

# pyfive: A pure-python HDF5 reader

Bryan Lawrence<sup>1</sup>, Ezequiel Cimadevilla<sup>2</sup>, Wout De Nolf<sup>6</sup>, David Hassell<sup>1</sup>, Jonathan Helmus<sup>3</sup>, Brian Maranville<sup>4</sup>, Kai Mühlbauer<sup>5</sup>, and Valeriu Predoi<sup>1</sup>

<sup>1</sup> NCAS-CMS, Meteorology Department, University of Reading, Reading, United Kingdom. <sup>2</sup> Instituto de Física de Cantabria (IFCA), CSIC-Universidad de Cantabria, Santander, Spain. <sup>3</sup> TBD <sup>4</sup> NIST Center for Neutron Research <sup>5</sup> Institute of Geosciences, Meteorology Section, University of Bonn, Germany. <sup>6</sup> European Synchrotron Radiation Facility (ESRF), Grenoble, France.

DOI: 10.xxxxxx/draft

## Software

- Review
- Repository
- Archive

Editor: Open Journals

## Reviewers:

- @openjournals

Submitted: 01 January 1970

Published: unpublished

## License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

## Summary

pyfive (<https://pyfive.readthedocs.io/en/latest/>) is an open-source thread-safe pure Python package for reading data stored in HDF5. While it is not a complete implementation of all the specifications and capabilities of HDF5, it includes all the core functionality necessary to read gridded datasets, whether stored contiguously or with chunks, and to carry out the necessary decompression for the standard options. All data access is fully lazy, the data is only read from storage when the numpy data arrays are manipulated. Originally developed some years ago, the package has recently been upgraded to support lazy access, and to add missing features necessary for handling all the environmental data known to the authors. It is now a realistic option for production data access in environmental science and more widely. The API is based on that of h5py (which is a python shimmy over the HDF5 c-library, and hence is not thread-safe), with some API extensions to help optimise remote access. With these extensions, coupled with thread safety, many of the limitations precluding the efficient use of HDF5 (and NetCDF4) on cloud storage have been removed.

## Statement of need

HDF5<sup>1</sup> (Folk et al., 2011) is probably the most important data format in physical science, used across the piste. It is particularly important in environmental science, particularly given the fact that NetCDF4<sup>2</sup> (Rew et al., 2006) is HDF5 under the hood. From satellite missions, to climate models and radar systems, the default binary format has been HDF5 for decades. While newer formats are starting to get mindshare, there are petabytes, if not exabytes of existing HDF5, and there are still many good use-cases for creating new data in HDF5. However, despite the history, there are few libraries for reading HDF5 file data that do not depend on the official HDF5 library maintained by the HDFGroup, and in particular, apart from pyfive, in Python there are none that cover the needs of environmental science. While the HDF5 c-library is reliable and performant, and battle-tested over decades, there are some caveats to depending upon it: Firstly, it is not thread-safe, secondly, the code is large and complex, and should anything happen to the financial stability of The HDF5group, it is not obvious the C-code could be maintained. Finally, the code complexity also meant that it is not suitable for developing bespoke code for data recovery in the case of partially corrupt data. From a long-term curation perspective both of these last two constraints are a concern.

The original implementation of pyfive (by JH), which included all the low-level functionality to deal with the internals of an HDF5 file was developed with POSIX access in mind. The

<sup>1</sup><https://www.hdfgroup.org/solutions/hdf5/>

<sup>2</sup><https://www.unidata.ucar.edu/software/netcdf>

recent upgrades were developed with the use-case of performant remote access to curated data as the primary motivation, but with additional motivations of having a lightweight HDF5 reader capable of deploying in resource or operating-system constrained environments (such as mobile), and one that could be maintained long-term as a reference reader for curation purposes. The lightweight deployment consequences of a pure-python HDF5 reader need no further introduction, but as additional motivation we now expand on the issues around remote access and curation.

Thread safety has become a concern given the wide use of Dask<sup>3</sup> in python based analysis workflows, and this, coupled with a lack of user knowledge about how to efficiently use HDF5, has led to a community perception that HDF5 is not fit for remote access (especially on cloud storage). Issues with thread safety arise from the underlying HDF5 c-library, and cannot be resolved in any solution depending on that library, hence the desire for a pure python solution. Remote access has been bedevilled by the widespread need to access remotely data which has been chunked and compressed, combined with the use of HDF5 data which was left in the state it was when the data was produced - often with default unsuitable chunking (Rew, 2013) and with interleaved chunk indexes and data. Solutions have mainly consisted of reformatting the data (and rechunking it at the same time) or utilising kerchunk mediated direct access to chunked HDF5 data<sup>4</sup>. However, in practice using kerchunk requires the data provider to generate kerchunk indices to support remote users, and it leads to issues of synchronicity between indices and changing datasets.

This version of pyfive was developed with these use-cases in mind. There is now full support for lazy loading of chunked data, and methods are provided to give users all the benefits of using kerchunk, but without the need for a priori generation. Because pyfive can access and cache (in the client) the b-tree (index) on a variable-by-variable basis, most of the benefits of kerchunk are gained without any of the constraints. However, the kerchunk index is always a contiguous object accessible with one get transaction, this is not necessarily the case with the b-tree, unless the source data has been repacked to ensure contiguous metadata using a tool like h5repack. Much of the community is unaware of the possibility of repacking the index metadata, and this together with relatively opaque information about the internal structure of files (and hence the necessity or other wise of such repacking), means that repacking is rarely done. To help with this process, pyfive also includes extensions to expose information about how data and indexes are distributed in the files. With these tools, index extraction with pyfive can be comparable in performance to obtaining a kerchunk index, and completely opaque to the user.

With the use of pyfive, suitably repacked and rechunked HDF5 data can now be considered 'cloud-optimised', insofar as with lazy loading, improved index handling, and thread-safety, there are no "format-induced" constraints on performance during remote access. To aid in discovering whether or not a given HDF5 dataset is cloud-optimised, pyfive also now provides simple methods to expose information about file layout - both in API extensions, and in a new p5dump utility, which provides (in the default view) functionality similar to ncdump, and when used with p5dump -s, information about storage characteristics.

The issues of the dependency on a complex code maintained by one private company in the context of maintaining data access (over decades, and potentially centuries), can only be mitigated by ensuring that the data format is well documented, that data writers use only the documented features, and that public code exists which can be relatively easily maintained. The HDF5group have provided good documentation for the core features of HDF5 which include all those of interest to the weather and climate community who motivated this reboot of pyfive, and while there is a community of developers beyond the HDF5 group (including some at the publicly funded Unidata institution), recent events suggest that given most of those developers and their existing funding are US based, some spreading of risk would be

<sup>3</sup><https://www.dask.org/>

<sup>4</sup><https://fsspec.github.io/kerchunk/>

91 desirable. To that end, a pure-python code, which is relatively small and maintained by an  
92 international constituency, alongside the existing c-code, provides some assurance that the  
93 community can maintain HDF5 access for the foreseeable future. A pure python code also  
94 makes it easier to develop scripts which can work around data and metadata damage should  
95 they occur.

## 96 Remote Access

97 A notable feature of the recent pyfive upgrade is that it was carried out with thread-safety  
98 and remote access using fsspec (filesystem-spec.readthedocs.io) in mind. We provide two  
99 examples of using pyfive to access remote data, one in S3, and one behind a modern http  
100 web server:

101 For accessing the data on S3 storage, we will have to set up an s3fs virtual file system, then  
102 pass it to Pyfive:

```
import pyfive
import s3fs

# storage options for an anon S3 bucket
storage_options = {
    'anon': True,
    'client_kwargs': {'endpoint_url': "https://s3server.ac.uk"}
}
fs = s3fs.S3FileSystem(**storage_options)
file_uri = "s3-bucket/myfile.nc"
with fs.open(file_uri, 'rb') as s3_file:
    nc = pyfive.File(s3_file)
    dataset = nc[var]
```

103 for an HTTPS data server, the usage is similar:

```
import fsspec
import pyfive

fs = fsspec.filesystem('http')
with fs.open("https://site.com/myfile.nc", 'rb') as http_file:
    nc = pyfive.File(http_file)
    dataset = nc[var]
```

## 104 Cloud Optimisation

105 To be fully cloud optimised - as defined by Stern et al. (2022) - an HDF5 file needs to have  
106 a contiguous index for each variable, and the chunks for each variable need to be sensibly  
107 chosen and broadly contiguous within the file. When these criteria are met, indexes can be  
108 read efficiently, and middleware such as fsspec can make sensible use of readahead caching  
109 strategies.

110 HDF5 data files direct from simulations and instruments are often not in this state as information  
111 about the number of variables, the number of chunks per variable, and the compressed size  
112 of those variables is not known as the data is being produced. In such cases the data is also  
113 often not chunked along the dimensions being added to as the file is written (since it would  
114 have to be buffered first).

115 Of course, once the file is produced, such information is available. Metadata can be repacked to  
116 the front of the file and variables can be rechunked and made continuous - which is effectively  
117 the same process undertaken when HDF5 data is reformatted to other cloud optimised formats.

118 The HDF5 library provides a tool “h5repack” which can do this, provided it is driven with  
119 suitable informatin about required chunk shape and the expected size of metadata fields.  
120 Versions (>1.0) of pyfive supports both a method to query whether such repacking is  
121 necessary, and to extract necessary parameters.

122 In the following example we compare and contrast the unpacked and repacked version of a  
123 particularly pathological file, and in doing so howcase some of the pyfive API extensions  
124 which help us understand why it is pathological, and how to address those issues for repacking.

125 If we extract just a piece of the output of `p5dump -s` on this file (which has surface wind  
126 velocity at three hour intervals for one hundred years):

```
127 float64 time(time) ;
128         time:standard_name = "time" ;
129         time:_n_chunks = 292192 ;
130         time:_chunk_shape = (1,) ;
131         time:_btree_range = (31808, 19854095942) ;
132         time:_first_chunk = 9094 ;
133
134 float32 uas(time, lat, lon) ;
135         uas:_Storage = "Chunked" ;
136         uas:_n_chunks = 292192 ;
137         uas:_chunk_shape = (1, 143, 144) ;
138         uas:_btree_range = (28672, 19854809382) ;
139         uas:_first_chunk = 36520 ;
```

140 we can immediately see that this will be a problematic file! The b-tree index is clearly interleaved  
141 with the data (compare the first chunk address with last index addresses of the two variables),  
142 and with a chunk dimension of (1,), any effort to use the time-dimension to locate data of  
143 interest will involve a ludicrous number of 1 number reads (all underlying libraries read the data  
144 one chunk at a time). It would feel like waiting for the heat death of the universe if one was  
145 to attempt to manipulate this data stored on an object store!

146 It is relatively easy (albeit slow) to use h5repack to fix this - e.g see Hassell & Cimadevilla  
147 Alvarez (2025) - after which we see:

```
148 float64 time(time) ;
149         time:_Storage = "Chunked" ;
150         time:_n_chunks = 1 ;
151         time:_chunk_shape = (292192,) ;
152         time:_btree_range = (11861, 11861) ;
153         time:_first_chunk = 40989128 ;
154         time:_compression = "gzip(4)" ;
155 float32 uas(time, lat, lon) ;
156         uas:_Storage = "Chunked" ;
157         uas:_n_chunks = 5844 ;
158         uas:_chunk_shape = (50, 143, 144) ;
159         uas:_btree_range = (18663, 347943) ;
160         uas:_first_chunk = 41041196 ;
161         uas:_compression = "gzip(4)" ;
```

162 Now data follows indexes, the time dimension is one chunk, and there is a more sensible  
163 number of actual data chunks. While this file would probably benefit from splitting, with a  
164 contiguous set of indexes, it is now possible to exploit this data via S3.

165 All the metadata shown in this dump output arises from pyfive extensions to the  
166 pyfive.h5t.DatasetID class. pyfive also provides a simple flag: `contiguous_metadata` for  
167 a File instance, which can take values of True or False for any given file, which simplifies at  
168 least the “is the index packed at the front of the file?” part of the optimisation question -  
169 though inspection of chunking is a key part of the workflow necessary to determine whether or  
170 not a file really is optimised for cloud usage.

## 171 Acknowledgements

172 Most of the developments outlined here that have occurred since V0.5 (primarily authored  
173 by JH) have been supported by the UK Met Office and UKRI via 1) UK Excalibur Exascale  
174 programme (project ExcaliWork), 2) the UKRI Digital Research Infrastructure programme  
175 (project WacaSoft), and 3) the national capability funding of the UK National Center for  
176 Atmospheric Science (NCAS). Ongoing maintenance of pyfive is expected to continue under  
177 the auspices of that NCAS national capability funding.

## 178 References

- 179 Folk, M., Heber, G., Koziol, Q., Pourmal, E., & Robinson, D. (2011). An overview of the HDF5  
180 technology suite and its applications. *Proceedings of the EDBT/ICDT 2011 Workshop on*  
181 *Array Databases*, 36–47. <https://doi.org/10.1145/1966895.1966900>
- 182 Hassell, D., & Cimadevilla Alvarez, E. (2025). *Cmip7repack: Repack CMIP7 netCDF-4*  
183 *datasets*. Zenodo. <https://doi.org/10.5281/zenodo.17550920>
- 184 Rew, R. (2013). Chunking data: Choosing shapes. In *News @ Unidata*. [https://www.unidata.ucar.edu/blogs/](https://www.unidata.ucar.edu/blogs/developer/en//entry/chunking_data_choosing_shapes)  
185 [developer/en//entry/chunking\\_data\\_choosing\\_shapes](https://www.unidata.ucar.edu/blogs/developer/en//entry/chunking_data_choosing_shapes).
- 186 Rew, R., Hartnett, E., & Caron, J. (2006). NetCDF-4: Software implementing an enhanced  
187 data model for the geosciences. *22nd International Conference on Interactive Information*  
188 *Processing Systems for Meteorology, Oceanography and Hydrology*.
- 189 Stern, C., Abernathey, R., Hamman, J., Wegener, R., Lepore, C., Harkins, S., & Merose, A.  
190 (2022). Pangeo forge: Crowdsourcing analysis-ready, cloud optimized data production.  
191 *Frontiers in Climate*, 3. <https://doi.org/10.3389/fclim.2021.782909>