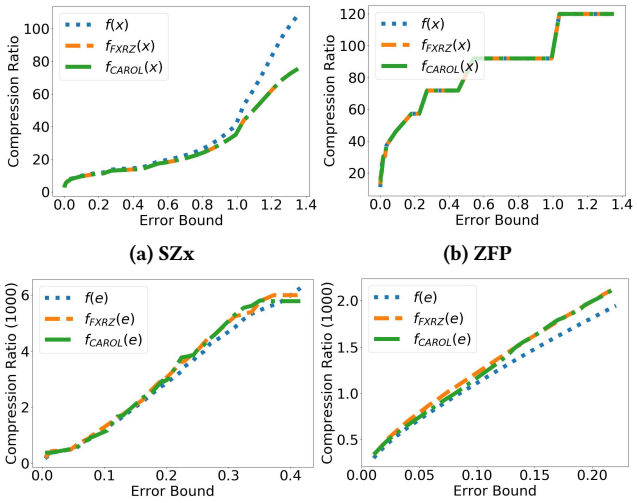
We compare the end-to-end performance - in accuracy and execution time - of CAROL and FXRZ. Recall that these machine learning-based frameworks consist of three steps: (1) *data collection*, (2) *model training* and (3) *compression ratio prediction*. The first two steps are one-time setup operations, while compression ratio prediction is a recurring step that consists of *features extraction* from the input data and *model inference* through a random forest traversal.

6.2.1 End-to-end Accuracy. CAROL aims to maintain the accuracy of FXRZ while lowering the execution time of all three phases: data collection, model training and compression ratio prediction. FXRZ mainly targets *single domain* use cases, where the training and testing data come from a single application. The experiments presented in [22] are of two kinds: *single field* experiments, where compression ratio estimation is performed across different time steps for a single data field of an application, and *multiple fields* experiments, where compression ratio estimation is performed across different data fields of an application.

Single domain experiments: We conduct our experiments on four fields of the NYX dataset: Baryon Density (BD), Dark Matter Density (BDB), Temperature (Temp) and Velocity-X (V-X). For each field,

Table 3: Single domain experiments: estimation error (α) of FXRZ and CAROL on 4 fields of the NYX dataset.

	SZx		ZFP		SZ3		SPERR	
	FXRZ	*CAROL*	*FXRZ*	*CAROL*	*FXRZ*	*CAROL*	*FXRZ*	*CAROL*
BD	10.0%	8.5%	5.3%	4.6%	27.0%	26.0%	17.0%	18.3%
DMD	16.5%	17.0%	5.5%	8.0%	19.0%	18.8%	21.9%	21.8%
Temp	10.0%	10.0%	3.4%	4.2%	23.0%	24.0%	19.0%	17.8%
V-X	8.0%	10.0%	2.5%	2.5%	24.0%	25.0%	17.0%	18.2%
Average	**11.1%**	**11.4%**	**4.2%**	**4.8%**	**23.3%**	**23.5%**	**18.7%**	**19.0%**



(a) SZx (b) ZFP

(c) SZ3 (d) SPERR

Figure 7: Multiple domain experiments on Velocity-X data from Miranda: requested compression ratio (f), and compression ratio achieved by FXRZ (f_{FXRZ}) and CAROL (f_{CAROL})

we use 6 time steps (3GB) as training data and one later time step (0.5 GB) as testing data. Table 3 shows the *estimation error* (α) of FXRZ and CAROL, computed using formula (1). On average, across the four compressors considered, the accuracy achieved by our framework is similar to that of FXRZ (less than 1% difference). CAROL performs similarly to FXRZ or slightly better than it (up to 2% better) on Baryon Density and Temperature, and similarly or slightly worse (up to 2.5% worse) on Dark Matter Density and Velocity-X. Like FXRZ, CAROL performs better (11.4% and 4.6%) on the high throughput group (SZx and ZFP), while incurring a higher estimation error (23.5% and 19%) on the high compression ratio group (SPERR and SZ3). The main reason for this behavior is that SZ3 and SPERR achieve compression ratios on the order of thousands, and the limited amount of training data for single domain experiments cannot adequately model the details of the compression functions. However, we note that CAROL still meets our objective of not degrading accuracy when compared to FXRZ.

Multiple domain experiments: To better assess the capabilities of CAROL, we extend FXRZ's single domain experiments to include data from different datasets. We use 4 data fields from NYX, 5 data fields from Miranda, and 2 independent simulations from Klacansky (HCCI and MRS) as training data (for a total of 14.2GB) and 2 data fields from Miranda (Velocity-X and Diffusivity) as testing data. Figure 7 plots the compression function achieved by the compressors ($f(s)$) (ground truth) along with the compression functions estimated by FXRZ ($f_{FXRZ}(s)$) and CAROL ($f_{CAROL}(s)$) when testing with the Velocity-X field of the Miranda dataset. We reported similar results on Miranda's Diffusivity field. As can be seen, in all cases the compression function estimations of CAROL and FXRZ overlap completely and are close to the ground truth compression function. In particular, CAROL achieves 10%, 1.5%, 7.8% and 5.8% estimation error (α) for SZx, ZFP, SPERR, and SZ3, respectively. The lowest estimation error is achieved on ZFP, which exhibits a step-wise compression function such that multiple error bounds result in the same compression ratio. In contrast, CAROL reports the

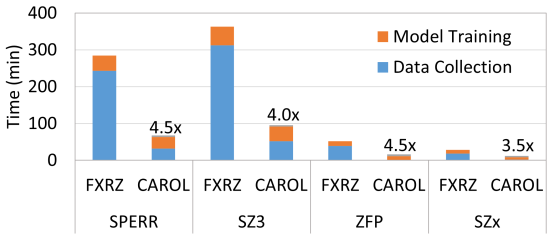


Figure 8: Training timing of FXRZ and CAROL. The numbers on top of the bars indicate the speedup of CAROL over FXRZ.

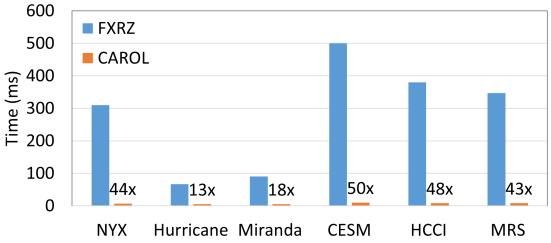


Figure 9: Feature extraction time (in ms) of FXRZ and CAROL. The forest traversal time is insignificant (~1ms). The numbers on top of the bars indicate speedup of CAROL over FXRZ.

highest estimation error on SZx. This is because SZx's byte-level delta-based compression algorithm is sensitive to changes in error bound. CAROL's predicted error bound is often within 2 decimal of the ground truth error bound; however, the resulting compression ratio is still on average 10% apart. *We note that the observed estimation error is lower for multiple domain experiments than for single domain experiments, thanks to the model's being trained with a more diverse set of data. By allowing better scaling of the model training time,* CAROL *facilitates the use of more training data.*

6.2.2 Timing distribution. Figure 8 shows the execution time (in minutes) of the model setup steps of FXRZ and CAROL on the four considered compressors. For FXRZ ,data collection is the dominant step, taking between 65% (SZx) and 85% (SPERR) of the total runtime. Our framework was able to speedup data collection by 6 to 26 times (on SZ3 and ZFP, respectively), from an average of 153 minutes to 21 minutes. In addition, CAROL reduced the model training time by 20% (for SZ3) to 45% (for SZx), from an average of 28 minutes to 21 minutes. Overall, CAROL reduced the setup time by 4.1 times. We noted that CAROL was able to achieve this speedup while utilizing only 1 CPU core, whereas FXRZ used all 36 cores.

Figure 9 shows that feature extraction time (in milliseconds) of FXRZ and CAROL. As noted before, model inference takes insignificant time (about 1 ms), so we only compare feature extraction time. Feature extraction time depends on the size of the input data (and is indepedent of the compressor). As can be seen across our benchmark datasets, CAROL's feature extraction is about 36× faster than FXRZ's. For large datasets like NYX, CESM and Klacansky, FXRZ's features extraction takes between 310 ms and 500 ms, while CAROL maintains feature extraction time lower than 10 ms in all cases. Low features extraction time allows for lower inference time.

6.3 Calibration and Estimation Accuracy

Our end-to-end evaluation demonstrates that CAROL achieves accuracy similar to FXRZ while only taking 1/4 of the setup time and 1/40 of the inference time (using a single CPU core). In this section, we discuss the main sources of speedup and prediction accuracy.

6.3.1 Secre Timing. Table 4 measure the time for the compressor to run and the time for SECRE to estimate the compression ratio. On the high throughput compressors, SECRE achieves a 14.8× and 15.8× speedup over SZx and ZFP true compression time, respectively. This speedup mainly comes from SECRE's sampling method: recall that SECRE performs compression on ≈1% of the data and uses those sampled data to extrapolate the compression ratio for the entire dataset. On the high throughput compressors, SECRE achieves 50.7× and 22.2× speedup over SZ3 and SPERR compression time, respectively. This more significant speedup comes from the fact that SECRE's surrogate compressor skips some key compression steps, such as outliers encoding for SPERR and zstd compression for SZ3. However, this speedup comes at the price of a higher estimation error. While SZx's and ZFP's estimation errors are less than 1%, SPERR's and SZ3's estimation errors can be as high as 60%. This large error introduces significant noise into the model.

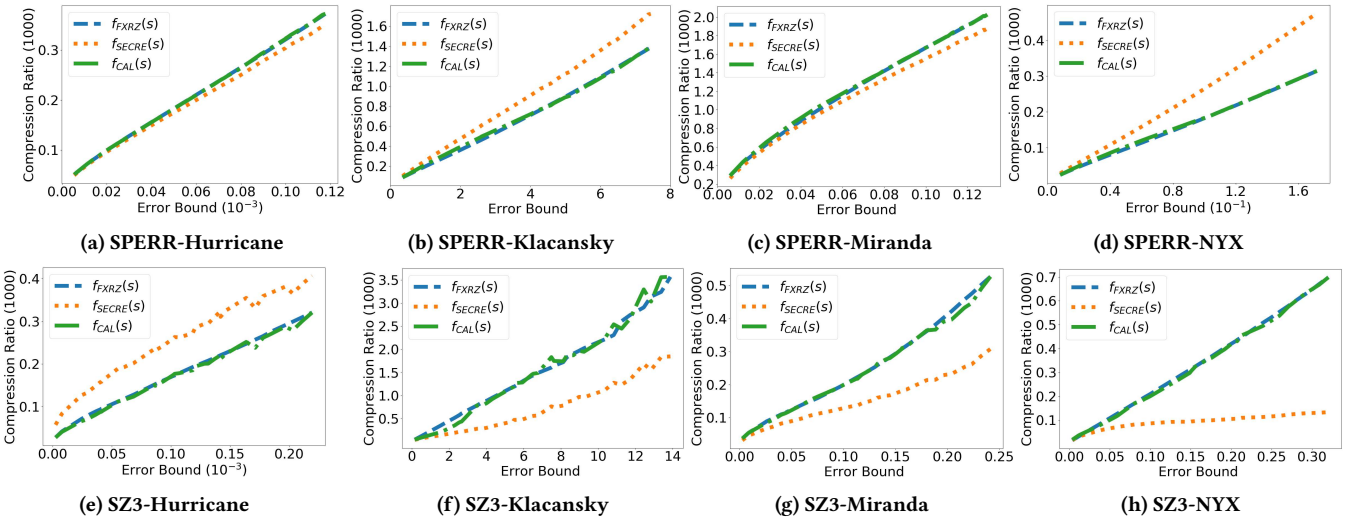
6.3.2 Calibration Timing and Accuracy. One of the design parameters of CAROL is the number of calibration points, i.e., the number of additional runs of the compressor to collect calibration data used to correct SECRE's estimation. Table 5 compares the performance and accuracy when running SECRE without calibration and with calibration, using 3, 4 or 5 calibration points. Specifically, we show the speedup reported over running the full compressor and the percentage estimation error α. We only show the results for SZ3 and SPERR because SZx and ZFP have estimation errors below 1% and do not need calibration.

The results from Table 5 show that SECRE incurs a 23% and a 47% estimation error on SZ3 and SPERR, respectively, while allowing for a 22.9× and a 48.8× speedup over running the full compressors. CAROL's calibration method using 3 additional data points reduces the estimation error to 1.8% and 7.5% while maintaining a 7.5× and 9.34× speedup over running the full compressors. As the number of calibration points increases to 4 and 5, the estimation error further decreases at the cost of a longer data collection time. From these results, we conclude that, for SPERR, three calibration points are sufficient to achieve good accuracy gains for both compressors, while 4 calibration points are enough to significantly limit the estimation error while maintaining good performance gains.

Figure 10 plots the real compression ratio, the compression ratio estimated by SECRE and the compression ratio calibrated by CAROL. As can be seen, in all cases CAROL is able to correctly determine

Table 4: Training data collection time (in seconds) using full compressor (*full*) and SECRE surrogate estimation (*est*).

	SZx		**ZFP**		**SZ3**		**SPERR**	
	Full	*Est*	*Full*	*Est*	*Full*	*Est*	*Full*	*Est*
Miranda	59	5.2	95	9	1407	27	683	27
NYX	1038	65	2243	139	17357	336	13956	632
Hurricane	54	5.4	43	4.2	1328	28	640	29
CESM	145	11	295	16	4462	94	2560	116
Klacansky	73	5.45	126	9.1	1923	37	1074	46
Speedup	**14.8×**		**15.8×**		**50.7×**		**22.2×**	



(a) SPERR-Hurricane (b) SPERR-Klacansky (c) SPERR-Miranda (d) SPERR-NYX

(e) SZ3-Hurricane (f) SZ3-Klacansky (g) SZ3-Miranda (h) SZ3-NYX

Figure 10: Compression ratio of SECRE and full compressor on viscosity data from Miranda.

Table 5: Effectiveness of calibration: speedup ($\mathcal{S}$) over running the full compressor and estimation error (α) in percentage terms.

SZ3								
	Estimation		**Estimation with Calibration**					
			3 points		**4 points**		**5 points**	
	$\mathcal{S}$	α	$\mathcal{S}$	α	$\mathcal{S}$	α	$\mathcal{S}$	α
Miranda	50×	27.2%	9.4×	2.4%	6.1×	1.4%	5.2×	1.2%
NYX	51×	56.9%	9.4×	15.5%	6×	5.8%	5.2×	4.5%
Hurricane	46×	60%	9.2×	3.26%	6×	2.5%	5.1×	2%
CESM	47×	36.8%	9.3×	4.3%	6×	2%	5.1×	1.4%
Klacansky	50×	54.6%	9.4×	11.8%	6.1×	6.6%	5.2×	5.4%
Average	**48.8×**	**47.1%**	**9.3×**	**7.5%**	**6×**	**3.6%**	**5.2×**	**2.9%**
SPERR								
	Estimation		**Estimation with Calibration**					
			3 points		**4 points**		**5 points**	
	$\mathcal{S}$	α	$\mathcal{S}$	α	$\mathcal{S}$	α	$\mathcal{S}$	α
Miranda	25.1×	7.24%	7.8×	2%	5.4×	0.5%	4.7×	0.5%
NYX	23×	38.6%	7.3×	2.7%	5.2×	0.6%	4.5×	0.5%
Hurricane	21×	16%	7.2×	0.4%	5.1×	0.2%	4.5×	0.2%
Klacansky	22.7×	31.3%	7.5×	2.1%	5.3×	0.7%	4.6×	0.5%
Average	**22.9×**	**23.2%**	**7.5×**	**1.8%**	**5.3×**	**0.5%**	**4.6×**	**0.4%**

if SECRE overestimates or underestimates the compression ratio and correctly fix the estimation error. When running SZ3 (figure 10e,10f, and 10g), using a 4 point calibration allows us to correct the unstable prediction of SECRE.

7 CONCLUSION AND FUTURE WORK

In this work we propose CAROL, a fixed-ratio compression framework that significantly improves execution time over state-of-the-art solutions. We evaluate our framework on four compressors and five real-world scientific datasets. Our results show that CAROL achieves significant improvements over state-of-the-art solutions while maintaining or improving prediction accuracy. Bellow are key insights from our work:

- *Performance*: Data collection is the most time-consuming step. Thus, its acceleration has the most impact on performance.
- *Compressor Behavior 1*: High compression ratio compressors (e.g., SPERR and SZ3) tend to incur higher estimation errors, and require calibration to achieve good estimation accuracy.
- *Compressor Behavior 2*: High throughput compressors (e.g., SZx and ZFP) tend to incur low estimation error (less than 1%), making calibration not necessary to achieving good accuracy.
- *Compressor Behavior 3*: When a surrogate model is not available, CAROL is still a viable solution, especially for high throughput compressors. In this case, full compression will be first performed on sampled data, and then our proposed calibration method will be used to reduce the estimation error. The key to an accurate estimation is that the sampling method has to match the target compressor's compression window (Table 1).
- *Model Accuracy*: FXRZ's randomized grid-search and CAROL's Bayesian optimization produce models with similar prediction accuracy (less than 1% difference on average).
- *Model Training Time and Prediction Accuracy*: Overall prediction accuracy is dependent on the quality and the diverseness of the training data. Bayesian Optimization allows for scalable training time and incremental model improvement while limiting the computational needs.
- *Feature Extraction*: To maintain low prediction time, the data sampling method used must be tailored to the hardware platform.

Future extensions to CAROL include utilizing different machine learning models and designing a feedback loop enabling on-the-fly model improvement.

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, and by the National Science Foundation under grants OAC-2003709, OAC-2104023, OAC-2311875 and CCF-1907863. We acknowledge the computing resources provided on Bebop (operated by Laboratory Computing Resource Center at Argonne).

REFERENCES

[1] [n.d.]. Event Horizon Telescope. https://eventhorizontelescope.org/. Accessed: 2023-12-11.

[2] [n.d.]. Hurricane Isabel. http://vis.computer.org/vis2004contest/data.html. Accessed: 2023-12-11.

[3] [n.d.]. Klacansky. https://klacansky.com/open-scivis-datasets/category-all.html. Accessed: 2023-12-11.

[4] [n.d.]. Miranda application. https://wci.llnl.gov/simulation/computer-codes/miranda. Accessed: 2023-12-11.

[5] [n.d.]. NYX simulation. https://amrex-astro.github.io/Nyx. Accessed: 2023-12-11.

[6] Ioannis Alagiannis, Renata Borovica-Gajic, Miguel Branco, Stratos Idreos, and Anastasia Ailamaki. 2015. NoDB: efficient query execution on raw data files. *Commun. ACM* 58, 12 (nov 2015), 112–121. https://doi.org/10.1145/2830508

[7] Jianfei Chen, Lianmin Zheng, Zhewei Yao, Dequan Wang, Ion Stoica, Michael Mahoney, and Joseph Gonzalez. 2021. Actnn: Reducing training memory footprint via 2-bit activation compressed training. In *International Conference on Machine Learning*. PMLR, 1803–1813.

[8] Yu Cheng and Florin Rusu. 2014. Parallel in-situ data processing with speculative loading. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (Snowbird, Utah, USA) *(SIGMOD '14)*. Association for Computing Machinery, New York, NY, USA, 1287–1298. https://doi.org/10.1145/2588555.2593673

[9] Jack Dongarra, Bernard Tourancheau, Franck Cappello, Sheng Di, Sihuan Li, Xin Liang, Ali Murat Gok, Dingwen Tao, Chun Hong Yoon, Xin-Chuan Wu, Yuri Alexeev, and Frederic T Chong. 2019. Use cases of lossy compression for floating-point data in scientific data sets. *Int. J. High Perform. Comput. Appl.* 33, 6 (nov 2019), 1201–1220. https://doi.org/10.1177/1094342019853336

[10] Jim Gray, David T. Liu, Maria Nieto-Santisteban, Alex Szalay, David J. DeWitt, and Gerd Heber. 2005. Scientific data management in the coming decade. *SIGMOD Rec.* 34, 4 (dec 2005), 34–41. https://doi.org/10.1145/1107499.1107503

[11] Yafan Huang, Sheng Di, Xiaodong Yu, Guanpeng Li, and Franck Cappello. 2023. cuSZp: An Ultra-Fast GPU Error-Bounded Lossy Compression Framework with Optimized End-to-End Performance. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'23)*. Association for Computing Machinery, Denver, CO, USA. https://doi.org/10.1145/3581784.3607048

[12] Yafan Huang, Shengjian Guo, Sheng Di, Guanpeng Li, and Franck Cappello. 2022. Mitigating silent data corruptions in HPC applications across multiple program inputs *(SC '22)*. IEEE Press, Article 17, 14 pages.

[13] Sian Jin, Chengming Zhang, Xintong Jiang, Yunhe Feng, Hui Guan, Guanpeng Li, Shuaiwen Leon Song, and Dingwen Tao. 2021. COMET: a novel memory-efficient deep learning training framework by using error-bounded lossy compression. *Proc. VLDB Endow.* 15, 4 (dec 2021), 886–899. https://doi.org/10.14778/3503585.3503597

[14] J. E. Kay, C. Deser, A. Phillips, A. Mai, C. Hannay, G. Strand, J. M. Arblaster, S. C. Bates, G. Danabasoglu, J. Edwards, M. Holland, P. Kushner, J.-F. Lamarque, D. Lawrence, K. Lindsay, A. Middleton, E. Munoz, R. Neale, K. Oleson, L. Polvani, and M. Vertenstein. 2015. 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* 96, 8 (2015), 1333 – 1349. https://doi.org/10.1175/BAMS-D-13-00255.1

[15] Arham Khan, Sheng Di, Kyle Chard, Ian Foster, and Franck Cappello. 2023. SECRE: Surrogate-based Error-controlled Lossy Compression Ratio Estimation Framework. In *2023 IEEE 30th International Conference on High Performance Computing, Data, and Analytics (HiPC)*.

[16] Sihuan Li, Sheng Di, Kai Zhao, Xin Liang, Zizhong Chen, and Franck Cappello. 2020. Towards End-to-end SDC Detection for HPC Applications Equipped with Lossy Compression. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)*. 326–336. https://doi.org/10.1109/CLUSTER49012.2020.00043

[17] Shaomeng Li, Peter Lindstrom, and John Clyne. 2023. Lossy Scientific Data Compression With SPERR. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 1007–1017. https://doi.org/10.1109/IPDPS54959.2023.00104

[18] Xin Liang, Sheng Di, Franck Cappello, Mukund Raj, Chunhui Liu, Kenji Ono, Zizhong Chen, Tom Peterka, and Hanqi Guo. 2023. Toward Feature-Preserving Vector Field Compression. *IEEE Transactions on Visualization and Computer Graphics* 29, 12 (2023), 5434–5450. https://doi.org/10.1109/TVCG.2022.3214821

[19] Xin Liang, Sheng Di, Dingwen Tao, Sihuan Li, Shaomeng Li, Hanqi Guo, Zizhong Chen, and Franck Cappello. 2018. Error-Controlled Lossy Compression Optimized for High Compression Ratios of Scientific Datasets. In *2018 IEEE International Conference on Big Data (Big Data)*. 438–447. https://doi.org/10.1109/BigData.2018.8622520

[20] Xin Liang, Kai Zhao, Sheng Di, Sihuan Li, Robert Underwood, Ali M. Gok, Jiannan Tian, Junjing Deng, Jon C. Calhoun, Dingwen Tao, Zizhong Chen, and Franck Cappello. 2023. SZ3: A Modular Framework for Composing Prediction-Based Error-Bounded Lossy Compressors. *IEEE Transactions on Big Data* 9, 2 (2023), 485–498. https://doi.org/10.1109/TBDATA.2022.3201176

[21] Peter Lindstrom. 2014. Fixed-Rate Compressed Floating-Point Arrays. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (2014), 2674–2683. https://doi.org/10.1109/TVCG.2014.2346458

[22] Md Hasanur Rahman, Sheng Di, Kai Zhao, Robert Underwood, Guanpeng Li, and Franck Cappello. 2023. A Feature-Driven Fixed-Ratio Lossy Compression Framework for Real-World Scientific Datasets. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. 1461–1474. https://doi.org/10.1109/ICDE55515.2023.00116

[23] Seung Son, Zhengzhang Chen, William Hendrix, Ankit Agrawal, Weikeng Liao, and Alok Choudhary. 2014. Data Compression for the Exascale Computing Era - Survey. *Supercomput. Front. Innov.: Int. J.* 1, 2 (jul 2014), 76–88. https://doi.org/10.14529/jsfi140205

[24] Robert Underwood, Sheng Di, Jon C. Calhoun, and Franck Cappello. 2020. FRaZ: A Generic High-Fidelity Fixed-Ratio Lossy Compression Framework for Scientific Floating-point Data. In *34th IEEE International Parallel and Distributed Processing Symposium*. IEEE, New Orleans.

[25] Xiaodong Yu, Sheng Di, Kai Zhao, Jiannan Tian, Dingwen Tao, Xin Liang, and Franck Cappello. 2022. Ultrafast Error-Bounded Lossy Compression for Scientific Datasets *(HPDC '22)*. Association for Computing Machinery, New York, NY, USA, 159–171. https://doi.org/10.1145/3502181.3531473

[26] Kai Zhao, Sheng Di, Xin Lian, Sihuan Li, Dingwen Tao, Julie Bessac, Zizhong Chen, and Franck Cappello. 2020. SDRBench: Scientific Data Reduction Benchmark for Lossy Compressors. In *2020 IEEE International Conference on Big Data (Big Data)*. 2716–2724. https://doi.org/10.1109/BigData50022.2020.9378449