An Algorithm for Feature Extraction from Large Scale Meteorological Data Stores on Irregular Grids

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

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

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

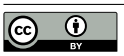
Martin Schultz
Forschungszentrum Juelich (FZJ) and University of Cologne
m.schultz@fz-juelich.de

Abstract

In recent years, the volume of meteorological data has grown rapidly due to higher model resolutions and more frequent data output. To handle this increase efficiently, the European Centre for Medium-Range Weather Forecasts has developed new ways to extract data without accessing full meteorological fields through their feature extraction capabilities. This includes in particular the Polytope algorithm, which lets users extract time series or regional subsets directly from the data backends, greatly reducing data transfer and processing time. However, the current algorithm only works on structured iso-latitude grids such as octahedral or HEALPix grids. Many modern meteorological models, by contrast, use unstructured grids where points do not align along latitudes, such as Lambert conformal or ICON icosahedral grids. In this paper, we extend the Polytope algorithm to support these irregular grids. We first explain why the original approach was limited to structured data before introducing a new extraction method, which uses the well-studied quadtree data structure. As a popular data structure in geospatial applications, we focus on describing how quadtrees can be integrated within the Polytope feature extraction framework to handle complex grid geometries. We then assess the performance and scalability of this algorithm, which performs on par with state-of-the-art alternatives used in geospatial applications. In particular, the extended algorithm outperforms other methods when accessing large or complex regions and therefore offers a compelling alternative to current data extraction techniques for large-scale datasets. Finally, we conclude by discussing how this algorithm can be integrated into an operational data service.

CCS Concepts

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



This work is licensed under a Creative Commons Attribution 4.0 International License.
PASC '26, Bern, Switzerland
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2734-4/2026/06
https://doi.org/10.1145/3815572.3815740

Keywords

Data Extraction, Data Management, Numerical Weather Prediction

ACM Reference Format:
Mathilde Leuridan, James Hawkes, Tiago Quintino, and Martin Schultz. 2026. An Algorithm for Feature Extraction from Large Scale Meteorological Data Stores on Irregular Grids. In *Platform for Advanced Scientific Computing Conference (PASC '26), June 29-July 01, 2026, Bern, Switzerland*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3815572.3815740

1 Introduction

In recent years, scientific data production has reached unprecedented scales, with scientific data archives around the world growing by orders of magnitude. In the fields of numerical weather prediction (NWP) and climate science in particular, initiatives such as kilometer-scale modeling [14, 23, 29, 50] and AI-driven weather forecasting [6, 25, 44] have pushed daily data generation into the petabyte range. With these increasing data volumes, and with the rise of data-driven science and engineering, efficient data access has become essential for enabling scientific discovery, innovation and timely decision-making.

In meteorology, data is typically stored as two-dimensional horizontal slices of the atmosphere, called fields. Traditionally, meteorological data access systems return entire fields to users, without the possibility to subset these data units. However, with model resolution increases, field sizes start to grow quadratically. This fast growth can then quickly strain the data systems, especially when serving many users at once. To maintain scalability, modern data frameworks are shifting towards the use of chunking [3, 38, 43] to enable efficient data access. This approach, however, depends on prior knowledge of common access patterns to optimize chunk layout.

To support arbitrary data access without such assumptions, the European Centre for Medium Range Weather Forecasts (ECMWF) developed the Polytope feature extraction service [27, 28]. Instead of relying on pre-defined chunks, Polytope dynamically computes which data bytes to query from meteorological data files. This enables the flexible and efficient retrieval of any high-dimensional shape from large weather datacubes, supporting fast on-demand access for dynamic digital twins and helping build more streamlined, user-centric workflows. Polytope is an efficient data retrieval technique that reduces both the load on the system, as well as the

user-side post-processing needs [26]. Using this algorithm, a scientist estimating storm risk for a specific city can, for example, extract only the relevant grid points, rather than downloading full forecast fields, as required by traditional systems. This minimises data transfer and simplifies downstream analysis. Polytope has not only demonstrated strong performance across a variety of use cases, but also scales well to different extraction shapes and dimensions. However, despite its advantages, Polytope currently only supports extraction from datacubes on iso-latitude grids. Whilst this includes widely used grids like the octahedral [36], HEALPix [15], and regular latitude-longitude grids, it is a significant limitation of the current algorithm. In particular, it implies that feature extraction is not currently possible on more complex irregular grids, such as the icosahedral grid [32] used by the ICON model [60], the tripolar and curvilinear grids [30] used by the NEMO model [34] or the Lambert conformal conic projection [9, 57].

In this paper, we address this constraint by extending Polytope to support arbitrary unstructured grids. We first provide a more complete motivation of this work, before explaining the origin of the limitation in the current algorithm and reviewing related work. We then introduce the extended algorithm, demonstrate its use on various irregular grids, and evaluate its performance. Finally, we discuss integration into the existing service and consider performance and scalability implications on the overall data extraction system.

2 Background

To motivate our work, we begin this section by quickly describing the original Polytope algorithm, before then providing a few examples of irregular grids. These examples highlight a key limitation of the current Polytope algorithm, which relies on the assumption that the datacube can be decomposed into tensor products of 1-dimensional spaces. Since irregular and unstructured grids are inherently embedded into 2-dimensional space however, a different extraction approach is required for them. To address this inconsistency, we will need to adapt the current query algorithm to operate in higher dimensional spaces. We therefore conclude the section with a review of related work on 2-dimensional point search and spatial indexing problems.

2.1 Original Polytope Algorithm

The Polytope algorithm [27] is a feature extraction algorithm which, for any arbitrary n-dimensional request shape, computes the set of datacube points contained within it. Using computational geometry concepts, it first decomposes the request shape into a set of convex sub-polytopes. At this point, each sub-polytope is then successively sliced along all of the datacube coordinate axes. This reduces the sub-polytope dimension by dimension until it has been completely decomposed into a set of one-dimensional line segments, as shown in Figure 2. The datacube points contained within these segments are then identified and returned to the user as the full set of points contained in the original request shape.

The algorithm relies on the important assumptions that we can both perform these axis-aligned slicing operations efficiently, and then quickly identify the points contained within the resulting one-dimensional segments.

2.2 Irregular Grids

While many NWP models are based on iso-latitude grids, other grid types offer interesting advantages, including improved spatial uniformity and a more stable convergence of solutions. These encompass icosahedral [54] and Yin–Yang grids [22], as well as projected grids such as Lambert conformal [57], Mercator [16] and polar stereographic projections [33], which are commonly used in limited area modelling [8, 42, 46] for their regional accuracy and numerical stability. Examples of such grids, which are not iso-latitude and we therefore treat as irregular, are shown in Figure 1(a).

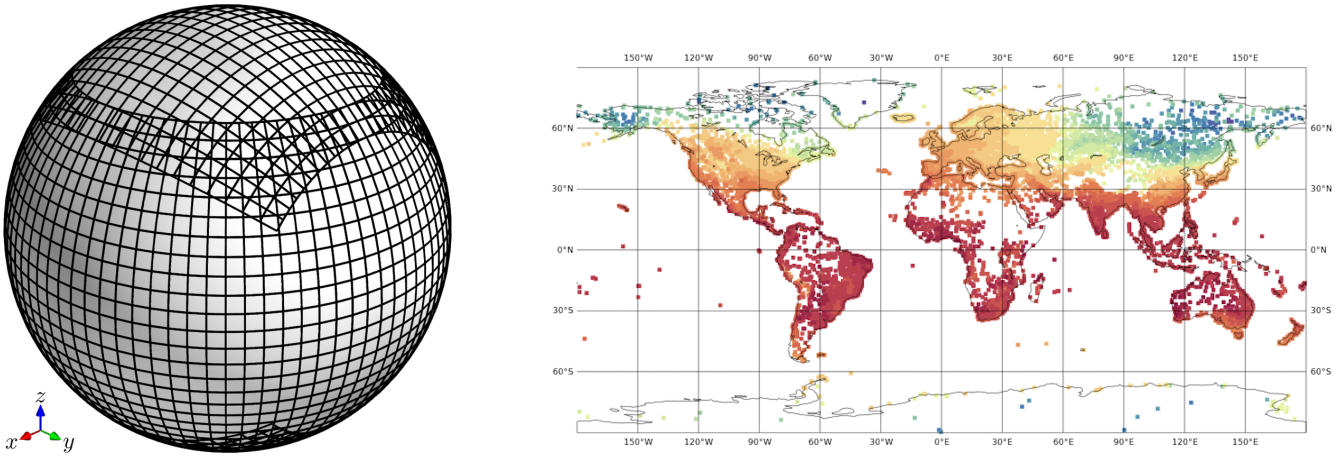
Although they are not iso-latitude, these grids still retain a clearly defined spatial arrangement. By contrast, some earth science datasets lack any regular 2-dimensional structure and are thus classified as unstructured grids [56]. Examples include observational datasets and more complex spatial configurations such as the ocean ORCA grids [35], which are especially tuned to lie within specific regions. Examples of such completely unstructured grids are shown in Figure 1(b).

In all of these examples, the points are clearly unaligned and do not follow a coherent structure along the coordinate axes. The current Polytope algorithm, however, relies on slicing polytopes along each of the datacube axes, including the grid axes, at constant coordinate values. As described in [27] and shown in Figure 2, this process eventually requires identifying points that lie within the extents of a line segment on an axis. On structured grids, this works well because multiple points can usually be found within such extents at the same time and for a relatively cheap cost. However, for unstructured and irregular grids, or more generally any non–axis-aligned point clouds, each point would need to be checked individually, since there is no guarantee that several points will fall within a given extent. This makes the operation inefficient and unsuitable in practice for unstructured data. Therefore, a different approach is needed to perform an operation comparable to the Polytope slicing in these cases.

Fundamentally, the problem arises because we try to simplify the search by reducing it to a one-dimensional space, where binary search can then be applied efficiently. This works well for uniform iso-latitude grids, since their layout can be mapped clearly onto a line. Irregular and unstructured grids however do not behave this same way as they are inherently two-dimensional objects. Forcing them into a one-dimensional order therefore loses important spatial relationships and as a result, one-dimensional methods like binary search tend to break down and become inefficient. Instead, we need to approach the problem in its natural dimensionality by using spatial indexing and search techniques that are built for higher dimensions. This includes the use of specialised data structures or other spatial partitioning methods [4, 12, 17, 52, 53], which allow for efficient point searches in irregular and unstructured spaces as they preserve the spatial aspect of the problem.

2.3 Related Work

Over the years, scientists have developed a variety of data structures and query algorithms to efficiently work with spatial point cloud data. With the rapid growth of geospatial data in particular, specialized systems and techniques have emerged to improve how this data is indexed and queried.



(a) Example of a Yin-Yang grid [2].

(b) Example of a temperature observations dataset [13].

Figure 1: Irregular grid examples.

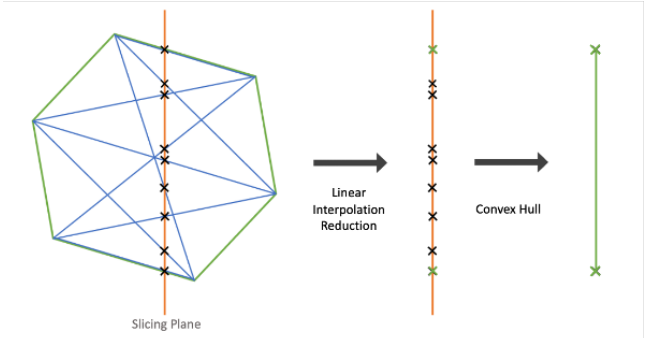


Figure 2: 1-dimensional slicing step.

One of the fundamental operations in this context is the point-in-polygon test [51], used to determine whether a point lies within a given polygon. Since its first introduction, this method has been refined many times with techniques such as ray tracing [20, 24] to increase efficiency for different use cases. In the geospatial field, the point-in-polygon test is often used in polygon range queries [40, 61]. Such queries aim to identify which points in a point cloud fall within a user-defined region of interest, and are in fact the 2-dimensional equivalent of the problem that the Polytope algorithm is trying to solve. Note, however, that the naïve approach of solving this problem by performing expensive point-in-polygon checks for every point in a point cloud becomes prohibitively expensive for high-resolution datasets with thousands or millions of samples.
Instead, researchers have developed specialised spatial data structures that capture how points are distributed in space and use this additional information to avoid exhaustive and unnecessary point-in-polygon checks. Examples include k-d trees [4], R-trees [17] and quadtrees [49], along with their higher-dimensional variants [5, 31, 37]. Such spatial indexing structures have been widely adopted in geospatial databases and scientific data analysis libraries for over a decade. Notable examples include the use of k-d trees and octrees in the Point Cloud Library [47], R-trees in GeoPandas [21], k-d trees in the NextGEMS processing pipeline [50], a range of spatial tree structures within the PostGIS framework [41], and octree-based indexing in OpenVDB [39]. Each of these structures offers unique trade-offs, and we now review some of their strenghts and limitations to identify the most suitable approach for designing an alternative slicing algorithm to the current Polytope algorithm.

k-d trees. A k-d tree is a binary search tree for locating points in k-dimensional space. Each node splits the space into two half-spaces by an axis-aligned hyperplane. The splitting coordinate typically changes with depth and median splits are common for balance, though other heuristics exist. Recursion continues until leaf nodes contain a single or a small number of data points. This data structure naturally generalises to any number of dimensions, although it is most efficient in low dimensions where trees stay shallow. For very large or high-dimensional datasets, the tree can become deep as only one dimension is split per level. These trees enable fast nearest-neighbor searches and range queries by pruning subtrees whose subspace lies outside of the query. Axis-aligned box queries are especially efficient on this data structure because intersection tests with the splitting hyperplanes are simple cheap comparison operations.

R-trees. An R-tree is a hierarchical tree data structure, which stores information about how objects are clustered in k-dimensional space. In such a tree, each node stores the minimum bounding box that encloses all of its children. The tree therefore represents rectangular regions in space rather than individual points. This design makes R-trees efficient for queries on region intersection, containment and unions, particularly for rectangular or box-shaped ranges. However, exact point queries or searches with arbitrary polygons are less efficient, since they first require approximating each point with a bounding box before checking which of these boxes intersect the query region, effectively building a box coverage of the polygon. For each box of the coverage, we would then finally

need to check point containment to verify that the points in the bounding boxes are really inside of the polygon.

quadtrees. A quadtree is a spatial tree data structure used for storing two-dimensional point data. Starting from a bounding box of the domain, it is built by recursively sub-dividing the space into four equal sized quadrants. The recursion step is repeated until the resulting quadrant only contains a small number of points, when the process terminates and the points contained inside of the quadrant are stored as leaf nodes of the tree. This construction guarantees that at any given depth, all quadrants in the tree cover the same area, which ensures a uniform partitioning of the original space. Since large empty regions can easily be pruned during recursion, quadtrees are well suited for range queries, as well as for applications such as collision detection and spatial mapping. Their efficiency is however strongly correlated to the underlying data point distribution as, with uniformly spaced data, the tree remains balanced and shallow, while more clustered points lead to an unbalanced tree that requires deeper traversal to answer a query.

The main objective of the algorithm developed in this paper is to efficiently identify all of the points contained within a given polygon from a high-dimensional unstructured point cloud. The different spatial data structures described above offer complementary strengths, but not all are equally well suited to this task.
R-trees, for example, are designed to capture relationships between extended regions by storing minimum bounding boxes. Whilst this makes them effective for rectangular range queries, it complicates polygon containment problems. As already mentioned, to handle such problems in an R-tree, each point must first be represented as a bounding box, and polygon queries then require additional intersection tests. This adds unnecessary overhead because the problem is fundamentally point-based. k-d trees represent a more natural approach, as they explicitly store point locations by recursively partitioning the space along coordinate axes. However, in large and fairly homogeneous spaces, such trees become very deep, since only one dimension is split at each level. Querying then requires traversing many nodes, making this data structure computationally expensive for problems such as retrieving all points contained within a polygon. Quadtrees offer a more suitable solution in this situation. Indeed, they not only provide a balanced representation of the space, especially when points are uniformly distributed, but also enable efficient pruning of uninteresting regions while still keeping tree depth manageable. This makes polygon containment queries, such as the one we are trying to perform, both practical and efficient.
Following this discussion, we therefore choose quadtrees as the foundation of our work and design an algorithm that leverages their properties to solve the required polygon containment query.

3 Extended Slicing Algorithm

In this section, we introduce a higher-dimensional slicing algorithm that operates directly in higher dimensional spaces. This is in contrast to the original Polytope slicing mechanism, where the problem was always first reduced to lower dimensions. The method described here builds on established spatial data structures to enable efficient queries, with quadtrees chosen as the most suitable for polygon range searches. As discussed previously, such spatial data structures are well-studied and have already been applied to a range of geospatial problems. In this work however, we focus on their adaptation and integration within the Polytope framework, showing how they can be applied to large-scale meteorological data extraction from complex grid geometries in an operational setting. We first present the algorithm, before illustrating its extraction capabilities. We then finally conclude the section by evaluating the performance and scalability of the algorithm on a few example Numerical Weather Prediction (NWP) datasets.

3.1 Modified Algorithm

Our proposed algorithm extends the Polytope algorithm [27] to higher dimensions. Whilst the overall framework, including tree construction, remains the same, the slicing step requires fundamental changes. The core assumption of the original method, which is that points within a query range can be located directly as we reduce the problem to 1-dimensional spaces, breaks down in a higher-dimensional setting. This limitation makes the original algorithm inefficient and impractical, if not completely unusable. To overcome this, we instead introduce a new algorithm, which uses a quadtree structure to efficiently identify points contained within polygons. This not only addresses the weaknesses of the original approach, but also preserves scalability of the algorithm. Although the approach naturally generalises to higher dimensions, in the following we focus on two-dimensional unstructured latitude-longitude subspaces since vertical levels, as well as other dimensions such as time or ensemble number, are always well aligned in NWP and climate models.

3.1.1 Pre-Computation Step. As a first step, we introduce a precomputation phase in which the available data in the datacube is organised into a quadtree with leaf quadrants that each contain a single grid point. This pre-processing is necessary because, unlike earlier methods that could efficiently query raw data directly, our algorithm requires a static spatial index to remain computationally cheap. Indeed, building the quadtree in advance encodes the spatial distribution of the data into a hierarchical structure, which can then be reused efficiently during polygon and range queries. By shifting this cost to pre-processing, we ensure that subsequent queries on the same dataset are faster and, crucially, also remain scalable when dealing with larger data volumes.
This pre-computation phase is especially effective for static grids, such as those on which NWP models are defined, where the quadtree only needs to be computed once and can then be reused across many queries. More dynamic grids, such as the ones defining observational datasets from mobile platforms like ships or aircraft, might change between queries and thus require the quadtree to be recomputed more frequently. Whilst this reduces the benefits of caching the quadtree, the algorithm described here still applies in such cases, with performance depending on how often the underlying grid changes.

3.1.2 Dynamic Step. Building on the precomputed quadtree, we introduce a slicing algorithm to identify all of the points contained within a user-defined polygon. The general idea is to recursively

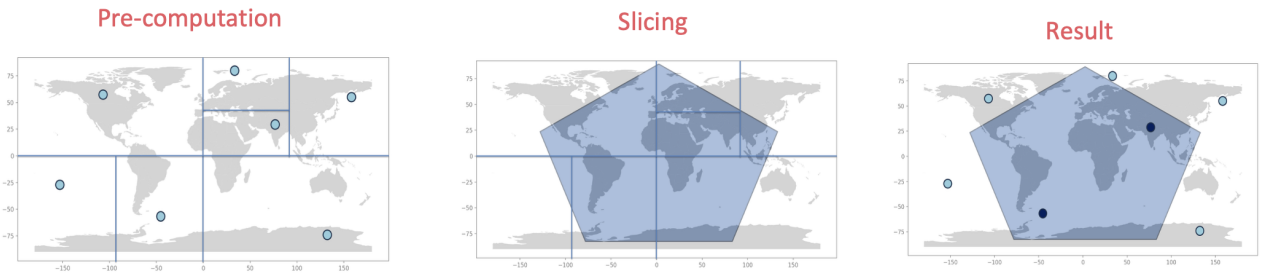


Figure 3: Unstructured grid slicing algorithm.

slice the polygon along the boundary of each of the quadtree quadrants. At each step, we act on a given quadtree node, where a vertical and a horizontal slice divide the polygon into four sub-polygons, each fully contained in one of the child quadrants of the current node. This subdivision continues until the leaves of the tree are reached, where a point-in-polygon test then confirms whether the leaf point lies inside of the final sub-polygons. The full procedure is formalised in Algorithm 1 and illustrated in Figure 3.

Our approach extends two core components of the original Polytope slicing algorithm [27]. First, the point inclusion step is generalised from one-dimensional bound checks to a full two-dimensional point-in-polygon test. Similarly to the original Polytope algorithm however, the polygon that is being sliced at this step is guaranteed to be convex. We can therefore apply similar strategies as in the original algorithm to reduce the problem to a one-dimensional line segment. As illustrated in Figure 4, the point-in-polygon test can be reformulated as two simple bound checks. First, we verify whether the vertical coordinate of the point lies within the vertical extent of the polygon and, if so, we then determine the corresponding horizontal extent of the polygon at that vertical position and check whether the point's horizontal coordinate falls within this interval. Second, the one-dimensional hyperplane slicing operation is extended to two dimensions, as shown in Algorithm 2. Instead of reducing the dimensionality of the polygon by projecting it to a lower-dimensional space, we preserve the full dimensionality of the problem here by slicing the polygon along each dimension at every recursion step. This operation produces four 2-dimensional sub-polygons, which replaces the original algorithm's 1-dimensional projection and thereby ensures that the multi-dimensional nature of the problem is maintained. Note that we perform similar steps to the original 1-dimensional algorithm here, and in particular, we still compare pairs of point on either side of the slice plane before taking the convex hull. We therefore maintain the same complexity as in the original algorithm [26] of $O(m^2)$, where m is the number of nodes defining the input polygon. These two extensions allow the algorithm to take full advantage of the hierarchical structure of the quadtree, thus making polygonal queries in two-dimensional space both efficient and scalable.

3.1.3 Optimisation Step. As an optimisation step to improve performance, the algorithm checks whether a sub-polygon exactly overlaps the full area of a quadtree node. If this is the case, the polygon fully covers the associated node quadrant, which implies that all points stored in the leaf nodes of that branch must lie inside of the polygon. These points can therefore directly be added to the search result without any further tree traversal or point-in-polygon tests. This pruning step allows an early return, avoiding unnecessary recursion, as well as helps reduce significant computational overhead, especially for larger polygon queries.

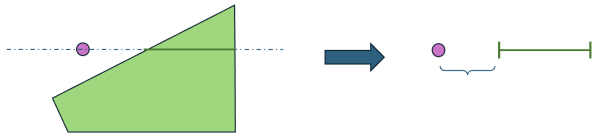


Figure 4: Custom point-in-polygon test.

3.2 Examples

We now demonstrate example extractions with our algorithm on two non-iso-latitude grids, namely a Lambert conformal projection grid and an ICON icosahedral grid. Both grids are used in the Destination Earth On-Demand Extremes digital twin to run specialised local meteorological models. Unlike regular latitude–longitude grids, these exhibit more complex two-dimensional mappings, as shown in Figure 5, where the grid points are clearly not aligned with the coordinate axes.

The Lambert Conformal projection is constructed by projecting a regular iso-latitude grid onto a cone and flattening it into the plane [9]. In principle, one could design a mapping to transform polygons into this space, apply the original Polytope algorithm in a decomposed 1D form, and then map the results back. However, this would require additional transformations and is not straightforward in practice. Our quadtree-based method sidesteps this complexity, as illustrated in Figure 5(a), which shows extractions directly on the projected grid.

The ICON grid is even less structured. It is built from a nested icosahedral decomposition of the sphere, whose vertices are by construction never aligned with latitude or longitude [32]. Unlike the Lambert projection, there is no simple analytical mapping to transform this grid into an iso-latitude representation. In practice,

Algorithm 1 Polytope Intersection Algorithm

Input: quadtree node n, polygon $\mathcal{P}$.

if n is a leaf **then**
    if n's quadrant contains a data point **then**
        if data point is contained in $\mathcal{P}$ **then**
            Add data point to results
        end if
    end if
else
    Slice $\mathcal{P}$ into polygons $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_4$ that fit inside n's children's quadrants
    for child node c_i in n's four children **do**
        Repeat algorithm with new inputs c_i and $\mathcal{P}_i$
    end for
end if

Algorithm 2 Slice Algorithm

Input: polygon $\mathcal{P}$, slice value x_f.

Separate $\mathcal{P}$'s vertices into sets S_l and S_r of points to the left and right of x_f.
for pairs $(p_l, p_r) \in S_l \times S_r$ **do**
    Find intersection point p_m of the line through p_l and p_r with the line $x = x_f$.
    Add p_m to a set S_m.
end for
Take convex hull of the sets $S_l \cup S_m$ and $S_r \cup S_m$ to get left and right sliced polygons.

ICON grids are distributed as precomputed files that store the point locations explicitly. This irregularity makes polygon slicing particularly challenging. Nevertheless, as Figure 5(b) shows, our algorithm successfully extracts subsets of points on this unstructured grid without requiring such a mapping.

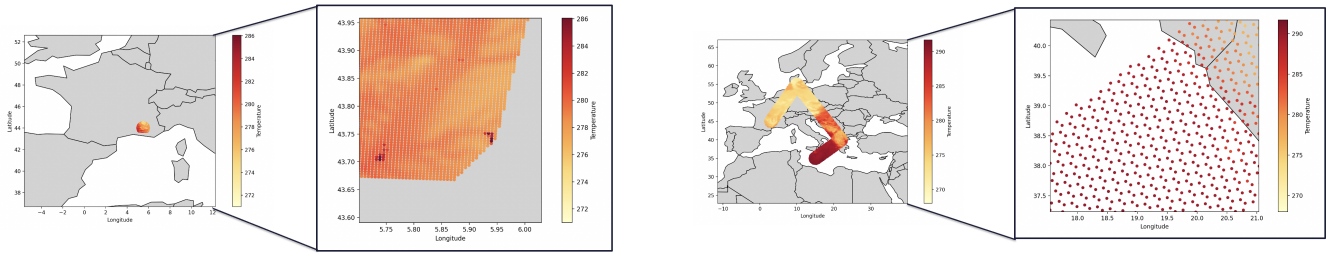
These examples highlight that the algorithm works reliably not only on regular grids but also on complex, non-axis-aligned structures. This allows us to return only the small fraction of bytes relevant to a user's query, even for very large, high-resolution datasets, preserving the original motivation behind the Polytope algorithm.

3.3 Performance

The performance of the extended algorithm depends on two important aspects, which are the complexity of the input shape and the complexity of the underlying dataset. Dataset complexity is influenced by grid resolution and point distribution, which both determine the number and density of points to be processed. Shape complexity is defined by factors such as its area, the number of vertices and the number of unions defining the shape. To assess how these factors affect performance in an operational setting, we first analyse their individual scaling behaviour. We then compare the algorithm against state-of-the-art point-in-polygon approaches, which serve a similar purpose. These reference methods rely on standard point-in-polygon tests, often optimised by limiting computations to points within a polygon's bounding box, which is an approach that we also adopt for our comparison [55, 58, 59]. The algorithm implementation used for this performance analysis is written in Rust and does not make use of any parallelism.

3.3.1 Scaling. The scaling behaviour of the algorithm is influenced by two main groups of factors, those related to the size and structure of the dataset and those linked to the complexity of the input polygon. While shape-related factors primarily affect the dynamic extraction time, dataset-related factors impact both the quadtree construction, or pre-computation phase, as well as the dynamic extraction time.

We first investigate how the characteristics of the underlying dataset affect total extraction times. In this experiment, a box of constant size is extracted from uniform point clouds of varying densities to assess how Algorithm 1 scales. As shown in Figure 6(a), both the quadtree construction time and the polygon point extraction time scale linearly with the number of points in such a uniformly distributed point cloud. This behaviour is expected as a homogeneous point distribution produces a well-balanced quadtree, whose traversal depth increases proportionally with the number of points. However, real-world datasets rarely exhibit such uniformity. To assess the effect of spatial inhomogeneity, we generate point clouds of one million points drawn from normal distributions with varying standard deviations around the centre. Figure 6(b) shows that when points are clustered and less evenly distributed, the quadtree becomes less balanced, leading to higher construction times. On the other hand, when points are more dispersed in space, the time required to identify points within the input shape increases. Although this behaviour may seem counterintuitive, it arises because a more dispersed distribution reduces opportunities for early termination in the traversal. Indeed, since fewer quadrants are fully contained within the selection area, more of the quadtree must be explored during the algorithm runtime.



(a) Extraction on a local Lambert conformal projection grid.

(b) Extraction on an ICON grid.

Figure 5: Extraction examples on irregular grids.

Besides the point cloud properties, the remaining key factor which influences the algorithm's performance is the complexity of the input polygon. Polygon complexity is mainly determined by two characteristics, the number of vertices defining its boundary and the number of polygon unions which compose the overall shape.

We first examine the impact of the vertex count. To isolate this effect, we use polygonal approximations of a circle with progressively more vertices, keeping the enclosed area approximately constant to avoid bias. The results, shown in Figure 7(a), confirm that quadtree construction time remains constant, as the underlying dataset is unchanged. For polygons with few vertices, total extraction time is initially dominated by data retrieval from the storage backend via the GribJump software [11], while the computational cost of determining points within the polygon is low. This occurs because simpler polygons cover larger areas, leading to larger data extractions. As polygon complexity increases, backend data retrieval time stabilises, which indicates a consistent number of retrieved points, whilst the time to compute points contained in the polygon grows quadratically. This behaviour aligns with the algorithm's theoretical scaling, as the slicing step involves an increasing number of polygon–line intersection operations.

The second factor to investigate is the number of unions composing the shape. To study this, we repeat the extraction for the same circular region, represented as a union of an increasing number of sub-polygons. As shown in Figure 7(b), both quadtree construction and data retrieval times remain constant, since the underlying point cloud and extracted area are unchanged. However, the time required to compute points within the polygon grows linearly with the number of unions, which then directly impacts the total extraction time. This linear relationship can be explained by the fact that each sub-polygon is processed independently using Algorithm 1.

3.3.2 Method Comparison. Having examined the key factors influencing extraction performance from both the dataset and polygon perspectives, we now compare the algorithm against state-of-the-art, highly optimised point-in-polygon (PIP) methods [18, 19, 61]. Specifically, we benchmark against the Rust geo library implementation [45], which stores the input polygon in memory and employs bounding-box skipping to avoid unnecessary point checks. To evaluate performance, we perform two sets of experiments, one using progressively larger rectangular boxes and another using triangles of increasing size but contained within the same bounding box. In both cases, the polygons are simple, allowing the baseline PIP tests to perform efficiently.

Figure 8(a) presents the results for the box experiments. The quadtree-based approach performs comparably to, and often slightly better than, the optimised PIP implementation, particularly for larger boxes containing more points, where the number of required PIP checks increases substantially. Similar trends are observed for the triangle experiments in Figure 8(b), where the quadtree method outperforms the PIP tests again. In particular, we note that the performance gap widens for larger extraction areas. Both of these results also confirm that the quadtree algorithm scales linearly with the area of the extracted region.

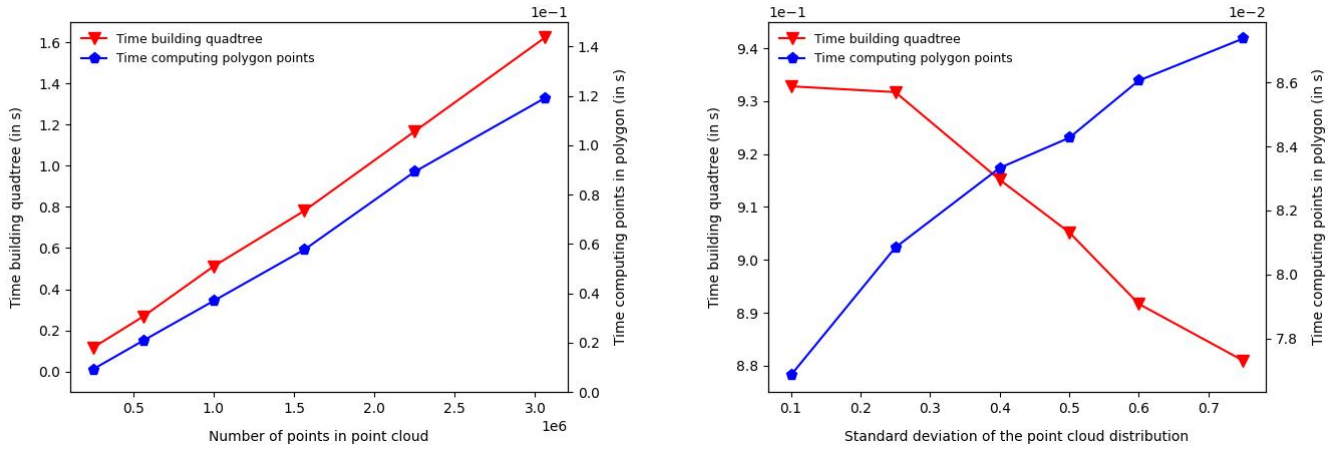
It is finally worth noting that standard PIP tests degrade in performance as polygon complexity increases, due to the rising cost of ray-casting operations. In contrast, the quadtree approach stays more efficient while offering additional functionality, such as nearest-neighbour searches [1, 48], at minimal additional computational costs. We therefore conclude that our quadtree-based algorithm achieves performance on par with leading PIP methods used in GIS and Earth science data analysis, while providing enhanced versatility for meteorological feature extraction across diverse grid structures.

4 Discussion

As shown throughout this paper, the new slicing algorithm enables point searches within arbitrary polygons on fully unstructured spaces, without assumptions on the spatial arrangement of points. This represents a major improvement over the original Polytope algorithm and, in particular, makes it possible to serve datasets that were previously unsupported. Since many meteorological services use diverse and non-standard grids, the relaxed constraints of our approach extend the Polytope service to virtually all dataset types. However, deploying this algorithm in real-world scenarios, especially as part of an operational service, requires addressing several practical considerations, which we now discuss.

4.1 Data Extraction as a Service

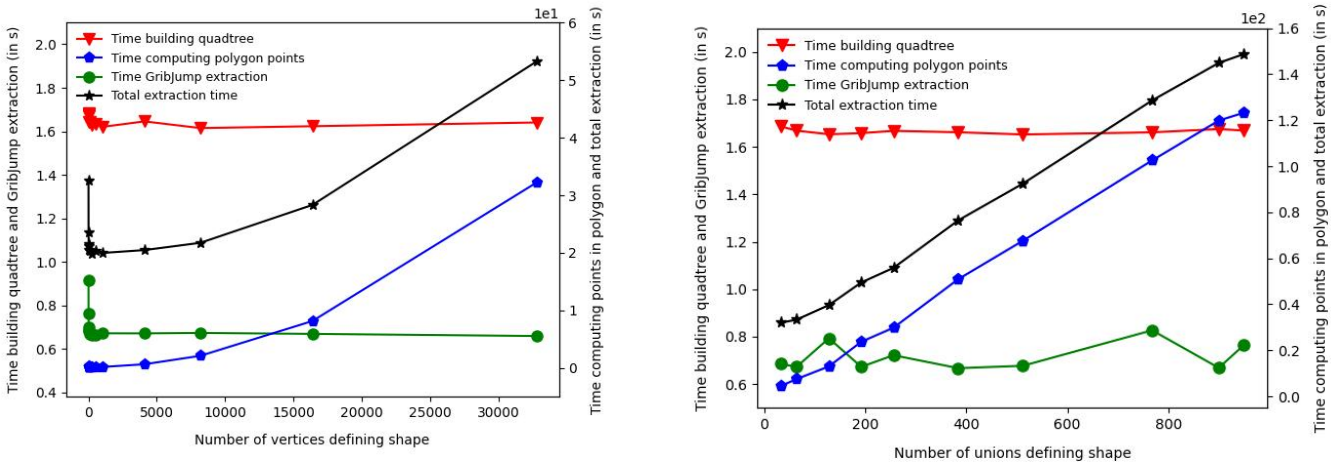
To make the described slicing capability accessible in practice, the algorithm must be integrated into an operational service that allows users to extract subsets of data directly from large multidimensional



(a) Varying homogeneous point density.

(b) Varying inhomogeneous point distribution.

Figure 6: Extraction times of a box for varying point clouds.



(a) Varying number of vertices.

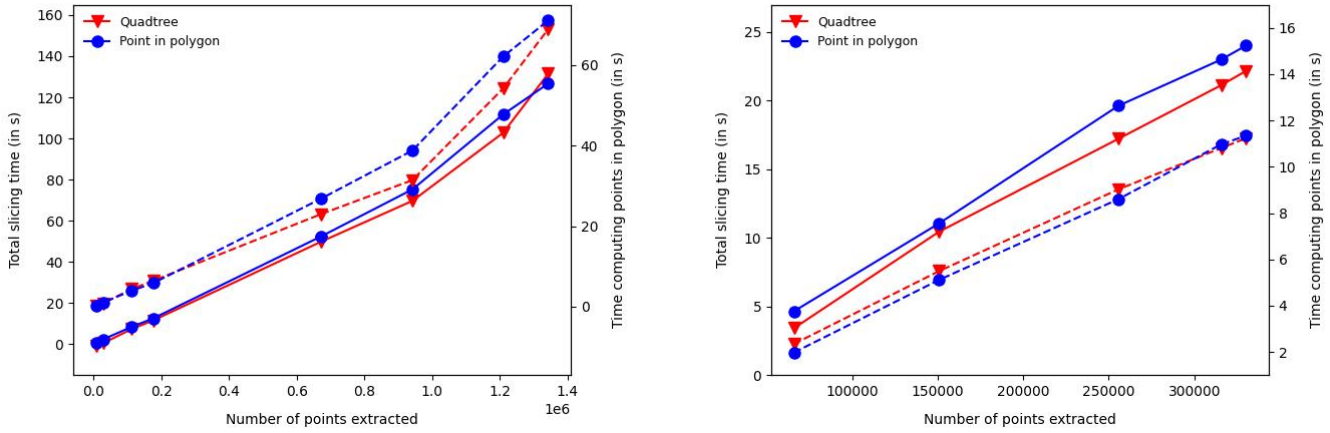
(b) Varying number of sub-polygon unions.

Figure 7: Extraction times for a disk defined with a varying number of vertices (a) and a varying number of sub-polygon unions (b).

datasets. Such a service should efficiently handle different grid structures while maintaining responsiveness for on-demand queries.

Whilst the new slicing algorithm greatly extends applicability to unstructured grids, it also introduces significant additional computational costs due to the preprocessing required to construct the quadtree. In contrast, the original Polytope slicer remains more efficient for structured, iso-latitude grids where such preprocessing is unnecessary. On top of these few seconds of additional preprocessing, the original Polytope slicer will moreover benefit from knowledge of the data structure during slicing and typically achieves a quadratic speedup over the fully unstructured version. To maintain both flexibility and high performance, a complete data extraction service must therefore integrate both algorithms in a single mechanism by dynamically selecting the most suitable slicer based on the dataset and its grid type.

Beyond considerations of computational efficiency, using this new algorithm as part of an operational service introduces additional memory management challenges. Since the method depends on precomputed quadtrees, these data structures must be stored persistently for static reuse or regenerated dynamically for each incoming request. Depending on system architecture, this can be adressed by caching quadtrees for commonly accessed grids, distributing them



(a) Increasing square boxes.

(b) Skewed triangles contained in same bounding box.

Figure 8: Comparison of extraction times between quadtree approach (Algorithm 1) and optimised point in polygon test for different shapes, where the solid lines show the time computing the points in the polygon and the dashed lines show the total slicing time.

across sub-services or applying compression when decompression costs are acceptable.

Considering these aspects is crucial to ensure that an operational service can efficiently deliver data to users, minimising computational overhead while preserving scalability and versatility across supported grid types.

4.2 Future Work

The service described above works reliably as long as the grid type and configuration of the dataset on which we want to apply the algorithm are known in advance. A key direction for future work is to automate this step through a data discovery mechanism capable of identifying the underlying grid of any dataset. This could for example be achieved by introducing a new data structure that stores metadata about each dataset, including its grid specification, as well as other relevant spatial characteristics. Such an approach would enable a fully dynamic service that automatically selects the appropriate slicer without requiring prior user input. Beyond improving user-friendliness, this would more importantly enable streamlined and interactive access to data, paving the way for integration of the service within a digital twin ecosystem. As the ultimate objective is to power digital twins by providing users with seamless, automated, and transparent access to datasets, this system would be a first step towards this goal by allowing them to interact with the data efficiently and intuitively.

Beyond system-level considerations, an important direction for future work is to further optimise the algorithm in order to improve performance and scalability to very large datasets. This includes parallelisation, by running point-in-polygon tests at the quadtree leaf nodes concurrently for example. Recent work has also shown that quadtree construction and traversal can be parallelised [10], and exploring how such methods could be integrated into our algorithm may result in significant performance improvements. In addition, GPU-optimised quadtree traversal [7] represents another promising research avenue. Besides parallelisation, further optimisations could involve exposing the quadtree compression level as a user-controlled parameter and studying how optimal compression varies with grid types and resolutions. Finally, while the algorithm was designed to operate on fully unstructured grids, many real-world datasets exhibit at least partial structure. Future work could therefore investigate how this structure, such as that present in Lambert or other projected grids, can be exploited to further optimise performance by using, for example, more specialised quadtree variants or alternative spatial data structures in our algorithm.

Acknowledgments

The authors would like to thank Professor Stefan Wesner for helpful discussions about this work.

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] Kunio Aizawa and Shojiro Tanaka. 2008. A constant-time algorithm for finding neighbors in quadtrees. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31, 7 (2008), 1178–1183.

[2] Marius Almanstötter, Tobias Melson, Hans-Thomas Janka, and Ewald Müller. 2018. Parallelized solution method of the three-dimensional gravitational potential on the Yin–Yang grid. *The Astrophysical Journal* 863, 2 (2018), 142.

[3] Cédric Bastoul and Paul Feautrier. 2003. Improving data locality by chunking. In *International Conference on Compiler Construction*. Springer, 320–334.

[4] Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM* 18, 9 (1975), 509–517.

[5] Stefan Berchtold, Daniel A Keim, and Hans-Peter Kriegel. 1996. The X-tree: An index structure for high-dimensional data. (1996).

[6] Kaifeng Bi, Lingxi Xie, Hengheng Zhang, Xin Chen, Xiaotao Gu, and Qi Tian. 2022. Pangu-weather: A 3d high-resolution model for fast and accurate global weather forecast. *arXiv preprint arXiv:2211.02556* (2022).

[7] Wade Brainerd, Theresa Foley, Manuel Kraemer, Henry Moreton, and Matthias Nießner. 2016. Efficient GPU rendering of subdivision surfaces using adaptive quadtrees. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 1–12.

[8] Ramon De Elia, René Laprise, and Bertrand Denis. 2002. Forecasting skill limits of nested, limited-area models: A perfect-model approach. *Monthly Weather Review* 130, 8 (2002), 2006–2023.

[9] Charles H Deetz. 1918. The Lambert conformal conic projection. *US Department of Commerce, US Coast and Geodetic Survey, Special Publication* 47 (1918), 1–61.

[10] Landon Dyken, Will Usher, Steve Petruzza, Stavros Sintos, and Sidharth Kumar. 2025. Accelerating Web-Based Graph Drawing with Bottom-Up GPU Quadtree Construction. In *2025 IEEE 18th Pacific Visualization Conference (PacificVis)*. IEEE, 24–29.

[11] ECMWF. 2024. GribJump Software. https://github.com/ecmwf/gribjump Accessed on 2025-11-08.

[12] Ahmed Eldawy, Louai Alarabi, and Mohamed F Mokbel. 2015. Spatial partitioning techniques in SpatialHadoop. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1602–1605.

[13] European Centre for Medium-Range Weather Forecasts (ECMWF). 2025. Time–averaged geographical mean for 2 m temperature (4DVAR). ECMWF Charts / OBSTAT catalogue. https://charts.ecmwf.int/catalogue/packages/obstat/ Dataset accessed: https://charts.ecmwf.int/catalogue/packages/obstat/ (last accessed 2025-11-08).

[14] 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 prediction system into the heart of Destination Earth. *Procedia Computer Science* 240 (2024), 99–109.

[15] Krzysztof M Gorski, Eric Hivon, Anthony J Banday, Benjamin D Wandelt, Frode K Hansen, Mstvos Reinecke, and Matthia Bartelmann. 2005. HEALPix: A framework for high-resolution discretization and fast analysis of data distributed on the sphere. *The Astrophysical Journal* 622, 2 (2005), 759.

[16] Erik W Grafarend. 1995. The optimal universal transverse Mercator projection. *Manuscripta geodaetica* 20, 6 (1995), 421–468.

[17] Antonin Guttman. 1984. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*. 47–57.

[18] Eric Haines. 1994. Point in Polygon Strategies. *Graphics Gems* 4, 1 (1994), 24–46.

[19] Jianqiang Hao, Jianzhi Sun, Yi Chen, Qiang Cai, and Li Tan. 2018. Optimal reliable point-in-polygon test and differential coding boolean operations on polygons. *Symmetry* 10, 10 (2018), 477.

[20] Paul S Heckbert and Pat Hanrahan. 1984. Beam tracing polygonal objects. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*. 119–127.

[21] Kelsey Jordahl, Joris Van Den Bossche, Martin Fleischmann, James McBride, Jacob Wasserman, Matt Richards, Adrian Garcia Badaracco, Alan D. Snow, Jeffrey Gerard, Jeff Tratner, Matthew Perry, Brendan Ward, Carson Farmer, Geir Arne Hjelle, Micah Cochran, Mike Taves, Sean Gillies, Giacomo Caria, Lucas Culbertson, Matt Bartos, Nick Eubank, Ray Bell, Sangarshanan, John Flavin, Sergio Rey, Maxalbert, Aleksey Bilogur, Christopher Ren, Dani Arribas-Bel, and Daniel Mesejo-León. 2022. *geopandas/geopandas: v0.12.1.* doi:10.5281/zenodo.7262879

[22] Akira Kageyama and Tetsuya Sato. 2004. "Yin-Yang grid": An overset grid in spherical geometry. *Geochemistry, Geophysics, Geosystems* 5, 9 (2004).

[23] Daniel Klocke, Claudia Frauen, Jan Frederik Engels, Dmitry Alexeev, René Redler, Reiner Schnur, Helmuth Haak, Luis Kornblueh, Nils Brüggemann, Fatemeh Chegini, et al. 2025. Computing the Full Earth System at 1km Resolution. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 125–136.

[24] Moritz Laass. 2021. Point in Polygon Tests Using Hardware Accelerated Ray Tracing. In *Proceedings of the 29th International Conference on Advances in Geographic Information Systems*. 666–667.

[25] 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).

[26] 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 *Proceedings of the Platform for Advanced Scientific Computing Conference*. 1–9.

[27] Mathilde Leuridan, James Hawkes, Simon Smart, Emanuele Danovaro, Martin Schultz, and Tiago Quintino. 2025. Polytope: an algorithm for efficient feature extraction on hypercubes. *Journal of Big Data* 12, 1 (2025), 1–25.

[28] Mathilde Leuridan, Adam Warde, Oisin-M, James Hawkes, James Varndell, Peter Tsrunchev, Chris Bradley, Dušan Figala, Iain Russell, Kai Kratz, Simon Smart, and Yordan Radev. 2025. *ecmwf/polytope: 2.1.1.* doi:10.5281/zenodo.17483028

[29] Lingcheng Li, Gautam Bisht, Dalei Hao, and L Ruby Leung. 2024. Global 1 km land surface parameters for kilometer-scale Earth system modeling. *Earth System Science Data* 16, 4 (2024), 2007–2032.

[30] Xiaolan Li, Yongqiang Yu, Hailong Liu, and Pengfei Lin. 2017. Sensitivity of Atlantic meridional overturning circulation to the dynamical framework in an ocean general circulation model. *Journal of Meteorological Research* 31, 3 (2017), 490–501.

[31] King-Ip Lin, Hosagrahar V Jagadish, and Christos Faloutsos. 1994. The TV-tree: An index structure for high-dimensional data. *The VLDB Journal* 3, 4 (1994), 517–542.

[32] Leonidas Linardakis, Daniel Reinert, and Almut Gassmann. 2011. *Icon grid documentation.* Technical Report. Tech. rep., DKRZ, Hamburg, available at: https://www. dkrz. de/SciVis/hd-cp ....

[33] Richard J Lisle and Peter R Leyshon. 2004. *Stereographic projection techniques for geologists and civil engineers.* Cambridge University Press.

[34] Gurvan Madec et al. 2015. NEMO ocean engine. (2015).

[35] Gurvan Madec and Maurice Imbard. 1996. A global ocean mesh to overcome the North Pole singularity. *Climate Dynamics* 12, 6 (1996), 381–388.

[36] 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.

[37] Donald Meagher. 1982. Geometric modeling using octree encoding. *Computer graphics and image processing* 19, 2 (1982), 129–147.

[38] Alistair Miles, jakirkham, Joe Hamman, Dimitri Papadopoulos Orfanos, David Stansby, M Bussonnier, Josh Moore, Davis Bennett, Tom Augspurger, Norman Rzepka, Deepak Cherian, Sanket Verma, James Bourbeau, Andrew Fulton, Ryan Abernathey, Gregory Lee, Hannes Spitz, Mads R. B. Kristensen, Max Jones, Zain Patel, Saransh Chopra, Matthew Rocklin, AWA BRANDON AWA, Nathan Zimmerman, Martin Durant, Juan Nunez-Iglesias, Elliott Sales de Andrade, and Vincent Schut. 2025. *zarr-developers/zarr-python: v2.18.5.* doi:10.5281/zenodo.15101732

[39] Ken Museth, Jeff Lait, John Johanson, Jeff Budsberg, Ron Henderson, Mihai Alden, Peter Cucka, David Hill, and Andrew Pearce. 2013. OpenVDB: an open-source data structure and toolkit for high-resolution volumes. In *Acm siggraph 2013 courses*. 1–1.

[40] Stig Nordbeck and Bengt Rystedt. 1967. Computer cartography point-in-polygon programs. *BIT Numerical Mathematics* 7, 1 (1967), 39–64.

[41] Regina Obe and Leo Hsu. 2021. *PostGIS in action.* Simon and Schuster.

[42] P Oddo and Nadia Pinardi. 2008. Lateral open boundary conditions for nested limited area models: A scale selective approach. *Ocean modelling* 20, 2 (2008), 134–156.

[43] Cédric Penard, Flavien Gouillon, Xavier Delaunay, and Sylvain Herlédan. 2025. A new sub-chunking strategy for fast netCDF-4 access in local, remote and cloud infrastructures, chunkindex V1. 1.0. *EGUsphere* 2025 (2025), 1–24.

[44] Ilan Price, Alvaro Sanchez-Gonzalez, Ferran Alet, Tom R Andersson, Andrew El-Kadi, Dominic Masters, Timo Ewalds, Jacklynn Stott, Shakir Mohamed, Peter Battaglia, et al. 2023. Gencast: Diffusion-based ensemble forecasting for medium-range weather. *arXiv preprint arXiv:2312.15796* (2023).

[45] The GeoRust Project. 2025. *geo: geospatial primitives and algorithms.* GeoRust. https://crates.io/crates/geo Rust crate providing geospatial primitives and algorithms.

[46] Burkhard Rockel, Andreas Will, and Andreas Hense. 2008. The regional climate model COSMO-CLM (CCLM). *Meteorologische zeitschrift* 17, 4 (2008), 347.

[47] Radu Bogdan Rusu and Steve Cousins. 2011. 3d is here: Point cloud library (pcl). In *2011 IEEE international conference on robotics and automation*. IEEE, 1–4.

[48] Hanan Samet. 1982. Neighbor finding techniques for images represented by quadtrees. *Computer graphics and image processing* 18, 1 (1982), 37–57.

[49] Hanan Samet. 1984. The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR)* 16, 2 (1984), 187–260.

[50] Hans Segura, Xabier Pedruzo-Bagazgoitia, Philipp Weiss, Sebastian K Müller, Thomas Rackow, Junhong Lee, Edgar Dolores-Tesillos, Imme Benedict, Matthias Aengenheyster, Razvan Aguridan, et al. 2025. nextGEMS: entering the era of kilometer-scale Earth system modeling. *Geoscientific Model Development* 18, 20 (2025), 7735–7761.

[51] Moshe Shimrat. 1962. Algorithm 112: position of point relative to polygon. *Commun. ACM* 5, 8 (1962), 434.

[52] Hoang Vo, Ablimit Aji, and Fusheng Wang. 2014. SATO: a spatial data partitioning framework for scalable query processing. In *Proceedings of the 22nd ACM SIGSPATIAL international conference on advances in geographic information systems*. 545–548.

[53] Tin Vu and Ahmed Eldawy. 2020. R*-grove: Balanced spatial partitioning for large-scale datasets. *Frontiers in big data* 3 (2020), 28.

[54] Ning Wang and Jin-Luen Lee. 2011. Geometric properties of the icosahedral-hexagonal grid on the two-sphere. *SIAM Journal on Scientific Computing* 33, 5 (2011), 2536–2559.

[55] Yong Wang, Zhenling Liu, Hongyan Liao, and Chengjun Li. 2015. Improving the performance of GIS polygon overlay computation with MapReduce for spatial big data processing. *Cluster Computing* 18, 2 (2015), 507–516.
[56] Nigel P Weatherill. 1999. Unstructured Grids.
[57] RT Williams. 1995. Lambert and Mercator map projections in geology and geophysics. *Computers & Geosciences* 21, 3 (1995), 353–364.
[58] Sheng Yang, Jun-Hai Yong, Jiaguang Sun, Hejin Gu, and Jean-Claude Paul. 2010. A point-in-polygon method based on a quasi-closest point. *Computers & Geosciences* 36, 2 (2010), 205–213.
[59] Borut Žalik and Ivana Kolingerova. 2001. A cell-based point-in-polygon algorithm suitable for large sets of points. *Computers & Geosciences* 27, 10 (2001), 1135–1145.
[60] Günther Zängl, Daniel Reinert, Pilar Rípodas, and Michael Baldauf. 2015. The ICON (ICOsahedral Non-hydrostatic) modelling framework of DWD and MPI-M: Description of the non-hydrostatic dynamical core. *Quarterly Journal of the Royal Meteorological Society* 141, 687 (2015), 563–579.
[61] Jianting Zhang and Simin You. 2012. Speeding up large-scale point-in-polygon test based spatial join on GPUs. In *Proceedings of the 1st ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data*. 23–32.