RESEARCH-ARTICLE

Performance Analysis of an Efficient Algorithm for Feature Extraction from Large Scale Meteorological Data Stores

MATHILDE LEURIDAN, European Centre for Medium-Range Weather Forecasts, Reading, U.K.

CHRISTOPHER BRADLEY, European Centre for Medium-Range Weather Forecasts, Reading, U.K.

JAMES HAWKES, European Centre for Medium-Range Weather Forecasts, Reading, U.K.

TIAGO QUINTINO, European Centre for Medium-Range Weather Forecasts, Reading, U.K.

MARTIN SCHULTZ, Research Center Juelich GmbH, Jülich, Germany

Open Access Support provided by:

European Centre for Medium-Range Weather Forecasts

Research Center Juelich GmbH

Published: 16 June 2025

Citation in BibTeX format

PASC '25: Platform for Advanced Scientific Computing Conference
June 16 - 18, 2025
Brugg-Windisch, Switzerland

Conference Sponsors:
SIGHPC

Performance Analysis of an Efficient Algorithm for Feature Extraction from Large Scale Meteorological Data Stores

Mathilde Leuridan
European Center for Medium-Range Weather Forecasts (ECMWF) and University of Cologne
mathilde.leuridan@ecmwf.int

Christopher Bradley
European Center for Medium-Range Weather Forecasts (ECMWF)
chris.bradley@ecmwf.int

James Hawkes
European Center for Medium-Range Weather Forecasts (ECMWF)
james.hawkes@ecmwf.int

Tiago Quintino
European Center for Medium-Range Weather Forecasts (ECMWF)
tiago.quintino@ecmwf.int

Martin Schultz
Forschungszentrum Juelich (FZJ) and University of Cologne
m.schultz@fz-juelich.de

ABSTRACT

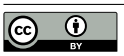
In recent years, Numerical Weather Prediction (NWP) has undergone a major shift with the rapid move towards kilometer-scale global weather forecasts and the emergence of AI-based forecasting models. Together, these trends will contribute to a significant increase in the daily data volume generated by NWP models. Ensuring efficient and timely access to this growing data requires innovative data extraction techniques. As an alternative to traditional data extraction algorithms, the European Centre for Medium-Range Weather Forecasts (ECMWF) has introduced the Polytope feature extraction algorithm. This algorithm is designed to reduce data transfer between systems to a bare minimum by allowing the extraction of non-orthogonal shapes of data. In this paper, we evaluate Polytope's suitability as a replacement for current extraction mechanisms in operational weather forecasting. We first adapt the Polytope algorithm to operate on ECMWF's FDB (Fields DataBase) meteorological data stores, before evaluating this integrated system's performance and scalability on real-time operational data. Our analysis shows that the low overhead of running the Polytope algorithm, which is in the order of a few seconds at most, is far outweighed by the benefits of significantly reducing the size of the extracted data by up to several orders of magnitude compared to traditional bounding box methods. Our ensuing discussion focuses on quantifying the strengths and limitations of each individual part of the system to identify potential bottlenecks and areas for future improvement.

CCS CONCEPTS

• **Mathematics of computing** → **Mathematical software performance**; • **Information systems** → **Data access methods**; **Extraction, transformation and loading**.

KEYWORDS

Data Extraction, Data Management, Numerical Weather Prediction



PASC '25, June 16–18, 2025, Brugg-Windisch, Switzerland

ACM ISBN 979-8-4007-1886-1/2025/06
https://doi.org/10.1145/3732775.3733573

ACM Reference Format:
Mathilde Leuridan, Christopher Bradley, James Hawkes, Tiago Quintino, and Martin Schultz. 2025. Performance Analysis of an Efficient Algorithm for Feature Extraction from Large Scale Meteorological Data Stores. In *Platform for Advanced Scientific Computing Conference (PASC '25), June 16–18, 2025, Brugg-Windisch, Switzerland*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3732775.3733573

1 INTRODUCTION

Meteorological data ranks among one of the largest sources of scientific data today. Modern weather forecasting models now generate high-resolution data for hundreds of parameters and ensemble members, extending over medium-range periods of up to 10-15 days. Each day, these models produce hundreds of terabytes, forming massive high-dimensional datacubes. As we push towards kilometer-scale global weather forecasting [19, 27] and increasingly leverage AI [11, 12] to enable more frequent model runs, the volume of this data is set to continue growing exponentially over the coming years.

Once available, this data is used in a wide range of applications, from disaster prevention to climate research or resource management. In many cases, users want to extract data over specific regions of interest. For instance, one user might want to request precipitation levels for a local area to evaluate the risk of flash floods [1], while another needs wind speed data over wind farm locations in the Baltic Sea to estimate energy production [20] over the next two days.

In most current systems however, data access is limited to orthogonal queries [6, 8]. Users are required to specify ranges along the datacube's metadata axes, rather than being able to extract non-orthogonal shapes, such as the ones described above, directly. As a result, users often retrieve bounding boxes around their area of interest, leading to the extraction of excess data.

To address this limitation, ECMWF introduced a feature extraction algorithm called Polytope [13]. Polytope computes the exact metadata indices along the datacube's axes that fall within an arbitrary user-defined shape. When combined with a data store that enables individual byte access, this algorithm allows the system to read and retrieve only the specific bytes of interest to the user. Polytope is an integral component of both the ECMWF Software EnginE (ESEE), which is the smart software layer enabling all data provision and

workflow management services at ECMWF, and the Destination Earth Digital Twin Engine (DTE) [7].

In this paper, we integrate the Polytope feature extraction algorithm[1] with the FDB meteorological data store [21, 22]. In particular, we introduce a new component, GribJump[2], which enables single byte access to data stored in GRIB format [25] within the FDB. This operation, which was not previously supported by the FDB, is an essential step towards unlocking Polytope's full functionality of accessing only data within a given user-specified shape. We then perform a detailed study of the performance and scalability of this integrated system. There are many different aspects to consider during this analysis, from algorithmic to more technical considerations. We identify these factors and investigate how they affect the overall system's performance. Thereafter, we explore the impact of this data extraction technique on broader operational NWP systems, particularly in terms of reducing the data transfer between such systems and their users. We finally discuss the implications of our results before highlighting possible future improvements to Polytope and GribJump.

2 FEATURE EXTRACTION ON METEOROLOGICAL DATA STORES

2.1 Meteorological Data Storage

NWP models account for the daily production of hundreds of terabytes of data. ECMWF's Integrated Forecasting System (IFS) [16] for example, generates around 400 terabytes (TB) of meteorological data every day [5]. With expected improvements in model resolution, such as those planned under the European Union's Destination Earth initiative [10, 24], daily data output is expected to exceed a petabyte in the coming years.

Once generated, this data is stored in a meteorological object store for several days to remain easily accessible to users. A widely used system for this purpose is the FDB [21, 22], which supports ECMWF operations as well as large European projects such as Destination Earth. As an indexed data store, the FDB is optimized for efficient data retrieval using pre-defined metadata schemas. All data within the FDB is stored in GRIB (General Regularly distributed Information in Binary form) [25] messages, where each message represents a distinct atmospheric field. The data within a message is fully specified by its associated metadata, such as the date and time at which the data was generated or the vertical level and time step of the forecast to which this data corresponds. Additionally, every message contains a latitude-longitude grid, onto whose indices data values are mapped. This allows the data to be represented spatially across the Earth.

Many operational models are computed on high-resolution grids, containing millions of points. The O1280 octahedral grid used in ECMWF's IFS model, for example, consists of approximately 6.5 million points [14]. As a result, the individual fields stored within each message are about 12 megabytes (MB) in size, with a complete forecast over all 145 time steps for a single parameter adding up to over a gigabyte (GB).

[1] https://github.com/ecmwf/polytope
[2] https://github.com/ecmwf/gribjump

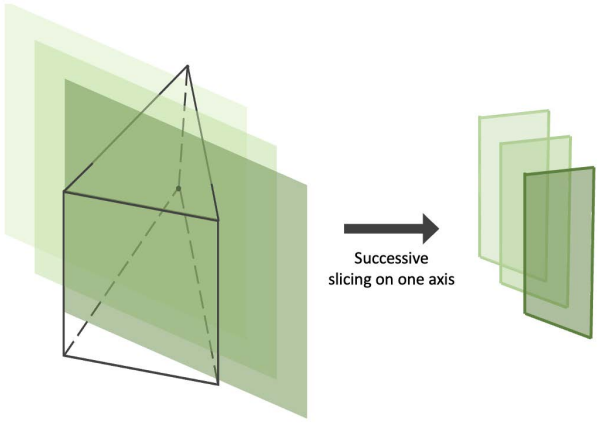


Figure 1: Polytope slicing mechanism, as illustrated in [13].

2.2 Polytope

Existing data access methods on the FDB only allow the retrieval of complete fields of data, requiring all of the data in those fields to be read and then returned to the user. As field sizes continue to grow due to increasing forecast resolution, it becomes more challenging to perform this operation for many users at a time as part of a time-sensitive workflow. To help mitigate such challenges, novel data extraction tools are needed.

One such tool is the recently developed Polytope feature extraction algorithm [13], designed to efficiently intersect arbitrary multi-dimensional polytopes with an iso-latitude datacube. The algorithm performs a series of recursive "slicing" steps, gradually reducing an n-dimensional polytope into a set of 1-dimensional points. At each step, several successive n-dimensional slices of the polytope are taken along one of the datacube's axes, as pictured in Figure 1. This process is then repeated for each axis until all points within the queried polytope are identified. Throughout the slicing process, a result tree is iteratively constructed to record all of the points found by the algorithm. This tree also provides guidance on the remaining slicing operations to be performed, detailing which lower-dimensional polytopes should intersect with which axes next.

The slicing algorithm itself does not directly extract data from the datacube however. It merely calculates the coordinates of the data points to be retrieved. For actual data extraction, the algorithm must be integrated with a compatible datacube backend that can both provide information on the available data within the datacube as well as support byte-level access to retrieve only the relevant data points.

2.3 GribJump

There is currently no well-established way to access individual bytes within fields stored on an FDB. To address this limitation, we introduce a new software tool, GribJump, designed to enable such byte-level access within the FDB. GribJump allows us to extract only the bytes found to be in the user-specified input shapes by the Polytope slicing algorithm instead of whole GRIB fields.

The core functionality of GribJump comes from its `extract` method. While data is written to the FDB after a model run, GribJump first

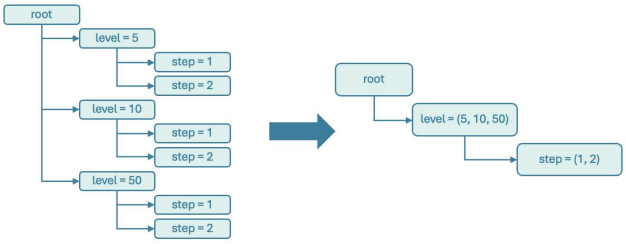


Figure 2: Tree compression example.

pre-computes an index on this data. This index is then used by the extract function to efficiently decode and locate individual bytes within fields stored in the FDB. Both simple packed data [4] and CCSDS compressed data [26] are currently supported by this mechanism.

For faster data access, GribJump has been highly optimized, particularly through parallelisation, allowing data retrieval tasks to run on multiple threads simultaneously based on the objects being accessed.

2.4 Polytope Tree Compression

As users request larger geometric shapes, the tree constructed during the Polytope slicing process can rapidly expand, resulting in a very broad structure. To address this and simplify the resulting tree, we propose a custom tree compression method, illustrated in Figure 2. This compression not only allows for a more compact representation of the results, but it is also used by the algorithm to avoid performing duplicate n-dimensional slices.

Note that not all axes of the datacube can be compressed however. In particular, compression can only be applied to axes that are defined as part of an orthogonal request shape. Due to how the tree is constructed within the Polytope algorithm, axes associated with unions can currently also not be compressed.

For meteorological data stored in an FDB, further constraints on the axis compression arise depending on the grid on which the data is stored. For most iso-latitude grids supported by Polytope, the latitude axis cannot be compressed as subsequent longitude values vary with the latitude values. The longitude axis, however, can always be compressed as it is the final axis stored on the FDB.

2.5 System Overview

In the remainder of this paper, we investigate the performance and scalability of the Polytope algorithm on data stored in an FDB. To support this analysis, we now first provide a brief overview of how both components of the system, Polytope and the FDB (via GribJump), operate together.

Initially, Polytope queries the FDB through GribJump's `axes` call to create an abstract representation of the datacube available in the FDB. This precomputation step occurs once, prior to any feature extraction, and is reusable for multiple requests as long as the data in the FDB remains unchanged.

Following this precomputation, the algorithm enters a dynamic, online phase. Here, Polytope takes a user-defined polytope, applies the slicing algorithm to the abstract datacube, and generates a result tree. This result tree, initially devoid of data, is then populated by querying the FDB through GribJump's `extract` call, after which the fully populated tree is returned to the user.

Figure 3 illustrates these interactions between Polytope and GribJump. This system is offered as part of the Polytope service at ECMWF and will also be available to users of the Destination Earth initiative through the earthkit software environment [18]. Note that this system is specific to data stored in GRIB format due to the fact that GribJump currently only supports byte access from the FDB in this data format. For datasets stored in other formats, such as climate datasets stored in netCDF [17], the Polytope algorithm can still be used to find points contained in given shapes, although specialised single byte access mechanisms will need to be implemented in order to provide an equivalent service to the system described above.

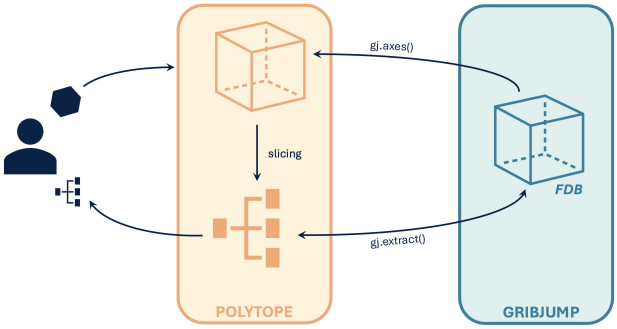


Figure 3: Overview of the Polytope and GribJump system.

3 PERFORMANCE AND SCALABILITY OF THE POLYTOPE SYSTEM

In this section, we analyse the entire Polytope system's performance and scalability. We start by examining the performance of the precomputation step of the algorithm. Note that, unlike the precomputation step, the algorithm's online phase can not be reused for multiple requests and it thus significantly affects the overall performance of the system. The main emphasis of this section is therefore on the online phase. To evaluate the performance of this step, we identify three key metrics: the slicing time, the tree construction time, and the GribJump extraction time. We then investigate how these quantities vary with the query shape to the algorithm.

3.1 Pre-computation phase

The performance of Polytope's pre-computation phase is tied to the performance of the FDB `axes` call, which itself depends heavily on how data is organised within the FDB according to its schema [3, 21, 22]. As more data on the FDB is exposed for Polytope to slice on, the cost of the FDB `axes` call will gradually increase. To access all of the current ECMWF operational data available on the FDB for example, this call takes about 4 seconds and needs to be repeated every 6 hours when new data becomes available. For larger climate datasets, calling `axes` might take longer, but the data is more

static so this cost can be paid less often as the pre-computation phase does not need to be performed as frequently. The data in the Destination Earth Climate digital twin for example is generated once and, while calling `axes` on this data takes close to 30 seconds, the output of this call can easily be cached in memory to be re-used when needed as no new data is produced in this dataset.

3.2 Online phase

In the online phase of the algorithm, three main statistics need to be considered: the slicing time, the tree construction time, and the GribJump extraction time. These metrics are influenced by various factors, ranging from technical aspects, such as the efficiency of GribJump's direct byte access to the FDB, to algorithmic factors like the dimensionality of the requested shape, how the shape is defined by the user, and the number of vertices specifying the shape.
To analyse these influences, we structure our discussion into two parts, distinguishing between technical and algorithmic impacts on the overall system performance. Finally, we conclude this section by illustrating how these combined factors affect extraction times for selected meteorological features.

3.2.1 Technical considerations. The primary technical considerations in the Polytope system stem from the GribJump component of the workflow, which directly interacts with the FDB to access data. Two critical questions arise at this stage. The first one being, how long does it take to request multiple data points from a single field, and the second one being, how does the extraction time then scale when accessing data across multiple fields.
Figure 4 addresses both of these questions. In Figure 4, in red, we observe the initial cost associated with reading and accessing data from a field. Once a field has been accessed however, we see that extracting additional points from the same field has a limited impact

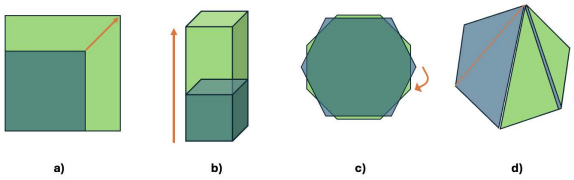


Figure 5: Schemas of the performance experiments.

on the extraction time. The blue lines in Figure 4 then show that the GribJump extraction time scales linearly with the number of fields accessed. We therefore conclude that the GribJump extraction time is mostly influenced by the number of distinct fields accessed, with minimal additional costs incurred when more points are accessed from each field.

3.2.2 Algorithmic considerations. As a computational geometry algorithm, the performance of Polytope is highly influenced by the characteristics of its query shape. In the following, we explore these dependencies in detail through various case studies, first for orthogonal boxes and then for non-orthogonal queries. For clarity, we provide a set of schemas showcasing some of the experiments we perform in Figure 5.

Orthogonal extractions. We begin by focusing our attention on orthogonal box extractions and how they scale in different dimensions.
We first investigate the effect of the shape's 2-dimensional area in Figure 6. In this figure, we extract increasingly large 2-dimensional

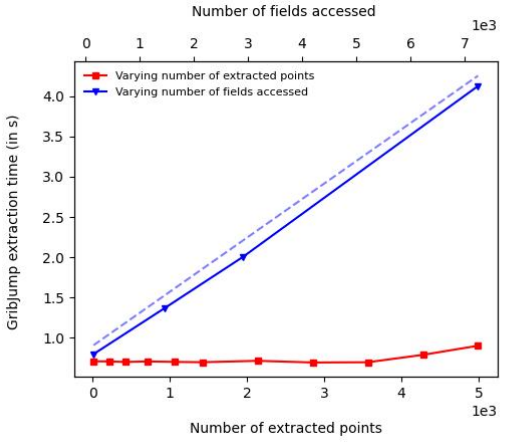


Figure 4: In red, GribJump extraction times for an increasing number of extracted points, where all points are extracted from the same field. In blue, GribJump extraction times for an increasing number of fields accessed, where a single point is extracted from each field and the dashed line shows the linear scaling behaviour of this extraction time.

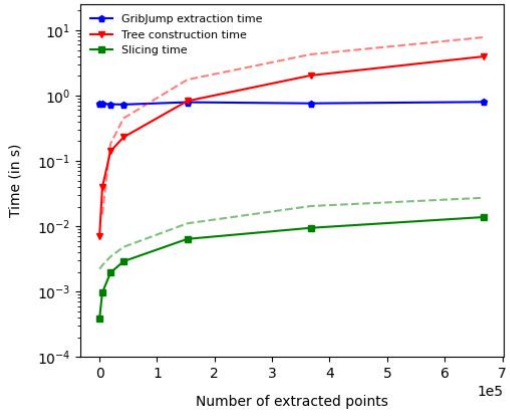


Figure 6: Polytope and GribJump extraction times for increasing 2D boxes (Figure 5a). The dashed red line represents the expected linear behaviour of the tree construction time, whilst the dashed green line represents the expected square root behaviour of the slicing time. The dashed lines are multiplied by a constant factor to enhance visibility.

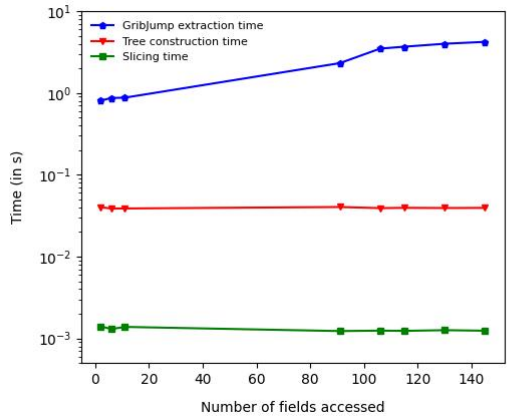


Figure 7: Polytope and GribJump extraction times for increasing 3D boxes (Figure 5b).

squares in latitude-longitude space. Each box is constructed by moving the top corner of the square further away from its lower corner. In Figure 6, in red, we observe a linear relationship between the increasing area and the tree construction time. This figure actually shows that most of the extraction time is spent constructing the result tree, with the slicing time only representing a small fraction of the overall extraction time. Note here that the tree construction time is relatively large as it not only includes the construction of the nodes of the tree, but also all of the time spent within the algorithm to set up subsequent slices. In particular, a large amount of time is spent ordering longitude values as we successively add them to the latitude tree nodes. This causes the observed linear behaviour of the tree construction time. In contrast, as we see in green in the figure, the slicing time scales with the square root of the number of extracted points. This is due to the fact that, as many meteorological grids are not iso-longitude, we do not compress the latitude axis in the extraction algorithm. We thus only perform slices for each latitude value found within the shape, which, because the sliced shapes are squares, approximately corresponds to the square root of the number of extracted points. As established before however, the extraction time is driven by the tree construction time and thus also scales linearly with the area here. Interestingly, the GribJump extraction time stays constant throughout the plot and is not impacted by the growing area of the square. This validates our earlier findings, as all points accessed here stem from the same field and thus most of the GribJump extraction cost comes from the initial object look up.

Next, we investigate the impact of slicing across multiple fields. In Figure 7, we extract 3-dimensional boxes in latitude, longitude and step. We construct these boxes by keeping the base latitude-longitude square constant, while varying the step extent of the boxes. As expected, we see in this figure that the GribJump extraction time gradually increases as we extract data from more fields. We however observe that neither the slicing nor the tree construction times vary significantly with the number of fields. This is due to the fact that we compress the boxes extracted here along the step axis, only slicing along this axis once. The resulting 2-dimensional latitude-longitude squares are the same for every box and the slicing and tree construction times are therefore not impacted by the step extension of the boxes. Figure 7 is thus a critical reminder that, as we move to higher-dimensional shapes, the performance of the system not only depends on the extracted volume, but also, perhaps more importantly, on how data is laid out in the FDB over various fields. This can further be witnessed in Figure 8, and Figure 8(a) in particular, where we scale the box extraction to higher dimensions, still keeping the base latitude-longitude square constant.

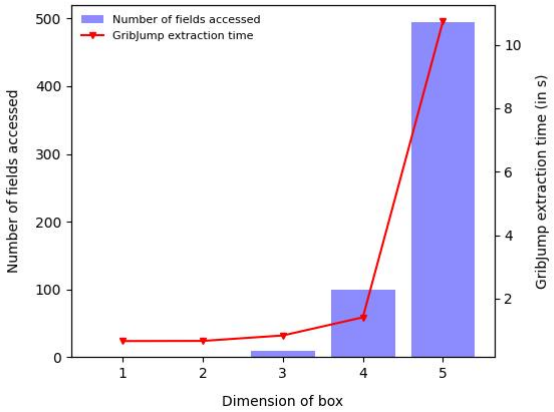
To understand how higher-dimensional slices impact the slicing time, we now extract boxes with increasing dimensions in Figure 8. We construct these boxes by successively taking the tensor product of the previous box with a range in a new dimension. In Figure 8(b), we show the number of n-dimensional slices performed for each dimension as well as the slicing time of each box. We then scale this total slicing time by the number of slices to estimate the slicing time of each individual n-dimensional slice. As we observe in the figure, 1- and 2-dimensional slices are relatively inexpensive, with slices in 4 and 5 dimensions becoming relatively slow to perform. The jump in costs between slices is due to the use of convex hulls in our slicing step, which gradually scales worse as the dimension increases. Moreover, since the latitude axis is not compressed, we need to perform all of the 2-dimensional slices along each latitude value when we move to higher-dimensional shapes. This explains the jump in both the number of slices and the total slicing time that we observe in Figure 8(b) as we extract a 2-dimensional box.

Axis compression in the request tree, as shown in Figure 2, has already been mentioned a few times throughout this section. To gain a better understanding of it, we investigate how this compression affects the different extraction times. In Figure 9, we extract boxes of increasing sizes while varying which of their axes are compressed. This shows that compression can significantly affect the performance of the slicing algorithm. In particular, we note that, as less axes are compressed, more nodes need to be created and the request tree becomes wider. This eventually results in a faster linear increase of the slicing time with the number of extracted points, which does not scale well to high-dimensional extractions.

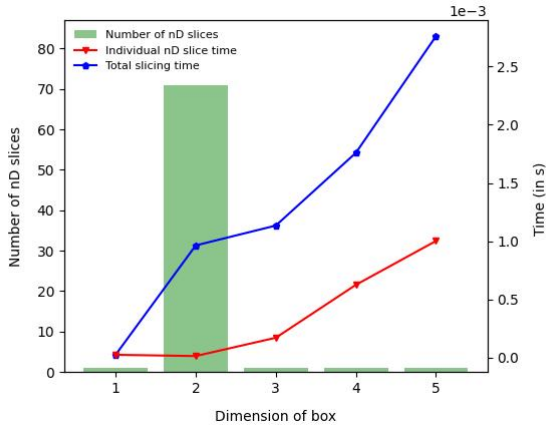
Non-orthogonal extractions. We now shift focus to explore how various shape properties affect the performance of non-orthogonal extractions.

We first analyse the effect of extracting very detailed polygons with many defining vertices. In Figure 10, we extract different polygons with an increasing number of vertices. As these polygons are constructed by approximating the same circle, they all have a very similar area so the extraction times should not change significantly. In particular, we see that the GribJump extraction time stays constant throughout, as expected. More interestingly, we observe that the slicing time of the algorithm grows with the square of the number of vertices. This can be explained by how we perform slices in the algorithm, taking the intersection of a plane with each line defined between pairs of vertices on either side of this plane.

Finally, we study the impact of constructive geometry operations on the slicing algorithm. In particular, we focus on the union operation here. In Figure 11, we extract the same 1024-sided polygon, but specified as the union of a growing set of sub-polygons. We



(a) GribJump extraction time and number of fields accessed for increasing n-dimensional boxes.



(b) Extraction times and number of nD slices performed for increasing n-dimensional boxes.

Figure 8: nD box extractions.

immediately notice that the slicing time increases linearly with the number of unions defining the shape. This is directly tied to the linear relationship between the number of unions and the number of slices to be performed, which we also observe in this figure. The linear behaviour is expected as we treat each sub-polygon in the union separately in the algorithm. We thus need to slice each sub-polygon separately, leading to a proportional increase in the number of slices.

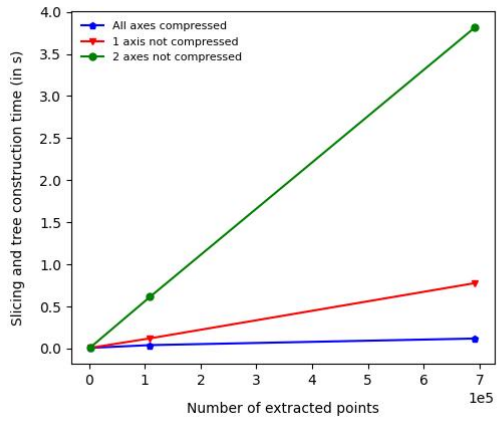


Figure 9: Extraction times of a 3D box with various levels of axis compression. The slicing and tree construction time is shown in blue for when all axes in the box are compressed, in red when the longitude axis is not compressed and in green when both the longitude and step axes are not compressed.

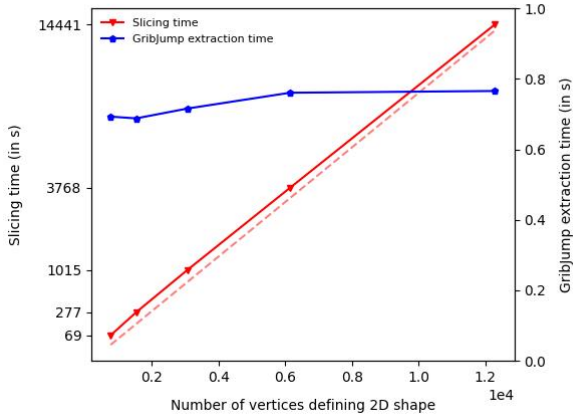


Figure 10: Extraction times of polygons of same area but with an increasing number of defining vertices (Figure 5c). Note that the slicing time axis appears in square root scale with the dashed red line representing the expected squared behaviour of the slicing time.

3.2.3 Extracting meteorological features. In real-world scenarios, the factors described above are combined to influence Polytope's total response time. To assess the algorithm's efficiency for meteorological applications, we examine several examples.
In Figure 12, we analyse data extraction times for different countries, identifying where time is spent. Countries are specified as unions of convex polygons with each polygon being treated independently by the algorithm and no compression of the involved axes. As expected, this impacts the slicing and tree construction times, which are more consequent for larger shapes, such as France

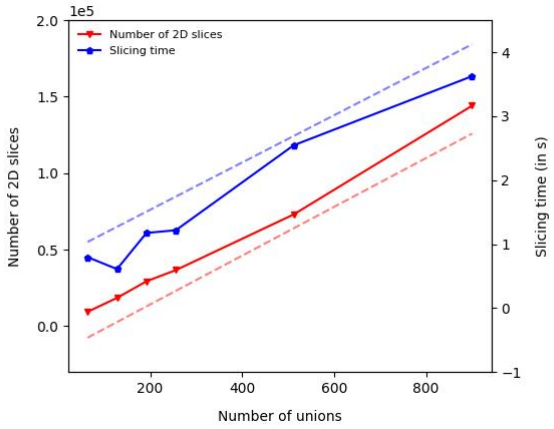


Figure 11: Slicing time and number of 2D slices for a polygon specified as the union of an increasing set of sub-polygons (Figure 5d). The dashed lines represent the corresponding expected linear behaviour of these quantities.

when compared to Switzerland, but also for shapes that have more irregularities, such as the UK when compared to the more rectangular Germany. This is further emphasized in Figure 12(b), where we show the stark contrast between the smaller and less-detailed countries in Europe and the large, more complex shape of Canada, which is exponentially more expensive to extract. Note that the fact that countries are defined as unions currently also affects the GribJump extraction time, which is quite considerable here for a single field extraction. This is due to the fact that each sub-polygon is treated separately as a different request to extract from GribJump and could be optimised in the future.

These examples focus on 2D extractions in latitude-longitude space, so the impact of accessing multiple data fields cannot be observed from this figure. Instead, in Figure 13(a), we address this point, showing a point timeseries extraction across multiple fields. Here, the slicing time does not vary and remains negligible, while the GribJump extraction time significantly increases as more fields are accessed, as expected. Finally, in Figure 13(b), we present an example timeseries extraction over the UK, where both more data points and fields are accessed. In this case, both slicing and GribJump extraction times increase, with the latter scaling linearly with the number of fields accessed.

In all examples, Polytope's extraction times were at most a few seconds, even for more complex cases, with the Canada retrieval taking under a minute. These results suggest that the slicing algorithm is an efficient data extraction method for various meteorological features.

4 I/O AND DATA REDUCTIONS

The main aim of the Polytope feature extraction algorithm is to improve access to large-scale meteorological data. A key aspect of this objective is to minimise the I/O load in operational weather systems by limiting data transfer to only what is essential for specific applications.

Figure 14 illustrates the data reduction achieved with Polytope compared to extracting a bounding box around several example countries. We observe a significant reduction of up to almost an order of magnitude for non-rectangular countries such as Italy. Even for more box-shaped countries like Germany, the reduction remains clearly visible on the graph in logarithmic scale. Note that the examples in the figure are of 2-dimensional extractions. As more fields are accessed however, the potential for data reduction can significantly increase. For high-dimensional shapes, such as spatio-temporal trajectories in particular, data reduction can easily exceed 99%.

This data reduction can then directly be translated into a faster access to data, especially for data spanning over multiple fields. For example, retrieving the point time series from above using Polytope took less than a second, compared to several seconds when retrieving the full fields from the FDB. For the point time series over all ensemble numbers, Polytope completed the extraction in about 4 seconds, while accessing the entire fields directly from the FDB took nearly 2 minutes.

Such improvements in data reduction and performance enable more efficient data access for users. By minimising data transfer between operational systems and their users, Polytope helps reduce the I/O load incurred per user on the storage system. This results in a more efficient data access pattern which can support a large number of users simultaneously.

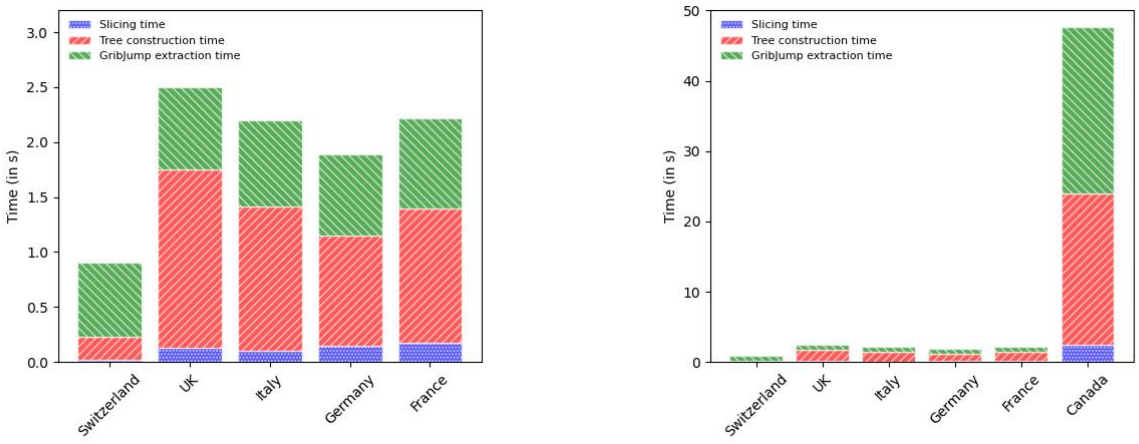
5 DISCUSSION AND FUTURE WORK

In the current context of open data initiatives such as Destination Earth, the Polytope feature extraction algorithm provides a reliable solution to a rising demand in meteorological data access. By limiting the amount of data read and transferred to users to the bare minimum, it represents an efficient and scalable alternative to traditional data extraction techniques. Beyond data retrieval, Polytope also introduces the possibility for interactive workflows as access to data in near real-time starts to become feasible. This paves the way for more responsive applications in the fields of meteorology and climate change.

While Polytope provides substantial benefits to operational weather forecasting systems however, the feature extraction algorithm still has a few notable limitations.

The first challenge we identified in this paper was in the pre-computation phase of the algorithm. In this part of the Polytope system, we experience a trade-off between speed and the range of shapes that can be extracted. To address this, one approach is to integrate the pre-computation phase within the online phase. In particular, we plan to optimise the FDB `axes` call to be called recursively as we build the request tree in future implementations of the slicing algorithm.

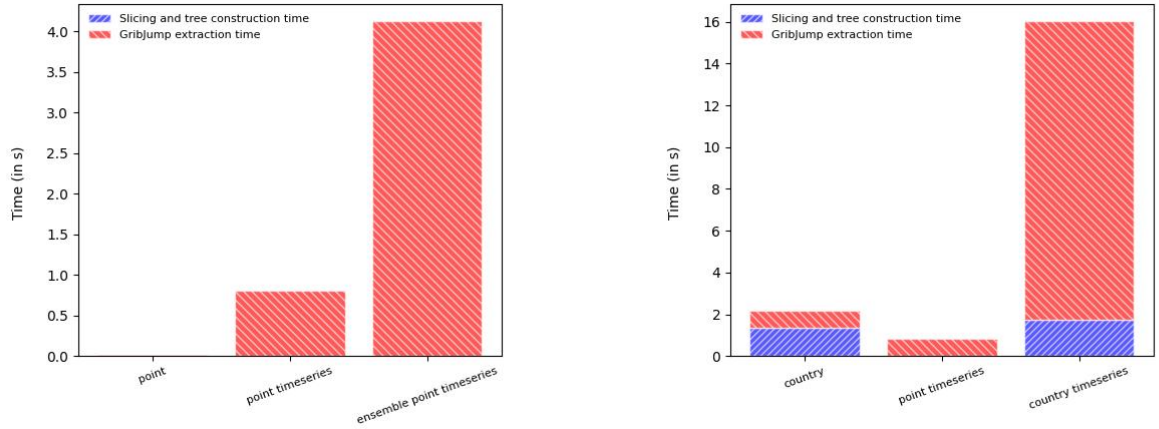
A second challenge that we identified during our performance analysis is that polygons with a large number of vertices are difficult to slice, and their extraction can quickly become computationally intractable. To overcome this, line-simplification methods [2], such as the Douglas-Peucker algorithm [9], can be used to approximate highly detailed polygons by polygons with fewer vertices. However,



(a) European countries

(b) Comparison between European countries and Canada

Figure 12: Extraction times for different country shapes.



(a) Point timeseries

(b) Country timeseries

Figure 13: Extraction times for different timeseries.

for Polytope to ensure that all points within a requested shape are returned to users, the line reduction algorithm that we use must meet an additional requirement. It should consistently overestimate the polygon boundaries to capture the entire shape, whilst still preserving the overall shape of the polygon in order to avoid unnecessary data extraction. As, to the best of our knowledge, there is no existing algorithm which fully meets this criteria, a key area of future work will involve developing and implementing a suitable polygon approximation technique specifically for this purpose.
Additional improvements to the feature extraction algorithm include optimising the tree compression and refining the interface between Polytope and GribJump. Our analysis indicated that tree compression significantly affects overall system performance. Extending this compression to handle more axes and complex shapes will thus be a crucial step toward increasing Polytope's scalability. Enabling GribJump to then work directly with this tree structure will also help reduce the current overhead from translating between different object types within the Polytope-GribJump interface. Furthermore, most of the slicing and tree construction in Polytope is currently implemented in Python and, due to the nature of the algorithm, existing compiled backends such as numpy [23] do not provide an advantage for performance. Significant speed-ups could instead be achieved by writing portions of Polytope in a more performant language, such as C++ or Rust [15].
In the longer term, Polytope's adaptability to different data sources

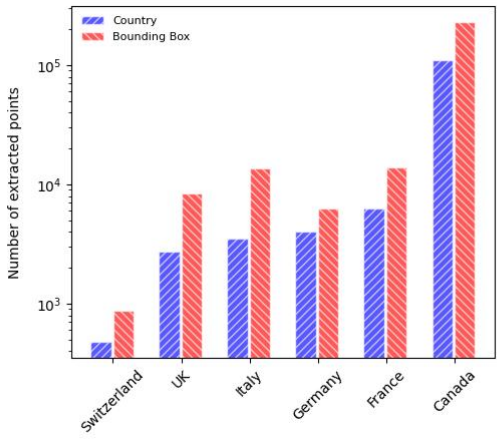


Figure 14: Number of extracted points for different country shapes and their bounding boxes. Note that the number of extracted points is shown in log scale.

is another important area for development. This includes both an assessment of necessary changes to GribJump for efficient data retrieval over networks instead of local file systems, as well as the development of alternative slicing algorithms for data stored on unstructured grids. These changes are particularly relevant for the Destination Earth initiative, where data on the data bridge is transferred over a network and certain datasets are stored on icosahedral and Lambert grids.

ACKNOWLEDGMENTS

The work presented in this paper has been produced in the context of the European Union's Destination Earth Initiative and relates to tasks entrusted by the European Union to the European Centre for Medium-Range Weather Forecasts implementing part of this Initiative with funding by the European Union. Views and opinions expressed are those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

REFERENCES

[1] Karamat Ali, Roshan M Bajracharyar, and Nani Raut. 2017. Advances and challenges in flash flood risk assessment: A review. *Journal of Geography & Natural Disasters* 7, 2 (2017), 1–6.
[2] Robert G Cromley. 1991. Hierarchical methods of line simplification. *Cartography and Geographic Information Systems* 18, 2 (1991), 125–131.
[3] ECMWF. 2017. FDB Documentation. https://fields-database.readthedocs.io/en/latest/ Accessed on 2024-11-12.
[4] ECMWF. 2024. GRIB2 Regulations. https://codes.ecmwf.int/grib/format/grib2/regulations/ Accessed on 2024-11-12.
[5] ECMWF. 2024. Key facts and figures. https://www.ecmwf.int/en/about/media-centre/key-facts-and-figures Accessed on 2024-11-12.
[6] Sandro Fiore, Alessandro D'Anca, Donatello Elia, Cosimo Palazzo, Dean Williams, Ian Foster, and Giovanni Aloisio. 2014. Ophidia: a full software stack for scientific data analytics. In *2014 international conference on high performance computing & simulation (HPCS)*. IEEE, 343–350.
[7] Thomas Geenen, Nils Wedi, Sebastian Milinski, Ioan Hadade, Balthasar Reuter, Simon Smart, James Hawkes, Emma Kuwertz, Tiago Quintino, Emanuele Danovaro, et al. 2024. Digital twins, the journey of an operational weather system into the heart of Destination Earth. *Procedia Computer Science* 240 (2024), 99–108.
[8] Luke Gosink, John Shalf, Kurt Stockinger, Kesheng Wu, and Wes Bethel. 2006. HDF5-FastQuery: Accelerating complex queries on HDF datasets using fast bitmap indices. In *18th International Conference on Scientific and Statistical Database Management (SSDBM'06)*. IEEE, 149–158.
[9] John Edward Hershberger and Jack Snoeyink. 1992. Speeding up the Douglas-Peucker line-simplification algorithm. (1992).
[10] Jörn Hoffmann, Peter Bauer, Irina Sandu, Nils Wedi, Thomas Geenen, and Daniel Thiemert. 2023. Destination Earth–A digital twin in support of climate services.
[11] Remi Lam, Alvaro Sanchez-Gonzalez, Matthew Willson, Peter Wirnsberger, Meire Fortunato, Ferran Alet, Suman Ravuri, Timo Ewalds, Zach Eaton-Rosen, Weihua Hu, et al. 2023. Learning skillful medium-range global weather forecasting. *Science* 382, 6677 (2023), 1416–1421.
[12] Simon Lang, Mihai Alexe, Matthew Chantry, Jesper Dramsch, Florian Pinault, Baudouin Raoult, Mariana CA Clare, Christian Lessig, Michael Maier-Gerber, Linus Magnusson, et al. 2024. AIFS-ECMWF's data-driven forecasting system. *arXiv preprint arXiv:2406.01465* (2024).
[13] Mathilde Leuridan, James Hawkes, Simon Smart, Emanuele Danovaro, and Tiago Quintino. 2023. Polytope: An Algorithm for Efficient Feature Extraction on Hypercubes. *arXiv preprint arXiv:2306.11553* (2023).
[14] Sylvie Malardel, Nils Wedi, Willem Deconinck, Michail Diamantakis, Christian Kühnlein, George Mozdzynski, Mats Hamrud, and Piotr Smolarkiewicz. 2016. A new grid for the IFS. *ECMWF newsletter* 146, 23-28 (2016), 321.
[15] Nicholas D Matsakis and Felix S Klock. 2014. The rust language. *ACM SIGAda Ada Letters* 34, 3 (2014), 103–104.
[16] Anders Persson and Federico Grazzini. 2007. User guide to ECMWF forecast products. *Meteorological Bulletin* 3, 2 (2007).
[17] Russ Rew and Glenn Davis. 1990. NetCDF: an interface for scientific data access. *IEEE computer graphics and applications* 10, 4 (1990), 76–82.
[18] Iain Russell, Tiago Quintino, Baudouin Raoult, Sándor Kertész, Pedro Maciel, James Varndell, Corentin Carton de Wiart, Edward Comyn-Platt, Olivier Iffrig, James Hawkes, et al. 2024. Introducing earthkit. https://www.ecmwf.int/en/newsletter/179/computing/introducing-earthkit
[19] Christoph Schär, Oliver Fuhrer, Andrea Arteaga, Nikolina Ban, Christophe Charpilloz, Salvatore Di Girolamo, Laureline Hentgen, Torsten Hoefler, Xavier Lapillonne, David Leutwyler, et al. 2020. Kilometer-scale climate models: Prospects and challenges. *Bulletin of the American Meteorological Society* 101, 5 (2020), E567–E587.
[20] Mark Schelbergen, Peter C Kalverla, Roland Schmehl, and Simon J Watson. 2020. Clustering wind profile shapes to estimate airborne wind energy production. *Wind Energy Science Discussions* 2020 (2020), 1–34.
[21] Simon D Smart, Tiago Quintino, and Baudouin Raoult. 2017. A scalable object store for meteorological and climate data. In *Proceedings of the Platform for Advanced Scientific Computing Conference*. 1–8.
[22] Simon D Smart, Tiago Quintino, and Baudouin Raoult. 2019. A high-performance distributed object-store for exascale numerical weather prediction and climate. In *Proceedings of the Platform for Advanced Scientific Computing Conference*. 1–11.
[23] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. 2011. The NumPy array: a structure for efficient numerical computation. *Computing in science & engineering* 13, 2 (2011), 22–30.
[24] Nils Wedi, Peter Bauer, Irina Sandu, Jörn Hoffmann, Sophia Sheridan, Rafael Cereceda, Tiago Quintino, Daniel Thiemert, and Thomas Geenen. 2022. Destination earth: High-performance computing for weather and climate. *Computing in Science & Engineering* 24, 6 (2022), 29–37.
[25] World Meteorological Organization (WMO). 2023. Manual on Codes, Volume I.2 – International Codes. https://library.wmo.int/idurl/4/35625
[26] Pen-Shu Yeh, Philippe Armbruster, Aaron Kiely, Bart Masschelein, Gilles Moury, Christoph Schaefer, and Carole Thiebaut. 2005. The new CCSDS image compression recommendation. In *2005 IEEE Aerospace Conference*. IEEE, 4138–4145.
[27] Xu Zhou, Kun Yang, Lin Ouyang, Yan Wang, Yaozhi Jiang, Xin Li, Deliang Chen, and Andreas Prein. 2021. Added value of kilometer-scale modeling over the third pole region: a CORDEX-CPTP pilot study. *Climate Dynamics* (2021), 1–15.