Towards universal building blocks for cloud-native digital-twins

Alexander Kmoch[1, 2], Wai Tik Chan[1], Guillaume Ameline[1,2], Justus Magin[3], Jean-Marc Delouis[3], Tina Odaka[3], Benoit Bovy[4], Anne Fouilloux[5], Evelyn Uuemaa[1, 2]

[1]University of Tartu, Institute of Ecology and Earth Sciences, Landscape Geoinformatics Lab, Tartu, Estonia - (alexander.kmoch, evelyn.uuemaa, wai.tik.chan)@ut.ee
[2]Geolynx, Tartu, Estonia - guillaume@geolynx.ee
[3]LOPS - Laboratoire d'Oceanographie Physique et Spatiale UMR 6523 CNRS-IFREMER-IRD-Univ.Brest-IUEM, Plouzane, France - (justus.magin, jean.marc.delouis, tina.odaka)@ifremer.fr
[4]Georode, Liege, Belgium - benbovy@gmail.com
[5]Simula, Oslo, Norway - annef@simula.no

Keywords: DGGS, OGC API, Zarr, Parquet, data cube, web service

Abstract

The exponential growth of Earth Observation (EO) data presents significant challenges for efficient data access, processing, and analysis. Current approaches often involve disparate data formats, coordinate systems, and access patterns, limiting interoperability and scalability. Firstly, the Zarr and Parquet data storage formats have seen wide adoption as a unifying cloud-native foundation for various domains in recent years, including climate, EO, bio-imaging, and genomics. Secondly, Discrete Global Grid Systems (DGGS) such as HEALPix, or ISEA-based hexagonal DGGS are being increasingly used to enable spatial data indexing beyond traditional grids, by using equal-area pixels and location- and refinement level encoding indices. Lastly, the recently published OGC API DGGS standard specifies a lightweight web service API for clients accessing data organised according to Discrete Global Grid Reference Systems (DGGRS).
We describe a scalable, interoperable, and extensible FOSS architecture for a modern geospatial data ecosystem, based on DGGS. As an example, we introduce *pydggsapi*, a Python server implementing this new OGC standard to serve large geospatial datasets from cloud-native Zarr and Parquet data stores indexed by a DGGS. This novel architecture combines the DGGS data cube paradigm, standardised OGC API DGGS web service access, and cloud-optimised data formats such as Zarr and Parquet as universal building blocks for geospatial data management, enabling seamless transitions between high-performance computing environments and lightweight client applications.

1. Introduction

Earth Observation (EO) data volumes continue to increase exponentially, driven by new satellite missions, higher sensor resolutions, and increased temporal coverage. The Copernicus program alone generates TB of data every day. Current approaches often involve disparate data formats, coordinate systems, and access patterns, limiting interoperability and scalability across different processing environments (Bauer-Marschallinger and Falkner, 2023). While data cubes have become valued abstractions for analysing large-scale geospatial data over space and time, they can currently only be implemented meaningfully in projected coordinate reference systems. This limitation restricts their spatial extent before introducing unacceptable areal distortions, particularly problematic for global-scale analyses where consistent area measurements are essential for statistical validity.

1.1 Discrete Global Grid Systems

The Discrete Global Grid System (DGGS) paradigm offers a solution to these challenges by providing a unified spatial reference framework based on hierarchical tessellation of the Earth's surface. The foundational methods and techniques underlying DGGS trace back to the 1960s and 1970s, with early explorations of triangular and hexagonal tessellations for spatial subdivision and hierarchical analysis methods (Christaller, 1966, Fuller, 1982, A. Rosenfeld and E. Luczak, 1976). During the 1980s and 1990s, significant advances were made in the development of lattice structures specifically designed for global grids, and in combining these concepts into more coherent spatial data management and analysis frameworks (Bell et al., 1983, Dutton, 1989, Snyder, 1992, Goodchild and Shiren, 1992, Gray, 1995). The early 2000s marked a pivotal era with seminal contributions from Kimerling, White, and Sahr, who formalised the concepts of Discrete Global Grids, geodesic grids, and global tessellations (Sahr et al., 2003). Particularly influential was Sahr's work on Geodesic Discrete Global Grid Systems, which introduced hexagonal hierarchical grids and led to the development of DGGRID. Concurrently, HEALPix (Górski et al., 2005) emerged from the astronomy community, demonstrating the practical utility of equal-area pixelization for large-scale data management.

Similar to the trajectory of neural networks, DGGS remained largely theoretical for decades due to computational constraints. However, recent advances in computing power and cloud infrastructure have made these systems not only feasible but exceptionally powerful for working with massive geospatial data. This computational enablement has spurred the emergence of industry-driven implementations, including Google's S2 geometry library (Veach et al., 2017) and Uber's H3 hexagonal grid system (Uber Technologies Inc, 2018). rHEALPix was introduced as one of the first DGGS explicitly defined for the ellipsoid (Gibb et al., 2016). More recent developments include IGEO7 (Kmoch et al., 2025c) - an equal-area alternative for H3, Felix Palmer's A5 system (Palmer, 2025), which provides

an aperture-4 grid based on a pure pentagonal basis, and Ecere's DGGAL (Discrete Global Grid Abstraction Library) framework (Ecere Corporation, 2025), which offers a generalized approach to DGGS implementation with support for multiple grid types and projection methods.

1.2 Equal-area DGGS zones as pixels for data cubes

Most DGGS implementations are constructed through hierarchical subdivision of Platonic solids (Mahdavi-Amiri et al., 2015), and often use the icosahedron for hexagonal systems (ISEA family) or the cube for quadrilateral approaches. HEALPix represents a notable exception, employing direct sphere pixelization that partitions the celestial sphere into equal-area zones without intermediate polyhedral projection. Furthermore, the majority of DGGS operate on the sphere rather than the ellipsoid, necessitating authalic (equal-area) conversions to maintain geometric fidelity when working with ellipsoidal Earth models. Systems like A5 and DGGAL already incorporate these conversions directly into their implementation. The OGC defined that equal-area means a distortion under 1%. For DGGRID-based systems and HEALPix, even though defined on the sphere, their distortions are already under 0.5%, and there is already functionality available that allows integration of authalic conversion (Karney, 2024).

Indexing/addressing	Data cube ready
- H3 Uber (not EA, 7x) - S2 Google (not EA, 4x) - A5 (IR, 4x) - Geohash (not EA, 4x) - Quadkey (not EA, 4x) - ISEA3H, 3x, or ISEA4H, 4x (EA, ambiguous hierarchy)	- HEALPix (diamonds, 4x) - IGEO7/ISEA7H/IVEA7H (like H3 but equal-area, 7x) - rHEALPix (mixed zone geometries, 9x) - DGGAL _4R _9R systems (rhombus/diamond 4x, 9x)

Table 1. A proposed differentiation of DGGS systems for the two major use cases - spatial data binning and indexing (left), and treating zones as pixels for data cube use (right); 4x, 7x, etc., refer to a grid's refinement ratio (subdivision/aperture); IR - irregular subdivision, no clear parent-child relationship; EA - equal-area property.

The Table 1 proposes a clearer distinction between DGGS implementations designed for spatial indexing and aggregation versus those intended as foundational pixel structures for data cubes. Systems like Uber's H3, Google's S2, and recently, A5, excel at spatial indexing, enabling efficient point-in-polygon queries, binning and aggregation of vector data, and spatial joins. These systems focus on computational efficiency and practicality, but they prioritise relative spatial relationships for location-based services over equal-area properties where absolute area measurements matter more.

However, for raster-based Earth Observation data cubes, ocean and climate modelling, or other quantitative geospatial modelling and analyses, the equal-area property becomes essential rather than optional. Here, the pixels represent measurements of physical phenomena—surface temperature, precipitation, vegetation indices, or land cover for the same areal units, and we need the ability to perform direct statistical comparisons and aggregations across resolutions. This depends fundamentally on each zone representing an equal area of the Earth's surface. Non-equal-area grids introduce systematic biases in spatial statistics, complicating tasks such as multi-temporal analysis, change detection, and model validation.

The OGC introduced the terms "refinement level", in comparison to the term "resolution", to distinguish hierarchical subdivision from metric spatial resolution, and "zone" over the more classically used term "cell" (akin to grid cell). While each zone at a given refinement level has a well-defined area and corresponding metric resolution, the refinement level number itself indicates hierarchical position, analogous to zoom levels in Web Mercator systems, rather than absolute ground sampling distance. In this article, we aim to stay consistent with the newer terminology.

DGGS like HEALPix, IGEO7, and rHEALPix are designed explicitly as pixel frameworks where each zone serves as a data container with consistent geometric properties. This "data cube ready" approach enables seamless integration with array-based processing frameworks like Xarray and Dask (Kmoch et al., 2024), where operations assume regular dimensional structures. The hierarchical refinement ratio (3x, 4x, 7x, or 9x depending on the system) allows natural downsampling and upsampling operations while preserving area relationships—critical for applications ranging from climate reanalysis to continental-scale land use mapping. A5 is effectively also an equal-area DGGS, but the subdivision structure is irregular, and the elongated pentagonal shape makes neighbourhoods and related simulations and algorithms difficult to reason over.

1.3 Standardised web services for large-scale geospatial data

Conventional web service standards such as Web Coverage Service (WCS), Web Coverage Processing Service (WCPS), and Web Map Service (WMS) have provided standardised access to geospatial data (OGC and Baumann, 2010). However, these services typically rely on traditional coordinate reference systems and raster-based approaches that present challenges when working with global datasets at multiple resolutions or when combining heterogeneous data sources.

The Open Geospatial Consortium (OGC) has developed standards and best practices for DGGS implementation, including the DGGS Abstract Specification and the DGGS API. The OGC Testbed-16 Engineering Report (Open Geospatial Consortium, 2020) and the ESA technical study on Sentinel-2 ARD handling with DGGS further elaborate on implementation approaches and use cases (Purss et al., 2019, Salgues et al., 2023). Equal-area DGGS implementations, such as HEALPix (Górski et al., 2005), rHEALPix (Gibb et al., 2016) and ISEA (Snyder, 1992), have gained particular attention for Earth Observation applications due to their ability to maintain consistent area measurements across the globe (Kmoch et al., 2022), and their usability for large-scale EO analysis, including improved statistical analysis and multi-resolution data fusion.

1.4 Scientific computing on gridded EO data

In parallel, the scientific Python ecosystem has seen significant developments in data handling capabilities with tools like Xarray (Hoyer and Hamman, 2017), which provides labelled multi-dimensional array operations. The XDGGS extension for Xarray enables direct manipulation of DGGS-indexed data within this framework (Kmoch et al., 2024). Additionally, the XPublish concept provides a mechanism for exposing Xarray datasets through web service interfaces, bridging the gap between analytical environments and web-based access patterns.

In this paper, we present universal building blocks that combine DGGS with cloud-optimised data formats to create a comprehensive framework for EO data management. We introduce *pydggsapi*, an open-source Python server built with FastAPI that implements the OGC API DGGS standard, exposing cloud-optimised Analysis-Ready Data (ARD) such as Zarr archives and Parquet files indexed by DGGS zones. This architecture creates a seamless continuum between two distinct operational scales: on one end, data scientists can perform large-scale modelling by directly accessing DGGS-indexed Zarr archives in object storage using Xarray and Dask; on the other, lightweight web and mobile clients can consume the same underlying data through the standardised, RESTful pydggsapi interface. By maintaining a single, consistent data foundation while optimising for different computational and bandwidth constraints, we address key challenges in the EO data processing chain, from initial data organisation to final delivery and visualisation.

2. Methods and materials

2.1 Building blocks

We implemented *pydggsapi*, an open-source Python FastAPI service, that exposes the OGC API DGGS standard (Kmoch et al., 2025b) with a backend utilising Zarr arrays indexed by DGGS zones. The concept is inspired by the TiTiler and XPublish packages, which can employ a ”serverless” FastAPI web service routing interface on top of cloud-native and Xarray datasets. This architecture leverages complementary technologies as shown in Figure 1.

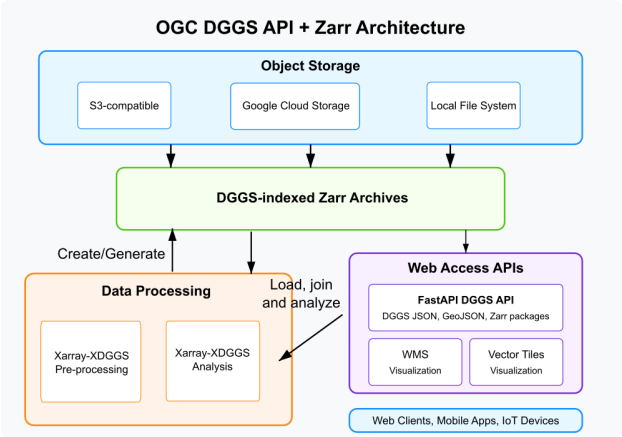


Figure 1. The proposed architecture of a modern cloud- and analysis-ready DGGS-based geospatial ecosystem.

The DGGS serves as a universal spatial index, providing a hierarchical, multiple-refinement-level grid system that consistently indexes geospatial data across the entire globe. This eliminates the need for reprojection when combining datasets from disparate sources and enables immediate analysis without preprocessing steps, and extends and improves upon the grid notion that is established in the met/ocean and climate communities.

Zarr and Parquet technology function as the cloud-optimised storage foundation, with its chunked, compressed array or table format enabling efficient parallel access to massive datasets stored in object storage systems while maintaining critical dimensional information and supporting selective data extraction at multiple refinement levels.

As a novel composite aspect, the architecture leverages already available software, such as Xarray-XDGGS, which is a package that implements 1-D DGGS indexed arrays to work with various open-source DGGS libraries and systems. Through the Xarray DataTree model and the representation in Zarr data groups, we can aggregate data towards higher-level DGGS refinement levels. The concept corresponds to image pyramids and overviews in other cloud-native formats like COG GeoTIFFs or PMTiles. As an example, we show use of vector tiles (MVT) to enable visual and data access. MVT can be very efficient for in-browser rendering by using WebGL with MapLibre GL JS as shown in Figure 4 a.

Ultimately, with *pydggsapi*, the system employs an OGC API DGGS interface that serves as the primary access point for web clients, providing both visualisation services and precise data access through standardised formats, including DGGS JSON, GeoJSON, or as hybrid Zarr ”packages” that maintain the original data structure while enabling efficient transfer. The web service API is meant for light-weight query and visualisation access for web- or mobile (or IoT)-based client applications.

2.2 Pydggsapi software architecture

pydggsapi is developed in Python. It provides an easy setup for users to publish DGGS-enabled data sources. It uses FastAPI as the core web-service platform to implement the OGC API DGGS standard. *pydggsapi* also aims to provide a flexible structure that can easily extend to support different DGGRS and data storage. Figure 2 shows the overall structure of *pydggsapi*.

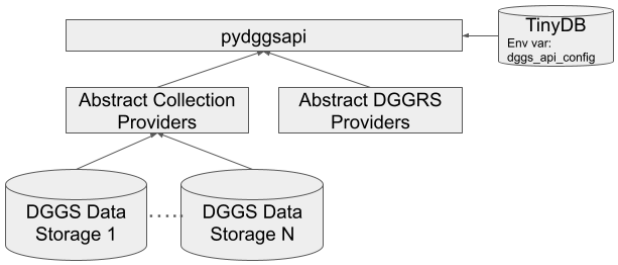


Figure 2. The general application structure of *pydggsapi*.

pydggsapi uses TinyDB, a JSON-based configuration database, to declare the supported DGGRS types and the available collections with their data sources. The application structure provides two abstract class definitions that allow developers to easily implement new DGGRS and data sources to work with *pydggsapi*. Currently, it supports:

1. DGGRS: IGEO7, H3, HEALPix, rHEALPix, IVEA7H, and ISEA7H_Z7 (i.e. DGGAL's implementation of IGEO7)
2. Collection/data source providers: Clickhouse database, Zarr (through *xdggs*), and Parquet

The HEALPix, rHEALPix, IVEA7H and ISEA7H_Z7 grids are implemented via the DGGAL Python implementation.

Our concept bridges two distinct operational scales - cloud-native big data processing and a more refined web service-based access for lightweight clients. Zarr arrays stored in object storage serve as the unified data foundation, enabling direct access for high-performance computing and cloud-based modelling workflows. The OGC API DGGS implementation, akin

to XPublish for Xarray, provides standardised, RESTful access to the same underlying data for a diverse range of web, mobile, and IoT clients and applications. This dual-scale approach ensures data consistency across use cases while optimising for different computational and bandwidth constraints.

The data collection provider components enable access to DGGS-indexed datasets through middleware that manages connections to cloud-storage Zarr archives. Based on the OGC ER-16 report, the application also showcases an experimental connector for the Clickhouse database to provide fast on-demand aggregation and analytical queries on DGGS-indexed database tables.

The main architecture relies on pre-aggregated pyramids, where the data access middleware reads metadata from Zarr archives. During initialisation, it extracts DGGS parameters from the Zarr archive's attributes, including the DGGS type (such as HEALPix or H3) and the indexing scheme used or additional index parameters (e.g., HEALPix' nested or ring scheme), available refinement levels, and the available data variables.

2.3 Storage optimisation through hierarchical indexing

The hierarchical nature of DGGS indices, whether alpha-numeric, like IGEO7's digit strings, or integer-based, like HEALPix's nested scheme (cf. Figure 3), inherently encodes spatial proximity through lexicographic or numerical ordering. This property directly mirrors the behaviour of space-filling curves such as Hilbert or Morton (Z-order) curves, where sequential index values correspond to spatially adjacent zones. Leveraging this spatial locality in the underlying storage format is useful for optimising read access patterns in cloud-native architectures.

For Zarr-based storage, chunking strategies should align with DGGS parent zones to maximise spatial coherence within chunks. When data is organised such that all child zones of a given parent reside within the same or adjacent chunks, queries at any refinement level benefit from minimised read operations. A practical approach involves chunking at a coarser DGGS refinement level (e.g., refinement level N-2 or N-3) while storing data at the target refinement level N, ensuring that hierarchical queries traverse contiguous storage blocks. The chunk size should balance between parallelisation granularity and I/O efficiency, typically targeting 10-100 MB per chunk depending on network characteristics and compression ratios.

Parquet's columnar format offers complementary advantages through row group partitioning aligned with DGGS zones. By partitioning datasets using DGGS zone identifiers as the primary key, range queries on zone indices translate directly to row group pruning, dramatically reducing data scanning. Parquet's statistics (min/max values per row group) enable efficient spatial filtering when zone IDs are properly ordered. For multi-refinement level datasets, storing the parent zone ID as an additional column enables rapid coarse-to-fine queries without reconstructing the hierarchical relationships at query time.

Pre-computed aggregate overviews represent another optimisation strategy. By additionally materialising statistics (mean, sum, count, etc.) at coarser DGGS refinement levels and storing them as separate Zarr arrays or Parquet tables, applications can rapidly retrieve lower-resolution summaries without accessing higher refinement level data. These overviews should follow a pyramid structure aligned with the DGGS refinement ratio (4x,

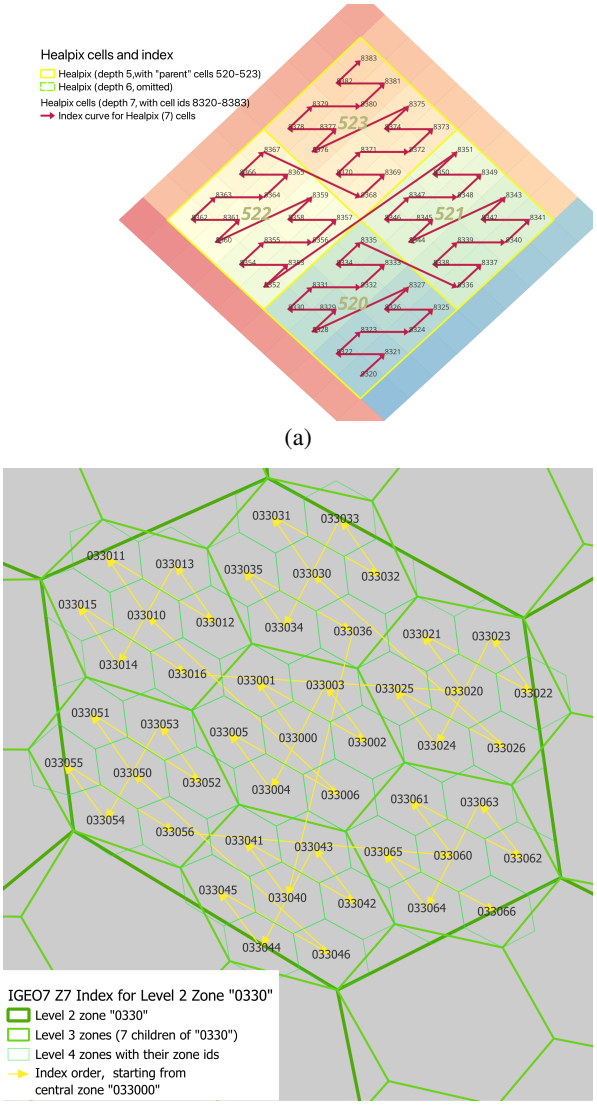


Figure 3. Examples of space-filling curves and hierarchical zone indexing in DGGRS - the alpha-numerical or bitwise/integer order should be utilised in the underlying storage format for read-access structuring and optimisation, a) HEALPix; b) IGEO7

7x, or 9x), with each level pre-aggregated from its children. When combined with lazy evaluation frameworks like Dask, this approach enables interactive visualisation and exploratory analysis at continental to global scales, loading higher refinement level data only when zooming to specific regions. The storage overhead of maintaining 3-4 refinement levels of overviews (approximately 25-35% additional space depending on refinement ratio) is offset by order-of-magnitude improvements in query latency for multi-scale applications.

The architecture aims to abstract the underlying storage system through the Zarr application library interface, providing consistent data retrieval whether data is stored in AWS S3, Azure Blob Storage, Google Cloud Storage, or local file systems. For each API request, the middleware identifies the target collection from the URL path, loads the Zarr metadata, attaches DGGS configuration to the request context, and queries the Zarr store instance purely on zone identifiers. To improve performance, the system maintains an in-memory cache of collection metadata, reducing storage access operations and speeding up repeated queries for common metadata.

This approach enables the FastAPI application to seamlessly serve OGC API DGGS requests while maintaining the performance benefits of cloud-native Zarr storage for Earth Observation data indexed by DGGS zones.

DGGS indexing automatically aligns diverse data products spatially and temporally, facilitating immediate analysis without further preprocessing. Via the OGC API DGGS, it is possible to provide convenient access to clients to zone-based summary queries, for example, as shown in Figure 4b., at a specific refinement level (i.e. higher resolution). The OGC API DGGS layer enables targeted data extraction for non-DGGS-capable clients, reducing data transfer volumes and simplifying implementation for client applications. Direct HPC access to cloud-stored Zarr archives is facilitated by high-throughput network connections and cloud-aware libraries like *fsspec* and *xarray*. These tools enable HPC applications, e.g. built on libraries like *Dask* to stream Zarr chunks directly from object storage APIs (e.g., S3), enabling efficient, highly parallelised batch processing over large areas and timescales without pre-staging the entire dataset.

3. Discussion

3.1 Universal Building Blocks for Earth Observation Data

Why universal building blocks? The combination of DGGS and Zarr creates building blocks for Earth Observation data management that address key challenges in the geospatial data pipeline. Maintaining a unified base of DGGS-indexed Zarr archives enables consistent data organisation across multiple use cases. This approach supports both large-scale and local applications through a dual-access pattern: direct access to cloud-stored Zarr archives for high-performance computing and API-mediated access for web clients with bandwidth constraints.

Pydggsapi also includes the well-known OpenAPI3/ Swagger documentation and client, which provides a low-barrier entry into experimenting with the API. In the near future, we will also provide training materials based on Python/Jupyter notebooks.

For large-scale EO applications requiring consistent area-based statistics across multiple resolutions and temporal dimensions, DGGS systems with equal-area guarantees and explicit pixel semantics provide the necessary foundation for cloud-native data cubes. However, indexing-focused systems remain well-suited for discrete feature aggregation and spatial lookup operations.

The indexing-only approach, while computationally elegant for vector operations and spatial queries, requires additional geometric corrections when used for continuous field data. Converting point observations or vector geometries into a DGGS index is fundamentally different from treating DGGS zones as native pixels in a multi-dimensional array. The latter enables direct application of well-established remote sensing and climate modelling workflows, leveraging decades of algorithmic development in raster analysis, while the former necessitates additional preprocessing and area-weighting steps that complicate reproducibility and scalability.

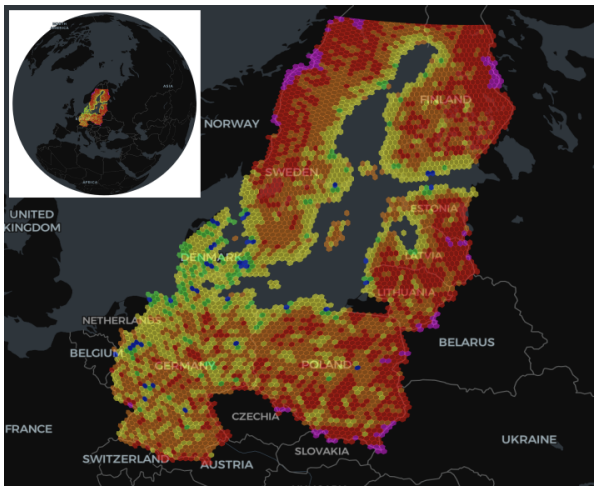
3.2 Example workflows

This architecture enables advanced geospatial analysis across multiple domains. For instance, a researcher studying catchment-scale erosion can query *pydggsapi* for all land cover and slope data within DGGS zones intersecting their basin boundary. The API backend accesses a massive, continental-scale Zarr dataset and extracts only the required zones, returning a manageable subset for analysis. Alternatively, the researcher can execute an Xarray batch job that directly accesses the cloud-stored Zarr archive, computes erosion risk statistics across the entire catchment, and returns comprehensive results—all without downloading the full dataset. Furthermore, users can easily coalesce data from multiple DGGS-enabled data sources through simple zone-ID-based joins, enabling trivial data federation. This capability significantly simplifies the integration of heterogeneous datasets from different providers, as the common DGGS indexing eliminates complex spatial alignment procedures.

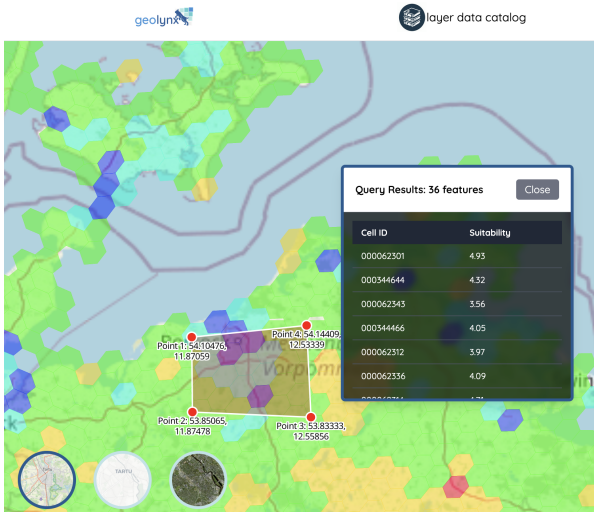
3.3 Standardization Requirements

The OGC DGGS working group is advancing standards for DGGS implementations, but additional components are still needed. A registry for DGGS Reference Systems, similar to the EPSG codes and Proj/WKT CRS definitions, is needed. Clear conventions for storing DGGS parameters in Zarr metadata fields must be established. For HEALPix, essential parameters include the desired refinement level and the indexing scheme (nested or ring), while more flexible systems like rHEALPix or ISEA-based DGGS may use additional parameters such as indexing/numbering scheme, origin, and rotation. The CF Metadata Conventions community has also picked up a discussion on grid parameter specification for NetCDF and Zarr archives (CF Metadata Conventions, 2025).

The OGC API DGGS implementation requires an indexing scheme that encodes the refinement level in the zone identifier. Systems like H3 and S2 have popularised this approach, and newer DGGS reference systems like IGEO7 (DGGRID/Z7 (Kmoch et al., 2025c)) and rHEALPix can also provide this functionality. However, a more accessible implementation is needed for HEALPix (namely *nuniq* or *zuniq*). A notable requirement in the OGC API DGGS standard is subzone ordering, which currently only DGGAL systems appear to fully support (Ecere Corporation, 2025). Most DGGS implementations have a space-filling curve index and rely on associating data values with zone identifiers during data transport. However, assuming that all child zones at a relatively higher refinement level are



(a)



(b)

Figure 4. Examples of DGGS web clients, a) MVT allows for lightweight browser visualisation; b) Cell-based summary queries

transferred for a parent zone, then the ordering of data points, even without zone IDs, could be considered reliable, especially via the alpha-numerical ordering of all child zones.

3.4 Technical Limitations and Future Work

The current implementation faces certain limitations that require further research and software engineering expertise. For example, Xarray's eager indexing of dimensions may limit its utility for very large DGGS archives, as all dimension indexes are loaded into memory. More efficient handling of multi-refinement level DGGS data may require extensions to the Xarray data model or more direct Zarr-native access approaches.

The use of Zarr groups/Xarray data trees to represent refinement level aggregations requires additional experimentation to ensure interoperability and efficient chunking/sharding, which takes DGGS zone topologies and parent-child-zone boundaries into account. The OGC ER-16 report outlines the methodology for providing descriptive statistics per zone, including standard deviation, ranges, min/max values, and variance. Implementing this efficiently within the Zarr data model across refinement levels would be useful to better convey the uncertainty of aggregated data.

Initial performance benchmarks do not indicate disadvantages compared to traditional approaches regarding data access performance. While this should be benchmarked more critically in the future, the data access patterns rely on state-of-the-art object storage connectivity. Combined with the inherent advantages of DGGS for global data analysis, these results suggest encouraging progress. There remains substantial potential for improvements in both usability and performance as the technology matures.

4. Conclusion

The integration of OGC API DGGS with cloud-native Zarr and Parquet storage represents a significant step toward universal building blocks for Earth Observation data management. This work demonstrates that combining equal-area DGGS as a unified spatial reference framework with cloud-optimised storage formats and standardised web service APIs creates a powerful, dual-access pattern that serves both high-performance computing and lightweight clients from a single, consistent data foundation.

The *pydggsapi* implementation showcases how this architecture enables seamless transitions between operational scales—from continental-scale batch processing using Xarray and Dask to targeted queries from web and mobile applications. The common DGGS indexing eliminates traditional barriers to data integration, making data federation across multiple providers trivial through simple zone-ID-based joins. This capability fundamentally simplifies the integration of heterogeneous datasets and reduces the complexity of multi-source geospatial analysis.

Looking forward, we envision this model as a path toward a new generation of value-added, GeoAI-ready data market APIs. As machine learning and artificial intelligence increasingly drive Earth Observation applications, the combination of Analysis-Ready Data in cloud-native formats, standardised DGGS indexing, and accessible web service interfaces provides an ideal foundation for training data preparation, model deployment, and result dissemination. The equal-area properties of DGGS ensure statistical validity across scales, while the dual-access architecture supports both the computational demands of model training and the lightweight requirements of inference services.

While challenges remain—particularly in standardisation, performance optimisation, and tooling maturity—the initial results are encouraging. The convergence of DGGS, cloud-native storage, and standardised APIs represents not merely an incremental improvement but a fundamental shift in how we organise, access, and analyse global geospatial data.

Code availability

The *pydggsapi* software and the examples are publicly available at: github.com/LandscapeGeoinformatics/pydggsapi/ (Kmoch et al., 2025b) and github.com/LandscapeGeoinformatics/FOSS4G-2025_DGGRID_and_pydggsapi_Workshop/ (Kmoch et al., 2025a). The GitHub repositories maintain the base implementation of *pydggsapi* and collate examples of DGGS-based data preprocessing and service configuration. They are open to public contributions.

Acknowledgements

This work was supported by the Estonian Research Agency (grant number PRG1764, PSG841), Estonian Ministry of Education and Research, Centre of Excellence for Sustainable Land Use (TK232) and by the European Union (ERC, WaterSmartLand, 101125476, Interreg-BSR, HyTruck, #C031), and by ESA (GRID4EARTH, contract 4000147951/25/I-NS). Views and opinions expressed are however, those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them. The authors are thankful for technical support from the High Performance Computing Center of the University of Tartu (University of Tartu, 2018). The authors also acknowledge the constructive communication with the OGC DGGS domain and standards working groups.

References

A. Rosenfeld, E. Luczak, 1976. Distance on a Hexagonal Grid. *IEEE Transactions on Computers*, 25(05), 532–533. doi.org/10.1109/TC.1976.1674642.

Bauer-Marschallinger, B., Falkner, K., 2023. Wasting petabytes: A survey of the Sentinel-2 UTM tiling grid and its spatial overhead. *ISPRS Journal of Photogrammetry and Remote Sensing*, 202, 682–690.

Bell, S. B. M., Diaz, B. M., Holroyd, F., Jackson, M. J., 1983. Spatially referenced methods of processing raster and vector data. *Image and Vision Computing*, 1(4), 211–220. doi.org/10.1016/0262-8856(83)90020-3.

CF Metadata Conventions, 2025. Convention for HEALPix grid parameters. github.com/cf-convention/cf-conventions/issues/433.

Christaller, W., 1966. *Central Places in Southern Germany*. First edition edn, Prentice Hall.

Dutton, G., 1989. Planetary modelling via hierarchical tessellation. *Auto-carto 9. Proc. symposium, Baltimore, MD, 1989*, 01760(508), 462–471.

Ecere Corporation, 2025. DGGAL, the Discrete Global Grid Abstraction Library. dggal.org.

Fuller, R., 1982. *Synergetics: Explorations in the Geometry of Thinking*. Collier books, Macmillan.

Gibb, R., Raichev, A., Speth, M., 2016. The rHEALPix DGGS preprint.

Goodchild, M. F., Shiren, Y., 1992. A hierarchical spatial data structure for global geographic information systems. *CVGIP: Graphical Models and Image Processing*, 54(1), 31–44.

Górski, K. M., Hivon, E., Banday, A. J., Wandelt, B. D., Hansen, F. K., Reinecke, M., Bartelmann, M., 2005. HEALPix: A Framework for High-Resolution Discretization and Fast Analysis of Data Distributed on the Sphere. *The Astrophysical Journal*, 622(2), 759. doi.org/10.1086/427976.

Gray, R. W., 1995. Exact transformation equations for Fuller's world map. *Cartographica*, 17–25.

Hoyer, S., Hamman, J., 2017. xarray: N-D labeled Arrays and Datasets in Python. 5(1), 10. Number: 1 Publisher: Ubiquity Press.

Karney, C. F. F., 2024. On auxiliary latitudes. *Survey Review*, 56(395), 165–180. doi.org/10.1080/00396265.2023.2217604.

Kmoch, A., Bovy, B., Magin, J., Abernathey, R., Coca-Castro, A., Strobl, P., Fouilloux, A., Loos, D., Uuemaa, E., Chan, W. T., Delouis, J.-M., Odaka, T., 2024. XDGGS: A community-developed Xarray package to support planetary DGGS data cube computations. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLVIII-4/W12-2024, 75–80. doi.org/10.5194/isprs-archives-XLVIII-4-W12-2024-75-2024.

Kmoch, A., Chan, W. T., Luik, I., Chrapkiewicz, K., Uuemaa, E., 2025a. Foss4g auckland dggs and pydggsapi workshop materials. [Zenodo] doi.org/10.5281/zenodo.17844019.

Kmoch, A., Chan, W. T., Migneault, F., 2025b. pydggsapi: A python FastAPI OGC DGGS API implementation. [software] github.com/LandscapeGeoinformatics/pydggsapi. last accessed December 1, 2025.

Kmoch, A., Sahr, K., Chan, W. T., Uuemaa, E., 2025c. IGEO7: A new hierarchically indexed hexagonal equal-area discrete global grid system. *AGILE: GIScience Series*, 6(32). doi.org/10.5194/agile-giss-6-32-2025.

Kmoch, A., Vasilyev, I., Virro, H., Uuemaa, E., 2022. Area and Shape Distortions in Open-Source Discrete Global Grid Systems. *Big Earth Data*. doi.org/10.1080/20964471.2022.2094926.

Mahdavi-Amiri, A., Alderson, T., Samavati, F., 2015. A Survey of Digital Earth. *Computers & Graphics*, 53, 95–117. doi.org/10.1016/j.cag.2015.08.005. Publisher: Elsevier.

OGC, Baumann, P., 2010. *OGC WCS 2.0 Interface Standard - Core, v2.0*. The Open Geospatial Consortium (OGC), http://www.opengeospatial.org/standards/is. Publication Title: WCS 2.0.

Open Geospatial Consortium, 2020. OGC Testbed-16: OpenAPI Engineering Report.

Palmer, F., 2025. A5: Pentagonal Discrete Global Grid System. a5geo.org.

Purss, M. B. J., Peterson, P. R., Strobl, P., Dow, C., Sabeur, Z. A., Gibb, R. G., Ben, J., 2019. Datacubes: A Discrete Global Grid Systems Perspective. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 54(1), 63–71. doi.org/10.3138/cart.54.1.2018-0017.

Sahr, K., White, D., Kimerling, A. J., 2003. Geodesic Discrete Global Grid Systems. *Cartography and Geographic Information Science*. doi.org/10.1559/152304003100011090.

Salgues, G., Cadau, E. G., Pessiot, L., Gaudissart, V., Enache, S., Gascon, F., Boccia, V., Strobl, P., 2023. A candidate dggs (discrete global grid system) for sentinel-2: First outcomes. *IGARSS 2023 - 2023 IEEE International Geoscience and Remote Sensing Symposium*, 4978–4981.

Snyder, J. P., 1992. An Equal-Area Map Projection For Polyhedral Globes. *Cartographica*, 29(1), 10–21. Publisher: University of Toronto Press.

Uber Technologies Inc, 2018. H3: Uber's Hexagonal Hierarchical Spatial Index. h3geo.org.

University of Tartu, 2018. Ut rocket. share.neic.no/10.23673/PH6N-0144.

Veach, E., Rosenstock, J., Engle, E., Snedegar, R., Basch, J., Manshreck, T., 2017. S2 Geometry. s2geometry.io.