# A Modular Software Framework for Compression of Structured Climate Data

Ugur Cayoglu[†, ‡]
Ugur.Cayoglu@kit.edu

Jennifer Schröter[‡]
Jennifer.Schroeter@kit.edu

Jörg Meyer[†]
Joerg.Meyer2@kit.edu

Achim Streit[†]
Achim.Streit@kit.edu
[†]Steinbuch Centre for Computing
Karlsruhe Institute of Technology
Germany

Peter Braesicke[‡]
Peter.Braesicke@kit.edu
[‡]Institute of Meteorology and Climate Research
Karlsruhe Institute of Technology
Germany

## ABSTRACT

Through the introduction of next-generation models the climate sciences have experienced a breakthrough in high-resolution simulations. In the past, the bottleneck was the numerical complexity of the models, nowadays it is the required storage space for the model output. One way to tackle the data storage challenge is through data compression.

In this article we introduce a modular framework for the compression of structured climate data. Our modular framework supports the creation of individual predictors, which can be customised and adjusted to the data at hand. We provide a framework for creating interfaces and customising components, which are building blocks of individualised compression modules that are optimised for particular applications. Furthermore, the framework provides additional features such as the execution of benchmarks and validity tests for sequential as well as parallel execution of compression algorithms.

## CCS CONCEPTS

• **Software and its engineering** → **Software design engineering**; **Software design techniques**; **Reusability**; *Designing software*;

## KEYWORDS

Usability, compression, software, framework, climate research

**ACM Reference Format:**
Ugur Cayoglu[†, ‡], Jennifer Schröter[‡], Jörg Meyer[†], Achim Streit[†], and Peter Braesicke[‡]. 2022. A Modular Software Framework for Compression of Structured Climate Data. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

New high-resolution climate models such as ICON-ART [13] simulate decades of weather dynamics and atmospheric interactions on a global scale. These models are validated with, for example, reanalysis datasets. One of them, the current **E**uropean **ReA**nalysis (ERA5) dataset outputs hourly data starting from 1979 to the present on a $1440 \times 721$ (about 31 km) horizontal and 137 level vertical (up to 0.01 hPa = 80 km) grid[1]. If we assume 16-Bit Integer values for each variable this amounts to 2.26 TiB p.a. per variable with support for 120 variables[2]. One way to tackle the storage problem is to use compression.

A successful compression algorithm would allow to save the output of even higher resolutioned models on the same available disk space or reduce the currently used disk space. Until recently, the development of compression methods focused rather on text files than numerical data. This has changed in recent years. A number of algorithms have been developed, which focus on the compression of floating-point values. Other developments specifically address data generated in climate sciences [1, 3, 5, 7]. Most of these methods use prediction-based compression [6, 11] which is particularly suited for climate data due to its gridded nature. Although these methods are based on the same principle of prediction-based compression, there is currently no easy way to test and adapt them to the data at hand. Our framework introduced here does provide this flexibility.

Our framework supports the creation of individual predictors, which can then be customised by the scientist. It defines necessary interfaces and components for the development of custom compression algorithms.

In the next section we will give a brief overview of prediction-based compression. Afterwards we will introduce our proposed framework. In Section 4 we will take a closer look at the implementation details. In the concluding section, we will outline how the community can contribute to the framework and give recommendations for future work.

[1]European Centre for Medium-Range Weather Forecasts (ECMWF) Newsletter No. 147 – Spring 2016 (p.7)
[2]While some of these variables are simulated, others can be deduced from simulated variables. For reference http://apps.ecmwf.int/codes/grib/param-db

## 2 PREDICTION-BASED COMPRESSION

In this section we will give a brief introduction to prediction-based compression. A prediction-based compression algorithm consists of the following steps:

(1) Reading in the data
(2) Mapping the data to integers
(3) Defining a traversal sequence
(4) Giving a prediction for each value
(5) Calculating the residual between prediction and true value
(6) Encoding these residuals and saving them on disk

Figure 1 depicts the state diagram of the above described compression algorithm. The decompression process follows through the states in reverse order. Prediction-based compression relies on the following premise:

The better the prediction is, the more zeros are leading the residual from step five. The number of zeros at the beginning of a binary number is called its leading zero count (LZC). The next step differs depending on whether lossy compression or lossless compression is desired. With lossless compression, the LZC and the residual are stored. With lossy compression, the deviation from the true value is compared with an error tolerance determined in advance and a decision is made whether or not the residual should be stored. The true value can then be reconstructed using the prediction model and residual.

After this brief introduction to prediction-based compression, we will now describe how this concept has been implemented in the framework.

## 3 FRAMEWORK

The main goal of the framework is to provide state of the art compression algorithms for environmental sciences. It should provide off-the-shelf solutions, but at the same time have a low barrier for customisation. In this section we will describe the structure of the proposed framework and how it achieves these goals.

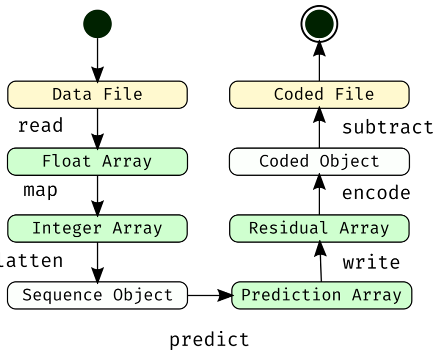

**Figure 1: State diagram of a prediction-based compression algorithm. The label on the arrows define transitions applied to the previous state. The coloured options emphasise the similarity of the states. Data files are yellow and multi-dimensional arrays are green.**

**Algorithm 1:** Default compression function using our framework.

```
Function compress(arr, mapper, sequencer, ...)
    iarr = mapper.map(arr) ;                          /* Mod 1 */
    seq = sequencer.flatten(iarr) ;                   /* Mod 2 */
    pred = [];
    for i = 0 to seq.size do
        p = predictor.predict();
        pred.append(p) ;                              /* Mod 3 */
        predictor.update(seq[i]);
    rarr = subtractor.subtract(iarr, seq, pred) ;     /* Mod 4 */
    coded = encoder.encode(rarr) ;                    /* Mod 5 */
    return coded;
```

The framework consists of two core components: objects and modifiers.

*Object.* The purpose of objects is to represent the current state of the data to be compressed like the states depicted in Figure 1. They may include metadata about previous states, but once created are immutable.

*Modifier.* Modifiers operate on objects and are the only way to transform one object to the other. Each modifier has exactly one method and can only operate on one kind of object. This prevents a mistake in change of modifiers and enhances interoperability.

An ordinary compression algorithm consists of five modifiers with the following tasks:

- **Mapper** *Mapping floating-point values to integers*
- **Sequencer** *Transforms an array into a data stream*
- **Predictor** *Predicts next datum on the data stream, based on past values*
- **Subtractor** *Calculates the residual between prediction and true value*
- **Encoder** *Prepares residuals to be written on disk*

The modifiers are designed for the tasks detailed in Section 2. The objects are the outcome of these tasks. The default compression function of the framework is shown in Algorithm 1. Since the interface of each modifier is standardised it is very easy to replace each modifier of the compression algorithm.

Next to these components the framework provides additional modules to help the scientist design a compression algorithm: ensemble predictors, quality assessment, parallel compression and random subsetting.

*Ensemble predictors.* It is rather unlikely that there is one predictor to rule them all. For example, if it is known that predictor $x$ performs very good for temperature, but bad for greenhouse gas ozone, the framework should provide the possibility to switch between predictors or to average the result of several predictors. For this case the framework supports ensemble predictors. An ensemble instance is defined by a list of predictors and a cost function. The predictors of the list are run in parallel during compression. The cost function determines the rank of these predictors. There might also be a consolidation function in case the prediction of the ensemble instance should be an average of all predictions instead of the one with the least costs. An example for an ensemble predictor is given in Algorithm 2. Here the predictors are ranked based on their performance on the data processed prior, given the pre-defined traversal sequence (step four in Section 2). Please note the similarities in the syntax of Alg. 1 and Alg. 2: Since there is no distinguishing property of ensemble and non-ensemble predictors, the framework supports the nesting of ensemble predictors.

*Quality Assessment.* The Quality Assessment (QA) module provides information about the achievable compression rate by the current setup and dataset. QA hooks into the compression process at step five (see Section 2) and calculates the LZC of the residual array. The residual array provides enough information about the performance of the predictors. The average LZC can then be compared to the Shannon Entropy [14] of the dataset. The Shannon Entropy quantifies the average amount of information represented by a random datum of the dataset.

*Parallel compression.* An additional module, which can support the scientist in search for a compression method is parallel processing. The proposed framework contains a parallelisation module, which can either chunk the data in blocks and compress each on a different thread or run a predictor on each thread with the same input file. This would lead to a less time-consuming search for a compression algorithm.

*Random subsetting.* Since the design of a compression algorithm is an iterative process, it would be a daunting task for the scientist to have to compress gigabytes of data on each test-run, only to realise that a certain parameter needs to be fixed or a predictor can be eliminated. Therefore the framework supports random subsetting of datasets. The subsetting is defined by its size, error margin and possible dimension constraints. The constraints limit the number of dimensions considered for a subset.

**Algorithm 2:** An ensemble compression algorithm using the best predictor from previous prediction [**?** ].

```
Function predict()
    pred = defaultpredictor;
    if lastbestpredictor is not NaN then
        pred = lastbestpredictor
    return pred.predict();
Function cost(prediction, truth)
    return abs(truth - prediction);
Function update(truth)
    predictions = dict();
    forall predictor p in predictors do
        prediction = p.predict();
        predictions[p] = cost(prediction, truth);
    sorted = sortByValue(predictions, ascending=True);
    lastbestpredictor = sorted[0]
```

These features should help the scientist define a custom compression method for the data at hand. The framework defines the necessary components and helper modules for customisation and grading of compression algorithms, while at the same time providing easy to use pre-defined algorithms. In the next section we will discuss the implementation of the framework.

## 4 IMPLEMENTATION

An implementation of the framework will be made available at [2]. The provided framework is implemented in Python 3 and uses as backend modules `scipy` [10], `pandas` [8] and `xarray` [4]. It has been tested with files in NetCDF format with Climate and Forecast Metadata Conventions. We hope the use of established open source software provides a good basis for uptake, future co-operations and possible extensions of the framework. For reasons of brevity we restrict details about our implementation to the core components.

The class diagram used for the implementation of the object components is shown in Fig. 2. As described in Section 3 the objects do not have the possibility to mutate itself or others. With the exception of `Float Array` none of the objects have methods to manipulate their content. The methods implemented in `Float Array` are for initialisation from common data types such as `numpy` [9] arrays or netcdf [12] data files.

The `Prediction Array` and `Residual Array` classes inherit from the `Integer Array` class. While these classes do not provide additional functionality compared to the `Integer Array`, they are necessary to provide easy distinction of objects on which each modifier can operate.

Figure 3 depicts possible modifiers to be used as components of the framework. This is a none exhaustive list of modifiers and should exemplify the large number of possible options in designing a compression algorithm. The modifiers which are implemented at the time of publication are emphasised. A documentation for each modifier is omitted for reasons of brevity, but is included in the provided implementation.

## 5 SUMMARY AND OUTLOOK

In the last couple of years the climate sciences have experienced a breakthrough in terms of possible fine-granular simulations. Next-generation climate models allow high-resolution simulations to be run on high-performance computers. This resulted in a significant increase of storage space. We present a modular framework for the compression of climate data to tackle this challenge. We briefly described the steps of a prediction-based compression methods in Section 2 and explained in Section 3 how these steps are implemented using the concept of modifiers and objects.

The framework provides all necessary components to design, test and grade various prediction-based compression algorithms. It further supports the use of ensemble predictors to merge predictions based on different predictors, quality assessment methods to help grade the prediction methods, parallel compression for concurrent execution of predictors as well as random subsetting for unbiased results during the design of a compression algorithm. Although the framework was currently only used with climate data, it is conceivable to use it in conjunction with environmental data such as measurement data or any other grid data.

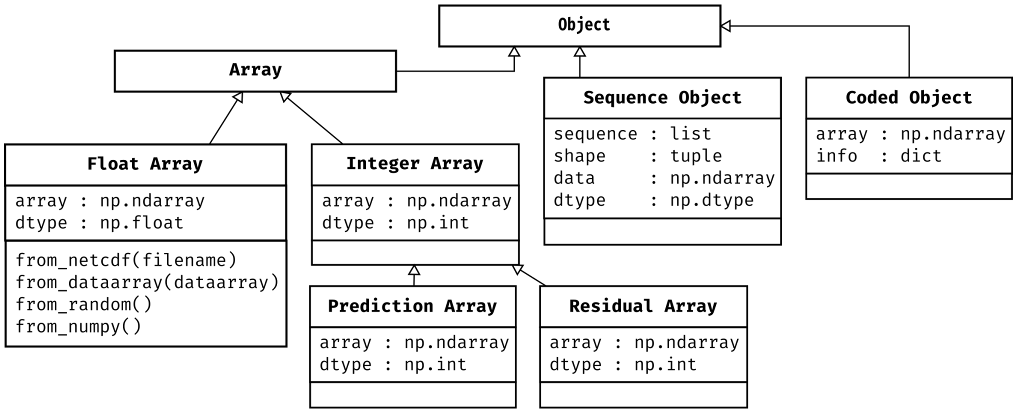

**Figure 2: UML class diagram for object components.**

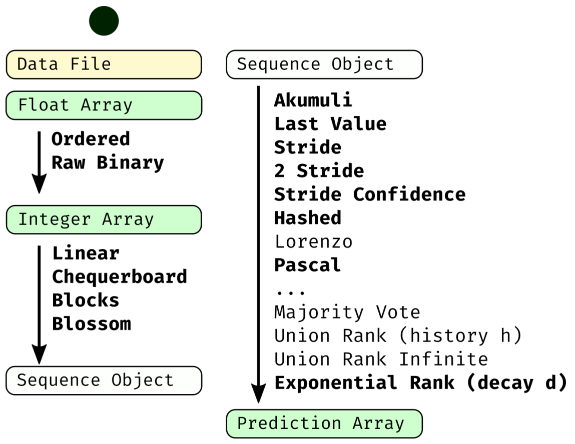

**Figure 3: A none exhaustive list of modifiers to exemplify the large number of possible combinations. Emphasised are the modifiers which are implemented and part of the framework at the time of publication.**

## CODE AVAILABILITY

An implementation of the framework described above will be made available under GNU GPLv3 license at [2].

## REFERENCES

[1] Allison H. Baker, Dorit M. Hammerling, Sheri A. Mickleson, Haiying Xu, Martin B. Stolpe, Phillipe Naveau, Ben Sanderson, Imme Ebert-Uphoff, Savini Samarasinghe, Francesco De Simone, Francesco Carbone, Christian N. Gencarelli, John M. Dennis, Jennifer E. Kay, and Peter Lindstrom. 2016. Evaluating Lossy Data Compression on Climate Simulation Data within a Large Ensemble. *Geosci. Model Dev. Discuss.* July (jul 2016), 1–38. https://doi.org/10.5194/gmd-2016-146

[2] Ugur Cayoglu. 2018. Prediction-based Compression Framework. https://github.com/ucyo/cframework. (2018). [Online; accessed 27-May-2018].

[3] Ugur Cayoglu, Peter Braesicke, Tobias Kerzenmacher, Jörg Meyer, and Achim Streit. 2017. Adaptive Lossy Compression of Complex Environmental Indices Using Seasonal Auto-Regressive Integrated Moving Average Models. In *2017 IEEE 13th International Conference on e-Science (e-Science)*. 315–324. https://doi.org/10.1109/eScience.2017.45

[4] Stephan Hoyer and Joe Hamman. 2017. xarray: N-D labeled arrays and datasets in Python. *Journal of Open Research Software* 5, 1 (2017). https://doi.org/10.5334/jors.148

[5] Xiaomeng Huang, Yufang Ni, Dexun Chen, Songbin Liu, Haohuan Fu, and Guangwen Yang. 2016. Czip: A Fast Lossless Compression Algorithm for Climate Data. *Int. J. Parallel Program.* 44, 6 (dec 2016), 1248–1267. https://doi.org/10.1007/s10766-016-0403-z

[6] Peter Lindstrom and Martin Isenburg. 2006. Fast and Efficient Compression of Floating-Point Data. *IEEE Trans. Vis. Comput. Graph.* 12, 5 (sep 2006), 1245–1250. https://doi.org/10.1109/TVCG.2006.143

[7] Songbin Liu, Xiaomeng Huang, Yufang Ni, Haohuan Fu, and Guangwen Yang. 2014. A High Performance Compression Method for Climate Data. In *2014 IEEE Int. Symp. Parallel Distrib. Process. with Appl.* IEEE, 68–77. https://doi.org/10.1109/ISPA.2014.18

[8] Wes McKinney. 2010. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman (Eds.). 51 – 56.

[9] Travis E Oliphant. 2006. *A guide to NumPy*. Vol. 1. Trelgol Publishing USA.

[10] Travis E. Oliphant. 2007. Python for Scientific Computing. *Computing in Science Engineering* 9, 3 (May 2007), 10–20. https://doi.org/10.1109/MCSE.2007.58

[11] Paruj Ratanaworabhan, Jian Ke, and Martin Burtscher. 2006. Fast Lossless Compression of Scientific Floating-Point Data. In *Data Compression Conf.* IEEE, 133–142. https://doi.org/10.1109/DCC.2006.35

[12] Russ Rew and Glenn Davis. 1990. NetCDF: an interface for scientific data access. *IEEE computer graphics and applications* 10, 4 (1990), 76–82.

[13] Jennifer Schröter, Daniel Rieger, Christian Stassen, Heike Vogel, Michael Weimer, Sven Werchner, Jochen Förstner, Florian Prill, Daniel Reinert, Günther Zängl, Marco Giorgetta, Roland Ruhnke, Bernhard Vogel, and Peter Braesicke. 2018. ICON-ART 2.1 – A flexible tracer framework and its application for composition studies in numerical weather forecasting and climate simulations. *Geoscientific Model Development Discussions* 2018 (2018), 1–37. https://doi.org/10.5194/gmd-2017-286

[14] Claude E. Shannon. 1948. A Mathematical Theory of Communication. *Bell Syst. Tech. J.* 27, 3 (jul 1948), 379–423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x arXiv:chao-dyn/9411012