LibPressio-Predict: Flexible and Fast Infrastructure For Inferring Compression Performance

Robert Underwood
runderwood@anl.gov
Argonne National Labratory
University of Chicago
Lemont, Illinois, USA

Sheng Di
sdi1@anl.gov
Argonne National Labratory
University of Chicago
Lemont, Illinois, USA

Sian Jin
sianjin@iu.edu
Indiana University
Bloomington, Indiana, USA

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

Arham Khan
arham@uchicago.edu
University of Chicago
Chicago, Illinois, USA

Franck Cappello
cappello@anl.gov
Argonne National Labratory
University of Chicago
Lemont, Illinois, USA

ABSTRACT

Over recent years, substantial efforts have gone into developing systems to infer compression performance without running compressors. These efforts have driven down the error in the estimates, reduced their runtimes, and improved their generality. However, these efforts are uncoordinated increasing the efforts required to perform comparisons between them. There may be subtle differences in sampling approaches, and nuances to the interfaces requiring efforts to port applications between them and to reproduce experiments. Additionally, many of these methods call for substantial amounts of training data to produce reliable estimates, as well as scalable codes to perform the training. In this work, we present LibPressio-Predict – a scalable library for use in applications using predictions of compression performance and a scalable tool LibPressio-Bench to run these experiments quickly at scale. We use this tool to evaluate 3 recent compression prediction approaches systematically with all 48 timesteps and 13 fields from the Hurricane Issable dataset.

CCS CONCEPTS

• **Theory of computation** → **Data compression**; • **Computing methodologies** → Machine learning; *Massively parallel and high-performance simulations.*

KEYWORDS

Lossy Compression, Prediction, HPC, I/O Optimization, SZ, LibPressio, ZFP

ACM Reference Format:
Robert Underwood, Sheng Di, Sian Jin, Md Hasanur Rahman, Arham Khan, and Franck Cappello. 2023. LibPressio-Predict: Flexible and Fast Infrastructure For Inferring Compression Performance. In *Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W 2023), November 12–17, 2023, Denver, CO, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3624062.3625124

1 INTRODUCTION

Increasingly lossy compression is being used as an optimization for I/O and memory operations in large high-performance computing (HPC) and AI applications[3, 21, 22]. For some of these applications, knowing the compression ratio, compression bandwidth, or some other metrics in advance of running the compressor would be very helpful. Existing work has shown that auto-tuning workflows such as OptZConfig[18] or choosing the best compressor among a group of two or more compressors [15], or writing to a shared file in parallel [6] all benefit from being able to quickly estimate the performance of compressors. Each of these applications places different demands on the prediction method: some need extremely fast but tolerate inaccurate predictions, and others can actually be somewhat slow relative to the time of invoking a compressor the same number of times provided the estimates are very accurate, and still other uses benefit special capabilities like error bounds on the estimate [2] or the ability to do counterfactual estimation for different compressor architectures.

In response to such diverse demands, many different estimation strategies have been developed each with distinct advantages and disadvantages [2, 6, 9, 15, 17, 20]. However, integrating these methods into applications or libraries often written in C++ or evaluating their individual strengths can be challenging. Many of these compression prediction schemes were implemented in mixes of programming languages, with inconsistent APIs, and subtle differences in implementation that make it difficult to correctly implement a comparison between these methods or for applications and libraries that wish to use these methods to adopt them without substantial code changes as methods evolve and implementations improve. This is largely the situation that motivated the introduction of MPI to standardize message passing implementation [1] in 1992, and the development of LibPressio for invoking lossy compressors in 2019 [16].

This work introduces a set of tools to address this challenge: LibPressio-Predict a lightweight and extendable framework for describing, implementing, and utilizing methods to predict compression performance; and LibPressio-Predict-Bench, an extendable, high-performance, embeddable, scalable, and resilient infrastructure to evaluate these prediction schemes. We use this infrastructure to evaluate some of the leading compression ratio prediction schemes on several different compressors and datasets.

This work aims to answer the following key challenging questions:

(1) How to generically enable maximum reuse of previously observed metrics in predictions to reduce the computational overhead as much as possible for using these methods for training or inference, especially in the case of interactive prediction scheme development on multiple compressors?
(2) How to minimize the I/O, compute, and memory overheads associated with training on large multi-terabyte datasets that do not easily fit in memory on any one node?
(3) How to best address fault tolerance in the distributed training process and make these kinds of faults recoverable?
(4) How to describe and refine the needs of use-cases of compression performance inference to metrics that can be used for comparisons to allow users to easily find estimation methods that are suitable for their needs?

The remainder of this work is organized as follows: In Section 2 we discuss applications of compression performance prediction and the methods used to implement this prediction. Then in Section 3 we describe why various existing systems such as ML libraries, workflow systems, and dataloaders alone are ill-suited for this task motivating the need for an integrated approach. After that in Section 4 we present the architecture of our approach providing an overview of our approach to these challenges. Section 5 then goes into the details of the implementation and demonstrates how to add a prediction method to LibPressio-Predict. Lastly, we evaluate how some of our optimizations lead to high-performance training and inference and conclude with an example evaluation of a few leading compression prediction methods available in LibPressio-Predict. We then conclude with future work in the compression ratio prediction space and call for contributions to our library from the community.

2 BACKGROUND

In this section, we describe the research background in two facets: existing applications of compression prediction and existing prediction methods of compression quality (such as compression ratio).

2.1 Applications of Compression Prediction

There are several possible tasks for which being able to predict compression ratios or other metrics is useful. Some of these uses include quickly and automatically determining a configuration for a compressor that projects a good performance while possibly meeting particular criteria [13, 18, 19], selecting the best-performing compressor quickly [15], or even accelerating writes to shared files such as compressed chunks in HDF5 [5, 6]. In this section, we elaborate on the use cases and the demands that they place on prediction schemes as documented in prior work [2].

One of the earliest use cases of using compressor performance prediction was choosing the best performing among a group of lossy compressors [15]. In this case, predictions are used as a replacement for running the compressors. This method needs to be fast but actually does not need to be tremendously accurate since it needs to only preserve the ranking of the best-performing compressor or group of compressors [2]. This is especially true if some of the metrics only have to be computed once for a wide variety of different predictions – a condition that we refer to as invalidation.

Quickly determining a compressor configuration that has high performance (e.g. compression ratio, compression throughput) while meeting a user's quality requirements is a common request among users[18]. One such example was the work by Rahman et al [13] which uses features of a dataset to quickly predict configurations that are likely to meet user requirements in this case the work primarily used random forests used to predict the compression ratio. However, in these use cases, the accuracy of the estimation is very important while the speed is only moderately important [2], especially for methods that are robust to invalidation.

One of the most recent use cases for compression performance prediction is accelerating writing to distinct offsets of shared files in parallel such as in HDF5 Parallel Write [6]. In this use case, predictions are used to precompute the offsets that are used and can fall back to appending the data in the event of mispredictions. Additionally, a safety factor can be used that will over-allocate storage and in turn decrease the chances of under-allocation. Additionally, one method by Ganguli [2] offers statistical bounds on the compression ratio estimation error allowing precise forecasting of the number of mispredictions. For these reasons, this method also does not need to be extremely accurate however it does need to be extremely fast relative to the time to invoke the compressor as there are few opportunities for invalidations.

Another particularly compelling use case is offered by Wang et al[20]: counterfactual analysis to predict the performance of compressors that do not yet exist. This is extremely useful in the effort of compressor design where hundreds of person-hours go into the design, testing, and evaluation of specialized lossy compressors for particular applications. If a prediction scheme can show with some confidence that a particular method will ultimately prove unfruitful for a particular application, it can be discarded early in the design process.

In short, different applications require different traits and capabilities of the prediction schemes that work best to accelerate them. Therefore there needs to be a variety of prediction approaches to best meet the needs of different applications, and different metrics to evaluate their suitability.

2.2 Methods

The desire to estimate and compute bounds for compression ratios and other compression statistics are not new. Shannon famously defined the entropy [14] which provides an upper bound on the compressibility of data for lossless compressors. In the years since Rate-Distortion theory has developed as a way to provide strong bounds on the compressibility of datasets. However, rate-distortion theory often has to make restrictive assumptions that make it ill-suited for estimating the compressibility of many scientific datasets and modern lossy compressors. To compensate for this many schemes have been designed to estimate the performance of compressors.

Some of these methods have training steps where models are fine-tuned for particular datasets to increase accuracy, while others take a more formulaic approach and forego training.

One of the earliest methods to predict compression ratios for modern lossy compressors was mentioned by Tao [15] and expanded on in [10]. It uses the average compression ratio for a particular compressor of blocks sampled from the input dataset. The performance of this method scales with the performance of the compressor, and for an appropriate sample count and block size (which in the original design was based on the internals of compressors), this can be a lightweight method to estimate compression ratios, however, it is not very accurate. However, for the use case for which it was proposed, it only needs to be accurate in determining which compressor performs the best.

The method by Krasowska [9]. This unlike the previous method uses no internals of the compressor and instead relies on two statistics of the input data: the quantized entropy and the local variogram which were fitted with a simple trained linear regression to samples of the data. This method was a substantial step forward in the prediction of compression ratios in that it was the first not to use any compressor internals beyond the notion of absolute error and proved far more accurate than prior sampling-based methods.

Beginning in 2022, there was an explosion of interest in methods to estimate compression ratio and other aspects of compression performance. An evolution of Krasowska [9] appears in Underwood and Bessac [17]. In this version, the variogram was exchanged for the truncation of the singular value decomposition (SVD) because of the availability of high-performance implementations of the SVD on which it is based and the statistical properties of the SVD which make it well suited for measuring the global amount of spatial information. This method also replaced the simple trained linear regression with a more sophisticated cubic spline regression. Together these improvements demonstrated vast speedups and improvements in accuracy on a variety of applications however even with highly optimzized GPU implementations of the SVD, it is unsuitable for applications that depend greatly on the speed of the prediction.

A related approach developed by Ganguli[2] uses a combination of three bespoke metrics (spatial correlation, spatial diversity, and spatial smoothness) and two existing metrics (coding gain and general distortion). This approach uses a trained mixture model and conformal prediction to both increase the robustness of statistical approaches but also to provide strong guarantees on the error to achieve unparalleled in-sample and out-of-sample prediction accuracy.

Around the same time ZPerf by Wang [20], introduced the capability to perform counterfactual analysis of the compression performance. It accomplished this by decomposing the stages that are common to compressors [3] in the formulation of compression performance and using estimates for each of the stages. This method was ultimately built on predictors largely derived from internals to the compressors from earlier papers from the same group [11, 12] which used Gaussian process modeling and deep neural networks respectively.

A related, but independently developed method SECRE [7] likewise takes the approach of modeling the various stages of the internals of the compressor but combines this with tightly coupled sampling and applies it to two additional compressors SZx (a high throughput version of SZ), and to SPERR a leading compressor based on wavelets.

FXRZ [13] combined primarily random forests with a series of facially neutral predictors (but closely aligned with predictors in SZ) to quickly estimate the compression ratio of various datasets. A key innovation of this work was to perform data augmentation – artificially accumulating additional training data by interpolation between observed values. This brought down the training cost for this class of model substantially while obtaining accuracy competitive with other approaches such as [18].

Jin [5] proposes a numerical model for prediction-based lossy compression. It decomposes the compression process into three stages: prediction, quantization, and encoding. By offering theoretical analysis encompassing Huffman encoding efficiency and subsequent lossless encoding efficiency, along with estimating the quantization code distribution based on prediction-quantization design, it can accurately predict the compression ratio of SZ. This theoretical foundation also enables its application in other areas, such as predicting compression time and I/O time [6].

We summarize the prior prediction methods in Table 1. However, with all of these methods, users have a challenging task to 1) evaluate which of these nearly a dozen methods is best for their use case and 2) integrate it efficiently train them for their problems, and how to efficiently integrate them into their applications in a way that allows them to adopt new methods quickly and compare them.

3 RELATED WORK

Our work holistically evaluates a series of prediction schemes. To do this efficiently, we need a collection of different types of components that are tightly integrated to offer high performance each of which has key similar work. As a result of the limitations of many of these prior approaches, most prior papers evaluating predictors for compression rely on bespoke programs [9, 17] that may or may not have been distributed using MPI.

One of the first steps of training a predictor is identifying, implementing, and evaluating at-scale a large number of data samples. The codes may be a mix of CPU and GPU and may be implemented in a variety of languages. In all likelihood, they were implemented using completely incompatible interfaces as were all the prediction schemes we considered in our work. Regardless of which other choices one makes to train and estimate using these prediction schemes you use to implement these metrics and schedule them, all of them will require a way to encode the different invalidation requirements for prediction schemes.

In other domains workflow systems such as SLURM, HT Condor, Swift-t, Pegasus, Legion, or Apache Hadoop/Spark. These systems generally allow the execution of graphs of tasks with various dependencies. Many of these are complex requiring extensive configuration making them difficult to embed into part of a larger C++ application. Embeddability is desirable because of wide variety of use cases of compressors from different contexts from which libraries like LibPressio are called – e.g. HDF5 plugins, high-level programming languages such as Julia, R, or Python, and in-situ embedding within scientific applications written in languages such

Table 1: Estimation Methods

method	training	sampling	black-box	goal	metrics	approach	features
Tao [15]	×	✓	~	fast	CR	trial-based	
Krasowska [9]	✓	×	✓	accurate	CR	regression	
Underwood [17]	✓	×	✓	accurate	CR	regression	
Ganguli [2]	✓	×	✓	accurate	CR	regresison	bounded
Jin [5, 6]	✓	×	×	fast	CR,Bandwidth	calculation	
Khan [7]	×	✓	×	fast	CR	calculation	
Rahman [13]	✓	✓	~	fast	various	machine learning	
Lu [11]	✓	✓	×	accurate	CR	regression	
Qin [12]	✓	✓	×	accurate	CR	deep learning	
Wang [20]	✓	✓	×	accurate	CR	calculation	counterfactuals

as C,C++, or Fortran – which may not easily or may not wish to configure these large complex systems alongside their codes and prefer a library-based approach and benefit from the ubiquity of support for components with a C foreign function interface such as by LibPressio and LibPresiso-Predict. Additionally, most of these systems do not feature the ability to coordinate I/O between tasks or in some cases lack the ability to allocate jobs primarily based on the availability of data or locations of cached resources. This is especially important because the datasets used for compression are large. Being able to use optimizations such as collective I/O, local caching, and neighbor caching is critical to achieving high performance of these codes at scale. Lastly, some of these systems lack or have administratively restricted the ability to dynamically add dependencies to currently running jobs as invalidations require additional computation. They may also lack the ability to checkpoint and restart the execution graph as it is executing to recover from failures (either due to bugs in the metrics implementation or hardware failures) that become more prevalent at scale with many diverse datasets. During our implementation, we observed many software faults in prediction schemes surfaced by the variety of data considered in our testing that we resolved over the course of our implementation. These faults required us to re-run our experiments, however, fine-grained checkpoint restart allows us to re-run only the affected results quickly.

Within the realm of AI, there are specialized data loaders (e.g. Nvidia's Dali) and prefetching systems (e.g. NoPFS [4]), but these tend to be highly specialized for the use case of training AI models and try to prefetch mini-batches of datasets to the GPU which may or may not be the case of particular metrics' and compressors' implementations in data compression.

There are many libraries for the training of complex models themselves. In R, libraries such as `car` or `fixest` are frequently used, and in Python `scikit-learn` and deep learning frameworks like `pytorch` and `tensorflow`. These tend to be non-trivial to integrate in C++ applications which are widely used in HPC. Libraries that are embeddable like Dlib [8] tend to lack features of these more robust solutions available in Python or R. However, all of these require some external system to provide the data that is itself used for training and inference and require the user to implement a checkpoint restart scheme on top of them if it is supported at all.

Answering these questions allows us to perform these comparisons more fairly and consistently across methods and evaluate methods on the axis of time, accuracy, and robustness.

4 OVERVIEW OF OUR APPROACH

The task of efficiently estimating compression ratios and other compressor performance metrics generally has several key steps that need to be implemented efficiently and tightly integrated to achieve high performance. First, the data or particular subsets of the data must be loaded and in some cases preprocessed with careful attention to hardware capabilities, parallelism, and memory limitations. Second, the various metrics must be computed or if appropriate loaded from a cache that will be processed to form estimates, and due to the volume of data, this often needs to be done in parallel at scale. Third, there may be a training process that fits the parameters of the method to the specific application or compressor. Finally, the trained predictor needs to be used on new datasets which will involve careful attention to how the application invalidates the pre-computation done prior to the training phase.

Our solution builds upon LibPressio which provides generic abstractions for compressors and metrics and divides this process into three main components: **libpressio-dataset** which handles hardware-efficient and memory-conscious sampling, loading, and preprocessing of datasets; **libpressio-predict** which handles the tracking of invalidation of metrics [1] and provides a consistent interface to training and inference; and **libpressio-predict-bench** which addresses the challenge of scaling the training process to many nodes in a resilient fashion. Together, these components are tightly integrated to dramatically simplify the process of developing and using predictors of compressors' performance. A sketch of their primary interactions can be found in Figure 1.

We will utilize this framework to evaluate a collection of existing compression techniques on the basis of both accuracy and runtime to understand their suitability for various applications.

4.1 LibPressio-Dataset

LibPressio-Dataset provides the abstraction for loading, sampling, and preprocessing data efficiently to nodes. The primary abstraction

[1]When a metric needs to be recomputed because the configuration of the compressor or some other condition has changed requiring a reobservation of the metric. See Section 2 for examples

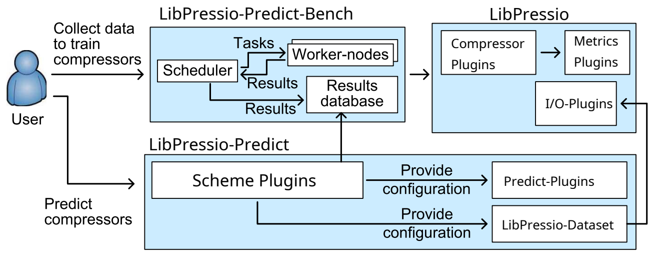

Figure 1: Architecture Sketch: The user can interact either with LibPressio-Predict to use compression performance estimation methods, or they can interact with it using predict-bench to allow training prediction methods at scale or simply to evaluate predictors for their use case

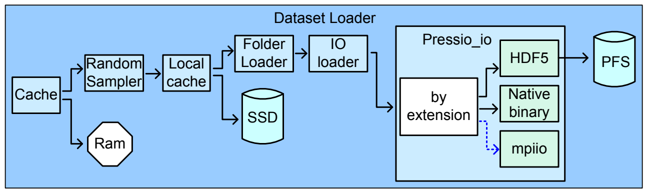

Figure 2: Sketch of a possible Dataset Loader pipeline. This pipeline uses multiple levels of cache to take advantage of deep memory tiers on modern supercomputers. Stages of the pipeline can be swapped out for hardware efficient versions such as replacing POSIX IO with MPIIO

is the `dataset_plugin` which has 4 primary methods `load_metadata`, `load_data` which load information such as shape, size, type, quality and the full data respectively, and variants `load_metadata_all` and `load_data_all` to do this for all datasets which may be able avoid repeated calls certain otherwise heavy operations and allow the batching of queries. There are also APIs to configure them, query metrics about them, and other common operations.

Like LibPressio compressors, `dataset_plugins` can be stacked to implement more complex data loading functionality as well as be extended by users without modifying the libpressio-dataset library to implement optimizations for specific hardware, to add other types of preprocessing specific to a particular case, or add new data sources.

Figure 2 shows one such load pipeline. Here the `io_loader` plugin uses a libpressio `io_plugin` to load a specific dataset using an implementation-specific method to load each file based on its file extension (e.g `.bin` uses `fread` where as `.h5` uses `H5Dread`), and can be configured to use hardware optimized loaders such as mpiio or parallel hdf5 instead. The `io_loader` is instructed to load specific datasets by the `folder_loader` which walks directories to load all of the datasets that match a pattern and attaches metadata to them about the files from which file each dataset came. As files are loaded onto each node, they can be cached onto the node's local SSD to enable faster restart times by a `local_cache` plugin which could be aware of node relationship and data placement. The results of this process could then be randomly sampled. Because job configuration only requires the metadata, operations like sampling can even appear near the end of the pipeline and still be implemented efficiently as the dataset loader can track datasets back to the individual flies that loaded them. Lastly, datasets themselves can leverage capabilities within LibPressio's core to move data to the appropriate device (CPU or GPU) once it is local to the node – future work should be able to allow this to be transparently replaced with a GPUDirect-like feature in the future and supported by the system.

4.2 LibPressio-Predict

LibPressio-Predict consists of 3 key components 1) additional LibPressio metrics modules that implement metrics specific to compression performance prediction 2) new `predict_plugins` that allow fitting and predicting compression ratios and other performance metrics, and 3) new `scheme_plugins` that enable users to quickly to determine what metrics need to be computed for a given prediction scheme based on the list of invalidations and configure a `predict_plugin` to use these metrics.

In addition to the new prediction schemes that implement the metrics from the papers that we compare, we've also added a new metadata field `predictors:invalidate` that allows both users and LibPressio-Predict to determine when a metrics result is no longer useful and implemented it for the existing metrics modules. If a compressor depends on a particular compressor setting such as `sz3:lorenzo` it can name that setting directly, but it can also use one of four special keys: 1) `predictors:error_dependent`, 2) `predictors:error_agnostic`, 3) `predictors:runtime`, and 4) `predictors:nondeterministic`[2]. Error dependent allows the user to specify that a particular metric is sensitive to errors in the data without having to exhaustively list the error metrics for a particular compressor. Error agnostic means the opposite, no errors can ever affect the results of this prediction. Runtime means that a metric depends on runtime factors that might change if performance-related settings are changed. Nondeterministic relates to a metric that the user may wish to run this metric multiple times or with different seeds to get an accurate metric – this will typically include runtime metrics, but also includes things like randomized SVD implementations. If a metric contains multiple different kinds of metrics (such as `error_stat` in the LibPressio library, these can also be listed under the particular type such as `predictors:error_dependent` or `predictors:error_agnostic`. Lastly, because we build on LibPressio Metrics, we can also utilize its external metrics framework to write new metrics in other languages to reuse existing code as much as possible. The interface to doing so is found below in Figure 3. Most typically users will provide error-agnostic metrics by overloading begin_compress_impl, and error-dependent ones by also overwriting end_decompress_impl and provide the response by get_metrics_results.

The `predict_plugin` is heavily inspired on the BaseEstimator from SciKit-Learn and has two primary methods `fit` and `predict` which train and perform predictions respectively. The API also requires that the state of the predictor be serializable and configurable like other LibPresio objects. LibPressio predicts currently provides two modules, the first is for "simple" methods where the prediction

[2]There is one more special key `predictors:training` which is used only by the user or framework for reporting invalidations, but is not listed in this option by a metric

```
#include <libpressio_predict/ext/cpp/predict.h>
class example_metric_plugin : public libpressio_metrics_plugin {
  /*hooks*/
  int begin_compress_impl(pressio_data const* in, pressio_data
↪   const*) ;
  int end_decompress_impl(pressio_data const* in, pressio_data const*
↪   out, int rc) ;

  /*results*/
  pressio_options get_metrics_results(pressio_options const &);

  /*configuration and metadata*/
  int set_options(pressio_options const& opts);
  pressio_options get_options() const;
  pressio_options get_configuration() const;

};
```

Figure 3: Major Functions in the Metrics API. Users can provide metrics using C++, or they can opt for the external metrics framework of LibPressio [18] at the cost of some overhead. Error dependent metrics will need to implement both begin_compress_impl and end_decompress_impl, and then get_metrics_results. Some metrics will implment get and set options to be parameterizable

comes directly from a metric without a training stage, and one that allows providing predictions from modules written in Python such as those that use Tensorflow, PyTorch or SciKit-Learn using an embedded interpreter with an optimized code path to share memory between the two. Like other components in LibPressio, this can be extended for example to include prediction schemes from C++, R, or Julia without modification to the core libpressio-predict library.

Lastly, the `scheme_plugin` facilitates the efficient combination of metrics with predictors without requiring users to fully know the details of these schemes and to introspect them. A code example that uses can be found in Figure 4. First, a user retrieves a scheme from the libpressio_predict registry. The registry also allows the user to enumerate the available schemes and retrieve individual predictors when developing a new scheme. With the scheme, they get a corresponding predictor for a particular compressor. This allows indicating an error if it is not supported in its current configuration. Next, they can re-load the results of prior training into the predictor. After the predictor is ready, the user indicates which if any entries may be invalidated here they choose: `pressio:abs` and all error agnostic metrics. Here we can also include a special key `predictors:training` which can be used to indicate that we should evaluate additional metrics required for training the predictor. error dependent appears in the list of metrics in addition to `pressio:abs` in case the scheme is unaware of how to specifically handle absolute error bounds, but can be ignored by the implementation since it recognizes a more specific option that affects the error and there are no others that it does not recognize. From this, they can get a metrics evaluator for the metrics that have been evaluated, and invoke it on the data buffer that has been loaded. Lastly, prediction is performed on the metrics resulting from the evaluation, and the results are returned.

```
#include <libpressio_predict.h>
using namespace std;
double predict(compressor* comp, pressio_data* data)
{
    pressio_predict plib;
    auto scm = plib->get_scheme("tao2019");
    auto pred = scm->get_predictor(comp));
    pred->set_options({
        {"predictors:state", prior_state}
    });
    string invs[] = {
        "pressio:abs",
        "predictors:error_dependent",
        "predictors:error_agnostic"
    };

    auto eval = scm->req_metrics_opts(invs);
    eval->set_options(comp->get_options());
    eval->compress(data, compressed);
    if(scm->do_decompress()) {
        eval->decompress(compressed, decompressed);
    }

    pressio_data results;
    pred->predict(
        extract(
            eval->get_metrics_results(),
            scm->req_metrics()
        ), &results);
    return *(double*)(results.data());
}
```

Figure 4: Inference Usage Sketch. Users get a reference to the prediction scheme that they wish to use. For this they can get a predictor for that metric. The predictor may require training, in this case, we load the predictor's state from a variable. Next, we ask the scheme what needs to be computed to use the predictor, then we can use the predictor to actually preform the prediction. The use of invalidations allows avoiding recomputing values where appropriate

4.3 LibPressio-Predict-Bench

Lastly, LibPressio-Predict-Bench combines these various components into a single scalable distributed system for both training models and evaluates them on standard metrics using a k-fold validation [3]. It provides a system for distributing the tasks, scheduling them, and check-pointing the results as they complete.

Configuration is handled via LibPressio object introspection which allows automatically converting the configuration flags into options structures for both the compressor and the dataset.

Checkpointing is enabled via an embedded SQLite database. A database was chosen both because of atomicity guarantees in the case of failures – no accidental partial results – but also the ability to query and partially restore the key state – the metrics results. Computing the metrics frequently dominates the time required to train and predict models of compressibility and from experience tends to be the most fault-prone and thus is the portion most in need of checkpoint restart capabilities.

[3]A common statistical technique where data is partitioned into k chunks. k-1 of them are used for training and 1 is used for validation. Training and validation are performed k times once with each chunk in the validation set. Often prediction quality metrics are only considered for the various validation sets

However, how is the state indexed in the database? We introduce a new capability into LibPressio to hash option structures with a fast cryptographic hash. Unlike the hash functions in standard libraries which make no guarantees about the hash stability of objects between executions, cryptographic hashes provide this stability. The option structure is walked in a deterministic order, and all entries with a consistent value are hashed [4] We compute these hashes once upfront before execution begins to avoid the overhead of repeated hashing. We then use these hashes to index into the database. Individual results are uniquely identified by their compressor configuration, dataset configuration, experimental metadata, and replicate ID.

As data loading times tend to dominate task runtimes for most compressors [5] even with our optimized data loading scheme on the first load, we attempt to schedule as many jobs with the same data to the same workers when they are available. More advanced scheduling could be implemented with small modifications to LibDistributed a distributed MPI-based task queue developed for use in OptZConfig [18]. When multiple workers are not available, we can fall back to single-node processing.

5 METHODOLOGY

For our evaluation, we ported three of the leading predictions to use LibPressio-Predict. Specifically, we implemented Rahman 2023 [13], Khan 2023 [7], and Jin 2022 [5, 6] described in Section 2.2.

We will evaluate metrics schemes along two axes: time and quality. For time, we will record the mean and standard deviation of the error-agnostic time, the error-dependent time, the training time, the fit time, and the inference time on various field and timesteps of Hurricane. These stages correspond to the time required to: (1) **Error-agnostic time** – compute predictors whose values are completely independent of the configuration of the compressors (2) **Error-dependent time** – compute predictors whose values are dependent on compressors settings that effect the error allowed by the compressor (3) **Training time** – compute predictors whose values are only required for training, but not inference – most commonly running the compressor (4) **Fit Time** – refers to the time required to fit the model to trained data (5) **Inference Time**– infer a single value. If a predictor is exactly the value of a metric this is N/A. Not all prediction schemes have each of these stages. These timings correspond to entries in the formula available in [2] which can be used to consider the speedups for various applications.

Next, we consider metrics, datasets, and compressors. For quality, we will use the Median Absolute Percentage Error (MedAPE) used in [2, 9, 17]. MedAPE computes the median of the absolute value of the percentage errors and is robust to outliers and the scale of the metrics. For datasets, will consider multiple fields and timesteps on the Hurricane dataset from [23]. This dataset represents a variety of different data structures and most importantly sparsity patterns. Sparsity patterns have substantial effects on the compressibility of datasets. We will consider both a conservative (1e-6) and a more liberal (1e-4) absolute error bound (`pressio:abs`) as there can be different compression ratios for each[6]. We only consider SZ3 and ZFP in this work as these two compressors are two of the most commonly used compressors and are best supported by the compression prediction schemes we evaluate. Blackbox methods such as [2, 9, 17] support many more compressors, but due to time constraints, we were not able to include these in our analysis.

6 RESULTS

In this section, we use our tools to conduct a systematic evaluation of the prediction schemes. We organize our results as follows. First, we will look at the compressor performance as a baseline. Then we will turn our attention to the runtime performance of the estimation methods. Lastly, we turn our attention to the prediction quality. Table 2 summarizes our key findings.

As a baseline, what do the compressors achieve? The runtime of SZ3 on this entire dataset (all fields and timesteps) averaged 322.8 ± 30.1 ms whereas decompression averages 101.98 ± 26.72. ZFP tends to be faster on the entire dataset and averages 65.49 ± 25.33 ms whereas decompression averages 33.86 ± 16.21. This is the number that sampling methods aim to defeat. Why estimate compression ratios if you can run the compressor to determine them faster and have higher accuracy? This does not mean that estimation methods need to be faster than a single run of the compressor. Methods like [2, 9, 17] all leverage the ability to compute a subset of error-agnostic metrics up front, and then use them to conduct many different predictions in order to achieve speedups over running the compressors multiple times to get each of these observations.

What about runtime? We see that both Khan and Rahman have error-dependent, error-agnostic, and inference times that are substantially below the compression time. We found that Jin took longer than we expected and longer than the execution of the compressor. When profiling the Jin results revealed a substantial amount of overhead associated with the management of C++ shared pointers in the multi-dimensional iterator code used in the implementation. While the original implementation also used shared pointers, the optimizer was unable to elide as much more of this code in the original and resulting in much higher overhead.

Other works that use training such as [17] are competitive in terms of their error-dependent metrics with less than 43ms. However, this work requires the computation of the SVD truncation which takes closer to 771ms making it suitable for cases where multiple compression operations are performed on the same data.

Next, what we can observe about the quality of the predictions? Before we present these results, it is important to note that these results represent a kind of worst-case scenario for prediction. First, Hurricane features a mix of sparse and dense data fields. Sparse fields can be substantially more compressible than dense fields and prediction methods need a tool to compensate for this diversity to handle this. Next, unlike most prior work in this space, we focused on the ability to perform out-of-sample prediction – that is prediction on a wide variety of datasets instead of datasets that are largely heterogenous rather than relying on similarity of fields.

[4]We exclude `void*` objects which are used to store objects like CUDA streams and MPI_Comm objects. Generally, libpressio objects provide other optional parameters to recreate these objects as needed

[5]MGARD and TThresh being notable exceptions

[6]In principle a value range relative error bound which would have normalized differences between fields could be used, but not all compression prediction codes are accurate with this use

Table 2: Hurricane Performance Results using a 10-Fold Cross-Validation. Timing results are shown ± standard deviation

	method	Error-Dependent (ms)	Error-Agnostic (ms)	Training (ms)	Fit (ms)	Inference (ms)	Compression/ Decompression (ms)	MedAPE (%)
sz3							322.8 ± 30.1/101.98 ± 26.72	
sz3	khan [7]	5 ± .47	N/A	N/A	N/A	N/A		232.57
sz3	sian [6]	518 ± .43	N/A	N/A	N/A	N/A		25.88
sz3	rahman [13]	N/A	7 ± 0.51	322.8 ± 30.1	370.34 ± 14.90	0.135 ± 0.0438		20.20
zfp							65.49 ± 25.33/33.86 ± 16.21	
zfp	khan [7]	5 ± .47	N/A	N/A	N/A	N/A		381.12
zfp	sian [6]	N/A	N/A	N/A	N/A	N/A		N/A
zfp	rahman [13]	N/A	7 ± .51	65.49 ± 25.33	360.49 ± 14.98	.09 ± .04		13.86

Now what is the estimation error? The best error we observe is with a training-based approach: Rahman with a 20.20% error for SZ3 and likewise a 13.86% error for ZFP. We attribute the vastly superior performance to the sparsity correction factor it uses[13]. Likewise, we would expect methods like [2] to also do well in this use case because this method uses a mixture regression approach to reduce the error as is reported in their paper which reports a worst-case error of less than 12.5% in a similar but not quite identical case with an out-of-sample prediction for a subset of hurricane data. However, Rahman achieves a much lower training and error-agnostic+error-dependent time than this work.

The best non-training-based method that achieves a MedAPE is Jin for SZ3 and Khan for ZFP. These approaches tend not to estimate as well as the training-based methods for a few reasons: 1) These methods tend to suffer on datasets with different degrees of sparsity. This is intuitive because when they sample the data there is no guarantee that they sample the portions of the data that are representative of the compressibility of the dataset – especially when there is a relatively small region that is highly uncompressable. 2) These methods tend to suffer even further because it is difficult to accurately estimate the compressibility of data after quantization and prediction. The method by Jin does so well on the SZ3 compressor because in part it uses parts of the first few stages of the SZ3 compressor and excludes the more expensive stages encoding stages, and it does not need to solve as challenging a problem to estimate the compressibility of what remains and can build off of existing knowledge of rate-distortion.

7 CONCLUSION

In this work, we discuss and provide a set of tools that enables efficient use and comparison between different compression prediction techniques. LibPressio-Predict allows easily interchanging prediction schemes with a consistent API. LibPressio-Dataset allows the loading of these datasets in a configurable way that takes advantage of the hardware. Finally, LibPressio-Predict-Bench allows scalably training and evaluating them on large datasets at scale on a supercomputer.

We use these tools to conduct a more systematic comparison and exploration of how the different compression ratio prediction methods are achieved. With this exploration, we find that the mixture of sparsity and density in datasets can contribute to large errors in estimation. We highlight two methods from the literature (the sparsity factor in [13], and the mixture model in [2] to address this issue. We identify a gap in the literature for a both highly accurate and fast prediction scheme.

For future work, we invite the community to contribute their prediction schemes to our framework and we desire to expand our study in several ways: 1) We would like to consider in-sample prediction cases in addition to the out-of-sample use cases we provide here. Including in-sample use cases would highlight a "best-case" scenario for prediction performance to complement our current results. 2) We would like to expand our analysis to non-weather datasets and explore a wider variety of scientific data from wider domains. Different datasets have different structural patterns that are best exploited by different kinds of compressors. By expanding our evaluation we can better capture the range of prediction performance in real-world applications. 3) We would like to address the performance disparities due to the optimizer to ensure better performance parity with the evaluation in [6]. Given the similarity of this aspect to the internals of SZ3, we expect that the performance of the corresponding modules in SZ3 would also be improved by our solution closing the gap between SZ2 and SZ3's runtime performance. 4) Some of the methods support predicting other metrics such as bandwidth. As these metrics will leverage non-deterministic and runtime metrics, there will need to be refinements to the invalidation model to best handle these use-cases such as machine-dependent metrics that are agnostic to any dataset.

ACKNOWLEDGMENTS

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy (DOE) Office of Science and the National Nuclear Security Administration, and DOE Advanced Scientific Computing Research (ASCR), under contract DE-AC02-06CH11357. This work was also supported by the National Science Foundation under Grant OAC-2003709 and OAC-2104023, and CSSI-2311875. This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

REFERENCES

[1] 2021. MPI: A Message-Passing Interface Standard Version 4. https://www.mpi-forum.org/docs/mpi-4.0/mpi40-report.pdf

[2] Ganguli Arkaprabha, robert underwood, Julie Bessac, David Krasowska, Jon C. Calhoun, Sheng Di, and Franck Cappello. 2023. A Lightweight, Effective Compressibility Estimation Method for Error-bounded Lossy Compression. IEEE, Sante Fe, New Mexico.

[3] 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. *The International Journal of High Performance Computing Applications* 33, 6 (Nov. 2019), 1201–1220. https://doi.org/10.1177/1094342019853336 Number: 6.

[4] Nikoli Dryden, Roman Böhringer, Tal Ben-Nun, and Torsten Hoefler. 2021. Clairvoyant prefetching for distributed machine learning I/O. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, St. Louis Missouri, 1–15. https://doi.org/10.1145/3458817.3476181

[5] Sian Jin, Sheng Di, Jiannan Tian, Suren Byna, Dingwen Tao, and Franck Cappello. 2022. Improving Prediction-Based Lossy Compression Dramatically via Ratio-Quality Modeling. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. 2494–2507. https://doi.org/10.1109/ICDE53745.2022.00232 ISSN: 2375-026X.

[6] Sian Jin, Dingwen Tao, Houjun Tang, Sheng Di, Suren Byna, Zarija Lukic, and Franck Cappello. 2022. Accelerating Parallel Write via Deeply Integrating Predictive Lossy Compression with HDF5. http://arxiv.org/abs/2206.14761 arXiv:2206.14761 [cs].

[7] Arham Khan. 2023. SECRE: Surrogate-based Error-controlled LossyCompression Ratio Estimation Framework. IEEE, Sante Fe, New Mexico.

[8] Davis King. 2018. dlib C++ Library - Optimization. http://dlib.net/optimization.html#global_function_search

[9] David Krasowska, Julie Bessac, Robert Underwood, Jon C. Calhoun, Sheng Di, and Franck Cappello. 2021. Exploring Lossy Compressibility through Statistical Correlations of Scientific Datasets. In *2021 7th International Workshop on Data Analysis and Reduction for Big Scientific Data (DRBSD-7)*. 47–53. https://doi.org/10.1109/DRBSD754563.2021.00011

[10] Xin Liang, Sheng Di, Dingwen Tao, Sihuan Li, Bogdan Nicolae, Zizhong Chen, and Franck Cappello. 2019. Improving Performance of Data Dumping with Lossy Compression for Scientific Simulation. In *2019 IEEE International Conference on Cluster Computing (CLUSTER)*. 1–11. https://doi.org/10.1109/CLUSTER.2019.8891037 ISSN: 2168-9253.

[11] Tao Lu, Qing Liu, Xubin He, Huizhang Luo, Eric Suchyta, Jong Choi, Norbert Podhorszki, Scott Klasky, Mathew Wolf, Tong Liu, and Zhenbo Qiao. 2018. Understanding and Modeling Lossy Compression Schemes on HPC Scientific Data. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, Vancouver, BC, 348–357. https://doi.org/10.1109/IPDPS.2018.00044

[12] Zhenlu Qin, Jinzhen Wang, Qing Liu, Jieyang Chen, Dave Pugmire, Norbert Podhorszki, and Scott Klasky. 2020. Estimating Lossy Compressibility of Scientific Data Using Deep Neural Networks. *IEEE Letters of the Computer Society* 3, 1 (Jan. 2020), 5–8. https://doi.org/10.1109/LOCS.2020.2971940 Number: 1 Conference Name: IEEE Letters of the Computer Society.

[13] Md Hasanur Rahman, Sheng Di, Kai Zhao, Robert Underwood, Li Guanpeng, and Franck Cappello. 2023. A Feature-Driven Fixed-Ratio Lossy Compression Framework for Real-World Scientific Datasets. IEEE Computer Society, Anaheim, California. https://doi.org/10.1109/ICDE55515.2023.00116

[14] Claude Shannon and Warren Weaver. 1948. The Mathematical Theory of Communication. (1948), 131.

[15] Dingwen Tao, Sheng Di, Xin Liang, Z. Chen, and Franck Cappello. 2019. Optimizing Lossy Compression Rate-Distortion from Automatic Online Selection between SZ and ZFP. *IEEE Transactions on Parallel and Distributed Systems* 30, 8 (Aug. 2019), 1857–1871. https://doi.org/10.1109/TPDS.2019.2894404 Number: 8 Citation Key Alias: taoOptimizingLossyCompression2019.

[16] Robert Underwood. 2019. CODARcode/libpressio. https://github.com/CODARcode/libpressio Programmers: _:n1461 original-date: 2019-08-19T14:18:32Z.

[17] Robert Underwood, Julie Bessac, David Krasowska, Jon C Calhoun, Sheng Di, and Franck Cappello. 2023. Black-box statistical prediction of lossy compression ratios for scientific data. *The International Journal of High Performance Computing Applications* 37, 3-4 (July 2023), 412–433. https://doi.org/10.1177/10943420231179417

[18] Robert Underwood, Jon C Calhoun, Sheng Di, Amy Apon, and Franck Cappello. 2022. OptZConfig: Efficient Parallel Optimization of Lossy Compression Configuration. *IEEE Transactions on Parallel and Distributed Systems* (2022), 1–1. https://doi.org/10.1109/TPDS.2022.3154096 Conference Name: IEEE Transactions on Parallel and Distributed Systems.

[19] 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. IEEE, New Orleans.

[20] Jinzhen Wang, Qi Chen, Tong Liu, Qing Liu, and Xubin He. 2023. Zperf: A Statistical Gray-Box Approach to Performance Modeling and Extrapolation for Scientific Lossy Compression. *IEEE Trans. Comput.* (2023), 1–14. https://doi.org/10.1109/TC.2023.3257517 Conference Name: IEEE Transactions on Computers.

[21] Xiaoxia Wu, Zhewei Yao, Minjia Zhang, Conglong Li, and Yuxiong He. 2022. XTC: Extreme Compression for Pre-trained Transformers Made Simple and Efficient. https://openreview.net/forum?id=xNeAhc2CNAl

[22] Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. [n. d.]. ZeroQuant: Efficient and Affordable Post-Training Quantization for Large-Scale Transformers. ([n. d.]).

[23] 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

Received 5 September 2023; accepted 13 September 2023