



FoxBench: Benchmark for n-Dimensional Array File Formats in Data Analytics Environments

Arne Osterthun ¹ and Matthias Pohl ¹





Abstract: For effective data exchange and transfer, choosing the proper file format is crucial. Different domains have specific standards for file formats. While CSV files are commonly used, they lack reusability. Data files are well-suited for computing clusters as they enable efficient data processing and storage across multiple interconnected systems. Data analytics pipelines can be time-consuming due to handling large volumes of data. Timely data access is crucial for efficient processing and analysis. Earth system science (ESS) data commonly manifests as dense or sparse n-dimensional data. Dense n-dimensional data is conventionally stored in arrays, while sparse n-dimensional data is typically housed in data frames. In the realm of ESS, an array of file formats is leveraged for the storage of dense n-dimensional data, including NetCDF4, TileDB, and Zarr. The paper at hand aims to evaluate data file formats for retrieving multidimensional data, specifically focusing on tools within the ESS domain. The insights from this exploration will be applicable to other data analytics projects.

Keywords: Benchmark, Data Access, Storage, Cost-based Valuation, File Formats, Big Data

1 Introduction

The use of databases provides structured access to data, but the process of structuring the data can be labor-intensive and resource-heavy. In the context of research and data analytics projects, handling large volumes of data is often the norm. Rather than relying on traditional databases, many of these projects opt for simple storage or data lake technologies to manage distributed data effectively. This choice is driven by the fact that specialized distributed storage solutions, such as Apache Hadoop HDFS, are often not implemented in research projects where data originates from diverse stakeholders with various requirements. The specific needs of these projects encompass data access by specialized research software, file transfer, statistical analysis, and visualizations, all of which must be accommodated. Additionally, software tools often require modifications to operate seamlessly with specialized distributed file systems. Moreover, in cloud environments, the use of simple storage services like object stores is integral to the construction of efficient data analytics pipelines. Earth System Science (ESS) is a domain that often works with large datasets at TB to PB scale, resulting in data-intensive processing pipelines. Data lakes serve as a solution for handling large datasets that are impractical to download easily. ESS data commonly presents itself in the form of dense and sparse n-dimensional data. Dense n-dimensional data is typically stored in arrays, while sparse n-dimensional data is often

¹ German Aerospace Center (DLR), Institute of Data Science, Mälzerstr. 3-5, 07745 Jena, Germany,
arne.osterthun@dlr.de,  <https://orcid.org/0000-0001-6455-9119>;
matthias.pohl@dlr.de,  <https://orcid.org/0000-0002-6241-7675>

stored in data frames. Within the ESS domain, a diverse range of file formats is utilized for storing dense n-dimensional data (e.g., GRIB, HDF5, NetCDF4, TileDB, and Zarr). For the exchange and transfer of data, the choice of file format is crucial for efficiency. Various application domains have their own distinct standards for file formats. Although CSV files are commonly utilized for exporting and transferring data, they lack efficiency in terms of reusability. On the other hand, data files are well-suited for distribution in computing clusters. In the context of data analytics pipelines, the process can often be time-consuming due to the complexity of handling, storing, and processing large volumes of data. Additionally, timely access to data is of paramount importance in data-intensive pipelines, as it directly impacts the speed and efficiency of data processing and analysis.

The primary objective of this paper is to examine prevalent data file formats for the retrieval of multidimensional data. The focus of the investigation will center on specific tools within the ESS domain, such as xarray and dask, with the aim of establishing a comparable environment. Importantly, the insights garnered from this exploration will be transferrable to other domains encompassing data analytics projects. We are pursuing the research question: *Which file format performs best-concerning data access time and storage consumption in domain-specific data analytics environments?*

Section 2 will provide an overview of related work focusing on the benchmarking of file formats. This will be followed by an exploration of file formats. In Section 4, we will introduce the benchmark framework, covering its layer components and necessary dependencies. The benchmark execution, data, file formats, system setup, and the results of the benchmarks will be described, along with a comparative analysis of the results in Section 5. Finally, an in-depth discussion of the benchmark results will be presented, along with an outlook on future work.

2 Related Work

The importance of benchmarking databases, explicitly highlighting the performance evaluation of data access and data ingestion as fundamental aspects of data management, is widely discussed [GB14, St93, BNE13, Cu10]. This includes benchmarking array database management systems (DBMS) conducted in the context of earth system science for specific databases [MMB16]. In this section, some related work that also focuses on file format benchmarking for querying spatial data is presented. The study designated by [Po09] delves into the examination of specific file systems, namely ADIOS and PLSF, as well as file formats such as NetCDF and HDF5. It aims to gauge read bandwidth (MB/s or GB/s) with respect to the number of writers. The outcome of the study involves the identification of distinct shifts in performance in relation to the varying number of writers. The study conducted by [AB23] provides a benchmark comparison for NetCDF, HDF5, and Zarr. The research evaluates the efficiency of creating, reading, and writing data files containing synthetic data of varying multidimensional sizes. It is important to note that this investigation represents a preliminary study. The study of [Bo17] involves benchmarking Apache Parquet

files within an HDFS cluster and VCF files that share similarities with CSV/TSV files. The investigation encompasses varying cluster sizes to gauge performance. The assessment focuses on measuring wall time and the time required for data loading. In the study conducted by [El11], the focus is on the analysis of the loading time of data files on different architectures of a Hadoop Cluster. The research also involved the comparison of custom cluster architectures, namely COHadoop and ParHadoop. Additionally, the study delved into evaluating the impact of different chunk sizes of data. Several studies are focusing on the development of an I/O-Controller designed to manage netCDF files in parallel. These also involved performance benchmarking of the read bandwidth of various processors and included a comparison with HDF5 files [Li03, Ga09, Ga11].

3 Background - File Formats

This section provides an overview of various file formats that are commonly utilized in general applications, with a particular focus on those relevant to the ESS domain. It will discuss the characteristics and potential uses of these formats. Additionally, these file formats are considered and evaluated in benchmarking processes.

Comma-separated values (**CSV**) is a plain text file format used for storing tabular data, such as spreadsheets or databases. Each line represents a single record, with fields separated by commas. CSV files can contain various data types, including integers, floats, strings, and dates. Unlike other formats, CSV does not store metadata about the data itself. It is commonly used for data exchange between applications and for storing log or configuration data, but it has limitations with complex data structures and large datasets. CSV files often use UTF-8 encoding for text fields.

HDF5 (hierarchical data format version 5)² is designed to systematically store substantial volumes of data in a file system-like structure. Its primary data models include groups, which can contain other groups or datasets, allowing for hierarchical organization. The **netCDF4**³ format is designed to store array-oriented data, using groups as the overarching structure for organizing other groups or variables. NetCDF4 achieves faster read times and shares similarities with HDF5. Unlike HDF5 datasets, variables in netCDF4 cannot be resized once created, but a workaround allows unlimited size in a specified dimension. Similar to HDF5 and Zarr, netCDF4 is self-describing and enables efficient data mapping in memory. NetCDF4 is built upon HDF5 and mostly links to the HDF5 library to execute read and write operations. In NetCDF4, we gain a more raster-data-oriented data model.

The data format **Zarr**⁴ is specifically designed to store large arrays⁴ of data efficiently. It shares key similarities with other popular formats, such as HDF5 and netCDF4, as it is hierarchical and self-describing. The format utilizes groups and chunks as the primary file

² <https://docs.hdfgroup.org/>

³ <https://www.unidata.ucar.edu/software/netcdf/>

⁴ <https://zarr.dev/>

structure, with each group containing datasets that are essentially multidimensional arrays of a homogeneous data type. One of its key advantages is the diverse options it provides for storing data, whether in memory, the file system, or other storage systems. It ensures a consistent interface across these storage options, offering convenience and flexibility. Furthermore, it boasts better performance in data access thanks to its implementation of chunk-wise separation of data to sub-files that support the usage of distributed storage.

TileDB⁵ is an open-source engine designed to handle multi-dimensional arrays, including dense and sparse data as well as tabular data. It offers fast storage and efficient data handling through multidimensional slicing, compression, encryption, and checksum filters. TileDB provides rapid data ingestion, parallel I/O, and hierarchical organization of array data. It offers APIs and seamless integration with popular platforms such as Spark and Dask.

Apache Parquet⁶ is an open-source columnar storage format designed to handle complex data processing at scale. Originally developed by Twitter and Cloudera, it has emerged as a fundamental component of the Hadoop ecosystem. This storage format excels in its efficient handling of data through column-oriented organization, enabling superior compression ratios and optimized query performance. Key features include predicate pushdown for selective data retrieval, robust schema evolution capabilities, and support for nested data structures. The architecture of Parquet is built around a hierarchical organization of row groups, column chunks, and pages, making it particularly well-suited for analytical workloads where queries typically access specific columns rather than entire datasets.

There are several additional data formats that can be classified as either domain-specific or commonly unutilized in data analytics environments. Firstly, domain-specific formats such as GRIB (General Regularly-distributed Information in Binary form)⁷ and ADF (Atmospheric Data Format) are specialized for particular fields, like meteorology and atmospheric science. These formats often face challenges related to compatibility with broader technology stacks typically used in data analytics, which can hinder their integration into more general analytical workflows. Moreover, there are formats like Avro⁸, a serialized data format that has been developed by the Apache Software Foundation. Avro is designed for data serialization in big data applications, yet it has not gained widespread adoption in various application domains. Additionally, the ROOT [An11, BI20] format, which is extensively utilized within the scientific research community is still under active development. The compatibility of the ROOT format with mainstream data analytics technology stacks is also limited, creating barriers to its integration into general-purpose data analysis frameworks.

⁵ <https://tiledb.com/>

⁶ <https://parquet.apache.org>

⁷ <https://community.wmo.int/en/activity-areas/wis/grib-edition-1>

⁸ <https://avro.apache.org/>

4 Benchmark Framework

4.1 Technology Stack

In this section, we will introduce the benchmark framework, providing an overview of its components and methodology. The framework encompasses a declaration of the technology stack utilized, as well as a systematic benchmark procedure designed for evaluations. A critical aspect of this framework is the application of the architecture to facilitate comparisons between various file formats. To ensure that the evaluations conducted are comparable, the framework delineates the specific structure of the technology stack. This structured approach is intended to streamline the benchmarking process, allowing for clearer insights and reliable results. The benchmarks are oriented around different storage engines and delve into detailed substructures that reflect their unique characteristics and performance metrics. This is achieved through the implementation of a five-layer architecture concept in databases, as referenced in [SSH11], along with considerations of data structures and data access methodologies as outlined in [Se73]. Each layer within the benchmark framework plays a pivotal role in influencing the performance of data access operations. The layered architecture not only serves to articulate the various technology stacks employed in the benchmarking process but also ensures a high degree of comparability across different configurations. This structured approach enhances the validity of the performance evaluations and aids in drawing meaningful conclusions about the efficacy of the technologies being compared.

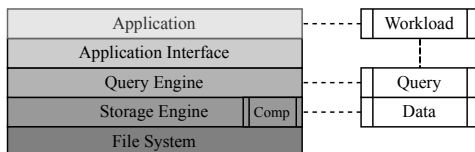


Fig. 1: Layer architecture of the technology stack of the benchmark framework

The architecture can be understood in several distinct layers, each serving a unique purpose in the data management and access process (see Figure 1). At the foundational level, the system interacts directly with the **file system** and the operational drivers of the underlying hardware. This includes managing how data is physically stored, retrieved, and organized on disk drives, as well as how the operating system communicates with the hardware components to ensure efficient input and output operations.

Building upon the file system, the second layer pertains to the **storage engine**, which is responsible for defining the logical structure and organization of the data. This layer includes the mechanisms used to translate physical data into logical representations that are meaningful for application development. Within the memory system, application objects are represented according to their internal memory architecture. This representation goes beyond basic application records and encompasses supplementary data structures, such as index entries. These auxiliary records support efficient data retrieval and manipulation,

ensuring that performance is optimized even as data complexity increases. The layer incorporates a compression mechanism that enhances functionality by reducing data size, thereby optimizing operational efficiency and resource utilization.

The middle layer of the architecture is characterized by the **query engine**. It serves as a vital component in conceptualizing and executing access paths to various data objects. The query engine interprets user requests and translates them into actionable operations, determining the most efficient means of accessing, joining, and aggregating data. Typically, access operations within the query engine are performed as scans that traverse files or utilize established access paths. These operations function as internal cursors, allowing the system to navigate through data collections methodically, thus enabling reliable data retrieval and processing.

The **application interface** layer plays a critical role in providing a uniform mechanism for accessing the query engines. It acts as an intermediary, ensuring that higher-level data models can interact seamlessly with stored data. This layer is designed to abstract the complexities of the underlying data structures, presenting a simplified interface for application developers.

At the topmost level of the architecture lies the **application environment**. This level provides users with a direct pathway to interact with the data, offering user-friendly interfaces and tools to access and manipulate information. The application environment is designed to facilitate various user interactions, ensuring a cohesive experience regardless of the complexity of the underlying layers.

In the context of benchmarking various file formats, the storage engine plays a pivotal role as it serves as the fundamental component that manages how data is stored, retrieved, and structured. The file formats themselves outline the specific architecture of the data objects, detailing both the organization of the actual data and the associated metadata contained within the files. Beginning with the storage engine, multiple additional layers can be constructed to enhance functionality and performance. These layers may vary based on the particular application environment in use; for instance, the choice of application layer, which interacts with the storage layer, will dictate what application interface is utilized. This, in turn, informs the design of the query engine, determining how queries are structured and executed. Moreover, it is crucial that the underlying file system is compatible with the chosen file formats and the storage engine. Compatibility ensures that the data can be effectively read and written, optimizing performance and reliability across the entire system. By considering all these interconnected elements, one can create a robust architecture tailored to facilitate efficient data management and access.

4.2 Workload, Queries & Data

When evaluating benchmark results alongside a technology stack that can be defined with the architecture in 4.1, it is essential to consider that these results are significantly influenced by the specific **workload** defined for the tests. In database environments, this workload is primarily determined by the nature and complexity of the **data queries** executed. Within the query engine layer, various internal processes take place in response to these queries. These processes are critical in transforming the input queries into efficient commands. Additionally, it is essential to emphasize that while reading data as specified by the queries is a fundamental operation, the writing of data also plays a vital role in the overall functionality of applications. This dual focus on **reading** and **writing** is crucial for ensuring data integrity and performance. The complexity of the queries is directly related to the underlying data model employed. A more intricate **data model** typically requires more sophisticated queries to retrieve and manipulate data effectively, which in turn increases the overall workload on the system. Furthermore, the **volume of data** being queried also has a significant impact on the workload; as the dataset grows, so does the challenge of efficiently retrieving and processing the required information. Therefore, both the data model and the amount of data are key factors that influence overall system performance in database environments.

4.3 Procedure Guideline

This section outlines the overall benchmark process, stressing its purpose as a means to evaluate performance across various technology stacks related to specific file formats. It establishes a structured approach while making clear that it should be treated as a guideline rather than a rigid step-by-step protocol. It is essential to understand that the benchmark process serves as a flexible framework to be adapted as necessary rather than a fixed recipe. Within the benchmark framework, emphasis should be placed on the critical tasks that significantly impact performance outcomes. These tasks should be clearly identified and prioritized to streamline the benchmarking efforts.

Identify and establish clear **definitions of technology stacks** that are relevant and comparable, taking into consideration the specific file formats being utilized. This ensures that the benchmarks are meaningful and applicable across different systems and implementations. Carefully **define the set of data** that will be used for benchmarking. The selection of benchmark data is crucial, as it should accurately represent the types of files and operations typically encountered in real-world scenarios. **Design and configure a comprehensive hardware system setup** that meets the specific requirements for implementing the chosen technology stack. This setup should include selecting appropriate components, such as a powerful CPU, adequate RAM, and sufficient storage solutions to ensure smooth operation during experiments. Engage in the **implementation of the chosen technology stacks**, ensuring all components are well-integrated and optimized for performance. Additionally, appropriate **preprocessing of the benchmark data** is essential to prepare it for effective

evaluation and to eliminate any factors that might skew results. **Formulate specific query sets** that will be utilized during the benchmarking process, encompassing a range of operations that reflect realistic use cases. These query sets should be thoughtfully crafted to assess various performance metrics thoroughly. **Execute the benchmarks** as planned, adhering to the defined parameters and maintaining control over variables as necessary. This step is critical to gathering reliable and valid performance data. Finally, rigorously **evaluate the results of the benchmarks**, focusing particularly on performance metrics related to reading and writing operations. This analysis will highlight the strengths and weaknesses of the various technology stacks, aiding in informed decision-making for future implementations.

5 Benchmarks

This section outlines the structure of the benchmark experiments, detailing the components and parameters used for evaluation. It defines the technology stacks for the specific file formats and describes the dataset, including its origin and characteristics. The data preprocessing into various file formats is summarized, and the system setup is outlined, covering hardware, software configurations, and environmental conditions. The benchmark procedure is explained, highlighting methodologies, metrics, and performance indicators. Finally, the results of the benchmarks are presented in a detailed manner.

5.1 Technology Stacks

In the following, we outline the technology stack that will be used for our analysis, specifically focusing on the data file formats that will be examined during the benchmarking process. A decision regarding the appropriate file formats has to be made, as the choice of these formats will significantly influence the construction of our technology stack. The technology stack is carefully selected based on typical requirements observed in data analytics applications within the ESS domain.

The selection of netCDF and Zarr formats is based on their widespread usage within the relevant application domain, making them suitable choices for data storage and management. It is important to note that netCDF and HDF5 share several similarities in their design and functionality. NetCDF4 incorporates technical enhancements derived from HDF5, allowing for improved performance and capabilities. However, HDF5 is not considered further in this context, as it is anticipated that netCDF4 will produce comparable outcomes to those of HDF5, making HDF5 unnecessary for our analysis and exploration. The Parquet format has been chosen for its efficiency in data analytics pipelines and allows for efficient data compression and encoding schemes, making it especially beneficial for large-scale data processing. Its ability to significantly reduce the amount of data read from disk and improved query performance makes it a preferred choice for analytics workloads. TileDB has been

identified as a prevalent solution in the management of scientific data. It serves as a robust backend for handling multidimensional data arrays, offering capabilities for handling large datasets efficiently. The underlying file format in TileDB is essential to consider during benchmarking, as it directly impacts performance, data retrieval speeds, and ease of data manipulation in scientific computations. The CSV format has been selected as a baseline due to its simplicity and widespread use. CSV files are easy to generate and read, making them accessible for quick data exchange and prototyping. By using CSV as a reference point, we can evaluate the performance and efficiency of more complex formats.

Based on the selected data file formats, several technology stacks can be developed and will facilitate comparable data pipelines that cater to various analytical requirements.

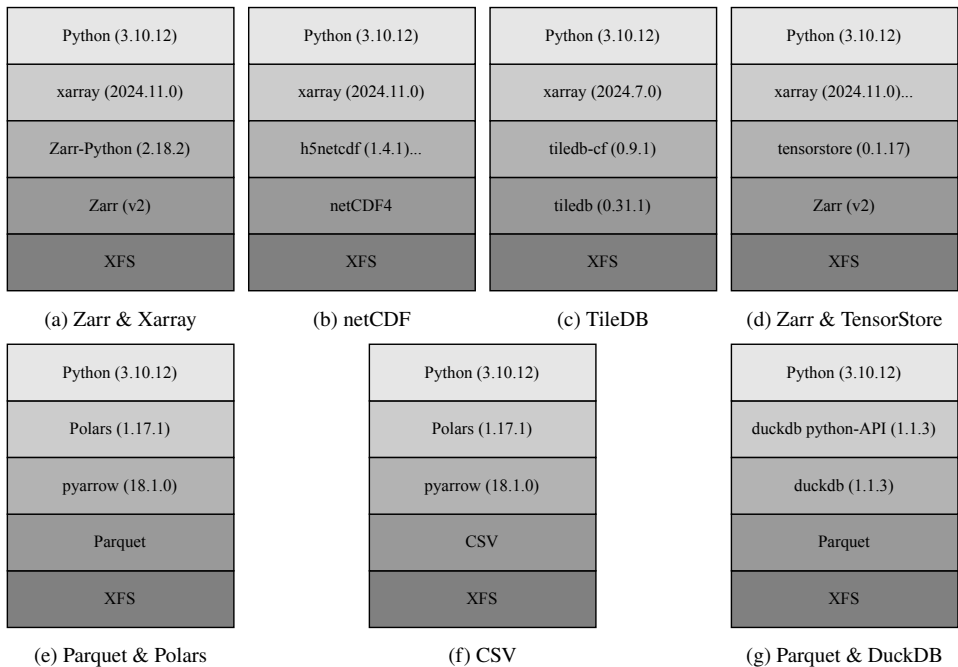


Fig. 2: Technology stacks selected for the benchmarks

The chosen programming environment for this application is Python, which is a versatile and widely used language in data science and analytics. While other languages like Julia and R are also prevalent in the ESS domain, Python has been selected as the primary requirement due to its rich ecosystem and community support. In this Python environment, `xarray`⁹ serves as a typical multidimensional data modeler and interface. It provides a powerful way to manipulate and analyze this data efficiently, making it easier to extract valuable insights. Moreover, `xarray` is compatible with various file formats, including Zarr, netCDF,

⁹ <https://xarray.dev/>

and tileDB (see Figure 2a, 2b and 2c). As a result, three distinct technology stacks can be defined, each tailored to work seamlessly with its corresponding query libraries for Zarr (Zarr-Python 2.18.2), netCDF (h5netcdf 1.4.1), and tileDB (tiledb-cf 0.9.1).

To enhance the existing technology stack, we can incorporate an additional query engine called TensorStore (version 0.1.17), which is specifically designed to handle multi-dimensional data with improved efficiency and scalability. Alongside this, we will integrate a compatible driver for xarray at the application interface(see Figure 2d).

Parquet and CSV files are not compatible with the xarray library. As an alternative, domain users may consider using Polars¹⁰, a high-performance data frame library that excels in handling large datasets. Polars offers seamless integration with both Parquet and CSV file formats, making it a versatile choice for data manipulation and analysis (see Figures 2e and 2f). Additionally, users looking for an efficient way to perform queries on Parquet files might explore the capabilities of DuckDB¹¹, an analytical query engine that can handle large-scale data and is particularly optimized for executing SQL queries directly on Parquet files. This combination of tools provides several options for effective data processing and analysis and define another technology stack (see Figure 2g).

5.2 System Setup

The benchmarking tests were conducted on a high-performance workstation, the detailed specifications of which are outlined in Table 1. While the relative performance comparisons derived from these benchmarks are reliable, it is important to note that the absolute scores could be improved through the implementation of more optimized system configurations.

Tab. 1: Specifications of the system setup

Type	Spec
Processor	AMD EPYC 7402P 24-Core
Memory	128GiB RAM
Storage	2x 8TB HDD in raid 0
OS	Ubuntu 20.04.6 LTS

5.3 Benchmark Data

In the field of earth system science, the use of multidimensional spatial data is highly prevalent. Having access to high-quality datasets is essential for accurate validation of scientific findings. Additionally, open accessibility of the dataset is imperative to ensure transparency, reproducibility, and the ability for other researchers to verify and build

¹⁰ <https://pola.rs/>

¹¹ <https://duckdb.org/>

upon previous findings. ERA-5 reanalysis data [C318] is a widely available reanalysis dataset for climate and weather data. The ERA-5 data is available through the Copernicus Climate Change Service (C3S) as GRIB and NetCDF4 files. It contains a large number of atmospheric, ocean-wave, and land-surface quantities from 1940 to today. The benchmark uses two years’ worth of data for five variables, as shown in Table 2.

Tab. 2: ERA-5 variables used in the benchmark

Variable	Dimensions	Unit	Description
2m temperature	[time, latitude, longitude]	K	Air temperature 2m above the surface.
Soil temperature	[time, latitude, longitude, altitude]	K	Soil temperature between 0-289cm above the surface.

Tab. 3: ERA-5 spatial and time dimensions

Dimensions	Domain	Resolution
time	[1990,1991]	hourly
latitude	[-90,90]	0.25°
longitude	[0,360]	0.25°
altitude	[.035, .175, .64, 1.445]	irregular

Spatial dimensions play a central role in user queries, particularly among scientific users who frequently request data pertaining to these dimensions. A comprehensive description of the dimensions can be found in Table 3. Furthermore, it is important to note that the dataset maintains an hourly time resolution. Due to the size limit per data request, we fetch each month separately, and merge them in the preprocessing step. During preprocessing, we merge the surface temperature variables into a single variable with a fourth dimension encoding the altitude, which is encoded as different variables in the original data. To ensure a level playing field across all formats, we use the same chunking ([time: 1, latitude: 50, longitude: 100, altitude: 1]) and compression algorithm (LZ4, level=6) for all datasets if applicable.

With appropriate preprocessing techniques and compression, variations in storage usage can be detected. NetCDF and Zarr share similarities, but TileDB requires 10-20% more storage capacity (see Table 4). Parquet and CSV stand out as significant outliers due to their differing storage consumption. In the weekly dataset, it is observed that using the Parquet format results in significantly higher storage consumption compared to other formats. However, the ratio is decreasing with larger datasets. On the other hand, CSV files present a contrast in this regard. These files tend to be quite large, resulting in substantial storage requirements. The lack of compression and the textual format of CSVs lead to increased file sizes, making them less favorable for scenarios where storage space is a primary concern.

Tab. 4: Storage consumption at different dataset scales

File Format	Week	Fortnight	Month	Quarter	Year
NetCDF4	1.13 GB	2.30 GB	5.19 GB	15.69 GB	63.29 GB
Zarr	0.86 GB	1.73 GB	3.79 GB	11.42 GB	45.79 GB
TileDB	1.06 GB	2.13 GB	4.64 GB	13.96 GB	55.87 GB
Parquet	7.68 GB	15.35 GB	33.32 GB	99.92 GB	148.00 GB
CSV	39.66 GB	79.32 GB	-	-	-

Tab. 5: List of queries used for the benchmarks

Query ID	Description
Q1	Get 2m temperature for a specific location and point in time.
Q2-7:	Get a [day,week,fortnight,month,quarter,year] worth of 2m temperature data at a specific location.
Q8-13:	Get a [day,week,fortnight,month,quarter,year] worth of 2m temperature data for an area on the map the size of Germany's bounding box.
Q14-19:	Get a [day,week,fortnight,month,quarter,year] worth of 2m temperature data for an area on the map the size of the USA's bounding box.
Q20-25:	Get a [day,week,fortnight,month,quarter,year] worth of 2m temperature data for an area on the map the size of the EU's bounding box.
Q26-28:	Get a [day,week,fortnight] worth of 2m temperature data for the whole map.
Q29-34:	Get a [day,week,fortnight,month,quarter,year] worth of soil temperature data at a specific location and altitude.
Q35-40:	Get a [day,week,fortnight,month,quarter,year] worth of soil temperature data at a specific location across all altitudes.

5.4 Benchmark Queries

The queries are chosen such that they emulate common data access patterns that occur during compute workloads. Typical compute workloads for this data include temporal and spatial aggregate operations like monthly temperature averages. The workload is designed as follows. The most basic workload entails querying data for a specific location and point in time (Q1), exemplified by a temperature information query. Expanding the temporal scope to encompass day, week, fortnight, month, quarter, and year (Q2-Q7) increases workload complexity. The workload is diversified by introducing modifications related to the spatial dimension. Successive increments in workload involve retrieving data for the coordinates of Germany (Q8-Q13), USA (Q14-Q19), Europe (Q26-Q28), and the entire global map (Q26-Q28), the latter within constrained time frames. The final set of queries includes altitude as an additional spatial dimension, encompassing specific points across various time frames (Q29-Q34) and all altitude information (Q35-Q40). The set of queries is listed in Table 5.

5.5 Results

We carried out a series of benchmark experiments to evaluate the performance and efficiency of the file formats within the defined technology stacks. The benchmark code is available on web resources [OP25], where a comprehensive overview of the results is shown with detailed tables (cf. Table 6) and plots that illustrate our findings.

5.5.1 Query Runtime

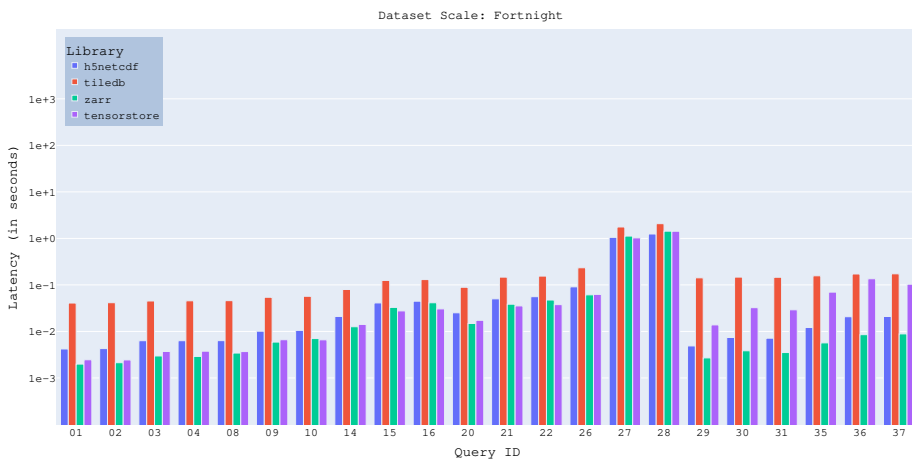


Fig. 3: Query runtime per file format with xarray stacks for fortnight data set size

To begin our analysis, we will conduct a thorough investigation of the query runtime results obtained during the benchmarking process. We examine the results of a benchmark run on two data scales: a small dataset spanning a fortnight and a larger dataset covering a full year. This comparison highlights how performance metrics differ with varying dataset sizes. The performance of TileDB is suboptimal for small queries but is near equal to NetCDF4 and Zarr for larger queries. Further, NetCDF4 exhibits a growing disparity compared to Zarr as the query result size increases. NetCDF and Zarr demonstrate notable scalability with respect to query result size. Finally, the performance of Zarr can be further improved on larger data scales with the stack, including TensorStore.

Although the total runtime has decreased, we can still discern the performance characteristics, as illustrated in Figure 9. The query runtime is directly proportional to the dataset size across all file formats. Across all dataset scales, TileDB exhibits markedly inferior performance, with the most significant disparity observed in large datasets. There appears to be a discernible overhead in query execution for TileDB, particularly noticeable in the context of small queries, as highlighted in Q31.

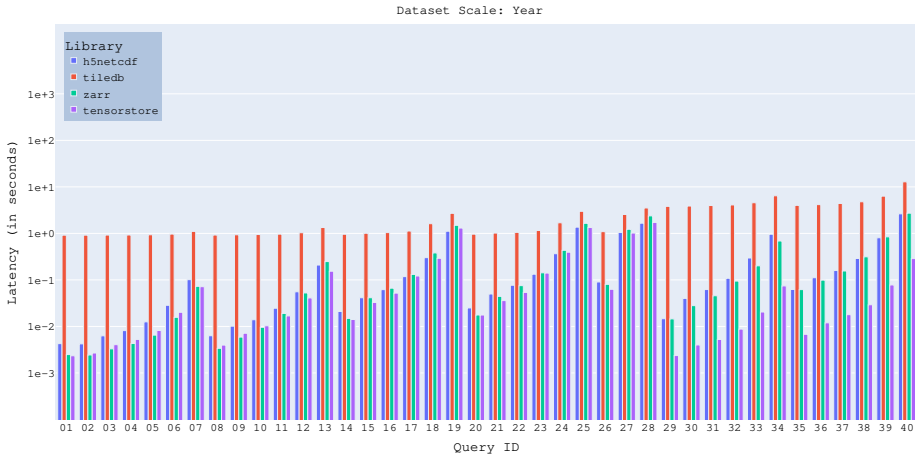


Fig. 4: Query runtime per file format with xarray stacks for year data set size

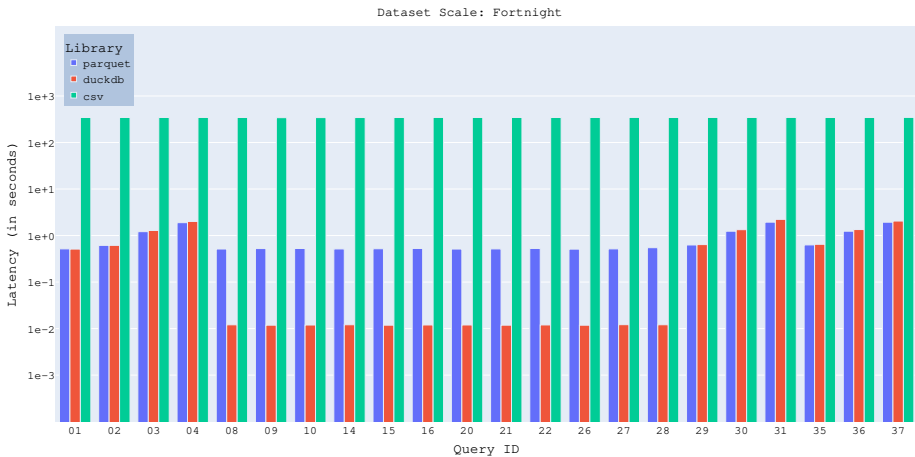


Fig. 5: Query runtime per file format with other stacks for fortnight data set size

When analyzing performance, the queries executed on Parquet files using Polars and DuckDB demonstrate a remarkable consistency in their execution time. It appears there is an inherent overhead associated with the processing that acts as a baseline for these operations. In contrast, when comparing various data formats, CSV files show significantly poorer performance across all technology stacks, highlighting their inefficiency.

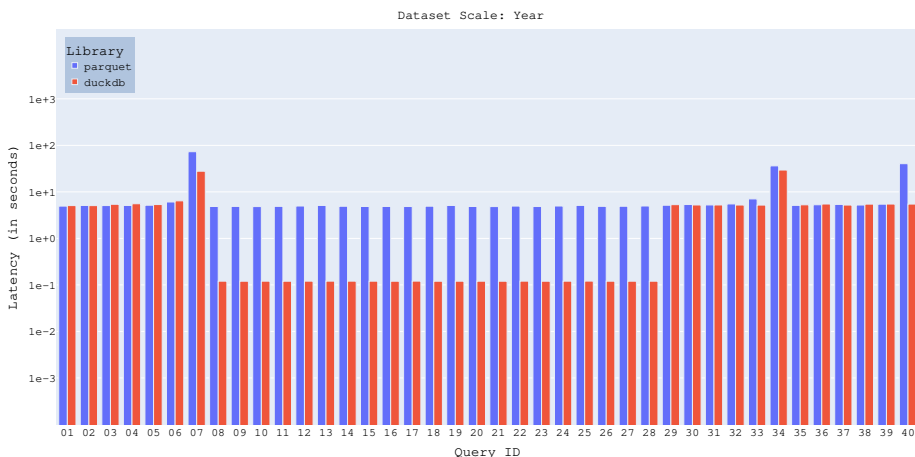


Fig. 6: Query runtime per file format with other stacks for year data set size

5.5.2 Write Runtime

In this section, we summarize the findings from our investigation into write runtime performance. We stored the result sets of the queries in the specified file format of a technology stack and assessed the duration of the writing process. The tensorstore library in the query engine does not include a writing component, making it impossible to trace related information. - as this also holds for duckdb, and CSV files became extremely large on the year scale, the results on the alternative stack could only be compared on the fortnight scale. Given that, this situation also applies to DuckDB, and due to the extensive size of CSV files when analyzed on a yearly scale, we had to limit our comparisons on the alternative stack to a fortnightly scale. Writing Parquet and CSV files with the Polars stack outperforms domain-specific formats for smaller result sets, demonstrating greater efficiency. In contrast, TileDB maintains consistent performance regardless of result set size, while Zarr and NetCDF scale in runtime as the number of results increases. This emphasizes the need to choose formats based on anticipated data size.

With larger datasets, the writing runtime for Parquet shows an increase, indicating reduced efficiency. In contrast, Zarr, NetCDF, and TileDB maintain stable writing runtimes, similar to their performance with smaller datasets, allowing for consistent and effective query results.

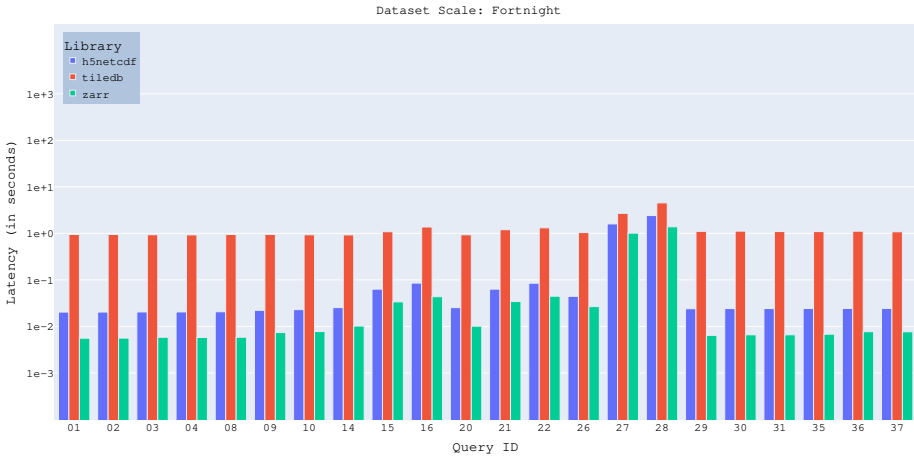


Fig. 7: Write runtime per file format with xarray stacks for fortnight data set size

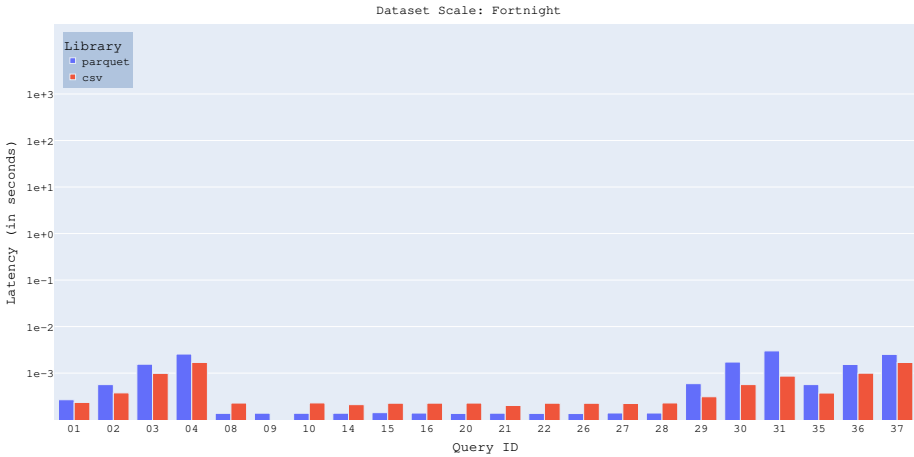


Fig. 8: Write runtime per file format with other stacks for fortnight data set size

6 Discussion

Benchmarking file formats for performance is challenging due to their reliance on specific technology stacks, making it impossible to evaluate them in isolation. Additionally, performance assessments are often tailored to specific workloads, which leads to varying results based on context. The performance of TileDB is suboptimal for small queries but is near equal to NetCDF4 and Zarr for larger queries. Further, NetCDF4 exhibits a growing disparity compared to Zarr as the query result size increases. Zarr and NetCDF demonstrate

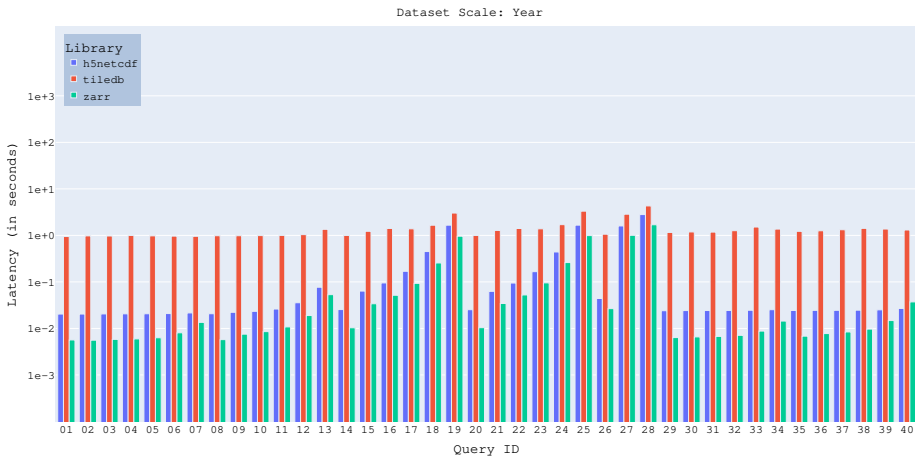


Fig. 9: Write runtime per file format with xarray stacks for year data set size

notable scalability with respect to query result size. Parquet consistently performs well with both Polars and DuckDB, making it a reliable option for data processing. However, it may not match the performance of domain-specific data formats tailored for specific use cases. Nevertheless, as data scales up, the performance of Parquet could improve, enabling it to handle larger datasets more efficiently.

It appears that performance differences between the file formats may be attributed to differences in the implementation of the access libraries rather than inherent properties of the formats themselves. The array formats all store data in a similar layout on a chunk level, with each chunk being a contiguous bit of storage. The divergence among the formats lies in how the chunks are stored on disk. For example, in Zarr, each chunk is stored as its own file, while in NetCDF4, all chunks are stored within one file, and in TileDB, multiple chunks are stored in a fragment file. This suggests that file formats are more about the features enabled by the way chunks are stored on disk, such as compatibility with object stores. At the same time, performance is dependent on the implementation. It should be noted that access to a single chunk is essentially the same across formats, though TileDB stands out as an outlier. It is also observed that implementation has a significant impact on performance, as TileDB is more efficient in its native library but not with xarray. The use of individual files for each chunk in Zarr allows for increased I/O performance scalability with additional compute resources, simplifying parallelization. Zarr's straightforward structure facilitates easy feature implementation, resulting in a well-developed and user-friendly toolkit. The relative comparability of the benchmarks remains valid. However, it is worth noting that optimizing system setups could lead to improvements in the absolute numbers recorded.

7 Conclusion

We have established a benchmark to investigate the performance implications of selecting a specific file format for the storage of n-dimensional array data using contemporary tools widely adopted for data analysis. Our research indicates that, currently, Zarr stands out as the preferred file format due to its seamless integration with xarray. Parquet exhibits significant potential for performance enhancement in the context of large-scale data management; however, further investigation is essential to validate this potential and determine its effectiveness comprehensively. In future investigations, it is worth considering the impact of compression on I/O performance for compute workloads. Furthermore, the integration of different technology stacks to accommodate more file formats, databases, or other application environments should be explored. For instance, xtensor or the environments of Julia and R can be utilized to compare various implementations of the same file formats or to contrast them with other data models and architectures. To enhance the benchmarking framework, we should include additional specialized array database management systems, such as rasdaman. This would enable a more thorough comparison against the baseline results, providing insights into the performance of these systems with complex array data. Further, it is crucial to evaluate different file systems to enhance performance [Po09], particularly focusing on distributed storage solutions (e.g., HDFS), as well as flash storage and NVMe-optimized file systems. Another key aspect is the use of distributed computing environments that can process multiple queries alongside distributed file systems.

Bibliography

- [AB23] Ambatipudi, Sriniket; Byna, Suren: A Comparison of HDF5, Zarr, and netCDF4 in Performing Common I/O Operations, February 2023. arXiv:2207.09503 [cs].
- [An11] Antcheva, I.; Ballintijn, M.; Bellenot, B.; Biskup, M.; Brun, R.; Buncic, N.; Canal, Ph; Casadei, D.; Couet, O.; Fine, V.; Franco, L.; Ganis, G.; Gheata, A.; Maline, D. Gonzalez; Goto, M.; Iwaszkiewicz, J.; Kreshuk, A.; Segura, D. Marcos; Maunder, R.; Moneta, L.; Naumann, A.; Offermann, E.; Onuchin, V.; Panacek, S.; Rademakers, F.; Russo, P.; Tadel, M.: ROOT - A C++ framework for petabyte data storage, statistical analysis and visualization. *Computer Physics Communications*, 182(6):1384–1385, June 2011. Elsevier Science.
- [BI20] Blomer, Jakob; Canal, Philippe; Naumann, Axel; Piparo, Danilo: Evolution of the ROOT Tree I/O. In (Doglioni, C; Kim, D; Stewart, GA; Silvestris, L; Jackson, P; Kamleh, W, eds): 24th International Conference on Computing in High Energy and Nuclear Physics (CHEP 2019). volume 245 of EPJ Web of Conferences. EDP Science, 2020.
- [BNE13] Boncz, Peter A.; Neumann, Thomas; Erling, Orri: TPC-H Analyzed: Hidden Messages and Lessons Learned from an Influential Benchmark. In: TPCTC'13. pp. 61–76, 2013.
- [Bo17] Boufeaa, Aikaterini; Finkers, Richard; van Kaauwen, Martijn; Kramer, Mark; Athanasiadis, Ioannis N.: Managing Variant Calling Files the Big Data Way: Using HDFS and Apache Parquet. In: Proceedings of the Fourth IEEE/ACM International Conference on Big Data Computing, Applications and Technologies. BDCAT '17, Association for Computing Machinery, New York, NY, USA, pp. 219–226, 2017. event-place: Austin, Texas, USA.

- [C318] C3S: ERA5 hourly data on single levels from 1940 to present, 2018.
- [Cu10] Cudré-Mauroux, Philippe; Kimura, Hideaki; Lim, Kian-Tat; Rogers, Jennie; Madden, Samuel; Stonebraker, Michael; Zdonik, Stanley B.; Brown, Paul G.: *SS-DB : A Standard Science DBMS Benchmark*. 2010.
- [El11] Eltabakh, Mohamed Y.; Tian, Yuanyuan; Özcan, Fatma; Gemulla, Rainer; Krettek, Aljoscha; McPherson, John: *CoHadoop: flexible data placement and its exploitation in Hadoop*. *Proc. VLDB Endow.*, 4(9):575–585, June 2011. Publisher: VLDB Endowment.
- [Ga09] Gao, Kui; Liao, Wei-keng; Choudhary, Alok; Ross, Robert; Latham, Robert: *Combining I/O operations for multiple array variables in parallel netCDF*. In: *2009 IEEE International Conference on Cluster Computing and Workshops*. pp. 1–10, August 2009. ISSN: 2168-9253.
- [Ga11] Gao, Kui; Jin, Chen; Choudhary, Alok; Liao, Wei-keng: *Supporting computational data model representation with high-performance I/O in parallel netCDF*. In: *2011 18th International Conference on High Performance Computing*. pp. 1–10, December 2011. ISSN: 1094-7256.
- [GB14] Gubichev, Andrey; Boncz, Peter: *Parameter Curation for Benchmark Queries*. In: *TPCTC'14*. pp. 113–129, 2014.
- [Li03] Li, Jianwei; Liao, Wei-keng; Choudhary, Alok; Ross, Robert; Thakur, Rajeev; Gropp, William; Latham, Rob: *Parallel netCDF: A Scientific High-Performance I/O Interface*, June 2003. arXiv:cs/0306048.
- [MMB16] Merticariu, George; Misev, Dimitar; Baumann, Peter: *Towards a General Array Database Benchmark: Measuring Storage Access*. In (Rabl, Tilmann; Nambiar, Raghunath; Baru, Chaitanya; Bhandarkar, Milind; Poess, Meikel; Pyne, Saumyadipta, eds): *Big Data Benchmarking*, volume 10044, pp. 40–67. Springer International Publishing, Cham, 2016. Series Title: *Lecture Notes in Computer Science*.
- [OP25] Osterhun, Arne; Pohl, Matthias: *FoxBench - Benchmark of Array File Formats (Code Repository)*, 2025. <https://doi.org/10.5281/zenodo.14644865>.
- [Po09] Polte, Milo; Lofstead, Jay; Bent, John; Gibson, Garth; Klasky, Scott A.; Liu, Qing; Parashar, Manish; Podhorszki, Norbert; Schwan, Karsten; Wingate, Meghan; Wolf, Matthew: *...and eat it too: high read performance in write-optimized HPC I/O middleware file formats*. In: *Proceedings of the 4th Annual Workshop on Petascale Data Storage*. PDSW '09, Association for Computing Machinery, New York, NY, USA, pp. 21–25, 2009. event-place: Portland, Oregon.
- [Se73] Senko, Michael E.; Altman, Edward B.; Astrahan, Morton M.; Fehder, PI L: *Data structures and accessing in data-base systems, I: Evolution of information systems*. *IBM Systems Journal*, 12(1):30–44, 1973.
- [SSH11] Saake, Gunter; Sattler, Kai-Uwe; Heuer, Andreas: *Datenbanken: Implementierungstechniken*. mitp-Verlag, 2011.
- [St93] Stonebraker, Michael; Frew, Jim; Gardels, Kenn; Meredith, Jeff: *The SEQUOIA 2000 storage benchmark*. In: *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*. ACM, Washington D.C. USA, pp. 2–11, June 1993.

Tab. 6: Benchmark results per file format with xarray stacks for year data set size

Query	Read				Write			
	Zarr	netCDF	tileDB	tensorstore	Zarr	netCDF	tileDB	tensorstore
01	0.0025	0.0043	0.9161	0.0023	0.0057	0.0204	0.945	-
02	0.0024	0.0042	0.9155	0.0027	0.0056	0.0204	0.9757	-
03	0.0033	0.0063	0.9207	0.0041	0.0058	0.0205	0.9705	-
04	0.0043	0.0081	0.9239	0.0053	0.006	0.0206	0.9934	-
05	0.0065	0.0125	0.9327	0.0082	0.0063	0.0207	0.9766	-
06	0.0157	0.0284	0.9663	0.0201	0.0081	0.0209	0.9647	-
07	0.0728	0.1012	1.1028	0.0716	0.0135	0.0215	0.9558	-
08	0.0034	0.0063	0.9221	0.004	0.0057	0.0207	0.983	-
09	0.0059	0.0101	0.9327	0.0071	0.0075	0.0222	0.9836	-
10	0.0096	0.0139	0.9457	0.0103	0.0085	0.0233	0.9896	-
11	0.019	0.0246	0.963	0.0168	0.0108	0.0261	0.9977	-
12	0.0525	0.0556	1.0326	0.041	0.0189	0.0357	1.0389	-
13	0.2477	0.2086	1.3356	0.1535	0.0532	0.0768	1.3497	-
14	0.0149	0.021	0.9568	0.0141	0.0103	0.0254	0.996	-
15	0.0414	0.0414	1.0062	0.0327	0.0339	0.0635	1.2244	-
16	0.0665	0.0618	1.04	0.052	0.0516	0.095	1.4102	-
17	0.1311	0.1178	1.1254	0.1215	0.0931	0.1692	1.3851	-
18	0.3806	0.301	1.6154	0.2913	0.2555	0.4494	1.6556	-
19	1.491	1.1136	2.6996	1.303	0.9597	1.6599	3.0214	-
20	0.0176	0.025	0.9611	0.0176	0.0104	0.0253	0.9947	-
21	0.0442	0.0498	1.0148	0.0359	0.0343	0.0627	1.2795	-
22	0.0753	0.0767	1.0522	0.0538	0.0524	0.0945	1.4075	-
23	0.1421	0.1328	1.1519	0.1403	0.0954	0.1661	1.3845	-
24	0.4326	0.3658	1.6929	0.3932	0.2606	0.4401	1.7013	-
25	1.6452	1.3637	2.983	1.3362	0.9905	1.661	3.316	-
26	0.0803	0.0903	1.0896	0.0627	0.0266	0.044	1.061	-
27	1.2264	1.0437	2.5499	1.0162	1.0042	1.5959	2.8549	-
28	2.3788	1.6576	3.5204	1.7204	1.6915	2.8037	4.3003	-
29	0.0145	0.0146	3.8104	0.0024	0.0064	0.024	1.1574	-
30	0.028	0.0401	3.8729	0.004	0.0066	0.0242	1.1869	-
31	0.0457	0.062	3.9699	0.0052	0.0067	0.0243	1.1768	-
32	0.0943	0.1084	4.0873	0.0088	0.0071	0.0244	1.2631	-
33	0.2009	0.2959	4.5643	0.0204	0.0088	0.0246	1.5128	-
34	0.6871	0.9573	6.4294	0.0744	0.0144	0.0253	1.3659	-
35	0.0619	0.0623	4.0102	0.0068	0.0068	0.0243	1.2231	-
36	0.099	0.1114	4.1521	0.012	0.0077	0.0245	1.2581	-
37	0.1555	0.159	4.4038	0.0179	0.0084	0.0245	1.3226	-
38	0.3141	0.2888	4.7861	0.0292	0.0097	0.0247	1.4161	-
39	0.8434	0.8058	6.3211	0.078	0.0148	0.0251	1.37	-
40	2.723	2.6326	12.848	0.2874	0.0372	0.0269	1.307	-