



AI-DGGS DISASTER MANAGEMENT PILOT REPORT

ENGINEERING REPORT

PUBLISHED

Submission Date: 2026-02-12

Approval Date: 2026-03-04

Publication Date: 2026-07-31

Editor: Stelios Contarinis, Sina Taghavikish

Notice: This document is not an OGC Standard. This document is an OGC Public Engineering Report created as a deliverable in an OGC Interoperability Initiative and is *not an official position* of the OGC membership. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard.

Further, any OGC Engineering Report should not be referenced as required or mandatory technology in procurements. However, the discussions in this document could very well lead to the definition of an OGC Standard.

License Agreement

Use of this document is subject to the license agreement at <https://www.ogc.org/license>

Copyright notice

Copyright © 2026 Open Geospatial Consortium

To obtain additional rights of use, visit <https://www.ogc.org/legal>

Note

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

CONTENTS

I. OVERVIEW	ix
II. EXECUTIVE SUMMARY	ix
III. KEYWORDS	x
IV. CONTRIBUTORS	x
IV.A. Sponsoring Organizations	xi
V. FUTURE OUTLOOK	xii
V.A. Strategic Standardization Roadmap	xii
VI. VALUE PROPOSITION	xiii
1. INTRODUCTION	2
1.1. Operational Challenges in Modern Disaster Management	2
1.2. The Case for DGGs: A Digital-First Spatial Foundation	3
1.3. Participants and Deliverables	4
2. AI-DGGs PILOT ARCHITECTURE (THE SERVER BACKBONE)	7
2.1. The Case for DGGs: A World Model Paradigm Shift	7
2.2. DGGs-centric System Design	8
2.3. The Foundational Stack (Server Backbone)	9
2.4. Functional Analysis Capabilities	9
2.5. The Spatial Tokenizer: Data Models and Encodings	10
2.6. The Integration Glue: Intelligent COP Schema	10
3. AI ORCHESTRATION (THE INTELLIGENCE FRONTIER)	13
3.1. Overview: Giving AI a “Geospatial Brain”	13
3.2. Model Context Protocol (MCP): The “Action Protocol”	13
3.3. The Metadata Discovery Gap	14
3.4. Retrieval-Augmented Generation (RAG): The “Context Engine”	14
3.5. Curing “Scale Blindness” through Progressive Refinement	15
3.6. Semantic Reasoning and Cascading Risk Analysis	16
3.7. Participant Implementations (Intelligence Frontier)	16
4. DGGs FRICTIONS (TECHNICAL IMPLEMENTATION CHALLENGES)	19
4.1. Overview: Reconciling Theory with Reality	19
4.2. Friction 1: Geometric Alignment (Spherical vs. Ellipsoidal Models)	19
4.3. Friction 2: Sparse Data and Storage Bloat (The “Data Hole” Problem)	20

4.4. Friction 3: The Stacking Paradox (Non-Nested Hierarchies)	20
4.5. Friction 4: Cross-Platform Integrity (Logic and Logic Drift)	21
4.6. Friction 5: Spatial Temporal Resolution and Range Queries	21
4.7. Friction 6: The Interpretation Gap (Contextual Blindness)	22
4.8. Summary of Implementation Challenges	23
5. DGGS DEMONSTRATORS AND WEB RESOURCES	26
5.1. Overview of the Distributed Pilot Environment	26
5.2. The AI-DGGS Pilot Hub	26
5.3. The Server Backbone (DGGS APIs)	26
5.4. The Intelligence Frontier (AI Clients)	35
5.5. The Integration Glue (Schema & Tools)	42
5.6. Open Source Technical Resources	43
6. STANDARDIZATION ROADMAP AND FUTURE OUTLOOK	45
6.1. Overview: From Data Retrieval to Geospatial Reasoning	45
6.2. The AI-DGGS Standardization Roadmap	45
6.3. Operational Alignment Matrix: From Frictions to Standardization Actions	49
6.4. Future Research and Operational Outlook	49
7. SECURITY, PRIVACY, AND ETHICAL CONSIDERATIONS	53
7.1. Implementation Status and Disclaimer	53
7.2. The IPT Framework: Proposed Concepts for Trust	53
7.3. Theoretical Ethical AI and Decision Safeguards	53
7.4. Data Protection and Privacy Considerations	54
7.5. Summary of Proposed Trust Requirements	54
BIBLIOGRAPHY	57
ANNEX A ABBREVIATIONS/ACRONYMS	60
ANNEX B DGGS: A PARADIGM SHIFT IN GEOGRAPHIC INFORMATION	63
B.1. The Legacy of Layers	63
B.2. World-Centric Representation	63
B.3. The Power of Single-Index Lookups	64
B.4. Analysis-Ready Integration	64
B.5. Spatial Tokens and Digital Fingerprints	65
B.6. The Substrate for GeoAI	65
B.7. Conclusion	66
ANNEX C HOW TO SELECT WHICH DGGS TO USE?	68
C.1. Useful Selection Aspects	68
C.2. Decision Logic for DGGRS Selection	71
C.3. DGGRS Implementation Matrix	72
ANNEX D SERVER BACKBONE: LIBRARY AND TOOLKIT SPECIFICATIONS	75

ANNEX E SERVER ANALYSIS CAPABILITIES	77
ANNEX F DGGGS-ZARR ENCODING DRAFT SPECIFICATION	81
F.1. Introduction	81
F.2. Current DGGGS metadata approaches for CF conventions, Xarray(xdgggs) and Zarr	81
F.3. Core DGGGS Metadata Parameters (Zarr)	83
F.4. Profiles Zarr	85
F.5. JSON Examples	85
F.6. Status	86
F.7. Summary Recommendations for Next-Gen NetCDF-CF DGGGS Conventions	86
F.8. Changes proposed to the Next-Gen CF-Conventions for arbitrary DGGGS support	87
F.9. Future work	89
F.10. References	89
ANNEX G CRIM – DGGGS API IMPLEMENTATION	92
G.1. Architecture and Design	92
G.2. Implemented Capabilities	92
G.3. Interoperability and Testing	94
G.4. Performance and Challenges	96
G.5. Outlook and Resources	97
ANNEX H ECERE – DGGGS API IMPLEMENTATION	100
H.1. Architecture and Design	100
H.2. Implemented Capabilities	101
H.3. Interoperability and Testing	101
H.4. Performance and Challenges	102
H.5. Outlook and Resources	102
ANNEX I GEOINSIGHT – DGGGS API IMPLEMENTATION	106
I.1. Architecture and Design	106
I.2. Implemented Capabilities	107
I.3. Interoperability and Testing	110
I.4. Performance and Challenges	110
I.5. Outlook and Resources	111
ANNEX J GEOLYNX – DGGGS API IMPLEMENTATION	113
J.1. Architecture and Design	113
J.2. 2. Implemented Capabilities	114
J.3. 3. Interoperability and Testing	116
J.4. 4. Performance and Challenges	117
J.5. 5. Outlook and Resources	119
ANNEX K GEOMATYS – DGGGS API IMPLEMENTATION	121
K.1. Architecture and Design	121
K.2. Implemented Capabilities	122
K.3. Interoperability and Testing	124

K.4. Performance and Challenges	125
K.5. Outlook and Resources	125
ANNEX L SAFE SOFTWARE – DGGS API IMPLEMENTATION	127
L.1. Architecture and Design	127
L.2. Implemented Capabilities	127
L.3. Interoperability and Testing	133
L.4. Performance and Challenges	134
L.5. Outlook and Resources	135
ANNEX M COMPUSULT – AI-ENABLED DGGS CLIENT	138
M.1. System Overview	138
M.2. DGGS Integration Logic	141
M.3. AI and Explainability Functions	142
M.4. User Interaction and Testing	142
M.5. Conclusions and Future Work	145
ANNEX N TERRAFRAME – AI-ENABLED DGGS CLIENT	148
N.1. System Overview	148
N.2. DGGS Integration Logic	148
N.3. AI and Explainability Functions	149
N.4. User Interaction and Testing	149
N.5. Outlook and Resources	151
ANNEX O CS GROUP – AI-ENABLED DGGS CLIENT	154
O.1. System Overview	154
O.2. DGGS Integration Logic	156
O.3. AI and Explainability Functions	157
O.4. User Interaction and Testing	158
O.5. Technology Integration Experiments (TIEs)	160
O.6. DGGS Metadata and Naming Conventions	161
O.7. Outlook and Resources	162
ANNEX P HARTIS – AI-ENABLED DGGS CLIENT	164
P.1. System Overview	164
P.2. System Workflow	164
P.3. DGGS Integration and Spatial Processing	166
P.4. DGGS Playground	166
P.5. AI Reasoning and Explainability	167
P.6. User Interaction and Operational Scenarios	168
P.7. Outlook and Relevance to OGC Standardization	170
P.8. Demonstrators and Resources	170
ANNEX Q H3 OGC API – DGGS IMPLEMENTATION AND INTEROPERABILITY CONSIDERATIONS	173
Q.1. Scope and Rationale	173
Q.2. Earth reference model	173

Q.3. Zone Boundaries and Topology	175
Q.4. Topological sub-zones	178
Q.5. Towards an H3 DGGRS definition and OGC Best Practice	178
ANNEX R H3 SUBZONE ALGORITHM	180
R.1. H3 subzone, the missing feature of H3	180
R.2. Computing H3 subzones	180
ANNEX S APACHE-SIS JAVA DGGRS API	187
S.1. Overview	187
S.2. Discrete Global Grid Reference System API	188
S.3. Reference System	190
S.4. Resource and Data API	190
ANNEX T DGGAL JAVA BINDING	193
T.1. Panama binding	193
T.2. Relationship between DGGAL API and Apache SIS API	193
ANNEX U CLIENT/SERVER DGGRS AND FORMATS BENCHMARKS	199
U.1. Method	199
U.2. Formats performances	200
U.3. Response size by format and zone depth	200
U.4. Response size ratio compared with JSON	200
U.5. Response time by format and zone depth	201
U.6. DGGRS performances	202
U.7. Conclusions	205
ANNEX V PROTOTYPING AN EXTENSION FOR SIMPLER DATA ACCESS	208
V.1. Why a new extension ?	208
V.2. Summary zone values extension	208
V.3. Zone resolution extension	209
ANNEX W HIERARCHICAL PROGRESSIVE SURVEYS INTEROPERABILITY	212
W.1. What is HIPS ?	212
W.2. Examind connector	212
W.3. Results	212
ANNEX X COMMON OPERATING PICTURE (COP)	216
X.1. Context and Rationale	216
X.2. Keywords	216
X.3. Background	216
X.4. Evolution of COP Standards	217
X.5. COP as an Information Sharing Mechanism	219
X.6. COP as an Orchestrator Platform	221
X.7. Identity, Provenance, and Trust (IPT) Integration	222
X.8. Small Local Language Models (SLLM) Integration	224
X.9. Extended Schema Proposal	225

X.10. Benefits and Expected Outcomes	226
X.11. Security and Ethical Considerations	226
X.12. Annex References	226
ANNEX Y FOUNDATION OF THE FLOOD IMPACT INDEX (FII)	229
Y.1. DGGS-Native Framing	229
Y.2. Pixel to DGGS Transfer for Flood Evidence	230
Y.3. Data Classes and Indicators	230
Y.4. Normalization and Synthesis Logic	231
Y.5. Hybrid Risk Products and AI-Enabled Serving	232
Y.6. Core Data Sources	232
Y.7. Operational Use in the AI-DGGS Client	233
INDEX	235



OVERVIEW

The AI-DGGS for Disaster Management Pilot, conducted under the OGC Research Program and sponsored by Natural Resources Canada (NRCan), the European Space Agency (ESA), Centre national d'études spatiales (CNES), and the United States Geological Survey (USGS), examines the convergence of **Discrete Global Grid Systems (DGGS)**, Artificial Intelligence (AI), and OGC APIs. The initiative anchors its research in the **Red River Basin** of Manitoba, Canada—a region prone to high-consequence flooding—to simulate how standards-based AI can advance flood risk assessment and explainable scenario modeling.

Traditional geospatial systems are often rooted in the legacy of paper maps, leading to distortions and the “Raster-Vector Divide.” DGGS offers a digital-first spatial foundation, treating each cell as a unique identifier and a unit of analysis. The recent publication of the **OGC API – DGGS** transforms this theoretical framework into a practical engine for global spatial analytics.

Within this pilot, DGGS serves as the **spatial tokenizer** for AI. By indexing authoritative data to specific grid cells, AI models (LLMs) can access location-indexed facts instead of relying on pre-trained knowledge. This architecture effectively cures “**Scale Blindness**”—the tendency for AI to request computationally impossible amounts of data—by using the grid to provide progressive refinement and context-aware filtering.

Linking a standards-based architecture with real-world operational scenarios, the pilot establishes a **common analytical framework** that improves the interoperability, transparency, and explainability of geospatial intelligence. The combination of the geospatial analytical framework provided by DGGS with AI analytical tools allowed this pilot to demonstrate examples of how human operators can ask natural language questions and receive answers about natural hazard risk management that are grounded in physical reality.



EXECUTIVE SUMMARY

The Artificial Intelligence-Augmented Discrete Global Grid Systems (AI-DGGS) for Disaster Management Pilot set out to bridge the “**Discovery Gap**”: the barrier that prevents emergency responders from getting instant answers to plain-language questions because data is buried in fragmented, incompatible systems. In a crisis, decision-makers are often data-rich but information-poor.

This pilot successfully demonstrated a breakthrough in disaster response by **giving Artificial Intelligence a “Geospatial Brain.”** By adopting **DGGS** as a common “Digital Fabric,” participants proved that diverse datasets—from RADARSAT Constellation Mission (RCM) radar imagery to socio-demographic population counts—can be unified into a single, cell-based representation of the Earth.

The result is a shift from “searching for data” to “finding answers.” By integrating AI reasoning via **Retrieval-Augmented Generation (RAG)** and the **Model Context Protocol (MCP)**, the pilot

enabled non-expert users to pose natural-language queries and receive **transparent, data-grounded answers** visualized directly on a map.

To deliver this capability, the pilot established a coordinated architecture consisting of a “**Server Backbone**” (optimized DGGS data providers) and an “**Intelligence Frontier**” (AI-aware client applications). Through Technology Integration Experiments (TIEs), the project demonstrated that true interoperability among independent vendors is achievable. The outcomes point toward a future where **grid-native, agentic AI infrastructures** strengthen climate resilience and establish a **Common Operating Picture (COP)** that is auditable, traceable, and trusted.



KEYWORDS

The following are keywords to be used by search engines and document catalogues.

Discrete Global Grid Systems (DGGS), OGC API – DGGS, AI-enabled flood risk assessment, Earth Observation, climate resilience, geospatial modeling, interoperability, disaster management, artificial intelligence (AI), machine learning (ML), chat-bot, large language models (LLM), retrieval-augmented generation (RAG), explainable AI, AI-enabled decision support, model context protocol (MCP), agentic AI, common operating picture (COP), Spatial Knowledge Graph, OpenAPI, natural hazard modeling



CONTRIBUTORS

All questions regarding this submission should be directed to the editor or the submitters:

NAME	AFFILIATION	OGC MEMBER
Stelios Contarinis	HARTIS Integrated Nautical Services	Yes
Sina Taghavikish	Open Geospatial Consortium	Yes
Lucio Colaiacomo	4113 Engineering	Yes
Jérôme Jacovella-St-Louis	Ecere Corporation	Yes
Johann Sorel	Geomatys	Yes
Michael Jendryke	GeoInsight	Yes

NAME	AFFILIATION	OGC MEMBER
João Manuel	GeoInsight	Yes
Alexander Knoch	Geolynx	Yes
Wai Tik Chan	Geolynx	Yes
Francis Charette-Migneault	Computer Research Institute of Montreal (CRIM)	Yes
Dean Hintz	Safe Software	Yes
Dave Campanas	Safe Software	Yes
Kailin Opaleychuk	Safe Software	Yes
Loukas Katikas	HARTIS Integrated Nautical Services	Yes
Vassilis Andronis	HARTIS Integrated Nautical Services	Yes
Nathan McEachen	TerraFrame	Yes
Justin Smethie	TerraFrame	Yes
Nicolas Vila	CS Group	Yes
Craig Coffin	Compusult	Yes
Jim Antonisse	WiSC Enterprises, LLC	Yes
Graham Wilkes	Natural Resources Canada (NRCan)	Yes
Ryan Ahola	Natural Resources Canada (NRCan)	Yes

IV.A. Sponsoring Organizations

Also the following organizations provided the funding, strategic vision, and oversight for this initiative:

ORGANIZATION	ROLE	URL
Natural Resources Canada (NRCan)	Sponsor	natural-resources.canada.ca
European Space Agency (ESA)	Sponsor	esa.int
Centre National d'Études Spatiales (CNES)	Sponsor	cnes.fr
United States Geological Survey (USGS)	Sponsor	usgs.gov



FUTURE OUTLOOK

The AI-DGGS Pilot demonstrated that transitioning from natural-language chatbots to autonomous **Geospatial Analysts** requires moving from passive data retrieval toward **Geospatial Reasoning**. To support this, OGC standards must establish a **Digital Fabric**—a world-centric model ensuring AI logic is anchored in physical reality, multi-grid interoperability, and scientific provenance.

To date OGC has excelled at developing standards to support general data interoperability. In order to better support **information interoperability** particularly in the context of AI reasoning support, more work is needed to develop domain semantics, schemas and ontologies for disaster management. This will enable both human users and AI agents to have a more **consistent context** with which to interrogate and aggregate cumulative disaster impact and management information.

V.A. Strategic Standardization Roadmap

The following six pillars address the technical “frictions” identified during the pilot to bridge the gap between AI reasoning and high-resolution geospatial grids.

Table 1 — Standardization roadmap pillars.

ROADMAP PILLAR(S)	RESULTING CAPABILITY
1. Authoritative DGGRS Register	Mathematically consistent cell alignment across vendors, enabling reliable federated AI reasoning

ROADMAP PILLAR(S)	RESULTING CAPABILITY
2. Discovery-oriented extensions & quantization transparency	Agents can autonomously interpret data meaning, units, aggregation logic, and uncertainty
3. OGC Best Practice for H3	AI-generated analyses align precisely with real-world terrain and infrastructure
4. Temporal regridding parameters	Multi-source datasets can be synchronized in time without bandwidth-heavy preprocessing
5. Auditable Common Operating Picture (COP)	AI decisions are explainable, reproducible, and suitable for high-stakes operational use
6. Advanced analytical extensions	Servers return pre-computed, multi-dimensional indicators enabling scalable AI workflows

For more details, refer to the Clause 6 section.



VALUE PROPOSITION

One Digital Fabric. Instant Answers.

This Report represents the **first comprehensive interoperability experiment** for DGGs and Agentic AI within the OGC framework. It proves that multi-vendor interoperability, scenario simulation, and traceability can coexist within a shared geospatial ecosystem.

By resolving implementation “frictions” and leveraging the **Model Context Protocol (MCP)**, the pilot sets a precedent for AI-driven, standards-based solutions that transform raw data into actionable emergency intelligence.



1

INTRODUCTION

1.1. Operational Challenges in Modern Disaster Management

Flood events and other disasters are becoming increasingly frequent and severe, demanding **robust and timely planning responses**. However, in the wake of a crisis, decision-makers are often “data-rich but information-poor.” This pilot identifies this barrier as the “**Discovery Gap**”—the disconnect between the vast proliferation of Earth Observation (EO) data and the ability of responders to extract instant, actionable answers to plain-language questions.

Despite the abundance of geospatial data, current systems remain fragmented. Data arrives in incompatible formats and different coordinate systems, processed through tools that rarely interoperate. This lack of uniform frameworks restricts **cross-agency coordination** and slows the adoption of intelligent, scenario-based decision support systems.

This Report addresses these challenges by integrating **OGC-compliant DGGs services**, AI modeling pipelines, and explainable analytics. By establishing a shared Common Operating Picture (COP), the pilot provides a unified framework for dynamic disaster risk assessment and scenario communication.

1.1.1. Red River Basin: A Historical Testbed for Flood Modeling

The Red River Basin provides a high-consequence geographic context for testing AI-DGGs-enabled flood risk assessment. Its unique northward flow, flat topography, and climatic conditions make it prone to **recurrent large-scale flooding** that challenges communities on both sides of the Canada–US border.

To anchor the pilot in real-world conditions, the participants simulated an operational experiment modeled after the most notable historical events in the basin.

- **1826 Flood** – The largest known flood on record, inundating vast portions of the basin.
- **1950 Flood** – Displaced over 100,000 people in Winnipeg and prompted major projects like the Red River Floodway.
- **1997 “Flood of the Century”** – Required large-scale emergency measures and record-breaking international cooperation.
- **2011 Flood** – Characterized by widespread overland flooding and prolonged high water levels.

- **2022 Flood** — Severe flooding across the province resulting from the cumulative effects of multiple storms. Also a wide range of high quality data was collected from recently modernized systems. [NASA Report Manitoba Province Operation Report](#)



Figure 1 — An aerial photo showing flooding in the St. Vital area, 1950 (Source: Government of Manitoba).

By integrating DGGS for consistent spatial partitioning and AI for predictive analytics, the pilot simulates scenarios similar to these historic events. The goal is to improve **decision-making capabilities for civil protection agencies** by providing a “Geospatial Brain” that understands the interplay between hydrology, infrastructure, and socio-demographic vulnerability.

1.2. The Case for DGGS: A Digital-First Spatial Foundation

Discrete Global Grid Systems (DGGS) offer more than a new way to partition the Earth; they provide a “**Digital Fabric**” designed for the realities of modern geospatial integration. DGGS supply a globally consistent spatial framework that allows observations, models, and sensor data to be combined at multiple scales without the distortions or “Raster-Vector Divide” inherent in traditional map projections.

DGGS enable **uniform, multiscale indexing** where every location on Earth is represented by a unique cell identifier. In the context of AI, this serves as a **Spatial Tokenizer**. By indexing authoritative data to specific grid cells, Artificial Intelligence can access location-indexed facts rather than relying solely on pre-trained knowledge.

Furthermore, DGGS provides a solution to “**Scale Blindness**”—the tendency for general-purpose AI to request computationally impossible amounts of data (e.g., asking for a 1-meter map of an entire province). By utilizing the grid’s hierarchical structure, the pilot enables an **Orchestration Layer** that provides progressive refinement, ensuring that AI responses are grounded, efficient, and mathematically accurate.

Ultimately, DGGS provide the foundation needed to establish a Common Operating Picture (COP) across diverse stakeholders. This uniform structure supports cross-agency collaboration and standardized reporting, ensuring that the “Server Backbone” and the “Intelligence Frontier” work together as a single, interoperable engine.

1.3. Participants and Deliverables

The AI-DGGS Pilot brought together a diverse group of OGC members and industry partners to develop a distributed environment for disaster management. The following table summarizes the key participants and their primary contributions.

Table 2

DELIVERABLE ID	PARTICIPANT	DESCRIPTION
D001	Hartis	Report and Pilot Website
D001	4113 Engineering	COP Extended Schema and Lead Architecture
D101	Ecere	DGGS API & DGGAL Library Development
D103	Compusult	AI-enabled DGGS Client (UX and Chat-to-Map)
D104	TerraFrame	AI-enabled DGGS Client (Semantic Integration/SKG)
D105	CRIM	DGGS API (Data Integration and Pipelines)
D106	Safe Software	DGGS API (Workflow Orchestration via FME)
D121	GeoLynx	DGGS API (Open-Source Tooling/pydggsapi)
D122	GeoInsight	DGGS API (Performance and Rust implementation)

DELIVERABLE ID	PARTICIPANT	DESCRIPTION
D123	CS Group	AI-enabled DGGS Client (Agentic AI and MCP)
D124	Hartis	AI-enabled DGGS Client (Decision Intelligence/FII)
D125	Geomatys	DGGS API (Multi-Dimensional Experts/Apache SIS)

The project was executed in four primary phases:

1. **Architecture Design:** Defining the interface between DGGS servers and AI agents.
2. **API & AI Integration:** Deploying OGC API – DGGS endpoints and AI-ready clients.
3. **Testing:** Technology Integration Experiments (TIEs) to validate interoperability.
4. **Final Reporting:** Synthesis of findings into this Report.



2

AI-DGGS PILOT ARCHITECTURE (THE SERVER BACKBONE)

AI-DGGS PILOT ARCHITECTURE (THE SERVER BACKBONE)

2.1. The Case for DGGS: A World Model Paradigm Shift

To understand the architecture of this pilot, it is necessary to recognize that DGGS represents a fundamental paradigm shift in geospatial representation. Rather than viewing the Earth as a collection of static layers—an approach rooted in the legacy of paper maps and carried over into modern digital systems—DGGS serves as the most sophisticated **World Model** currently available, transitioning from traditional imagery to a structured global fabric (see Figure 2).

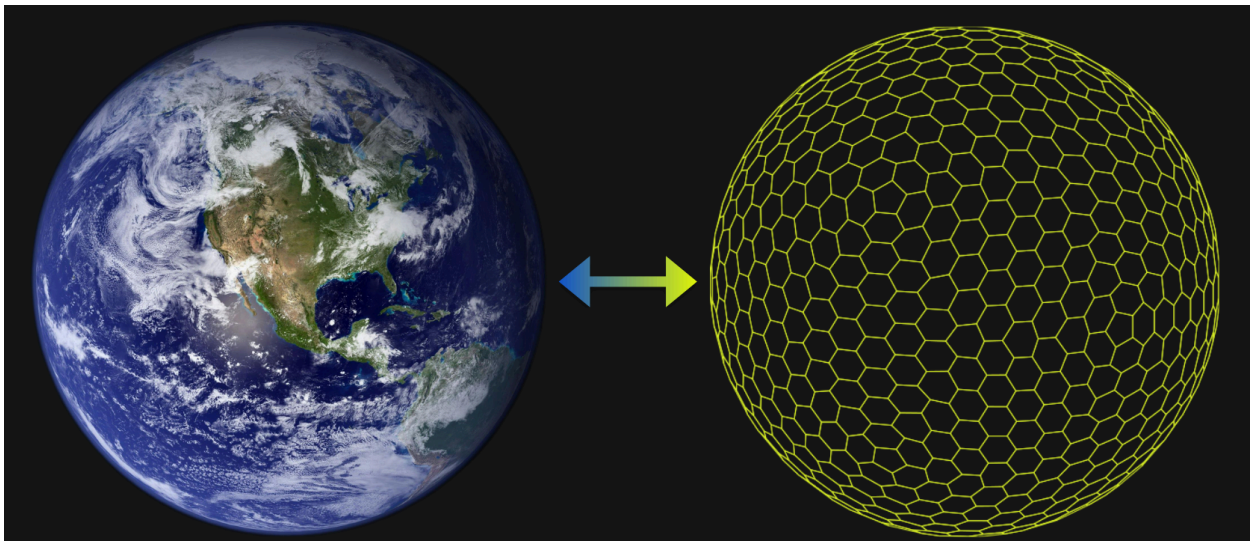


Figure 2 — The DGGS World Model paradigm shift: transitioning from legacy layers to a structured global fabric (courtesy of GeoInsight).

As a world model, DGGS delivers a structure that is consistent, readable, and analyzable by both humans and machines. As detailed by Jendryke in Annex B, DGGS moves away from a purely data-centric approach to provide a system of **containers** that can be filled with diverse data types. By flipping the traditional layer-based approach on its head, DGGS offers several critical advantages for disaster management.

- **Single-Index Lookups:** Drastically reducing the computational cost of spatial joins.
- **Regularity and Global Extent:** Providing a consistent analytical framework from the poles to the equator.

- **Hierarchy of Scales:** Allowing for the “Progressive Refinement” required to solve Scale Blindness.
- **Analysis-Ready Integration:** Ensuring that diverse datasets (radar, terrain, population) are inherently interoperable.

The convergence of DGGs and multi-source data results in a “digital fingerprint” for every zone on Earth. Within the context of this pilot, these are referred to as **Spatial Tokens**. As illustrated in Figure B.1, these tokens—machine-readable packages of localized facts—serve as the primary ingredient for the **Intelligence Frontier**, enabling GeoAI to reason across multiple variables within a unified spatial context.

2.2. DGGs-centric System Design

The pilot architecture follows a distributed, modular design where data ingestion, spatial quantization, and analytical modeling are decoupled from user interaction. This design ensures that heavy geospatial processing occurs close to the data source (the Backbone), while AI agents focus on reasoning and orchestration.

The system flow is categorized into three primary phases.

1. **Ingestion:** Raw Earth Observation (EO), external model outputs and thematic data are retrieved and quantized into DGGs zones.
2. **Modeling:** Analytical engines and ML pipelines derive indicators (e.g., flood risk scores) at the cell level.
3. **Interaction:** AI clients query standardized API endpoints to retrieve grounded facts for natural language responses.

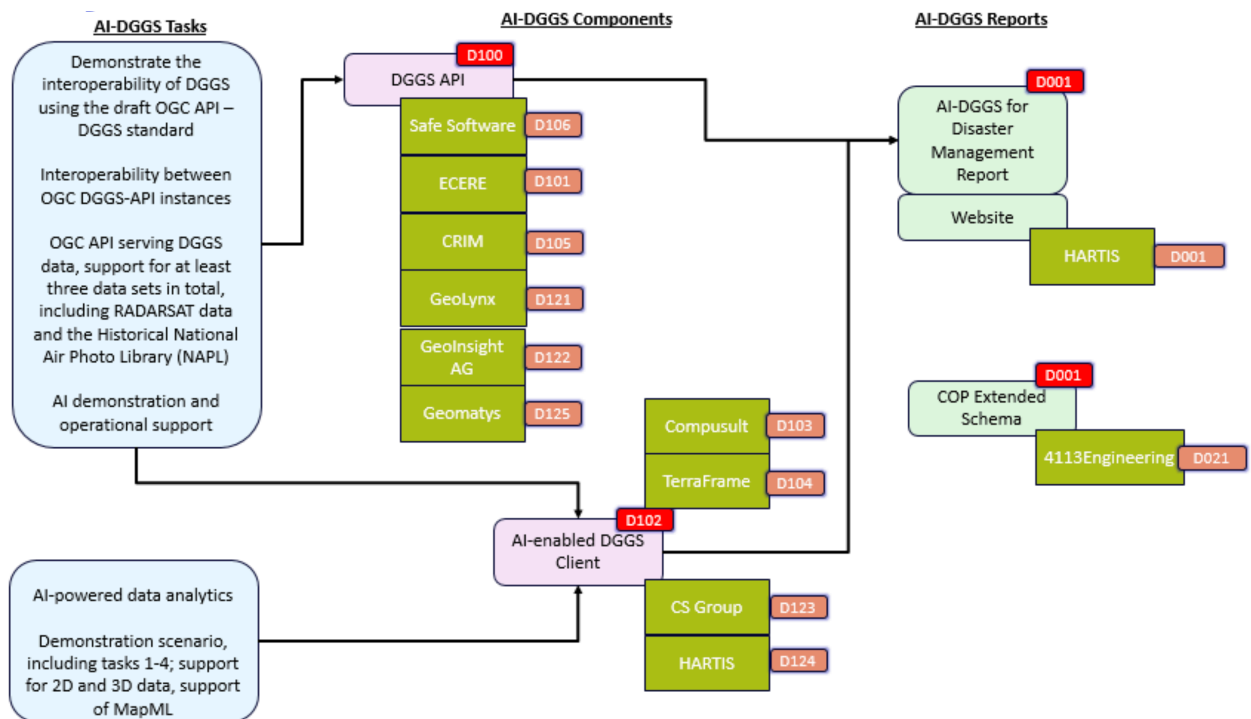


Figure 3 – AI-DGGS Pilot Architecture: Logic and Components

2.3. The Foundational Stack (Server Backbone)

The architecture is built upon a set of open-source libraries that provide low-level DGGS abstractions and interoperability frameworks.

- **Apache-SIS:** A Java library extending CRS (ISO-19111) and Gazetteer (ISO-19112) standards to support grid-native operations.
- **DGGAL (Discrete Global Grid Abstraction Library):** The core math engine developed by Ecere. It provides a unified interface for several DGGRS (ISEA/IVEA/RTEA 3H/7H/4R/9R, ®HEALPix, GNOSISGlobalGrid) and enables cross-platform validation.
- **GeoPlegma:** A Rust-based collaboration hub designed to reduce fragmentation between grid implementations like DGGRID, H3, and S2.
- **pydggsapi:** A Python FastAPI implementation of the OGC API – DGGS standard, enabling programmatic access to metadata and quantized coverages.

For a detailed breakdown of library features, git repositories, and supported bindings, see Annex D.

2.4. Functional Analysis Capabilities

DGGS servers in the pilot expose standardized capabilities aligned with **OGC API – DGGS**. These operations allow AI agents to navigate the grid and retrieve evidence.

- **Zone Discovery:** Identifying which “spatial tokens” intersect a geographic area.
- **Attribute Filtering:** Using **CQL2** to isolate zones meeting specific criteria (e.g., elevation > 200m).
- **Fact Retrieval:** Fetching data values for specific zones to ground AI reasoning.
- **Virtual Collections:** Using **OGC API – Processes** to generate on-the-fly indicators.
- **Federated Queries:** Enabling remote joins where one server acts as a client to another.

For specific HTTP request/response examples and API endpoint walkthroughs, see Annex E.

2.5. The Spatial Tokenizer: Data Models and Encodings

To facilitate AI ingestion, geography must be converted into discrete, machine-readable indices. The pilot explored three primary encoding styles for Spatial Tokens.

- **DGGS-JSON:** Optimized for rapid exchange and low-bandwidth transmission. It is the preferred format for real-time mobile responders receiving summarized risk alerts.
- **DGGS-JSON-FG:** A “feature-grid” style aligning with the **OGC API – Features** model. It supports richer metadata and properties, which is essential for **AI Explainability** and verifying data lineage.
- **Cloud-Native Grid (Zarr):** For high-volume and multi-dimensional data (e.g., time-series rainfall), the pilot extended **GeoZarr** and **CF-Conventions**. By integrating DGGRS attributes into Zarr arrays, the architecture supports high-performance “chunked” access to complex grid data.

The proposed Zarr metadata attribute schema and draft specifications are documented in Annex F.

2.6. The Integration Glue: Intelligent COP Schema

The **Common Operating Picture (COP)** serves as the “Integration Glue” of the pilot, evolving the OGC OWS Context Standard (12-080r2) into a machine-readable JSON-LD framework that aligns situational awareness for both human operators and AI agents. By replacing traditional coordinate bounding boxes with **DGGS Zone Identifiers (ZIRS)**, the schema allows Areas of Interest (AOI) to be defined as discrete sets of grid cells, enabling spatially consistent, multi-resolution views of the same geographic extent regardless of the underlying coordinate reference system.

The inclusion of **Identity, Provenance, and Trust (IPT)** metadata, contributed by **4113 Engineering**, ensures operational reliability by embedding verifiable digital signatures, security classifications, and chain-of-custody information directly into every data layer. This structured context acts as the primary knowledge base for **Retrieval-Augmented Generation (RAG)**, providing the “Spatial Conscience” required for AI agents to generate natural-language summaries that are strictly anchored in “Grid-Truth” and authoritative sensor data.

To maintain operational continuity, a **QGIS COP Plugin** was developed that serves as a bridge from legacy workflows, allowing desktop GIS analysts to export their layouts directly as AI-ready COP packages complete with DGGS grid references and IPT metadata. For the detailed technical specification of the COP schema extensions, implementation notes on the QGIS Plugin, and example JSON-LD instances, refer to Annex X.



3

AI ORCHESTRATION (THE INTELLIGENCE FRONTIER)

AI ORCHESTRATION (THE INTELLIGENCE FRONTIER)

3.1. Overview: Giving AI a “Geospatial Brain”

The core objective of the AI-DGGS Pilot was to demonstrate how Artificial Intelligence can interpret a natural hazard crisis as easily as it interprets text. By using DGGS as a common “Digital Fabric,” the pilot moved beyond simple natural-language summarization toward **Geospatial Reasoning**—the ability for an AI system to independently devise an analytical plan, execute multi-step workflows across federated servers, and interpret results as actionable intelligence.

A key takeaway from the pilot project is that DGGS services and Large Language Models (LLMs) are not inherently compatible. DGGS zone identifiers and index-based query responses are optimized for machine efficiency, but they are not directly interpretable or analyzable by LLMs in a meaningful way. As a result, an orchestration layer is required to mediate both user requests and system responses between the LLM and DGGS services. This mediation was essential for enabling multi-step reasoning, cross-layer analysis, and explainable outputs, effectively bridging the gap between spatial indexing systems and language-based reasoning systems.

To achieve this, the pilot established an **Orchestration Layer** that acts as the “Spatial Conscience” of the Large Language Model (LLM), reconciling high-level human intent with the physical realities and computational limits of geospatial data. Beyond syntactic translation, the orchestration layer also provides semantic grounding by mapping DGGS zones to real-world geographic features, administrative units, and domain concepts. This enables the LLM to reason over spatial results in context (e.g., populations, infrastructure, or risk zones) rather than opaque index values, which is critical for explainable and decision-support-oriented applications.

In practical terms, this orchestration layer is where the pilot’s key **AI Frontier Enablers** converge: standardized actionability, trustworthy context, scale-aware execution, and machine-interpretable semantics.

3.2. Model Context Protocol (MCP): The “Action Protocol”

A critical design choice in the pilot was the adoption of the **Model Context Protocol (MCP)**. This protocol standardizes how AI agents interact with OGC API servers, transforming static data endpoints into a **Self-Describing Infrastructure**.

Enabler: Actionability / Tool-use interoperability. MCP turns APIs into callable tools the agent can safely and consistently execute.

- **Standardized Discovery:** Instead of hard-coding DGGs logic into the AI, MCP allows agents to autonomously “read” the capabilities of a server (e.g., supported grids, available collections, and query parameters).
- **Autonomous Planning:** By treating API endpoints as executable tools, AI agents can plan complex disaster workflows—identifying which collections to query and which tools to chain—without manual human assistance.
- **Multi-Vendor Chaining:** The pilot demonstrated AI agents (specifically from **CS Group**) automatically invoking tools across independent servers (e.g., querying flood extents from **Safe Software** and population data from **Geomatys**) to answer a single user request.

3.3. The Metadata Discovery Gap

While the Model Context Protocol (MCP) successfully standardizes the “Action” layer, the pilot identified a significant **Semantic Gap** in current OGC API – DGGs implementations.

Enabler: AI-ready semantics. Agentic autonomy depends not only on calling endpoints, but on understanding what the fields **mean** and how they were derived.

As noted by CS Group (D123), the lack of descriptive metadata regarding field types, unit semantics, and quantization methods represents a primary barrier to agentic autonomy.

Without “AI-Ready” metadata, an agent can discover that a collection exists but cannot independently determine if the data (e.g., a “population” field) represents an absolute count, a density index, or a temporal average. To achieve true “Generic Discovery,” infrastructure must move toward a **Self-Describing** model where the meaning and quality of data are machine-readable.

3.4. Retrieval-Augmented Generation (RAG): The “Context Engine”

To prevent “AI Hallucinations,” the pilot implemented **Retrieval-Augmented Generation (RAG)** anchored in “Grid-Truth.”

Enabler: Trustworthy context / auditable grounding. DGGs provides stable identifiers (“spatial tokens”) that constrain generation to verifiable facts.

In this framework, the DGGs grid serves as the **Spatial Tokenizer**, providing a stable set of identifiers that the AI uses to access facts.

- **Descriptive Grounding:** Rather than guessing at coordinates or risk levels, the AI “reads” specific DGGs cell values directly. This ensures that natural-language summaries are factually accurate and auditable.
- **Grounding in Physical Reality:** By anchoring the narrative in actual sensor data (e.g., RCM radar signals or water level indicators), the AI can explain *why* a specific zone is hazardous (e.g., “High risk due to combined snowmelt and low-lying topography in Cell ID: N3456”).
- **Algorithmic Transparency and Bias:** To avoid “Inductive Bias” in AI reasoning, the pilot identified a requirement for metadata regarding the quantization process. As noted by Geolynx (Wai Tik Chan), agents must be able to identify if methods such as **K-Nearest Neighbor (K-NN)** or **Bilinear Interpolation** were used to regrid data into the DGGs. Without this, the AI cannot account for mathematical artifacts introduced during the data’s transition to the grid.
- **Semantic Enrichment via STAC Extensions:** The pilot advocated for a “batteries-included” metadata approach for /collections. CRIM (Francis Charette Migneault) proposed the embedding of specific **STAC Extensions** (Disaster, ML Models, and Processing) directly into the DGGs metadata. This allows an AI agent to verify not only the spatial extent but also the processing lineage and intended model compatibility of the data before execution.

3.5. Curing “Scale Blindness” through Progressive Refinement

One of the most significant findings of the pilot was the identification of “**Scale Blindness**” in general-purpose AI. Without a spatial intermediary, an AI might request a 1-meter resolution map of an entire country, inadvertently launching a “Resource Depletion Attack” on emergency servers.

Enabler: Scale-aware execution / safe-by-design efficiency. DGGs hierarchy allows the agent to reason across levels and “pay for resolution only when needed.”

The **DGGs Hierarchy** provided the solution.

1. **Coarse Discovery:** The AI first queries coarser grid levels (e.g., Level 5) to identify broad areas of interest.
2. **Targeted Refinement:** The orchestrator then guides the AI to drill down into high-resolution cells (e.g., Level 10) only where significant risk signals are detected.

3. **Efficiency:** This iterative process ensures that high-resolution data is only transmitted when necessary, protecting the stability of emergency communications.

3.6. Semantic Reasoning and Cascading Risk Analysis

By integrating DGGs with **Spatial Knowledge Graphs (SKGs)**, specifically demonstrated by TerraFrame, the pilot enabled AI to move beyond simple keyword matching to understand real-world relationships between objects.

Enabler: Semantic reasoning / impact propagation. SKGs let the agent infer dependencies and second-order effects, not just overlay layers.

- **Network Connectivity:** While the DGGs grid identifies *where* the water is, the SKG identifies *what* is affected.
- **Cascading Impacts:** The AI gained the ability to perform systemic reasoning—for example, understanding that a flooded electrical substation in “Cell A” will knock out power to a hospital located in “Cell B,” even if Cell B itself remains dry.
- **Operational Evidence:** This results in a comprehensive impact assessment that can prioritize infrastructure repair based on the cumulative population affected across the network.

3.7. Participant Implementations (Intelligence Frontier)

The following contributors developed the agentic clients that define the Intelligence Frontier, each operationalizing one or more of the pilot’s core **enablers** (actionability, grounding, scale-awareness, semantics, and cascading reasoning).

- **Compusult (D103):** Developed the “Chat-to-Map” user experience, using React-based clients to render DGGs zones and data layers from natural-language prompts. *Enabler: Human-to-geo translation into actionable views.*
- **TerraFrame (D104):** Integrated DGGs with SKGs and GeoSPARQL to automate semantic impact analysis on critical infrastructure. *Enabler: Semantic reasoning and cascading impact propagation.*
- **CS Group (D123):** Developed the backend MCP server and orchestration logic to enable autonomous tool selection and multi-server chaining. *Enabler: Action protocol + multi-vendor tool chaining.*

- **Hartis (D124):** Engineered the **Flood Impact Index (FII)** model, using RAG to ground AI analysis in fused multi-source grid data. *Enabler: Grid-truth grounding for auditable risk intelligence.*



4

DGGS FRICTIONS (TECHNICAL IMPLEMENTATION CHALLENGES)

DGGS FRICTIONS (TECHNICAL IMPLEMENTATION CHALLENGES)

4.1. Overview: Reconciling Theory with Reality

The OGC DGGS Standard provides a powerful blueprint for global spatial analytics; however, operationalizing this framework in high-stakes disaster scenarios reveals critical “Implementation Gaps.” The pilot identified six technical frictions where theoretical standards meet the practical constraints of bandwidth, mathematical precision, and AI reasoning. By documenting and resolving these frictions, the pilot establishes a path toward **Geospatial Grounding**: ensuring that an autonomous agent receives a mathematically identical and scientifically traceable response from any server in a federated network, regardless of the underlying implementation.

4.2. Friction 1: Geometric Alignment (Spherical vs. Ellipsoidal Models)

The Problem: A fundamental friction exists between widely used grid systems and standard geospatial reference frames. Grids like **H3** were originally designed using **authalic spherical models** to simplify calculations (see also Annex Q), whereas modern remote sensing and positioning infrastructure rely on the **WGS84 ellipsoidal model**. When mapping high-resolution data onto these grids without proper correction, small mathematical deviations occur. In a disaster scenario, these errors can misalign a flood risk zone by several meters, potentially mapping a life-threatening hazard to the wrong physical property or infrastructure asset.

The Integration Fix:

- **Coordinate Mapping:** The pilot established that WGS84 geodetic latitude must be mathematically converted to authalic latitude when interfacing with H3-based systems to ensure location accuracy.
- **Boundary Refinement:** To maintain topological integrity during coordinate transformations, zone boundaries were refined with additional intermediate points along the authalic sphere, preventing “creeping” errors at the edges of grid cells.

Outcome: This mathematical bridge ensures that AI-generated risk maps remain spatially consistent with ellipsoidal geodetic reference frames used by modern positioning and remote sensing systems. By carefully reconciling spherical grid abstractions with ellipsoid-based

coordinate systems, the pilot enabled accurate overlay of DGGS-derived analytics with legacy GIS infrastructure and high-precision GNSS observations.

4.3. Friction 2: Sparse Data and Storage Bloat (The “Data Hole” Problem)

The Problem: High-resolution satellite data, such as RCM radar imagery, is frequently “sparse,” containing large empty gaps where the sensor did not pass or where no significant signal was detected. Traditional web formats often treat these empty spaces as “heavy” data, resulting in bloated, uncompressed files. During an active crisis, these oversized files lead to bandwidth saturation and slow map response times, making it difficult for responders to access critical information on mobile devices in areas with degraded network connectivity.

The Performance Fix:

- **Optimized Encodings:** The pilot utilized **DGGS-UBJSON(-FG)** and **GeoParquet** to represent only valid data cells, effectively “ignoring” the empty space in the file structure.
- **High-Ratio Compression:** Applying high-ratio compression (such as GZIP) to “collapse” the repetitive null values inherent in sparse datasets, the pilot demonstrated that storage requirements could be reduced by orders of magnitude without losing signal integrity.

Outcome: These optimizations allow for near-instant delivery of flood-risk summaries to the field. Eliminating the “sparse data penalty,” the pilot proved that DGGS can efficiently handle massive satellite archives while maintaining the high-speed transmission required for real-time emergency response.

4.4. Friction 3: The Stacking Paradox (Non-Nested Hierarchies)

The Problem: In **Aperture 7** grids, a parent hexagon does not perfectly contain its seven “logical” children. This lack of perfect nesting creates a “Stacking Paradox” where, at every level of refinement, some child cells extend outside the parent boundary while neighboring children from adjacent parents bleed into it. If a system only counts the logical 7 children during data aggregation, it inadvertently introduces a significant statistical error by ignoring the actual spatial overlap of the grid.

The Accuracy Fix:

- **Quantifying Areal Error:** The pilot identified and measured that ignoring these overlaps results in a persistent error where **~6.52% of the analysis area** is replaced with data from the wrong spatial location.
- **Overlap-Aware Aggregation:** Participants (notably Ecere and Geomatys) implemented topological tools and scanline-based ordering to account for the actual **13 overlapping sub-zones** that contribute to a parent cell's value.

Outcome: Accounting for the geometric reality of grid overlaps rather than just the logical hierarchy, the pilot ensured that risk statistics and population counts are calculated over the correct area. This eliminates the 6.5% error margin that would otherwise degrade the reliability of multi-scale modeling.

4.5. Friction 4: Cross-Platform Integrity (Logic and Logic Drift)

The Problem: For disaster management tools to be effective, they must produce identical results across a federated ecosystem—from high-performance cloud servers to a responder's field tablet. Historically, grid mathematics and Zone ID generation have been implemented inconsistently across different programming languages (e.g., Python, Java, and C++). These subtle differences in implementation lead to “logic drift,” where the same coordinate might resolve to two different grid cells depending on the software used, destroying the trust required for a Common Operating Picture.

The Interoperability Fix:

- **DGGAL Validation:** The pilot successfully validated and utilized the **Discrete Global Grid Abstraction Library (DGGAL)** as a common mathematical foundation.
- **Universal Implementation:** Using DGGAL, the pilot demonstrated consistent results across **Python, Java, Rust, and WebAssembly (WASM)**, ensuring that the grid logic remains stable regardless of the deployment environment.

Outcome: Responders and decision-makers can trust that the high-speed grid data visualized in a web browser is mathematically identical to the data analyzed on a central server. This cross-platform integrity is essential for maintaining a single, auditable version of the truth during a distributed disaster response.

4.6. Friction 5: Spatial Temporal Resolution and Range Queries

The Problem: While DGGs excels as a global multi-resolution index, localized disaster management requires sub-meter spatial detail and high-cadence temporal modeling. Moving from modest resolutions, such as 100-meter cells, to sub-meter modeling creates an “exponential explosion” in the number of grid zones. During pilot testing, it was found that a single request for a hexagonal parent zone at a relative depth of 8 can return approximately 5.7 million sub-zones. Disseminating raw data at this level of detail through traditional API responses is practically impossible, as payloads can exceed 100MB per tile, leading to bandwidth saturation and system timeouts during active crises. Another aspect is temporal resolution. Current DGGs implementations often lack native support for 4D range queries (XYZT). While CQL2 can filter by time attributes, parsing these irregular time-steps is computationally expensive when an AI agent needs to query specific slices of a river’s crest across thousands of increments, such as days, hours, or minutes.

The Scalability Fix:

- **Linkages to Cloud Optimized Data Stores:** The pilot demonstrated a “Zonal Linkage” strategy where DGGs serves as a **Unified Discovery Index**. Coarse grid cells provide high-level summary statistics—such as maximum and mean flood levels—and act as **Intelligent Pointers**. The API response provides a direct metadata link to the spatial and temporal range associated with that zone stored in cloud-native formats such as Cloud Optimized GeoTIFF (COG), Zarr, or GeoParquet. Clients only read the specific high-resolution data they need from these optimized stores, offloading the heavy lifting from the grid API.
- **Updates to DGGs Standard for Temporal Support:** For improved spatio-temporal filtering, the pilot proposed enhancements to the DGGs standard to include better native support for time-range queries. This requires service providers to optimize for large volumes of time-series data cubes. Rapid retrieval is further enabled by hosting processing logic close to the cloud-native storage.

Outcome: Given these adaptations, DGGs can be used to support the high spatial and temporal resolutions needed for localized flood risk assessment. By serving as an indexing layer for 4D data cubes, DGGs enables AI agents to discover and reason across high-resolution datasets without being restricted by API transport limits.

4.7. Friction 6: The Interpretation Gap (Contextual Blindness)

The Problem: A significant hurdle identified during the pilot was the “Semantic Gap” between raw grid values and AI reasoning. While AI agents are proficient at retrieving numerical cell values, they suffer from “Contextual Blindness” regarding the underlying mathematical and scientific provenance of those numbers. For example, if an agent retrieves a value of “12.5,” it cannot natively distinguish if that represents an area-weighted **Mean** (flood intensity) or a cumulative **Sum** (total population). Furthermore, slight variations in how different quantization methods—such as K-Nearest Neighbor versus bilinear interpolation—associate data to zones can lead to inconsistent results. Without explicit machine-readable metadata regarding units and aggregation methods, AI agents risk applying incorrect scientific models, leading to “geospatial hallucinations” and unreliable insights.

The Provenance Fix:

- **Self-Describing Grids:** The pilot addressed this by prototyping **Self-Describing Grids**. This implementation embeds **STAC (SpatioTemporal Asset Catalog) extensions** and **Identity, Provenance, and Trust (IPT)** metadata directly within the DGGs API response. This provides the necessary machine-readable context for an agent to autonomously verify unit semantics (e.g., meters vs. feet) and aggregation math (e.g., instantaneous readings vs. temporal averages).
- **Quantization Provenance:** Including the specific mathematical steps used to derive a value within the cell’s metadata, the pilot enabled AI agents to verify the lineage and integrity of the data before performing multi-source fusion. This transforms the DGGs from a simple data container into a grounded knowledge engine.

Outcome: Making the grid values self-describing, the pilot ensured that AI-generated disaster insights are scientifically accurate and traceable. This metadata-heavy approach allows autonomous agents to safely perform multi-source integration, ensuring that all data used in a Common Operating Picture is “fit-for-purpose” and auditable.

4.8. Summary of Implementation Challenges

The following table maps the identified frictions to the specific technical strategies deployed to resolve them:

Table 3

FRICION	CATEGORY	RESOLUTION STRATEGY
Mathematical Model Mismatch (Geometric Alignment)	Georeferencing Precision	Standardized the conversion of WGS84 Geodetic coordinates to Authalic Spherical values for H3, and implemented boundary densification to maintain topological integrity.
Sparse Data Performance Penalty (The Data Hole Problem)	Bandwidth & Storage	Replaced bloated, uncompressed web formats with optimized UBJSON-FG and GeoParquet encodings that use high-ratio compression to “collapse” empty sensor gaps.
Hierarchical Misalignment (The Stacking Paradox)	Statistical Accuracy	Moved beyond “logical” indexing to implement intersection-based sub-zone discovery (13 overlapping zones vs. the logical 7), eliminating the ~6.52% areal error margin.
Cross-Language Logic Drift (Cross-Platform Integrity)	Runtime Interoperability	Utilized the DGGAL abstraction library to ensure identical Zone ID generation and mathematical results across Python, Java, Rust, and WebAssembly (WASM) environments.
Spatio-Temporal Scale Barrier (The 4D Resolution Wall)	Architectural Scalability	Demonstrated a Zonal Linkage strategy where DGGs cells serve as intelligent pointers to high-resolution, high-cadence temporal payloads in COG, Zarr, or GeoParquet stores.
Semantic Context Blindness (The Interpretation Gap)	Data Provenance & Trust	Prototyped Self-Describing Grids by embedding STAC and Identity, Provenance, and Trust (IPT) metadata to provide AI agents with machine-readable unit and aggregation math lineage.



5

DGGS DEMONSTRATORS AND WEB RESOURCES

DGGS DEMONSTRATORS AND WEB RESOURCES

5.1. Overview of the Distributed Pilot Environment

The AI-DGGS Pilot established a live, federated environment connecting six data servers (The Backbone) and four AI-enabled clients (The Frontier). This section provides a summary of the implementation roles and outcomes for each participant. For detailed technical configurations and endpoint specifications, refer to the respective annexes of each participant.

5.2. The AI-DGGS Pilot Hub

The central landing page provides a narrative walkthrough of the pilot's mission, the "Discovery Gap" challenge, and the final results.

- **Main Site:** [AI-DGGS Pilot Hub](#)
- **The Mission:** [The Mission Context](#)
- **The Challenge:** [Identifying the Discovery Gap](#)
- **The Technical Architecture:** [Interactive System Diagram](#)

5.3. The Server Backbone (DGGS APIs)

The following implementations provided the OGC API – DGGS infrastructure and raw data streams required for the flood risk scenarios.

5.3.1. CRIM – The Data Integrators (D105)

CRIM deployed a Python-based implementation using `pydggsapi`. Their focus was the high-volume ingestion of RCM Radar imagery and socio-demographic datasets into the **IGEO7** and **H3** grids.

- **Key Outcome:** Validated the use of GeoParquet for efficient storage of sparse spatiotemporal grid data and resolved geometric offsets between H3 and geodetic WGS84 coordinates.
- **Website Link:** [CRIM Website Detail](#)
- **Full Technical Detail:** See Annex G.

The [Computer Research Institute of Montreal \(CRIM\)](#) has developed its DGGs API implementation based on the open-source `pydggsapi` library offering a REST web interface through FastAPI. This API leverages the Python wrapper `dggrid4py` to perform DGGs operations with the C++ `DGGRID` CLI application. It also uses some `dgga1` functionalities to support additional DGGs.

The work has been contributed in collaboration with [Geolynx](#) and [Landscape Geoinformatics Lab](#) to enhance existing capabilities of `pydggsapi`. Amongst these improvements, support of DGGs-UBJSON responses, encoding compression for smaller responses, implementation of advanced queryable property logic, and handling of date-time-aware DGGs zone data retrieval were added. A particular focus has been placed on ensuring spatiotemporal DGGs support to explore the DGGs API's capabilities and operational challenges involved for long-term monitoring of Earth Observation products, Environment and Climate Change management. The implementation has also been containerized with Docker to allow integration within the [bird-house](#) ecosystem, which [CRIM](#) and other Canadian climate centers employ to deploy their geospatial web services deployment, such as on the [Hirondelle](#) server. This integration allows the DGGs API to be easily deployed alongside other [OWS](#) and [OGC API services](#), as well as to share it amongst other [DACCS Marble](#) nodes, therefore facilitating interoperability testing with other systems before wider adoption to production environments.

See Annex G for more details regarding the CRIM DGGs API implementation architecture and test experiments. See the [Geolynx DGGs API](#) implementation for more granular details about `pydggsapi` configuration.

Although the `pydggsapi` library supports multiple data sources and DGGs, CRIM's implementation focused primarily on the integration and quantization of [RADARSAT Constellation Mission – Analysis-Ready Data \(RCM-ARD\)](#) satellite imagery provided through [NRCan's EODMS STAC API](#). The IGEO7 (or ISEA7H with Z7 zone indexing) DGGs was selected to quantize the zones of the RCM-ARD scenes over the entirety of the available temporal extent (since 2025-04-01), over the Manitoba, Winnipeg region of Canada selected as the pilot area of interest. Zone statistics have been extracted from the RCM-ARD and quantized for IGEO7 resolutions 0 to 10. All zones and resolutions are then aggregated into a GeoParquet format to interface with the *collection provider* configuration of `pydggsapi`. An H3 Canada Population dataset from resolution 0 to 8 was also added, which contains population counts over Canada,

including the Manitoba region of interest. The Canada Population collection was added with non-temporal definitions (only spatial DGGs zones) to evaluate DGGs API operations across different dimensions and interpolation combinations (with and without time) when combined with the RCM-ARD collection. Different DGGs were employed between the collections (H3 and IGEO7) to evaluate challenges that can be encountered when dealing with different data quantization and aggregation strategies. The entire data processing pipeline to prepare the zone data is shared openly in a Jupyter Notebook found under the [CRIM OGC DGGs Pilot GitHub Repository](#). For more information regarding `pydggapi` capabilities, readers are encouraged to refer to the GeoLynx DGGs API details using the shared library.

The CRIM DGGs API implementation exposes [OGC API – DGGs endpoints](#) to perform zone queries, zone data retrieval, property subsetting, and CQL2 filtering on the RCM-ARD and Canada Population collections. The properties can be filtered based on desired zonal statistics and temporal extent (instances, ranges and both simultaneously). The implementation also supports zone-depth queries and inter-collection aggregations using the `/dgg` endpoint to retrieve multiple subzones data sources at once. It can return responses in multiple formats including DGGs-JSON, DGGs-UBJSON, GeoJSON FeatureCollection, and Zarr, with or without datetime dimension as applicable.

5.3.2. Ecere – The Standards Architects (D101)

Ecere provided the **DGGAL library** as a mathematical foundation for the pilot, and deployed an updated version of the GNOSIS Map Server implementing the OGC API – DGGs Standard.

- **Key Outcome:** Enhanced the **DGGAL library** with support for new DGGs (including Z7 index support) and new programming languages such as JavaScript
- **Website Link:** [Ecere Website Detail](#)
- **Full Technical Detail:** See Annex H.

Ecere served as the technical lead for grid geometry, deploying the **GNOSIS Map Server** to provide a robust implementation of the **OGC API – DGGs**, in addition to several other OGC API Standards. Their primary objective was to ensure a mathematically rigorous foundation for the pilot by supporting the widest variety of grid types, adding support for new grids to DGGAL including icosahedral equal-area (**ISEA/IVEA/RTEA 4H**) aperture 4 rhombic, aperture 7 hexagonal (**7H**) (with the Z7 hierarchical indexing as an option for compatibility with IGEO7 and DGGRID), and **HEALPix**. This complemented existing support for **rHEALPix**, **ISEA/IVEA/RTEA** aperture 9 rhombic (**9R**), aperture 3 hexagonal (**3H**), and the **GNOSIS Global Grid**. Ecere was also instrumental in clarifying how reconcile challenges and limitation of the widely used H3 grid and the reference H3 library with the implementation of a proper DGGs conforming to OGC API – DGGs (see Annex Q).

Ecere's software solutions, including both the GNOSIS Map Server and the **Discrete Global Grid Abstraction Library (DGGAL)**, are built upon the **eC programming language** enabling high-performance, native grid calculations. Usage of the DGGAL library from a wide variety of development and runtime environments is facilitated by readily available bindings for Java (thanks to contributions from Johann Sorel/Geomatys in this pilot), C, C++, Python, Rust and JavaScript (through WebAssembly). The DGGAL Python package (`pip install dggal`) with

minimal dependencies in particular greatly facilitated the adoption of the library. Several of these bindings were validated during this pilot as part of the implementations of both servers and clients by the other participants. The GNOSIS Map Server supports advanced **on-the-fly re-quantization** and resampling, allowing users to query data in any DGGRS while the source data is stored in a data store quantized to the GNOSIS Global Grid, which is well suited for this purpose as it does not involve an extra de-projection step. Key features of the Ecere's OGC API – DGGS implementation include **CQL2 filtering** for both Zone Queries and Zone Data Retrieval, as well as the ability for clients to set up “virtual collections” (e.g., calculating a Flood Impact Index) through the Collection Output capability of **OGC API – Processes** version 2.0 and then request data for specific zones of interest which automatically triggers on-demand processing specifically for those requested zones. The GNOSIS server implementation achieved high-speed response times for data packets at different refinement levels using **DGGS-JSON**.

Ecere also explored other DGGS research topics as part the new open-source DGGAL “High Vibes” Python project, which includes a second OGC API – DGGS server implementation by Ecere which is fully open-source, tools to import and export both raster and vector data from GeoTIFF and GeoJSON to highly scalable DGGS-quantized data stores, and an OGC API – DGGS client cloning data from a remote API to such local DGGS data store. As part of this work, Ecere implemented one of the first DGGS-JSON-FG implementation, quantizing vector data to DGGRS sub-zones without rasterizing the data, but clipping it to root zones. This concept is analogous to vector tiles but not limited to rectilinear shapes, allowing for example hexagonal data packets (“HexVecTiles”). Ecere also discovered a combinatorial solution to exact spatial overlap error (~6.52%) resulting from aggregating sub-zone based solely on the logical hierarchical indexing (7 primary child zones) for Aperture 7 grids such as H3 or IVEA7H. Ecere also conceptualized a DGGS Vision Transformer (DGGS ViT) based on root zone data packets and sub-zones analogous to traditional ViT used in geospatial foundational models, and initiated the development of a proof of concept implementation.

Next steps include refining the **Standardization Roadmap** for an authoritative DGGRS register, evolving the pilot's H3 integration findings into a formal **OGC Best Practice** document, support for additional grids to DGGAL such as A5, IVEA4H, HEALPix_NUNIQ and H3, continued refinement of a DGGS-JSON-FG vector pipeline, improvements to the DGGAL High Vibes tools and continued research on the topic of DGGS ViTs and their integration in AI-DGGS workflows.

5.3.3. GeoInsight – The Performance Engineers (D122)

GeoInsight provided a high-concurrency, Rust-based implementation focused on cloud-native efficiency and the transformation of “Big Geodata” into machine-readable **Spatial Tokens**.

- **Key Outcome:** Championed the “Cloud Native Pointer” architecture, utilizing DuckDB and S3 for near-instant data retrieval, proving that massive datasets can be served without the overhead of upfront transformation storage.
- **Website Link:** [GeoInsight Website Detail](#)
- **Full Technical Detail:** See Annex I.

GeoInsight's mission within the pilot was to unlock global-scale GeoAI by turning fragmented geospatial information into harmonized, AI-ready “digital fingerprints.” Their implementation

developed a **Spatial Tokenizer** designed to produce analysis-ready data by partitioning the Earth into stable, hierarchical cells that act as spatial keys. By aggregating heterogeneous sources—such as satellite imagery, climate indicators, and infrastructure data—into these zones, the system transforms continuous geography into discrete **Spatial Tokens** that AI models can analyze more efficiently than traditional coordinates.

The implementation follows a grid-agnostic, “Any Data, Any Grid” philosophy. Built on a high-performance **Rust** backend, the server uses the **GeoPlegma** library for grid generation and **DuckDB** for managing metadata and lightweight analytical queries. This hybrid approach allows the API to federate data from diverse sources while maintaining consistent access through standardized OGC API – DGGS endpoints. During the pilot, Geolnsight demonstrated how to serve massive datasets without bloating the grid by pointing to optimized storage formats like **Parquet** and **S3**, ensuring that responders can access grounded facts in active crisis environments.

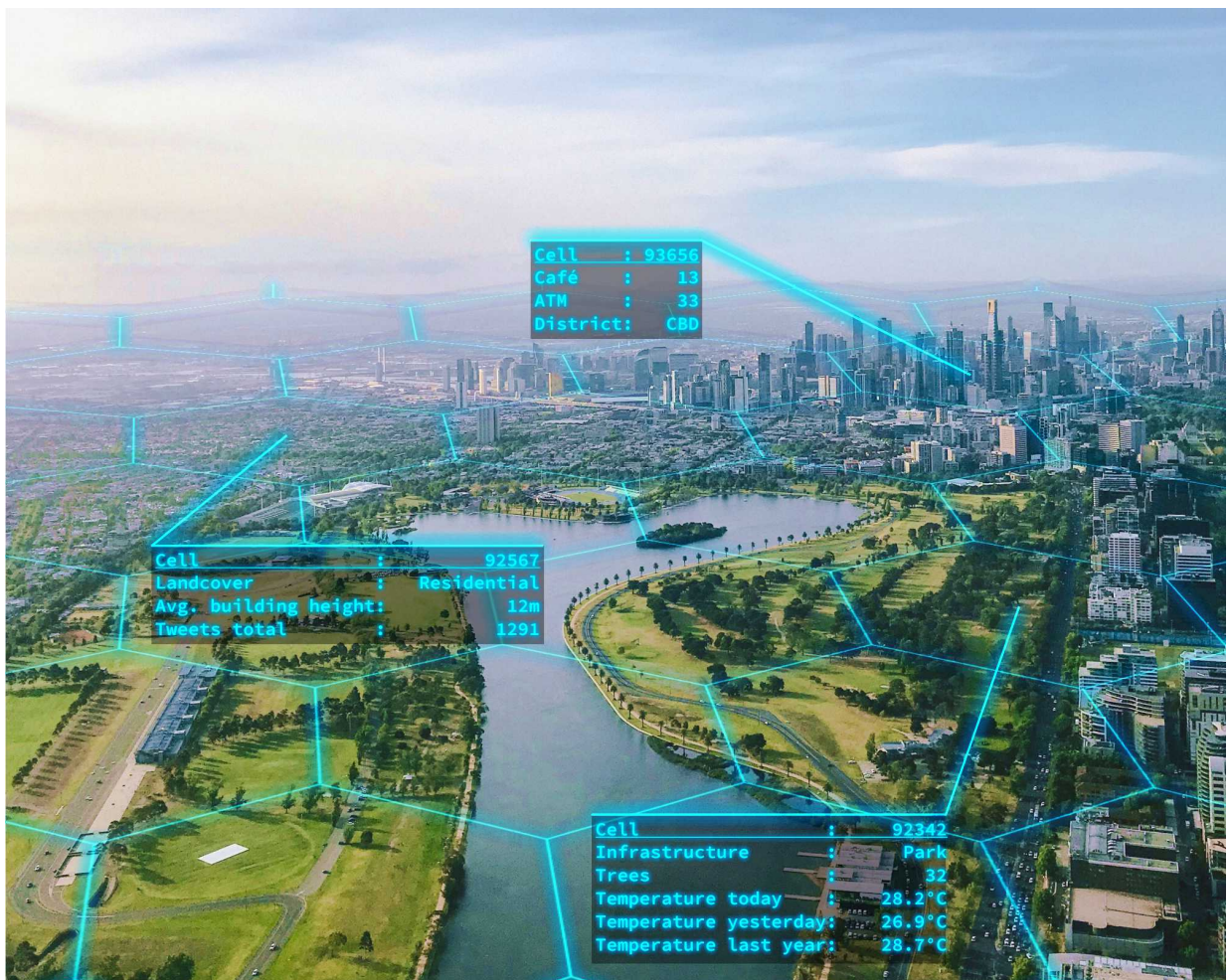


Figure 4 – Geolnsight Spatial Tokenizer: From Fragmented Data to Integrated Spatial Tokens

- **Technical Highlights**
 - **Performance Engineering:** Utilized Rust and DuckDB to achieve near-instant response times for on-the-fly quantization of Cloud Optimized GeoTIFFs (COGs).

- **Grid Agnostic:** Supports consistent analysis across multiple DGGRSs including H3, ISEA3H, ISEA9R, IVEA3H, IVEA9R, RTEA9R, and IGEO7.
- **Scalable Backends:** Validated a stable caching layer utilizing Parquet to reduce recomputation costs and enable the efficient re-use of AI-ready outputs.

5.3.4. GeoLynx – The Open-Source Innovators (D121)

GeoLynx developed the core `pydggsapi` components, providing foundational tools for the Python data science community to interact with Discrete Global Grid Systems.

- **Key Outcome:** Demonstrated successful inter-DGGRS conversion between IGEO7 and H3 at runtime, proving that data storage can be decoupled from client-specific grid requirements.
- **Website Link:** [GeoLynx Website Detail](#)
- **Full Technical Detail:** See Annex J.

GeoLynx, in collaboration with the Landscape Geoinformatics Lab (University of Tartu) and CRIM, developed **pydggsapi**, an open-source Python-based web API designed to provide an easy-to-use discovery service for DGGS data. Built on the high-performance **FastAPI** framework, the implementation serves as a primary middleware for the Python geospatial ecosystem, wrapping complex grid libraries such as **DGGRID** and **DGGAL** into standardized OGC API endpoints.

The implementation champion's a modular "Provider" architecture, allowing for a "plug-and-play" approach to different grid systems (H3, IGEO7, HEALPix, etc.) and storage backends. By supporting both database-driven (ClickHouse) and cloud-native file formats (Zarr, Parquet), GeoLynx demonstrated how large-scale research datasets can be made immediately analysis-ready. A unique feature of this implementation is the **Mapbox Vector Tiles API**, which allows for the seamless visualization of grid data in traditional GIS software like QGIS, bridging the gap between cell-based spatial tokens and standard mapping tools.

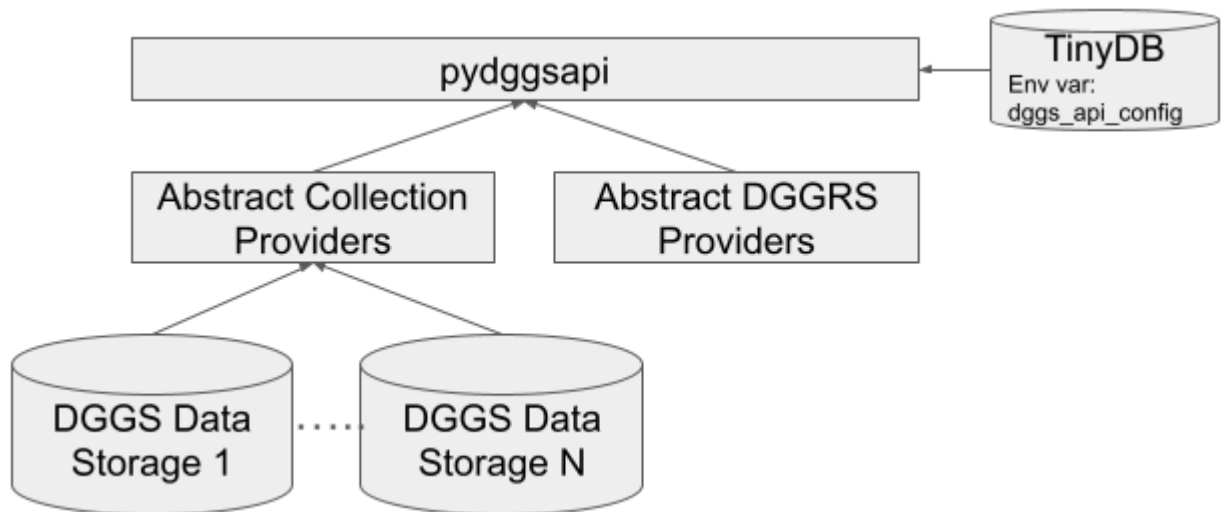


Figure 5 — Geolynx pydggsapi: Modular Architecture for Grids and Storage

- **Technical Highlights**
 - **Interoperability:** Enabled AI clients to navigate hierarchies across diverse reference systems through a unified spatial language.
 - **Scalable Backends:** Validated the use of chunked, file-based storage (Zarr/Parquet) for high-performance access to sparse satellite data.
 - **Grid Resampling:** Successfully executed experimental on-the-fly conversion from IGEO7 to H3, allowing clients to request data in their preferred grid regardless of the native storage format.

5.3.5. Geomatys — The Multi-Dimensional Experts (D125)

Geomatys utilized the Java-based **Examind Server** to address 5D data challenges, enabling AI agents to navigate complex spatial, temporal, and vertical dimensions within the grid.

- **Key Outcome:** Proposed and implemented the /zones discovery endpoint, transforming grid navigation from a technical requirement into a natural spatial search.
- **Website Link:** [Geomatys Website Detail](#)
- **Full Technical Detail:** See Annex K.

Geomatys acted as a critical anchor for the **Server Backbone**, focusing on the ingestion and serving of multi-dimensional coverage data. Utilizing the **Examind Server** and the **Apache SIS** (Spatial Information System) library, they demonstrated how high-velocity and high-dimensional datasets—such as time-series weather models and multi-layer elevation products—can be seamlessly re-sampled into any DGGRS on-the-fly. This “any data, any grid” approach proved

that server-side resampling is the key to decoupling large data archives from specific client-side grid requirements.

A major contribution of this implementation was the proposal of the **Discovery Endpoint**. Before this pilot, OGC API – DGGs required clients to know specific Zone IDs to retrieve data. Geomatys proved that by allowing agents to query via natural geography (e.g., a WKT Polygon), the barrier to entry for AI is significantly lowered. Additionally, Geomatys bridged the gap between Earth Observation and Astronomy by developing a connector for **Hierarchical Progressive Surveys (HIPS)**, proving the portability of the DGGs standard across scientific domains.

- **Technical Highlights**
 - **5D Scaling:** Validated the use of grid-native addressing for data containing Space, Time, and Height dimensions.
 - **Spatial Discovery:** Implemented the prototype /zones endpoint, allowing AI agents to “discover” the grid through bounding box queries.
 - **Standardization Leadership:** Contributed to the **Zarr DGGs metadata** conventions and developed the official Java bindings for the **DGGAL** library.

5.3.6. Safe Software – The Workflow Orchestrators (D106)

Safe Software utilized the **FME Data Virtualization Platform** to serve DGGs data via no-code automation, acting as the “universal translator” between legacy geospatial data and grid-native AI reasoning.

- **Key Outcome:** Validated that existing enterprise ETL platforms can be adapted to OGC DGGs standards without custom low-level coding.
- **Website Link:** [Safe Website Detail](#)
- **Full Technical Detail:** See Annex L.

Safe Software’s implementation demonstrates the feasibility of serving DGGs via no-code workflow orchestration. By leveraging the **FME (Feature Manipulation Engine)** platform and the **DGGAL** open-source library from Ecere, they created a suite of custom transformers that perform grid quantization, cell indexing, and spatiotemporal filtering. FME Data Virtualization allows for the rapid deployment of REST services such as OGC APIs by leveraging OpenAPI service descriptions. This architecture allows organizations to ingest high-volume datasets—such as the **Manitoba DEM**, **climate model data cubes**, and **daily flood models**—and expose them as compliant OGC API – DGGs endpoints with minimal integration friction. These custom transformers were published on [FME Hub](#) and are available to anyone who wants to work with DGGs from within FME workflows. This DGGAL based service includes support for: GNOSISGrid, ISEA9R, ISEA3H, IVEA9R, IVEA3H, RTEA9R, RTEA3H, rHEALPix grid reference systems.

The **DGGS service** is powered by **FME Flow** hosted on **AWS**, and supports the dynamic re-quantization of features into multiple resolutions. During the pilot, this capability was utilized to drive the Flood Impact Index (FII) and provide the AI-clients with ground truthed data streams. Through FME Flow, the implementation successfully delivered grid zone discovery, metadata access, and geometry delivery in **DGGSJSON**, **GeoJSON** and **GeoTIFF** formats, ensuring that the “Server Backbone” remains accessible to both human analysts and autonomous AI agents.

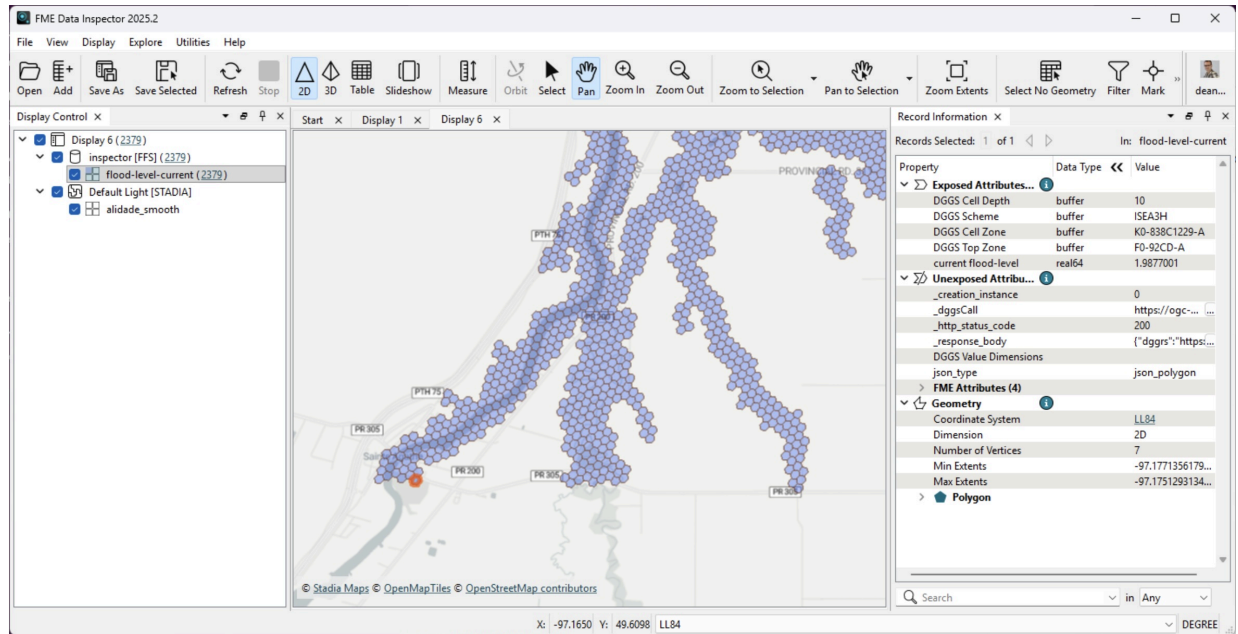


Figure 6 – FME Data Inspector Test Client viewing FME DGGS Service for Current Flood Level > 1 metre, Zoom Level 10

- **Technical Highlights**
 - **Compliance:** Full implementation of Draft OGC API – DGGS for zone discovery and data retrieval.
 - **Interoperability:**
 - Enabled AI clients to perform complex spatial/temporal and **CQL2 attribute queries** (e.g., filtering zones where flood depth > 1.5m).
 - Supported access to and quantization for a variety of source data streams including elevation, climate projections and flood model scenarios.
 - Supported AI clients such those from Hartis and Terraframe. Terraframe’s client leveraged Spatial Knowledge Graphs (SKG) which allowed it to identify cascading impacts where flooded zones compromised road or power networks. In this way the downstream effects could be assessed in terms of cumulative impacts.
 - **Automation:** Developed a repeatable pipeline that converts traditional vector and raster data into machine-readable **Spatial Tokens** on-the-fly.

5.4. The Intelligence Frontier (AI Clients)

The following participants developed the agentic reasoning and visualization tools that consumed the DGGs data streams.

5.4.1. Compusult – The UX Pioneers (D103)

Compusult developed the primary React-based chatbot, focusing on the “Chat-to-Map” user experience.

- **Key Outcome:** Successfully demonstrated natural-language geolocation and automated grid-cell rendering.
- **Website Link:** [Compusult Website Detail](#)
- **Full Technical Detail:** See Annex M.

For this pilot project, Compusult has developed an AI-enabled DGGs client to query and visualize data from OGC API – DGGs. The DGGs client is provided as a web-based application, and supports interoperability with independent DGGs endpoints. It demonstrates how DGGs, using OGC standards-based approaches, can support analysis of geospatial and statistical data to inform emergency management decision making.

- Development of an LLM based chat API;
- Ability to use AI to generate DGGs zone queries and zone data queries from a user-entered natural language text query;
- Rendering of DGGs zone data in GeoTIFF and GeoJSON formats in a map view;
- Support for several different DGGs, including H3, HEALPix, and Icosahedral Equal-Area projections;
- Interoperability experiments with at least two other participants;
- Use of AI to identify applicable data collections and filter attributes in user-entered queries; and
- Ability to filter on queryable parameters and search results using OGC Common Query Language (CQL2) expressions.

5.4.2. CS Group – The Autonomous Agent Architects (D123)

CS Group implemented advanced **Agentic AI workflows** using the Model Context Protocol (MCP) to demonstrate how autonomous agents can navigate a federated geospatial ecosystem without manual human intervention.

- **Key Outcome:** Successfully demonstrated the ability for an AI agent to independently “read” server capabilities and chain tools across multiple independent vendors to resolve complex queries.
- **Website Link:** [CS Group Website Detail](#)
- **Demonstrator** [CS Group AI-DGGs client demonstrator](#)
- **Full Technical Detail:** See Annex O.

CS Group’s contribution focused on shifting the paradigm from simple chatbots toward **Agentic AI**. Utilizing the **Model Context Protocol (MCP)** as the standard interface between the AI reasoning layer and the **Server Backbone**, they proved that AI can act as a proactive analyst. Rather than hard-coding spatial logic, the team developed a system where the AI chatbot autonomously discovers DGGs collections, search locations, and constructs valid OGC API – DGGs queries by interpreting machine-readable tool definitions. The AI agent is able to analyze simple queries or complex scenarios involving multiple collections from 1 or more DGGs API providers. In that case, the agent analyze and split the user input into a series of actions. The agent then choose which tools are necessary, and in which order to fulfill the user request. Tools output can be indeed used as input for next steps. For example:

- the first iteration is often the geolocation of the **area of interest (Aoi)**, extracted from the query. If no area is provided, the agent will ask the user for more details.

- Once extracted (eg. “**Winnipeg county**”) from the input prompt, the geolocation tool query the nominatim service to obtain its WGS84 coordinates and bounding box.
- This information is then used by another tool calling DGGs API to get the **cells identifiers** for this Aol.

Each tool call feeds a user session context, so that previous responses can be used to answer next questions or execute further steps if needed. This flexible way to interact with the DGGs API allows the end user to discover which collections are available, and dig progressively into the different collections, fields, and available DGGs.

The client is built on a Python-based stack using **FastMCP**, **LangChain** and **Streamlit**, providing an LLM-agnostic environment that integrates with both local models (Ollama) and cloud services (Google Gemini). During the pilot, CS Group demonstrated multi-vendor orchestration by chaining API calls across **Ecere**, **Geomatys**, and **Safe Software**. This architecture ensures that the AI’s reasoning process is explicit and traceable, providing a blueprint for future standards-based geospatial assistants that can handle high-velocity data across federated environments.

CS Group client managed to trace every tools that have been called to answer the user’s scenario. For validation and explainability it enables to check the different DGGs API calls, and for each the inputs and the output. Finally if needed it is possible to ask directly the agent to explain the reasoning that’s been used: explain why these tools have been called, for which purpose and write a detailed step-by-step summary.

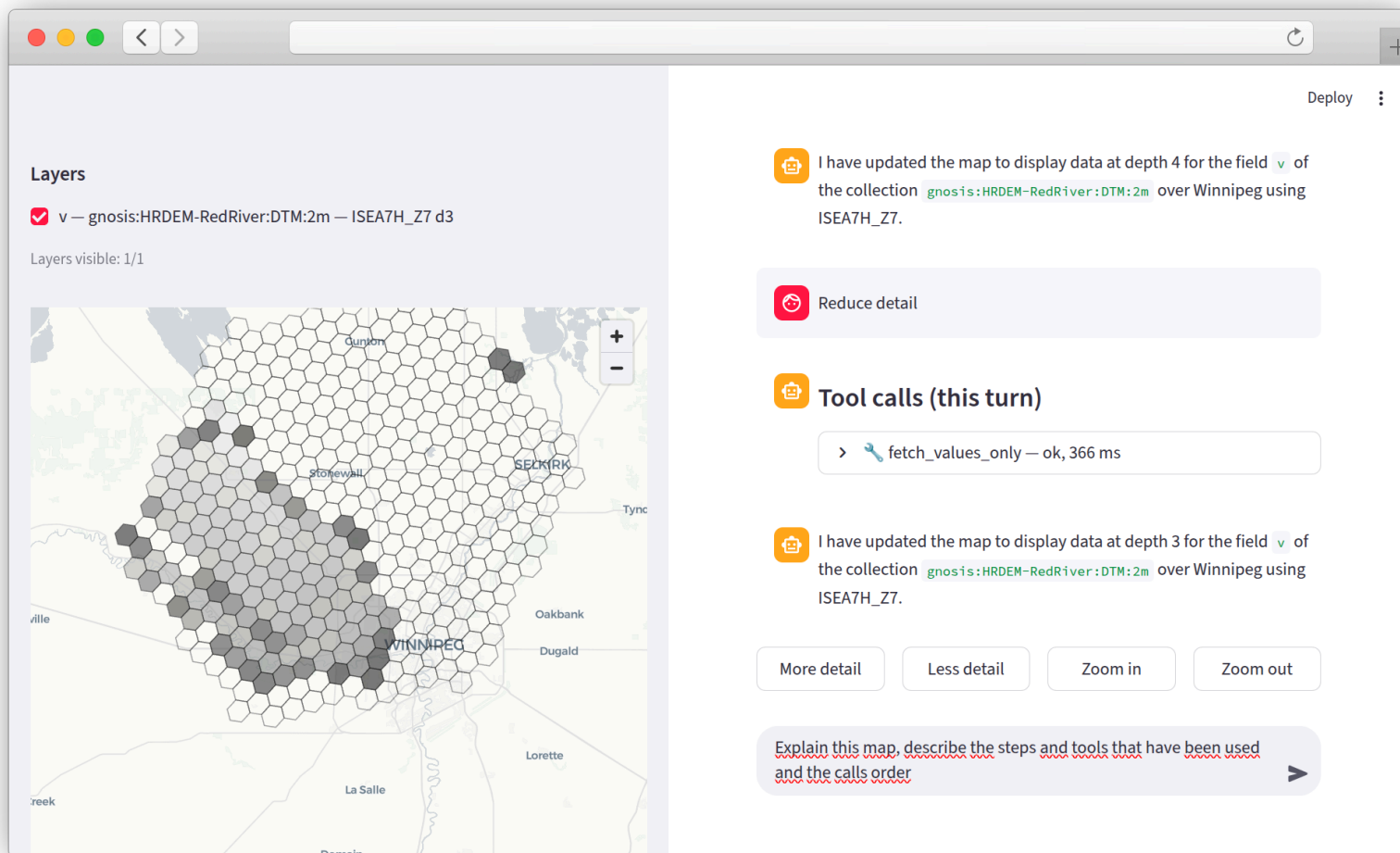


Figure 8 — CS Group AI-Client: Autonomous Tool Discovery and Sequential Reasoning via MCP

- **Technical Highlights**
 - **Agentic Workflows:** Moved beyond text generation to autonomous task planning, allowing the AI to determine the necessary steps to solve a spatial problem.
 - **Protocol Literacy:** Championed **MCP** to eliminate vendor-specific hard-coding, allowing the client to operate against any compliant server.
 - **Explainable Chains:** Provided a transparent audit trail of the tools invoked and data retrieved, ensuring that AI results are grounded in “Grid-Truth.”

5.4.3. Hartis — The Decision Intelligence Specialists (D124)

Hartis developed the **Flood Impact Index (FII)**, fusing multi-source grid data into actionable risk scores.

- **Key Outcome:** Validated that RAG-based AI can explain the “why” behind grid-based risk assessments.
- **Website Link:** [Hartis Website Detail](#)
- **Full Technical Detail:** See Annex P.

Acting as the **Decision Intelligence Specialists** for the pilot, Hartis developed an AI-enabled client (<https://ai-dggs.hartis.org/>) that provides an intelligent, explainable “Geospatial Brain” for flood-risk assessment. The primary objective was to bridge the “**Discovery Gap**” by allowing emergency responders to interact with petabytes of distributed geospatial data using natural language.

The client functions as both an interactive analytics dashboard and a conversational assistant, specifically designed to ground Large Language Models (LLMs) in “**Grid-Truth.**” By leveraging the **Model Context Protocol (MCP)**, the client translates plain-English questions into precise DGGs queries, consuming federated data streams—including flood probability, population density, topography, and land cover—to generate spatially coherent and auditable insights.

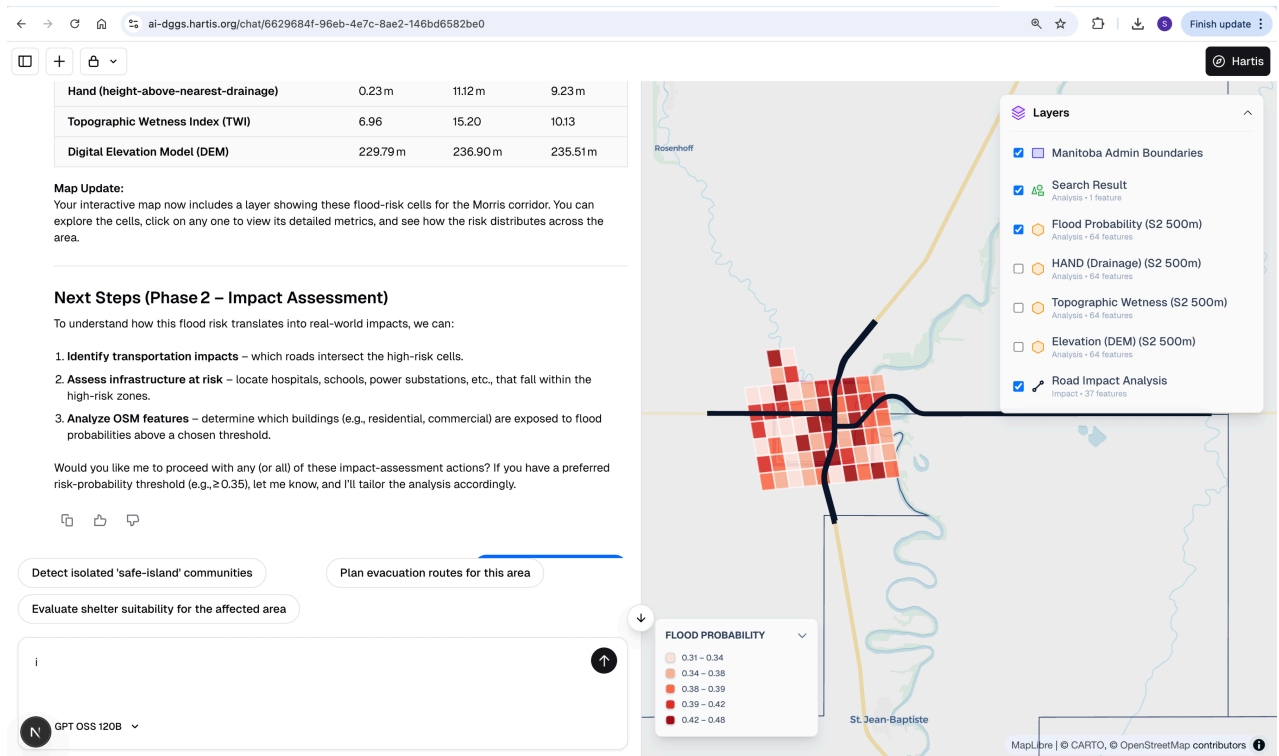


Figure 9 — Hartis "Chat-to-Map" Interface demonstrating the Flood Impact Index (FII) and MCP tool orchestration.

The architecture is built on a **Next.js (React)** frontend and a **Django/PostGIS** backend, utilizing **DGGAL** (WASM and Python) to handle high-speed grid mathematics. The intelligence layer is powered by **Retrieval-Augmented Generation (RAG)**, which ensures that the LLM does not “hallucinate” coordinates but instead “reads” the authoritative DGGs cell values directly from the **Server Backbone**.

To address “**Scale Blindness**,” the client acts as the AI’s “**Spatial Conscience**.” When a user asks a high-level question, the orchestrator uses MCP tools (such as `query_flood_zones` and `explain_flood_risk`) to perform progressive refinement—querying coarse grids for initial discovery before drilling down into high-resolution hexagons only where risk is detected. The back-end integrates the **Flood Impact Index (FII)**, a sophisticated model that fuses RCM radar imagery, GEPS weather forecasts, and CDEM terrain data into a single, cell-based risk score. During Technology Integration Experiments (TIEs), the client successfully federated queries across Geomatys, GeoLynx and Safe Software servers, validating true multi-vendor interoperability.

The resulting “**Chat-to-Map**” experience successfully transitioned geospatial interaction from human-driven data searching to AI-driven decision support. Users can identify vulnerable neighborhoods in Southern Manitoba, analyze cascading impacts on road networks, and receive summaries that link every recommendation to specific DGGs Zone IDs.

Moving forward, Hartis aims to evolve this client toward **Geospatial Foundation Models (GeoFMs)**. By training small, domain-specific models on the physical mechanics of flood dynamics and OGC standards, the system will provide even higher-speed reasoning at the edge, ensuring responders remain informed even in bandwidth-constrained disaster environments.

5.4.4. TerraFrame — The Semantic Integrators (D104)

TerraFrame linked DGGs zones with **Spatial Knowledge Graphs (SKGs)** to model infrastructure failure.

- **Key Outcome:** Proved that DGGs is most powerful when connected to semantic data, allowing AI to detect cascading risks.
- **Website Link:** [TerraFrame Website Detail](#)
- **Full Technical Detail:** See Annex N.

TerraFrame developed a web-based mapping client capable of translating natural language queries into requests to DGGs (Discrete Global Grid System) servers, returning results both visually on the map and as human-readable summaries. To accelerate development, the team successfully repurposed and extended existing components originally built for querying spatial knowledge graphs (SKGs). By integrating the strengths of DGGs for statistical aggregation with the semantic expressiveness of SKGs, the solution demonstrates how disaster impacts—such as flooding—can be analyzed not only within directly affected zones but also in terms of cascading effects. For example, flooding in one DGGs cell can compromise transportation infrastructure or power facilities, thereby affecting populations located outside the immediate flood area. This approach highlights the value of combined DGGs and SKG architectures in supporting downstream impact analysis for disaster resilience and planning.

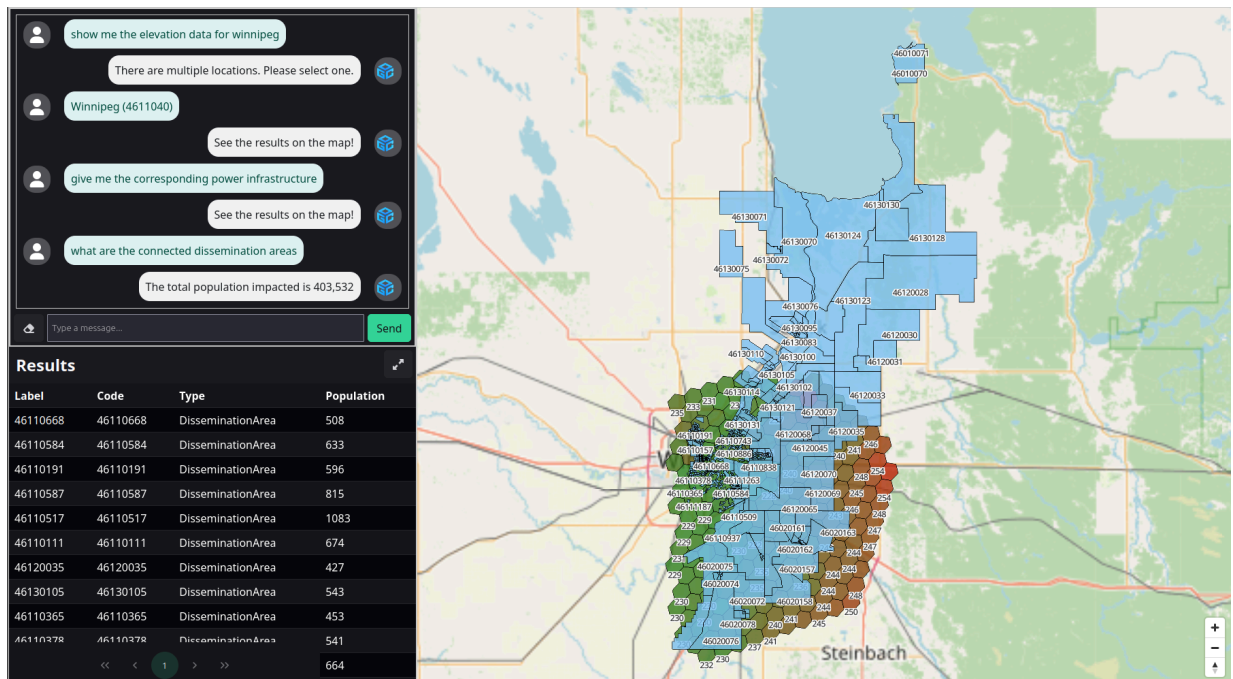


Figure 10 – TerraFrame AI-enabled DGGs Client Interface

The architecture of the TerraFrame client consists of an Angular front-end using MapLibre and DGGAL.js to display map features and DGGs zone data. The front-end communicates with a back-end server which orchestrates communication between the Anthropic Claude 3.7 LLM hosted on AWS Bedrock, Canada administrative hierarchy location data host on an Apache Fuseki server, and the various DGGs servers. The orchestration server is a SpringBoot server which uses basic Spring security to authenticate all users in the system. The system uses a tool spec approach to make the LLM aware of the server components it can use and to allow the LLM to make requests to individual components when it determines the need to do so based on the user questions. The TerraFrame client is integrated with Safe Software and GeolInsight DGGs servers.

The TerraFrame client enables users to query DGGs data using natural language—for instance, to identify flood risks. It further allows users to assess downstream impacts or cumulative effects by determining which feature networks are affected. This process can involve multiple layers of spatial indirection, such as assessing disruptions to transportation routes. Currently, power infrastructure and administrative boundaries are modeled within a spatial knowledge graph; however, a methodology has been developed to specifically evaluate the impact on evacuation and supply delivery routing. Providing users the ability to manually specify DGGs resolution (Zone level) remains an open challenge, as the optimal level depends on factors that are often not transparent to the end-user, such as the nature of the underlying data.

5.5. The Integration Glue (Schema & Tools)

The pilot's architecture is unified by the Common Operating Picture (COP) schema and technical bridges.

- **Lead Architecture & COP Schema:** Detailed walkthrough of the machine-readable “knowledge context.” [Website Detail](#)
- **Standardization Roadmap:** The 6-point plan to resolve DGGS frictions. [The Roadmap](#)
- **Identity, Provenance, and Trust (IPT):** Verification mechanisms for AI inputs. [IPT Overview](#)

5.6. Open Source Technical Resources

- **DGGAL Library:** [GitHub Repository](#)
- **GeoPlegma:** [GitHub Repository](#)
- **pydggsapi:** [Documentation](#)
- **CRIM Pilot Scripts:** [Data Preparation Notebooks](#)

NOTE: As per the pilot's commitment, all web-accessible prototypes and API endpoints will be maintained for 12 months post-pilot (until January 2027) to facilitate ongoing ISO/OGC review and stakeholder evaluation.



6

STANDARDIZATION ROADMAP AND FUTURE OUTLOOK

STANDARDIZATION ROADMAP AND FUTURE OUTLOOK

6.1. Overview: From Data Retrieval to Geospatial Reasoning

The AI-DGGS Pilot demonstrated that while baseline interoperability is achievable, transitioning from simple natural-language summarization to autonomous “**Geospatial Analysts**” requires a fundamental paradigm shift. Current OGC standards must evolve from passive data retrieval protocols into active frameworks for **Geospatial Reasoning**. To achieve agentic autonomy, OGC standards must prioritize **Autonomous AI Discovery**, resolving technical gaps to establish a **Digital Fabric**—a world-centric model ensuring AI logic is anchored in physical reality, multi-grid interoperability, and scientific provenance.

6.2. The AI-DGGS Standardization Roadmap

The pilot identifies six standardization actions required to bridge the gap between autonomous AI reasoning and the rigid requirements of high-resolution geospatial grids.

6.2.1. Authoritative DGGRS Register

A primary bottleneck for federated AI is the lack of a central mathematical source of truth for grid cell placement, as discrepancies between software libraries—notably between DGGAL and DGGRID implementations—frequently lead to misaligned risk indicators. To resolve this **Grid Inconsistency Friction** and enable Trustworthy Spatial Actionability, OGC should establish a formal **DGGRS Registry** modeled after the successful EPSG Registry for map projections.

Without a canonical reference, common grid definitions like H3 or rHEALPix remain collections of independent implementations rather than strict mathematical references, leading to subtle **spatial drift** caused by variations in floating-point precision or coordinate transformation logic across different programming environments. This drift is particularly problematic for **federated AI reasoning**, where an autonomous agent must aggregate data from multiple disparate servers that may represent the same physical area with mathematically incompatible cell boundaries.

By assigning unique **DGGRS Identification Codes**, OGC would enable servers to explicitly disclose the exact version, parameters, and mathematical constants used for their grid structure,

ensuring total transparency for high-stakes decision support. This infrastructure allows Agentic AI to verify that a “Flood Risk” token from one vendor and a “Population” token from another are perfectly co-registered before performing a spatial join, thereby eliminating the risk of geospatial hallucinations caused by underlying technical misalignment.

6.2.2. Discovery-Oriented Extensions and Quantization Transparency

Current standards primarily focus on spatial discovery via technical Zone IDs, which creates a significant “**Granularity Wall**” for autonomous agents that reason in geographic entities rather than technical indices. To resolve the **Metadata Discovery Gap** and activate **Descriptive Grounding**, the OGC API – DGGS should be extended to allow agents to query the **Zone Query (/zones)** capability via natural geographic features such as **BBOX or WKT Polygons**. This allows an AI to perform area-based data retrieval and receive high-level summary values directly without the computational burden of pre-calculating millions of technical Cell IDs, mirroring the ease of interaction found in traditional feature-based APIs and enabling the system to scale effectively across vast disaster zones.

Furthermore, the pilot identified a critical need to transition DGGS from a simple data container into a **Semantically-Aware Infrastructure** for autonomous discovery. To mitigate **Inductive Bias** and bridge the **Interpretation Gap**, the OGC must mandate “**AI-Ready**” **metadata profiles** that include explicit **Quantization Provenance** (e.g., documenting K-NN vs. Bilinear methods) and **Unit Semantics**. By embedding **STAC Extensions**, disaster-specific schemas, and formal ontologies into the metadata, we ensure that AI agents can independently verify the meaning, units, and scientific fitness of data for a specific crisis, preventing geospatial hallucinations and “Scale Blindness” caused by a lack of contextual awareness regarding how data was aggregated into the grid fabric.

6.2.3. OGC Best Practice for H3

There is a persistent conflict between the industry-standard H3 library, which utilizes an **authalic spherical model**, and OGC requirements for ellipsoidal (WGS84) precision. This discrepancy is a primary source of **Coordinate Model Ambiguity**, as geodetic latitudes on the WGS84 ellipsoid must be correctly mapped to the authalic sphere to ensure that zone boundaries remain mathematically consistent. Without a formal best practice, different software implementations may apply varying transformation logics, leading to “spatial drift” where AI-generated risk assessments fail to align with real-world terrain and critical infrastructure.

A major finding of the pilot is that the native H3 library lacks a **geometric sub-zone function** required to meet the rigorous hierarchical requirements of the OGC API – DGGS. Because H3 utilizes an aperture-7 tiling scheme, parent cells do not perfectly contain their logical children; instead, they exhibit a **fractal non-containment** where ~6.52% of the analysis area may technically belong to neighboring parent cells. To resolve this **Non-Containment Friction**, the pilot developed a **Geometric Sub-zone Algorithm** that performs a vertex-based boundary check to identify all descendant cells that are physically contained within an ancestral zone, regardless of their logical index.

The implementation of this algorithm, contributed by Ecere and Geomatys, see Annex R, utilizes **Spherical Linear Interpolation (SLERP)** to rigorously verify cell containment across

resolutions. By sampling vertices and interpolating intermediate points on the authalic sphere, the algorithm ensures that the “Digital Fabric” remains topologically sound, allowing AI agents to perform **Trustworthy Spatial Joins** between disparate datasets. This algorithmic fix, which has been shared with the open-source H3 project, provides the foundation for the **Physical Reality Anchoring Enabler** by ensuring that a “Flood Risk” predicted for a specific grid cell truly corresponds to the physical assets located within that boundary.

To formalize these findings, the OGC should publish a canonical **Best Practice for H3** that explicitly defines the geodetic-to-authalic transformation logic and standardizes the geometric sub-zone algorithm. This document will serve as the technical cornerstone for **Scientific Grounding**, ensuring that any organization using H3 within a federated OGC ecosystem can achieve mathematically identical results. By providing a clear reference for handling **vertex-based containment** and coordinate refinement, the OGC can transform H3 from a general-purpose indexing library into a scientifically robust DGGRS suitable for high-stakes disaster management and autonomous reasoning.

6.2.4. Temporal “Regridding” Parameters

Merging high-velocity satellite observations with static datasets is frequently hindered by inconsistent time-steps and **Temporal Fragmentation**, where a radar pass every few days must be reconciled with hourly weather forecasts and static infrastructure maps. To overcome this, the pilot recommends the implementation of formal “Temporal Resolution” and “Regridding” parameters within the OGC API — DGGS, enabling **Time-Aware Reasoning** by allowing AI agents to synchronize disparate data sources across a unified 4D timeline.

The current lack of temporal standards leads to a “**Temporal Explosion**” when moving from daily to minute-level time-steps, as the data volume required to represent high-resolution changes across millions of grid cells can quickly exceed bandwidth limits. By standardizing Temporal Resolution, a mechanism is provided for servers to define the “temporal bucket” (e.g., a 1-hour window), while **Temporal Regridding** parameters allow the client to specify how sparse observations should be handled—whether through snapping to the nearest time-step, linear interpolation, or returning the maximum observed value within a window.

This approach is essential for bypassing the **4D Resolution Wall**, as it allows DGGS cells to act as **Intelligent Pointers** to multi-dimensional data cubes stored in formats like Zarr or GeoParquet. Instead of transmitting the entire 4D volume, the server can provide a temporal summary (e.g., “Peak Flood Crest”) within the grid cell, while providing the URI for the full-resolution 4D payload. This ensures that Agentic AI can identify precisely which time-steps are contemporaneous and relevant to a user’s query, such as identifying if a predicted flood crest at 2:00 PM overlaps with a planned evacuation window, without the extreme overhead of manual 4D data alignment.

6.2.5. Auditable Common Operating Picture (COP)

The pilot recommends evolving the COP into a verifiable **Information Sharing Mechanism** that natively integrates the **Identity, Provenance, and Trust (IPT)** framework into a machine-readable JSON-LD schema to resolve the “provenance opacity” currently found in disaster response. A primary roadmap action is the mandate for **IPT Smart Certificates**, which transform the COP

from a static data snapshot into an immutable record of the full geospatial lifecycle, from initial acquisition to final dissemination.

To support high-stakes decision-making, the OGC should standardize **Trust-Based Decision Thresholds**, utilizing a composite scoring methodology that weights factors such as **Input Data Trust (30%)**, **Agent Identity (20%)**, and **Process Transparency (20%)**. Under this framework, the COP orchestrator can enforce safety-critical rules—such as requiring a trust score of ≥ 0.90 for life-safety decisions—ensuring that unverified AI inferences or “hallucinated” risk zones are flagged for human review before being shared across the command chain.

Future standard revisions should also extend the COP into an **Active Orchestration Platform** that utilizes the **Model Context Protocol (MCP)** to coordinate specialized **Small Local Language Models (SLLM)** and autonomous data integration agents. By capturing executable **Service Chains** rather than simple data snapshots, the auditable COP ensures total **Analytical Reproducibility**, allowing responders to re-execute the exact flood simulations and model parameters used to generate any given layer in the common operating picture.

6.2.6. Advanced Analytical Extensions

To be truly effective, DGGS must transition from a passive storage format to an active **Computational Engine**, resolving the **Limited Analytical Expressiveness Friction** by moving the “heavy lifting” of multi-source data synthesis closer to the data source. A core requirement for this evolution is the standardization of **Vector Discretization Rules**, ensuring that linear features (roads) and polygonal features (admin zones) are “tokenized” using consistent occupancy models (e.g., centroid-based vs. all-touched). Without standardized quantization rules for vectors, disparate servers cannot produce the mathematically identical outputs required for reliable federated analysis.

The discretization of vector infrastructure into the grid fabric is the primary prerequisite for **High-Performance Computational Joins**. By converting complex geometric intersections (e.g., ST_Intersects) into simple **Integer or String Matching**, the architecture enables “**Grid Algebra**” operations that run orders of magnitude faster than traditional GIS queries. This allows servers to perform real-time cascading risk analysis—such as intersecting flood hazard cells with discretized road networks—returning on-the-fly multi-source indicators to the AI agent without the bandwidth-heavy transfer of raw geometric layers.

This analytical capability also facilitates **Scale-Aware Progressive Refinement**, as discretized vectors can be integrated into the DGGS hierarchy to provide consistent summaries across resolutions. For example, a road network can be represented as “infrastructure density” at a coarse level for regional planning while maintaining precise cell-occupancy at finer levels for local evacuation routing. This native support for multi-resolution analysis ensures that **Agentic AI** receives a structured, uniform input—a **multidimensional digital fingerprint**—where infrastructure presence is just another attribute in the cell’s feature vector alongside elevation or flood depth.

Finally, the roadmap proposes the introduction of multidimensional reference frameworks, such as the **CompoundRS**, see Annex S, to natively integrate these 2D grid cells with temporal and vertical dimensions. Such extensions allow servers to return operational insights for 5D phenomena (Space, Time, and Height) rather than raw sensor values, operationalizing the **Compound Spatial Reasoning Enabler**. By standardizing both vector tokenization and

multidimensional algebra, the OGC ensures that the DGGs ecosystem can manage the mathematical rigor of complex planetary modeling at scale, allowing AI agents to focus on high-level strategy and natural language explanation.

6.3. Operational Alignment Matrix: From Frictions to Standardization Actions

The following matrix consolidates the core findings of the AI-DGGs Pilot, explicitly mapping the **DGGs Frictions** identified during implementation to the required **AI Frontier Enablers**, and to the corresponding **Standardization Actions** proposed by the pilot. This alignment ensures traceability from observed technical limitations to concrete, actionable standards evolution.

Table 4 — Operational Alignment Matrix mapping DGGs implementation frictions to AI Frontier enablers and corresponding standardization actions.

IDENTIFIED FRICTION	AI FRONTIER ENABLER	STANDARDIZATION ACTION	RESULTING CAPABILITY
Grid inconsistency across implementations	Trustworthy spatial actionability	Authoritative DGGs Register Clause 6.2.1	Mathematically consistent cell alignment across vendors, enabling reliable federated AI reasoning
Opaque metadata and semantic ambiguity	AI-ready semantics and generic discovery	Discovery-Oriented Extensions and Quantization Transparency Clause 6.2.2	Agents can autonomously interpret data meaning, units, aggregation logic, and uncertainty
Coordinate model mismatch (authalic vs ellipsoidal)	Physical reality anchoring	OGC Best Practice for H3 Clause 6.2.3	AI-generated analyses align precisely with real-world terrain and infrastructure
Temporal fragmentation of observations	Time-aware reasoning	Temporal “Regridding” Parameters Clause 6.2.4	Multi-source datasets can be synchronized in time without bandwidth-heavy preprocessing
Lack of traceability in AI reasoning	Trust, provenance, and auditability	Auditable Common Operating Picture (COP) Clause 6.2.5	AI decisions are explainable, reproducible, and suitable for high-stakes operational use
Limited analytical expressiveness of DGGs	Compound spatial and multi-dimensional reasoning	Advanced Analytical Extensions Clause 6.2.6	Servers return pre-computed, multi-dimensional indicators enabling scalable AI workflows

6.4. Future Research and Operational Outlook

6.4.1. Scientific Geospatial Foundation Models (GeoFMs)

A significant research horizon identified by this pilot involves the development of **Scientific Geospatial Foundation Models (GeoFMs)** which propose to move beyond pure linguistic prediction toward representations informed by geophysical laws. Future research could explore the integration of OGC standards with geophysical principles; while standard LLMs predict the next “token” based on linguistics, a Scientific GeoFM seeks to estimate the next “state” based on the underlying physics of the environment, such as fluid dynamics for floods or geomechanics for landslides.

The pilot explored three research pillars for building these models.

- **1. Self-Supervised Spatial Tokenisation Embeddings** establish a standardized “digital fingerprint” allowing AI to ingest, normalize heterogeneous spatial data into uniform formats.
- **2. Hierarchical Semantic Reasoning Logic** utilizes grid identifiers as a context-aware layer bypassing heavy coordinate math to reveal the scale, nature of a place instantly.
- **3. Generalization via DGGs Vision Transformers (ViT)** leverages grid hierarchies to learn global physical patterns, transitioning from location-memorization to generalized geospatial-logic.

To operationalize the first pillar, foundational research is required into Self-Supervised Geospatial Embeddings which learn representations anchored to geography through the fusion of multi-source imagery, elevation, and climate data. Current industry research [[Loc2Vec](#), [S2Vec](#), [Hex2Vec](#)] suggests that **DGGs serves as a potential geospatial tokenizer** to normalize diverse data into consistent, scale-aware feature vectors, providing the necessary mathematical substrate for models to ingest planetary-scale observations.

Regarding the second pillar, pilot participants discussed how treating Zone IDs as semantic labels may allow models to potentially bypass the heavy geometric overhead of traditional coordinate intersections and reprojections. For example, a model can learn that querying information for Parliament Hill in ISEA3H uses zone I0-1EEBFA7-C because its resolution ($\sim 0.39 \text{ km}^2$) is specifically relevant for that scale of inquiry. By training on such variations, it is posited that a model could develop a **‘how-to’ capability**, learning to resolve scale-aware queries correctly across the grid hierarchy rather than simply memorizing absolute geographic locations.

Building on the third pillar, experimental implementations of Vision Transformers (ViT) provide a pathway for generating what recent research identifies as **Earth Embeddings**. By treating **Root Zones as training batches** and hierarchical **sub-zones as individual tokens**, these architectures attempt to generalize terrain dynamics rather than just absolute locations. This pathway aims to enable an AI to discover similarities between regions, inferring hazard risks in one area based on learned physical patterns from comparable geographies elsewhere on the globe.

Figure 11 — A voxel-based Digital World Model: Transitioning from 2D surface tiling to multi-dimensional cubic volumes for deep-learning compatibility (courtesy of GeoInsight)

As noted by GeoInsight, high-resolution 4D processing may necessitate a transition toward **Voxel or Orthocubic grids** (see Figure 11), leveraging frameworks such as GeoSOT and TerraNexus for multi-dimensional efficiency. While hexagonal grids offer superior adjacency for 2D flow modeling, cubic voxel structures are often more compatible with the **3D convolutional neural networks** used to process 4D volumes. By deploying these models on the edge, responders could receive high-speed reasoning directly on mobile devices, working toward a future where DGGs enables AI to act as an intuitive Geospatial Analyst.

6.4.2. Streamlining Integration and Extensibility

Operationalizing these advanced intelligence capabilities requires a simultaneous focus on accessibility to ensure that the outputs of sophisticated GeoFMs can be utilized by the broader responder community. Consequently, efforts must focus on streamlining the **DGGs integration pipeline** to reduce the entry barrier for **grid-agnostic applications**, ensuring that high-resolution grid services can be leveraged across a wider array of federated software architectures without requiring specialized spatial libraries.

The introduction of **polyglot delivery extensions** is a key strategic priority, enabling servers to translate complex grid mathematics into standardized, “grid-blind” outputs like **GeoJSON**, **Vector Tiles (MVT)**, or **Cloud Optimized GeoTIFFs (COG)** on-the-fly. This allows traditional GIS tools and lightweight web clients to consume DGGs-derived intelligence through the formats they already natively support, effectively decoupling the underlying grid-based storage from the specific requirements of the consumption layer.

A frontier for this work is the formal integration of the **Model Context Protocol (MCP)**, providing the standardized “connective tissue” required for **agentic AI** to autonomously discover, query, and reason over federated DGGs infrastructures. By implementing MCP-compatible **Capability Disclosure**, DGGs servers can provide AI agents with machine-readable “tool definitions,” allowing them to independently plan and execute multi-step workflows—such as intersecting flood hazard zones with critical road infrastructure—without human-in-the-loop intervention.

Ultimately, this evolution transforms the DGGs API from a niche data endpoint into a **universal analytical service**. By lowering the technical threshold for both human analysts and autonomous systems, the OGC community ensures that the scientific rigor of high-resolution global grids is accessible to any application capable of making a standard web request, regardless of its underlying geometric sophistication.

7

SECURITY, PRIVACY, AND ETHICAL CONSIDERATIONS

SECURITY, PRIVACY, AND ETHICAL CONSIDERATIONS

7.1. Implementation Status and Disclaimer

During the course of this project, a thorough review was conducted to identify any potential security, privacy, and ethical concerns. After careful evaluation, it was determined that none of these considerations were relevant to the scope and nature of this project. Therefore, no specific measures or actions were required in these areas.

However, recognizing that these factors are critical for the eventual transition of AI-DGGS systems into production-level disaster management operations, the pilot conducted a theoretical review of the **Identity, Provenance, and Trust (IPT)** framework. The sections below outline the proposed safeguards and architectural concepts identified as “best practices” for future implementations.

7.2. The IPT Framework: Proposed Concepts for Trust

To address the potential for “**Provenance Opacity**” in future AI workflows, the pilot explored the integration of Identity, Provenance, and Trust (IPT) mechanisms based on OGC Testbed-19 and Testbed-20.

- **Identity (Proposed):** Utilizing **Smart Certificates** to identify every component in the ecosystem (data assets, API endpoints, and AI agents). This would ensure that an AI agent only interacts with verified and authorized servers.
- **Provenance (Proposed):** Recording a machine-readable **Provenance Chain** within the COP schema to track the “chain of custody” for an insight—from the raw sensor pass to the specific AI model version used for analysis.
- **Trust Scoring (Proposed):** Developing a composite trust score (0.0 to 1.0) based on data recency, agent identity verification, and algorithm transparency to help responders prioritize high-confidence information during a crisis.

7.3. Theoretical Ethical AI and Decision Safeguards

The pilot identified that while AI can act as a powerful **Geospatial Analyst**, its role in high-stakes modeling must be governed by ethical safeguards to prevent misinformation.

- **Hallucination Mitigation:** The pilot demonstrated that anchoring AI reasoning in the DGGs grid (**Retrieval-Augmented Generation**) is a primary method for reducing “hallucinations” by forcing the model to refer to documented cell values.
- **Trust Thresholds:** Future systems should enforce **Human-in-the-Loop (HITL)** validation for any AI-generated result that falls below a defined trust threshold (e.g., < 0.90 for life-safety evacuation routes).
- **Transparency:** A core ethical recommendation is that every AI-generated summary must remain **auditable**, providing a direct link back to the “Grid-Truth” data source provided by the Server Backbone.

7.4. Data Protection and Privacy Considerations

In the context of future cross-border operations (such as the Red River Basin), the following considerations were identified for future work.

- **Security Classification:** The ability to tag data within the COP schema with classification levels (e.g., **Public Release** vs. **Confidential**) to ensure sensitive infrastructure data is only exposed to authorized personnel.
- **Anonymization:** Ensuring that population density data used for exposure modeling is processed as aggregate grid-cell statistics rather than individual records, aligning with international privacy regulations like **GDPR** and **PIPEDA**.

7.5. Summary of Proposed Trust Requirements

The following table summarizes the theoretical requirements identified for a production-ready AI-DGGs environment:

Table 5

REQUIREMENT	PROPOSED STRATEGY
Verifiability	Smart Certificates for all participating actors and services.
Traceability	Machine-readable provenance chains recorded in the COP schema.
Accountability	Defined risk thresholds requiring human-in-the-loop validation.
Reliability	Metadata-driven trust scores to quantify data quality.



BIBLIOGRAPHY





BIBLIOGRAPHY

- [1] Government of Manitoba. (n.d.). Red River Basin – Historic floods. Manitoba Transportation and Infrastructure. <https://www.gov.mb.ca/mti/wms/floodcontrol/redriverbasin/historic.html>
- [2] Manitoba Historical Society. (2001). Duffs Ditch. Manitoba History, 42. https://www.mhs.mb.ca/docs/mb_history/42/duffsditch.shtml
- [3] Government of Canada. (n.d.). Flood mapping. Natural Resources Canada. <https://geo.ca/flood-mapping/>
- [4] NASA Earth Observatory. (2022, May 11). Red River flooding is worst in a decade. <https://earthobservatory.nasa.gov/images/149822/red-river-flooding-is-worst-in-a-decade>
- [5] CBC News. (2022, May 28). Analysis: Manitoba's 2022 flood among the most significant in a century. <https://www.cbc.ca/news/canada/manitoba/flood-2022-manitoba-analysis-1.6465372>
- [6] Jendryke, M., & McClure, S. C. (2021). Spatial prediction of sparse events using a discrete global grid system; a case study of hate crimes in the USA. International Journal of Digital Earth, 14(6), 789–805. <https://doi.org/10.1080/17538947.2021.1886356>
- [7] Babaei, H. (2017). Flood hazard maps in Manitoba: present status and future provincial plans with respect to climate change. National Research Council of Canada. Ocean, Coastal and River Engineering. <https://doi.org/10.4224/40000415>
- [8] Kienzler, R., Tizzei, L. P., Blumenstiel, B., Nagy, Z. A., Mukkavilli, S. K., Schmude, J., Freitag, M., Behrendt, M., Civitarese, D. S., Simumba, N., Kimura, D., & Hamann, H. (2023). TensorBank: Tensor Lakehouse for Foundation Model Training. arXiv. <https://doi.org/10.48550/ARXIV.2309.02094>
- [9] Sahr, K., White, D., Kimerling, A.J., (2003). Geodesic Discrete Global Grid Systems. Cartography and Geographic Information Science. <https://doi.org/10.1559/152304003100011090>
- [10] Mahdavi-Amiri, A., Alderson, T., Samavati, F., 2015. A Survey of Digital Earth. Computers & Graphics 53, 95–117. <https://doi.org/10.1016/j.cag.2015.08.005>
- [11] Purss, M.B.J.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, 63–71. <https://doi.org/10.3138/cart.54.1.2018-0017>

- [12] Kmoch, A., Vasilyev, I., Virro, H., Uuemaa, E., (2022). Area and Shape Distortions in Open-Source Discrete Global Grid Systems. *Big Earth Data*. <https://doi.org/10.1080/20964471.2022.2094926>
- [13] Kmoch, A., Matsibora, O., Vasilyev, I., Uuemaa, E., (2022). Applied open-source Discrete Global Grid Systems. *AGILE: GIScience Series* 3, 41. <https://doi.org/10.5194/agile-giss-3-41-2022>
- [14] 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. <https://doi.org/10.5194/isprs-archives-XLVIII-4-W12-2024-75-2024>
- [15] Kmoch, A., Sahr, K., Chan, W.T., Uuemaa, E., (2025). IGEO7: A new hierarchically indexed hexagonal equal-area discrete global grid system. *AGILE: GIScience Series* 6, 1–6. <https://doi.org/10.5194/agile-giss-6-32-2025>
- [16] Li, M., 2023. Hexagonal Discrete Global Grid Systems in Topographical and Hydrological Modeling. <https://dx.doi.org/10.11575/PRISM/40787>
- [17] Thalakkottukara, N.T., Thomas, J., Watkins, M.K., Holland, B.C., Oommen, T., Grover, H., 2024. Suitability of the height above nearest drainage (HAND) model for flood inundation mapping in data-scarce regions: a comparative analysis with hydrodynamic models. *Earth Science Informatics* 17, 1907–1921. <https://doi.org/10.1007/s12145-023-01218-x>
- [18] International Organization for Standardization. (2019). Geographic information — Referencing by coordinates (ISO 19111:2019). ISO, Geneva. <https://www.iso.org/standard/74039.html>
- [19] International Organization for Standardization. (2019). Geographic information — Spatial referencing by geographic identifiers (ISO 19112:2019). ISO, Geneva. <https://www.iso.org/standard/70742.html>
- [20] Open Geospatial Consortium. (2021). Topic 21 — Discrete Global Grid Systems — Part 1: Core Reference System and Operations and Equal Area Earth Reference System (OGC 20-040r3). Editor: Robert Gibb. <https://docs.ogc.org/as/20-040r3/20-040r3.html>
- [21] Open Geospatial Consortium. (2023). OGC API — Discrete Global Grid Systems — Part 1: Core (OGC 21-038r1). Editors: Matthew Brian John Purss, Jérôme Jacovella-St-Louis. <https://docs.ogc.org/is/21-038r1/21-038r1.html>



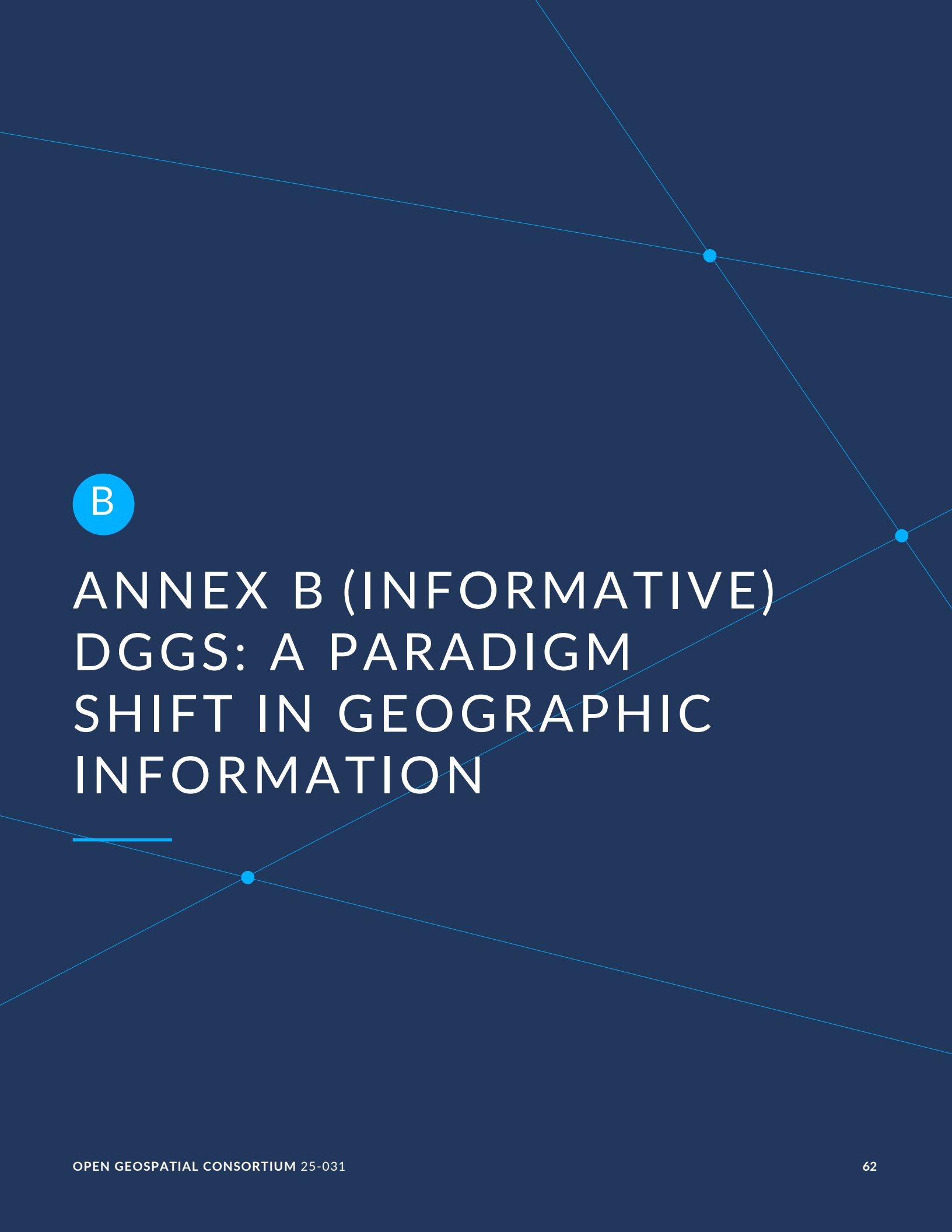
ANNEX A (NORMATIVE) ABBREVIATIONS/ACRONYMS



ANNEX A (NORMATIVE) ABBREVIATIONS/ACRONYMS

AI	Artificial Intelligence
AOI	Area of Interest
API	Application Programming Interface
CDEM	Canadian Digital Elevation Model
CF	Climate and Forecast (Conventions)
COG	Cloud Optimized GeoTIFF
COP	Common Operating Picture
CQL2	Common Query Language 2
CRS	Coordinate Reference System
DGGAL	Discrete Global Grid Abstraction Library
DGGRS	Discrete Global Grid Reference System
DGGS	Discrete Global Grid System
EO	Earth Observation
FAIR	Findable, Accessible, Interoperable, and Reusable
FII	Flood Impact Index
FME	Feature Manipulation Engine
GeoFM	Geospatial Foundation Model
GEPS	Global Ensemble Prediction System
GHSL	Global Human Settlement Layer
HAND	Height Above Nearest Drainage

HIPS	Hierarchical Progressive Surveys
IPT	Identity, Provenance, and Trust
LLM	Large Language Model
MCP	Model Context Protocol
ML	Machine Learning
OGC	Open Geospatial Consortium
OWS	OGC Web Services
RAG	Retrieval-Augmented Generation
RCM	RADARSAT Constellation Mission
SAR	Synthetic Aperture Radar
SKG	Spatial Knowledge Graph
SLLM	Small Local Language Model
STAC	SpatioTemporal Asset Catalog
TIE	Technology Integration Experiment
TWI	Topographic Wetness Index
WASM	WebAssembly
ZIRS	Zone and Inside Reference System



B

ANNEX B (INFORMATIVE) DGGs: A PARADIGM SHIFT IN GEOGRAPHIC INFORMATION

B

ANNEX B

(INFORMATIVE)

DGGS: A PARADIGM SHIFT IN GEOGRAPHIC INFORMATION

Contributor: Michael Jendryke (GeoInsight)

B.1. The Legacy of Layers

For decades, we have organized geographic information the way medieval cartographers organized their knowledge: in layers. A map of roads here, a layer of land use there, contour lines somewhere else. This approach made sense when maps were engraved on copper plates, when each theme required its own physical artifact. But this paradigm has persisted long past its technological necessity, carried forward into digital systems not because it is optimal, but because it is familiar. Discrete Global Grid Systems represent something fundamentally different. DGGS is not just another way to arrange data. It is a paradigm shift in how we conceive of geographic information itself.

Traditional GIS operates from a **data-centric worldview**. We ask “where is my data?” and “how do I project this layer onto that coordinate system?” The focus is on managing datasets, reconciling projections, and performing spatial joins between incompatible geometries. In this model, the world is whatever our data tells us it is, fragmented across countless reference systems, each with its own distortions and limitations. Analysis becomes an exercise in data wrangling, where half the effort goes into simply making different datasets speak to each other.

B.2. World-Centric Representation

DGGS inverts this entirely. It is **world-centric** rather than data-centric. Instead of starting with data and asking where it fits, DGGS starts with the world itself, partitioned into a consistent, global structure of discrete zones. These zones exist independent of any dataset. They are empty containers, waiting to be filled. This may seem like a subtle distinction, but its implications are profound. When you adopt DGGS, you are not resampling your data onto a grid. You are

describing the world through a coherent spatial vocabulary that remains consistent regardless of what data you choose to observe.

This consistency is what makes DGGs the most sophisticated world model we currently possess. Every zone in a DGGs has a precise mathematical definition. The relationships between zones are explicit and computable. Parent-child hierarchies connect scales seamlessly. Neighboring zones can be identified through simple operations. Unlike the patchwork of coordinate reference systems that dominate traditional GIS, where every projection distorts area, shape, or distance in different ways, a well-designed DGGs maintains consistent properties globally. When you describe a phenomenon in zone X at resolution Y, that description is reproducible, comparable, and meaningful in exactly the same way as a description of zone Z at the same resolution, whether Z is next door or on the opposite side of the planet.

B.3. The Power of Single-Index Lookups

This global consistency enables something that traditional GIS struggles with: **single-index lookups**. In a layer-based system, finding all the data relevant to a location requires querying multiple layers, each with its own indexing scheme, then spatially joining the results. In DGGs, you have one index: the zone identifier. Every piece of information about that part of the world, regardless of its source or theme, can be referenced by the same spatial key. This transforms geographic queries from complex multi-step operations into simple database lookups. You are no longer asking “which layers intersect this polygon?” You are asking “what do we know about zone 47382?”

The advantages compound rapidly. Because zones are discrete and bounded, they eliminate the ambiguities that plague vector geometries. There are no questions about whether a point falls exactly on a boundary, no concerns about floating-point precision in coordinate comparisons. Zones either contain a location or they don't. This binary clarity makes DGGs inherently machine-readable; digital. Algorithms do not need to handle edge cases about geometric intersections or topology errors. The structure is clean, predictable, and computationally efficient.

B.4. Analysis-Ready Integration

More importantly, DGGs makes geographic information **analysis-ready from the start**. In traditional GIS, preparing data for analysis often requires extensive preprocessing: reprojecting to common coordinate systems, harmonizing resolutions, cleaning topology, standardizing attributes and finding the spatial units of analysis. These steps are necessary because layers were created independently, for different purposes, in different contexts. DGGs eliminates this friction. Data ingested into a DGGs is immediately interoperable with any other data in the same grid. There is no need for reprojection, because everything already shares the same spatial reference. There is no need for harmonization, because zones provide a natural unit of

aggregation. Analysis becomes a matter of working with zone attributes rather than wrestling with geometries.

B.5. Spatial Tokens and Digital Fingerprints

The result of combining DGGs structure with observational data is what can be called a **Spatial Token**. Each zone accumulates attributes from various data sources, building up a comprehensive profile. A coastal zone might contain information about water temperature from satellite observations, population density from census data, infrastructure from OpenStreetMap, economic activity from mobile phone signals, and environmental risk from climate models. These attributes together form a **Digital Fingerprint** of that location (see [fig-annex-spatial-tokens]), a multidimensional description that captures the state and character of that part of the world. These Spatial Tokens are the fundamental units through which we can understand and model Earth systemically.

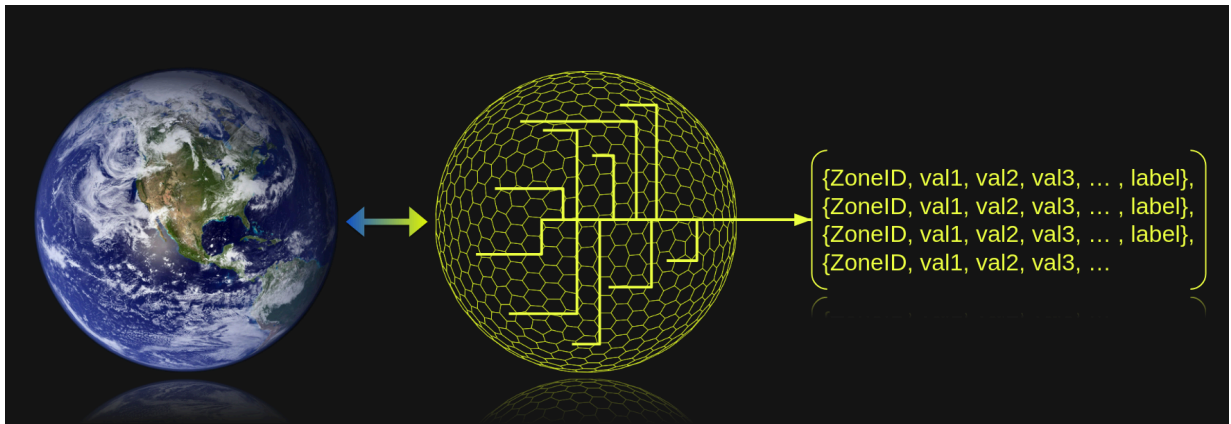


Figure B.1 — The generation of machine-readable Spatial Tokens: creating a digital fingerprint for Earth observations (courtesy of GeoInsight)

B.6. The Substrate for GeoAI

This token-based representation is particularly powerful for Artificial Intelligence. Modern AI excels at finding patterns in structured, high-dimensional data. Language models work with word tokens, vision models with image patches, time series models with temporal windows. But geographic AI has been hampered by the irregular, heterogeneous nature of traditional spatial data. Vector geometries vary wildly in size and shape. Raster grids use different resolutions and projections. Reconciling these differences before machine learning can even begin has been a persistent bottleneck.

Spatial Tokens solve this. They provide AI with a consistent, structured input format. Every zone is a feature vector of fixed dimensionality within a resolution level. The spatial relationships between zones form a graph structure that neural networks can exploit. Hierarchies across scales enable multi-resolution reasoning. Most importantly, the same zone identifiers allow models to be trained on one set of attributes and then applied to others, enabling transfer learning and generalization in ways that traditional spatial data structures cannot support.

This is why DGGs represents more than a technical improvement. It is the foundation for GeoAI at scale. Just as language models required tokenization to transform text into computable units, and just as computer vision required pixel grids to make images processable, geographic artificial intelligence requires Spatial Tokens to make the Earth itself computable. DGGs provides this substrate. It transforms the continuous, complex reality of planetary surfaces into discrete, structured information that both humans and machines can reason about systematically.

B.7. Conclusion

The shift from data-centric layers to world-centric zones is not merely about efficiency or convenience. It is about creating a shared language for describing our planet, one that is simultaneously human-interpretable and machine-actionable, locally detailed and globally consistent. DGGs gives us containers that structure observation rather than data that imposes structure. In doing so, it provides the world model that AI needs to understand Earth not as a collection of disconnected datasets, but as a unified, integrated system. That is the paradigm shift. That is what DGGs truly is.



ANNEX C (INFORMATIVE) HOW TO SELECT WHICH DGGS TO USE?



ANNEX C

(INFORMATIVE)

HOW TO SELECT WHICH DGGS TO USE?

Contributor: Jérôme Jacovella-St-Louis (Ecere Corporation), Alexander Kmoch (Geolynx)

Picking the right grid has important implications for successfully leveraging DGGS technology. Selection should be based on the specific trade-offs between grid parameters such as area variation, shape, refinement ratio, and zone nesting.

C.1. Useful Selection Aspects

C.1.1. Equal-area property

By OGC definition, a DGGS should ideally have < 1% area variation.

- **Not Equal-Area:** H3 ($\pm 50\%$ variation), GNOSIS Global Grid ($\pm 48\%$). These conform to the DGGS “Core” but not the “Equal-Area Earth DGGS” conformance class.
- **Good (<0.5% var):** Default ISEA/IVEA-based grids in DGGRID, HEALPix/rHEALPix (spherical DGGS without authalic conversion).
- **Very Good:** Grids with equal-area projection and authalic conversion (aka on ellipsoid). This includes implementations of IVEA, ISEA and RTEA supporting authalic conversion (including DGGAL and DGGRID through dggrid4py), HEALPix and rHEALPix (including DGGAL implementation) and A5.

C.1.2. Shapes

Within the same resolution or refinement level, different DGGRS offer different geometric properties.

- **Hexagons (6 direct neighbors):** Best for adjacency analysis, flow modeling, and compact representations.

- **Diamonds/Rhombus:** Most similar to classic quadrilateral pixels, making them familiar to users coming from traditional GIS backgrounds.

NOTE: While the projected zone shapes of rHEALPix are squares, on the surface of the globe the inverse projection results in mixed shapes such as circles, triangles and diverse quadrilaterals.

The shape of zones on the surface of the globe relate to the Isoperimetric Quotient (IPQ) compactness score of the DGGRS measuring how close to an ideal circle each zone is.

C.1.3. Resolution steps and Refinement Ratios

The refinement ratio represents the number of child cells that cover the same area as a parent cell. This determines how “abruptly” the resolution changes between zoom levels.

Table C.1 — Comparison of DGGS Refinement Ratios (Apertures)



Figure C.1

Figure 1: Refinement ratio 3
(Hexagons) (e.g., ISEA3H/IVEA3H)



Figure C.2

Figure 2: Refinement ratio
4 (Hexagons) (e.g., ISEA4H)

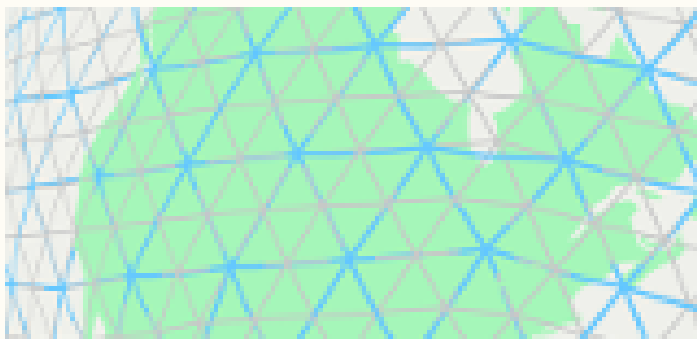


Figure C.3

Figure 3: Refinement ratio 4 (Triangles) (e.g., ISEA4T)

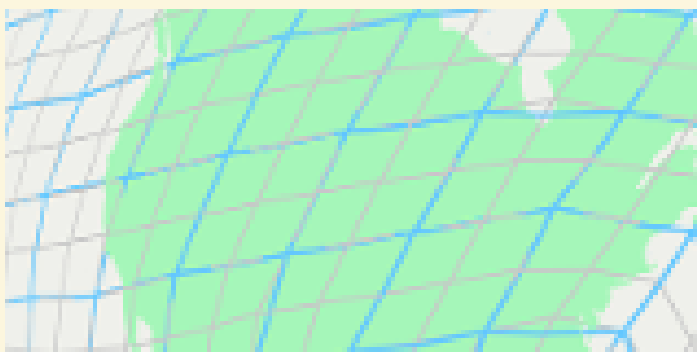


Figure C.4

Figure 4: Refinement ratio 4 (Diamonds/ Rhombus) (e.g., ISEA4D, HEALPIX)

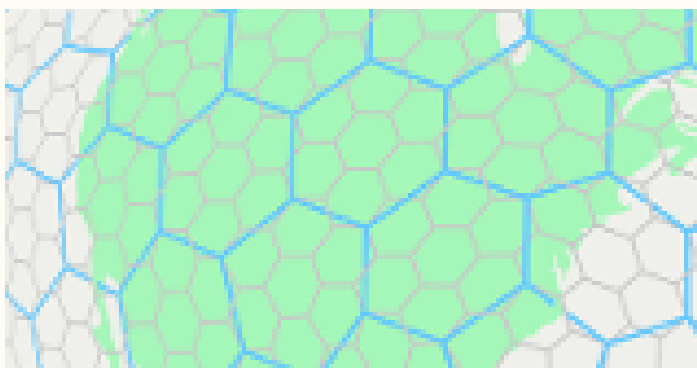


Figure C.5

Figure 5: Refinement ratio 7 (Hexagons) (e.g., H3, ISEA7H/IVEA7H)

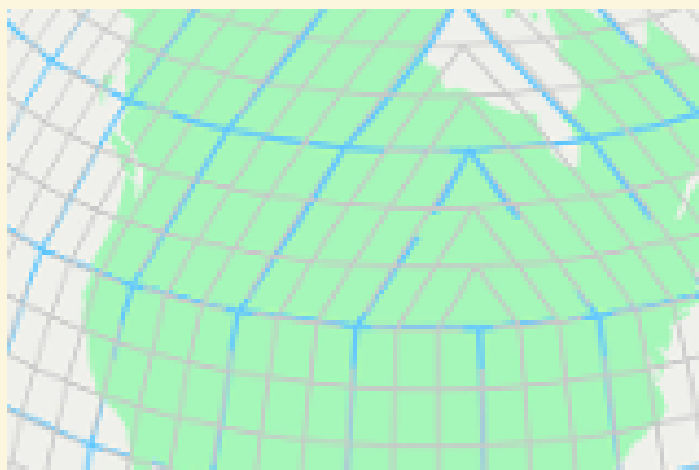


Figure 6: Refinement ratio 9
(Mixed shapes) (e.g., rHEALPix)

Figure C.6

- **3:1 (Aperture 3):** Provides a smooth transition between levels (Figure 1).
- **4:1 (Aperture 4):** Standard quad-tree style, similar to classic tiling systems (Figures 2, 3, 4).
- **7:1 (Aperture 7):** Used by H3; results in a more rapid increase in resolution per level (Figure 5).
- **9:1 (Aperture 9):** High jump in resolution, often used with mixed shapes (Figure 6).

C.1.4. Zone nesting

Zone nesting refers to how perfectly child zones fit into parent zones. **Fully nested** grids (like HEALPix or 4R) are ideal for cadastral applications and simpler indexing because child cells nest exactly within parent cell boundaries. **Almost nested** grids (like H3) have slight overlaps at the boundaries (Friction 3).

C.2. Decision Logic for DGGRS Selection

To simplify the selection process, the following flowchart provides a domain-specific decision tree. It guides users toward a well-justified DGGRS based on their primary needs—such as shape compactness, area precision, or technical constraints.

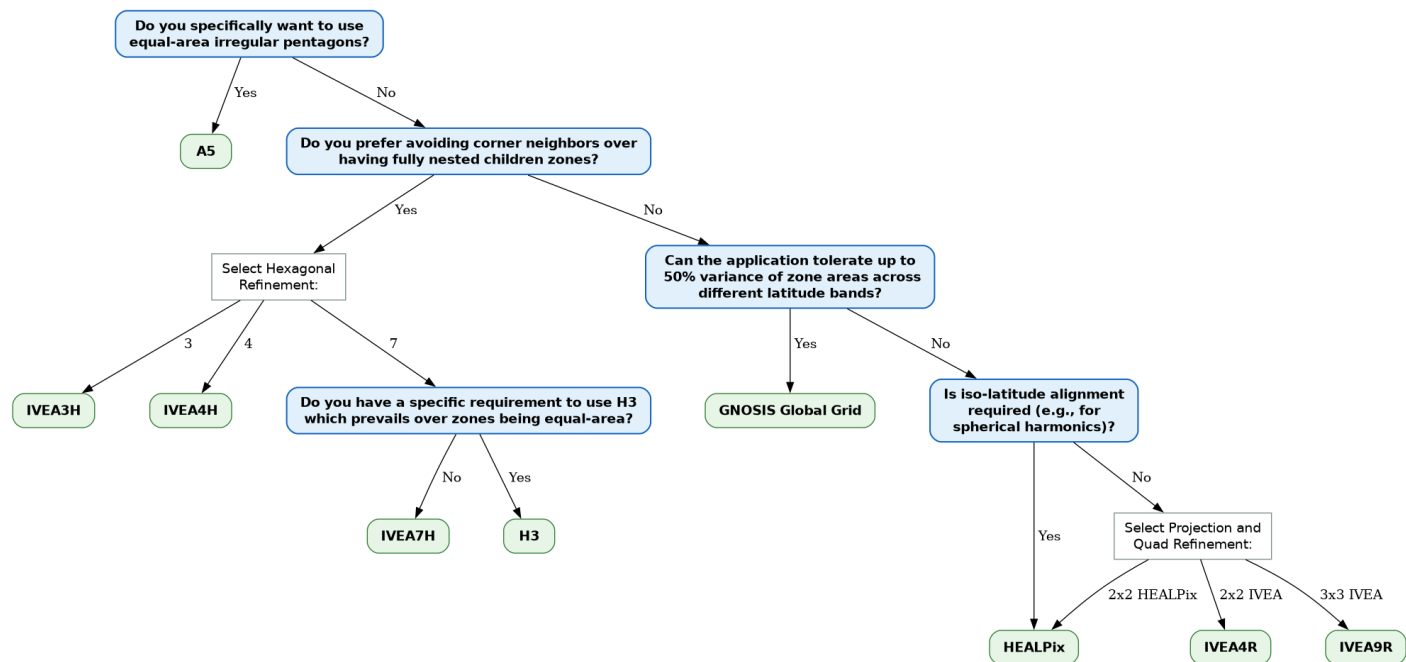


Figure C.7 – DGGRS Selection Flowchart: Balancing shape, aperture, and area variation.

C.3. DGGRS Implementation Matrix

The following matrix summarizes key properties and support across the pilot participants:

Table C.2

DGGRS	PROJECTION	ZONES	REFINEMENT RATIO	EQUAL AREA?	SERVERS	CLIENTS
<u>A5</u>	IVEA	Pentagons	4	Yes	Geomatys	Hartis
<u>GNOSIS GG</u>	Plate carrée	Rectangles	4	No (~48% var)	Ecere, Geomatys, Safe	Compusult
<u>H3</u>	Gnomonic	Hexagons	7	No (~50% var)	Geomatys, Geolynx/UT, GeoInsight, CRIM, Safe	Compusult, Hartis, CS Group
<u>S2</u>	QSC	Square	4	No	Geomatys, Geo Insight	Hartis
<u>HEALPix</u>	HEALPix	Square	4	Yes	Geomatys, Ecere,	Compusult, CS Group, Hartis

DGGRS	PROJECTION	ZONES	REFINEMENT RATIO	EQUAL AREA?	SERVERS	CLIENTS
					Geolynx/UT, GeoInsight, Safe	
<u>rHEALPix</u>	rHEALPix	Squares	9	Yes	Ecere, Geolynx/UT, Geomatys, Geo Insight, Safe	Compusult, Hartis
<u>IVEA3H</u>	IVEA	Hexagons	3	Yes	Ecere, Geomatys, Safe	Compusult, Hartis
<u>IVEA7H</u>	IVEA	Hexagons	7	Yes	Ecere, Geolynx/UT, Geomatys, Safe	Compusult, Hartis
<u>ISEA3H</u>	ISEA	Hexagons	3	Yes	Ecere, Geo Insight, Geomatys	Compusult, TerraFrame, Hartis
<u>ISEA7H</u>	ISEA	Hexagons	7	Yes	DGGRID/ IGEO7, Geolynx/ UT, Ecere, Geomatys, CRIM, Safe	Compusult, Hartis

NOTE: H3 and Area Variation: As detailed in the decision logic, users of H3 must be aware that it conforms to the “Core” DGGS standard but not the “Equal-Area Earth” class due to significant area variance (~50%). For scientific applications requiring strict area consistency, ISEA/IVEA-based hexagonal grids are preferred.



ANNEX D (INFORMATIVE) SERVER BACKBONE: LIBRARY AND TOOLKIT SPECIFICATIONS



ANNEX D

(INFORMATIVE)

SERVER BACKBONE: LIBRARY AND TOOLKIT SPECIFICATIONS

Contributors: Francis Charette-Migneault (CRIM), Jérôme Jacovella-St-Louis (Ecere), Johann Sorel (Geomatys), João Manuel (GeoInsight), Michael Jendryke (GeoInsight), Wai Tik Chan (GeoLynx), Alexander Kmoch (GeoLynx)

This annex provides a breakdown of the open-source libraries that constitute the mathematical and interoperability foundation of the AI-DGGS Pilot.

Table D.1

LIBRARY	PRIMARY RUNTIME	KEY PILOT CONTRIBUTIONS	REPOSITORY
Apache-SIS	Java	Developed a new DGGRS API extending ISO 19111 (CRS) and ISO 19112 (Spatial Referencing by Identifiers).	https://sis.apache.org/
DGGAL	eC (Native)	Provided bindings for Python, C, C++, Rust, Java and JavaScript/WASM. Implemented 7H, 4R, and HEALPix hierarchies.	https://github.com/ecere/dggal
GeoPlegma	Rust	Established an abstraction layer between H3, S2, and DGGRID to ensure cross-grid interoperability.	https://github.com/GeoPlegma/GeoPlegma
pydggsapi	Python	Built a FastAPI-based wrapper for OGC API – DGGS. Integrated with ClickHouse and Geo Parquet backends.	https://github.com/geolynx/pydggsapi
dggrid4py	Python	A wrapper for the DGGRID tool with higher-level operations to bridge with pydggsapi integration.	https://github.com/allixender/dggrid4py
DGGRID	C++	A command-line application that generates and manipulates icosahedral discrete global grids.	https://github.com/sahrk/DGGRID



ANNEX E (INFORMATIVE) SERVER ANALYSIS CAPABILITIES

E

ANNEX E (INFORMATIVE) SERVER ANALYSIS CAPABILITIES

Contributors: Francis Charette-Migneault (CRIM), Jérôme Jacovella-St-Louis (Ecere), Johann Sorel (Geomatys), João Manuel (GeoInsight), Michael Jendryke (GeoInsight), Wai Tik Chan (GeoLynx), Alexander Knoch (GeoLynx), Dean Hintz (Safe Software), Nathan McEachen (TerraFrame)

The following table summarizes DGGs server query, retrieval and analysis capabilities relevant to the AI-DGGs Pilot. These align with OGC API – DGGs and related standards. This includes querying zone identifier for a given bounding box, as well as data retrieval for a specific zone identifier. In addition, various filtering query and selection capabilities are described such as CQL2, subsetting and cross collection queries.

NOTE: Below endpoints are provided with /dgg prefix for simplicity, but could also be contextualized with /collections/{collectionId}/dgg depending on desired results.

Table E.1

CAPABILITY	EXAMPLE REQUEST / USE CASE	WHAT IT DOES / RETURNS
<u>Zone Query (ZQ)</u>	GET /dgg/{dggId}/zones?bbox=-10,35,30,60&zone-level=5	Returns the set of DGGs zones at level 5 whose geometries intersect the given bounding box for which data is available. Helps identify which zones cover or intersect an area of interest. (No filtering by attributes, only spatiotemporal extent, refinement level and optional parent zone)
<u>Filtering Zone Queries with CQL2 (ZQ)</u>	GET /dgg/{dggId}/zones?zone-level=4&filter=propertyX > 100 AND S_INTERSECTS(geometry, POLYGON...)&filter-lang=cql2-text	Extends zone query with the ability to filter on data attributes (e.g., only zones where propertyX > 100), plus spatial relations (e.g., only zones intersecting some geometry) if supporting those CQL2 spatial functions requirements class. The CQL2 filter restricts which zones are returned.
<u>Zone Data Retrieval (ZD)</u>	GET /dgg/{dggId}/zones/{zoneId}/data?zone-depth=2	Fetches data for a given zone, usually at a relative depth greater than zero where values for multiple sub-zones are returned. Can return values like elevation, temperature, or other stored attributes.

CAPABILITY	EXAMPLE REQUEST / USE CASE	WHAT IT DOES / RETURNS
Filtering Zone Data with CQL2 (ZD)	GET /dggs/{dggrsId}/zones/{zoneId}/data?zone-depth=3&filter=temperature > 20 AND date BETWEEN TIMESTAMP('2025-01-01T00:00Z') AND TIMESTAMP('2025-06-30T23:59Z')	Retrieves data within that zone (usually including multiple sub-zones) but replacing by NODATA values for sub-zones not satisfying the filter: e.g., temperature > 20°C during the first half of 2025.
Zone Data Subsetting (ZD)	GET /dggs/{dggrsId}/zones/{zoneId}/data?datetime=2025-04-01T00:00:00Z/2025-04-30T23:59:59Z&subset=atm_pressure_hpa(500:750)&properties=soil_moisture	Fetches data for a given zone, subsetting dimensions not part of the DGGRS like time (datetime or subset=time) and additional dimensions such as atmospheric pressure (subset=atm_pressure_hpa in this example), slicing for a single value, or trimming for an interval as in this example, and selecting specific fields (properties). Other combinations of temporal, vertical, and property subsetting are also supported. They must respect the dimension definitions of the contextual collection(s). Subsetting parameters can also be combined with other Zone Data parameters like zone-depth and filter.
OGC API – Processes: Collection Output with DGGS access / Virtual Collections	POST /processes/{processId}/execution?response=collection with payload specifying a process execution workflow (see coastal erosion example from FMSDI-3 project) which creates a virtual collection, from which data can then be accessed with the usual Zone Query or Zone Data retrieval requests for any supported DGGRS e.g., GET /collections/{virtualCollectionId}/dggs/{dggrsId}/zones/{zoneId}/data?zone-depth=3.	Enables users to derive new collections “on-the-fly” by submitting a workflow integrating data and processing resources (which could potentially be distributed), and then access the results using the DGGS API triggering processing on-demand for zones of interest.
Deriving Fields with CQL2	GET /dggs/{dggrsId}/zones/{zoneId}/data?alias[rainfall_density]=rainfall_mm / area_km2&properties=rainfall_density	Computes new fields from existing fields using CQL2 expressions. Example: derive rainfall density by dividing rainfall by zone area (assuming rainfall_density and area_km2 are properties available from the dataset or collection).
Cross-collection queries (local)	GET /dggs/{dggrsId}/zones?filter=soil_properties.ph > 7 AND land_cover.type = 'forest'	Combined filtering across two datasets (hosted by the same server). In this example, soil_properties and land_cover would be distinct collections, respectively providing properties ph and type. Returns zones satisfying both criteria.
Cross-collection queries (remote /	GET /dggs/zones?joinCollection[land_cover]=https://example.com/	Federated analysis combining data from remote DGGS collections with local ones

CAPABILITY	EXAMPLE REQUEST / USE CASE	WHAT IT DOES / RETURNS
<u>server acting as DGGs client</u>	<code>collections/land_cover&filter= land_cover.type = 'urban' AND soil_ properties.ph > 7</code>	(soil_properties in this example). Useful when data are distributed across multiple providers. Note that this is a preliminary proposed syntax for such queries which require more experimentation.



ANNEX F (INFORMATIVE) DGGS-ZARR ENCODING DRAFT SPECIFICATION



ANNEX F

(INFORMATIVE)

DGGS-ZARR ENCODING DRAFT SPECIFICATION

Contributors: Jérôme Jacovella-St-Louis (Ecere), Johann Sorel (Geomatys), Alexander Kmoch (GeoLynx), Francis Charette-Migneault (CRIM) Wai Tik Chan (Landscape Geoinformatics Lab, University of Tartu)

F.1. Introduction

The current landscape of DGGS standardization across domains and formats is still on-going. Here we summarize the specific metadata requirements for large Zarr archives, so, that a client reading from a DGGS-indexed (e.g. ellipsoidal HEALPix) Zarr archive can decode the structure correctly.

However, the integration of DGGS into EO, climate, and other geospatial data formats is currently in a transitional phase [1] where CF-Conventions provided an accepted foundation [2] (especially in the class CF climate/met/ocean communities), but more robust Zarr-centric DGGS conventions (e.g. via xdggs [3]) are explored with the necessary depth [4]. In addition, the Open Geospatial Consortium (OGC) has accepted Zarr and NetCDF formats as OGC API: DGGS web query response formats [5a],[5b]. However, the OGC's approach towards DGGS has a particular focus on "zones" and "subzones": The area of interest in data queries are typically expressed as a DGGS zone (i.e. a parent zone, a cell of larger extent) requesting data for all zones at relatively deeper level (finer refinement levels, higher resolution); in this context these are the sub-zones. OGC DGGS API aims to be as compact as possible and ideally wants to only transmit data values with zone ids being omitted, these subzones are expected to be ordered deterministically.

F.2. Current DGGs metadata approaches for CF conventions, Xarray(xdggs) and Zarr

F.2.1. CF Convention (1.13) for Healpix

With CF 1.13, Healpix as a grid_mapping with fundamental attributes like indexing_scheme and refinement_level was introduced. This limits the application to general DGGs. The ways to extend are likely open to discussion in the next version.

Table F.1

ATTRIBUTE	VALUE	NOTES
grid_mapping_name	healpix	
healpix_index	integer	The integer-valued HEALPix index coordinates are identified by the standard_name attribute value healpix_index
indexing_scheme	nested or ring or zuniq or nuniq	For “nested” and “ring” scheme, the grid resolution is controlled by the refinement level parameter refinement_level . 0 is the coarsest, the total number of cells on the sphere is $12 * 4^{\text{refinement_level}}$.
refinement_level	integer	Mandatory for indexing_scheme “nested” or “ring”, must be omitted for “nuniq” or “zuniq”.
earth_radius	6371000	implies spherical HEALPix
reference_body	identifier	to identify earth or astronomy

F.2.2. Xarray with xdggs

In the xdggs development community, a minimum set of metadata has been established for any DGGs to implement in xdggs. Still, it is very general and leaves the space for implementation to define additional attributes if needed. The mandatory attributes are :

Table F.2

ATTRIBUTE	NOTES
grid_name	the name of the DGGS grid.
level	the refinement level of the DGGS dataset.

Github: <https://github.com/xarray-contrib/xdggs/blob/main/xdggs/conventions/xdggs.py>

F.2.3. Zarr-conventions

There is a draft convention: [zarr-conventions/dggs](#) recently proposed by Justus Magin.

It uses deeply nested attributes, which netCDF-CF and grid mapping do not support. In the proposal, the convention supports the following attributes:

Attributes:

Table F.3

ATTRIBUTE	NOTES
name	The lower-cased name of the DGGS.
refinement_level	The refinement level as an unsigned integer.
ellipsoid	The ellipsoid used as a reference body (non-mandatory field)
indexing_scheme	
spatial_dimension	Name of the spatial dimension
coordinate	Name of the coordinate
compression	Compression type of the coordinate

xdggs provides an implementation of the draft convention [here](#)

F.3. Core DGGS Metadata Parameters (Zarr)

From the metadata review above, DGGS metadata are developed individually for different applications and are often missing the most crucial part or make it optional, which is the Geodetic Model.

To support massive archives and ensure the distinction between spherical and ellipsoidal models is respected, the following parameters are required within the `grid_mapping` variable derived from the Healpix CF convention or a dedicated dggs attribute structure:

Table F.4

ATTRIBUTE	NOTES
<code>dggs_name/standard_name</code>	The authoritative identifier of the system (e.g., healpix, h3, rhealpix).
<code>indexing_scheme</code>	Must specify the ordering logic (e.g., nested vs. ring for HEALPix, or the specific bit-layout version for H3) to ensure 1D arrays can be mapped to 2D/3D space.
<code>refinement_level</code>	The integer resolution of the grid. In a hierarchical Zarr store (multiscale), this may vary per group, but it is strictly required to interpret the scale of the cell index. Exception, if using the multi-order indexing types “nuniq” or “zuniq” then the <code>refinement_level</code> must be omitted.
<code>ellipsoid / reference_body</code>	Nested object. — <code>type</code> : Explicit declaration of sphere or ellipsoid. — <code>semimajor_axis/inverse_flattening</code> : Required numerical definitions if type is ellipsoidal. This is the critical missing link in CF conventions ; without it, high-resolution H3 or rHEALPix data cannot be accurately aligned with other geospatial layers.
<code>compression (or index_type)</code>	Describes how the coordinate array is stored. — <code>full_array</code> : The coordinate variable contains every cell ID. — <code>ranges</code> : The coordinate variable is a compressed array of [start, stop] intervals. This is essential for handling sparse data in large archives without storing billions of contiguous integers. — <code>sub-zone</code> — another option would be to assume it is of complete global coverage, then it would be implicit, because the order of all IDs within one <code>refinement_level</code> is deterministic
<code>base_zone</code>	required if <code>compression=sub-zone</code> , base zone identifier
<code>relative_depth</code>	required if <code>compression=sub-zone</code> , relative refinement level to the base zone identifier

One would also consider using a single attribute, “dggrs_definition”, that links to the complete DGGRS description, so clients can resolve the indexing method and ellipsoid/reference_body on the fly. But in this case, the clients must provide a default setting if the reference is unavailable.

F.4. Profiles Zarr

While Zarr 2 is still actively used, new projects such as [EOPF](#) have moved to Zarr 3. Therefore we added the Zarr3 profiles to match the latest zarr version.

Table F.5

PROFILE	AXIS SEMANTICS	ILLUSTRATIVE ATTRIBUTES	NOTES
zarr2 or zarr3	Geo/temporal axes in requested CRS	Standard CF/GeoZarr (grid_ mapping, CRS var)	No DGGS axis; not quantized.
zarr2-dggs or zarr3-dggs	First axis = sub-zone order	standard_name=dggrs_index, dggrs_definition=<DGGRS>, indexing_method=sub-zone, base_zone=<base_zone_id>, relative_depth=<level>	Needs base zone + depth.
zarr2-dggs-zoneids or zarr3-dggs-zoneids	First axis = global zone IDs (text or uint64)	standard_name=dggrs_index, dggrs_definition=<DGGRS>, indexing_method=zid	Portable if zone IDs are formally registered.

F.5. JSON Examples

```
{
  "zone_attrs": {
    "standard_name": "dggrs_index",
    "dggrs_definition": "<DGGRS URI or definition>",
    "indexing_method": "zid",
  },
  "zone": ["A6-0-B", "A6-0-C", "A5-0D"]
}
```

Listing F.1 – Zone IDs as text

```
{
  "zone_attrs": {
    "standard_name": "dggrs_index",
    "dggrs_definition": "<DGGRS URI or definition>",
    "indexing_method": "zid"
  },
}
```

```

    "zone": [576777411652222975, 585784609296351231]
}

```

Listing F.2 — Zone IDs as uint64

```

{
  "zone_attrs": {
    "standard_name": "dggrs_index",
    "dggrs_definition": "<DGGRS URI or definition>",
    "indexing_method": "sub-zone",
    "base_zone": "81267fffffffffff",
    "relative_depth": 6
  },
  "zone": [0, 1, 2, 3, 4]
}

```

Listing F.3 — Sub-zone order

F.6. Status

Active discussions are ongoing across CF conventions, HEALPix/HiPS, and EOPF (ESA/Destine DT). Goal: converge towards a **standard DGGS-Zarr profile**.

F.7. Summary Recommendations for Next-Gen NetCDF-CF DGGS Conventions

For the next iteration of CF-Conventions, we must move beyond the piecemeal addition of specific systems (like the current healpix entry) and towards a generic `grid_mapping_name = dggs`. The current trajectory of adding individual names for every system (e.g., H3, S2, rHEALPix) is unsustainable and creates unnecessary friction for software developers. A generic DGGS mapping container allows the software to act as a “driver,” where the specific logic is loaded based on attributes, rather than hardcoding support for every new standard name. This approach mirrors how projections are currently handled: `lambert_conformal_conic` is the logic, but the parameters determine the actual map. Similarly, `dggs` should be the logic, with attributes defining whether it behaves like H3, HEALPix, or a custom discrete global grid.

```

`json { "dggs": { "name": "healpix", "refinement_level": 5, "indexing_scheme": "nested",
"coordinate": "cell_ids", "compression": "ranges", "ellipsoid": { "semimajor_axis": 6378137.0,
"inverse_flattening": 298.257223563, "authority": "EPSG:4326" }, "ogcapi_query_context":
{ "type": "ogc_zone_subset", "parent_cell_id": 12456, "relative_depth": 3, "id_format":
"global_64bit" } } } `

```

```

> parameter "authority": rather alternative to specifying
major/minor axis if well-defined ellipsoid > section

```

"ogcapi_query_context": proposition only for DGGS API (Zarr response)

Figure F.1

The most critical metadata requirement for scientific accuracy is the explicit decoupling of the Topological Model (the grid structure) from the Geodetic Model (the shape of the Earth). Current conventions often dangerously imply that a DGGS index inherently maps to a unit sphere or WGS84. The new convention must enforce the presence of `semi_major_axis` and `inverse_flattening` within the `grid_mapping` variable for any non-spherical data. If a dataset uses HEALPix logic projected onto WGS84 (rHEALPix), the metadata must explicitly state this deviation from the standard spherical definition. Without this, overlaying DGGS cells with standard vectors (GeoJSON) or rasters (GeoTIFF) will inevitably result in geodetic misalignment, rendering the format useless for precision mapping.

To bridge the gap between specific xdggs/Zarr conventions and the abstract OGC DGGS API, we should allow to explicitly define the geometric model (ellipsoid) and the topological context (the “Zone” relationship). The OGC API’s concept of a zone request implies that the returned Zarr package is a subset of the global grid. To harmonize this, the metadata must carry both the absolute definition of the grid (incl. the ellipsoid) and the relative definition of the data package (the subset/subzone context). The compression: ranges method is particularly synergistic here: a “Subzone” request in the OGC API roughly translates to a single contiguous “Range” (or set of ranges) in the cell index space, or as a chunk or shard in the Zarr storage domain. The concept of a “subzone” — a specific parent cell and its hierarchical descendants is a critical factor for performance in cloud-native DGGS. If the data ordering on disk (or object storage) does not align with the hierarchical structure of the grid, the I/O cost of retrieving a specific region becomes prohibitive.

F.8. Changes proposed to the Next-Gen CF-Conventions for arbitrary DGGRS support

Discussion #447 was open on the CF-Conventions repository proposing a solution that would support both global zone identifiers and referencing by deterministic sub-zone order.

F.8.1. Using global zone IDs

This requires one contextual property:

- `dggrs_uri` (this is a URI like e.g., https://www.opengis.net/def/dggrs/OGC/1.0/HEALPix_NUNIQ)

and a dimension mapping allowed to be either text or integer:

- `dggrs_zone_id` (global zone ID as per the DGGRS)

What each property is named does not really matter, but it would be great if the names can be consistent across netCDF / Zarr / CoverageJSON.

This approach can package a whole dataset in a single file, but at the cost of one extra ID per individual set of data values. If the global zone id follows some kind of global space filling curve and the format supports a start..end range then the cost of the zone ID / zone can be avoided, assuming the global zone IDs do not have to be explicitly specified. Depending on the space filling curve, a single zone range may or may not correspond to “root zones” data packets as for OGC API — DGGS .../dggs/{dggrId}/zones/{zoneId}/data (e.g., if the space filling curve exits and re-enters a root zone later).

F.8.1.1. Relevant encoding profiles from OGC API — DGGS

netCDF profiles: netcdf3-dggs-zoneids and netcdf4-dggs-zoneids

Zarr profile: zarr2-dggs-zoneids

CoverageJSON profile: covjson-dggs-zoneids

F.8.2. Using deterministic sub-zone order

This approach is restricted to encoding data for sub-zones of a given root zone, leveraging the deterministic sub-zone order to avoid the need to pair each set of data values with a global zone identifier. If the format’s dimension mapping can express a continuous integer sequence without explicitly storing each integer, or through a basic compression algorithm, this can save significant amount of storage space and bandwidth. For a DGGRS which is not fully nested, this approach also comes at the cost of sub-zones overlapping multiple root zones being duplicated in multiple data packets, requiring an extra step to prune duplicates for translucent visualization or analytics.

Three contextual properties are required to implement the approach:

- dggrs_uri (same as for the global approach)
- dggrs_root_zone_id (this is always text)
- dggs_relative_depth (always an integer, sub-zone level — root zone level)

And a dimension mapping which is always a sequential integer from 0..nSubZones-1 (or 1 — nSubZones — only the order that matters, the convention just needs to specify 0 or 1-based index):

- dggrs_subzone_index (index into the deterministic sub-zone order of the root zone as per the DGGRS)

Since it avoids explicitly specifying the zone identifier, this approach can be significantly more compact, either through compression of the consecutive integer sequence, or if the sub-zone order index can be omitted using a start..end range. The payload is then only the data values

(except for the contextual elements for the whole root zone), but enforces one file/blob/shard with its contextual information per DGGRS root zone. A root zone + relative depth always corresponds to a `.../dggs/{dggrId}/zones/{zoneId}/data` data packet.

F.8.2.1. Relevant encoding profiles from OGC API – DGGS

netCDF profiles: `netcdf3-dggs` and `netcdf4-dggs`

Zarr profile: `zarr2-dggs`

CoverageJSON profile: `covjson-dggs`

F.9. Future work

- define approaches for storing multiple-resolution DGGS data, building on ongoing Zarr community discussions on hierarchical and multiscale datasets.
- define how to store data overlapping multiple root zones.
- define a DGGS tiling approach, inspired by `TileMatrixSet` concepts used in WMTS and OGC API – Tiles.

Future work should also continue aligning DGGS-Zarr conventions with emerging Zarr, NetCDF-CF, and OGC API – DGGS practices.

F.10. References

Zarr, xdggs, and (NetCDF-)CF conventions:

<https://github.com/zarr-conventions/dggs>

https://xdggs.readthedocs.io/en/latest/user_guide/conventions.html

<https://github.com/cf-convention/cf-conventions/pull/605>

<https://github.com/cf-convention/cf-conventions/issues/433>

<https://github.com/zarr-developers/zarr-specs/issues/125>

The OGC API: DGGS standard, supporting NetCDF and Zarr encoded DGGS data packages as response formats:

https://docs.ogc.org/is/21-038r1/21-038r1.html#rc_data-netcdf

https://docs.ogc.org/is/21-038r1/21-038r1.html#rc_data-zarr

https://docs.ogc.org/is/21-038r1/21-038r1.html#_776fdd86-042c-bc02-f7a6-db37514149f4

https://docs.ogc.org/is/21-038r1/21-038r1.html#_54c0ca16-41b8-d3cf-7089-6e1001d77fa5



ANNEX G (INFORMATIVE) CRIM – DGGS API IMPLEMENTATION



ANNEX G

(INFORMATIVE)

CRIM – DGGS API IMPLEMENTATION

Contributor: Francis Charette-Migneault ([Computer Research Institute of Montreal](#) – CRIM)

G.1. Architecture and Design

The CRIM DGGS API (<https://hirondelle.crim.ca/dggs-api/>) runs the `pydggsapi` Docker service along other OGC APIs such as [Weaver OGC API – Processes](#), a STAC API and multiple [WPS climate services](#). It is extensible by design with a docker-compose architecture that allows easy integration with other [geospatial and climate web applications](#) adapted to specific domains of study. The DGGS service relies mostly on the `dggrid4py` interface to invoke `DGGRID` functionalities for zone index resolution and retrieves the corresponding zone data from the pre-processed (Geo)Parquet files stored on the server. Through this microservice architecture disposition, CRIM aims to facilitate integration with other OGC APIs such as to allow performing on-demand calculation and quantization of zonal statistics with OGC API – Processes, and access them using DGGS API. The Parquet file format is selected for its efficient columnar storage that makes it suitable for handling a large amount of zones, while being readily accessible within `pydggsapi` and eventually exposing its metadata and data as an asset through STAC API.

G.2. Implemented Capabilities

The following capabilities are implemented in the CRIM DGGS API through `pydggsapi`.

Table G.1

CONFORMANCE CLASS	SUPPORTED BY THE API
https://www.opengis.net/spec/ogcapi-common-1/1.0/conf/core	#

CONFORMANCE CLASS	SUPPORTED BY THE API
https://www.opengis.net/spec/ogcapi-common-1/1.0/conf/landing-page	#
https://www.opengis.net/spec/ogcapi-common-1/1.0/conf/html	#
https://www.opengis.net/spec/ogcapi-common-1/1.0/conf/json	#
https://www.opengis.net/spec/ogcapi-common-1/1.0/conf/oas30	#
https://www.opengis.net/spec/ogcapi-common-2/1.0/conf/collections	#
https://www.opengis.net/spec/ogcapi-common-2/1.0/conf/umd-collection	#
https://www.opengis.net/spec/ogcapi-common-3/1.0/conf/queryables	#
https://www.opengis.net/spec/ogcapi-common-3/1.0/conf/returnables-and-receivables	#
https://www.opengis.net/spec/ogcapi-common-3/1.0/conf/schemas	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/root-dggs	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/collection-dggs	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/data-retrieval	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/data-subsetting	partially: datetime, properties and exclude-properties
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/data-custom-depths	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/data-cql2-filter	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/data-json	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/data-ubjson	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/data-dggs-jsonfg	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/data-dggs-ubjsonfg	#

CONFORMANCE CLASS	SUPPORTED BY THE API
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/data-geojson	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/data-zarr	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/zone-query	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/zone-query-cql2-filter	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/zone-uint64	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/zone-geojson	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/zone-geotiff	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/operation-ids	#

G.3. Interoperability and Testing

The principal tests of the CRIM DGGS API implementation were performed in collaboration with GeoLynx provided that both implementations leverage the same `pydggsapi` and `dggrid4py` libraries, but differ in their server framework and database configuration. Tests included verifying that both servers could simultaneously support `datetime`-aware and non-temporal DGGRS operations based on different Zarr and Parquet backends, while reporting DGGS collection metadata consistently, allowing zone data aggregations across these types of collections, and supporting different DGGS response formats. These tests allowed validating that both implementations could interoperate seamlessly with equivalent client requests, but demonstrated that DGGS temporal aggregations with non-temporal collections required careful handling and interpolations of the additional dimension to avoid misinterpretation of the data representations in DGGS responses.

Interoperability tests were also conducted with Ecere to validate lower-level operations such as zone index calculations and conversions between Z7 and 7H based DGGRS definitions using `dgga1` Python bindings. Additional tests were performed to validate authalic/geodetic latitude conversion to consider DGGRS and tools that rely on different ellipsoid reference systems. These tests helped identify discrepancies in zone index and coordinates resolutions between `dgga1`, `DGGRID` and `h3` DGGS backend libraries.

For instance, it has been observed that the `dgga1` and `DGGRID` libraries do not use the same grid generation initialization parameters, which leads to offsets in resulting zone geometries and

positions, although their nominal DGGRS definitions are supposedly equivalent. To mitigate these discrepancies, the following **DGGRID** metafile parameters have been identified to better align the tools. Provided that DGGs relies entirely on the agreement of a common DGGRS definition to communicate zone indices between servers and clients, rather than geospatial coordinates, it is critical that these parameters are well defined and documented to ensure interoperability. This reinforced the need for an official OGC DGGRS registry that precisely describes the parameters that characterizes each grid.

```
# grid initialization parameters
dggs_orient_specify_type SPECIFIED
dggs_vert0_lon 11.20
dggs_vert0_lat 58.282525588538994675786
dggs_vert0_azimuth 0.0

# DGGRS definition
dggs_type ISEA7H
dggs_res_spec 12
output_cell_label_type OUTPUT_ADDRESS_TYPE
output_address_type HIERNDX
output_hier_ndx_system Z7
output_hier_ndx_form DIGIT_STRING

# number of points per cell edge
# ensures that rounded polygon edges are well represented,
# or in the right orientation for antimeridian crossing
# (can be further increased for higher resolutions, but implies more
processing/storage of coordinates)
densification 5
precision 12

# grid clipping parameters
# when using geometry clipping to generate a grid subset, the following should
# be set to avoid incorrect zone coordinates precision at higher DGGs
resolutions
clip_cell_densification 5
clipper_scale_factor 100000000
```

**Listing G.1 — DGGRID metafile parameters to align with
DGGAL, specifically for the IGEO7 / ISEA7H+Z7 DGGRS**

Similarly to the above initialization parameter issue, distinct datum definitions are used between the core libraries, which led to different zone geometry results using either authalic or geodetic latitudes. Depending on the interpretation of the CRS and documentation of each tool, integrating these results directly in traditional GIS software produced diverging results, as shown in figure latitude zone shift. In order to align the coordinates on the same datum definition such as to interoperate with traditional GeoJSON output formats and between DGGs API implementations, adequate fixes have been applied to `dggrid4py` and `pydggsapi` to apply the appropriate latitude shift correction when converting between the two definitions. However, the discrepancies remain present with `h3` given the ambiguity surrounding its datum definition and official documentation.

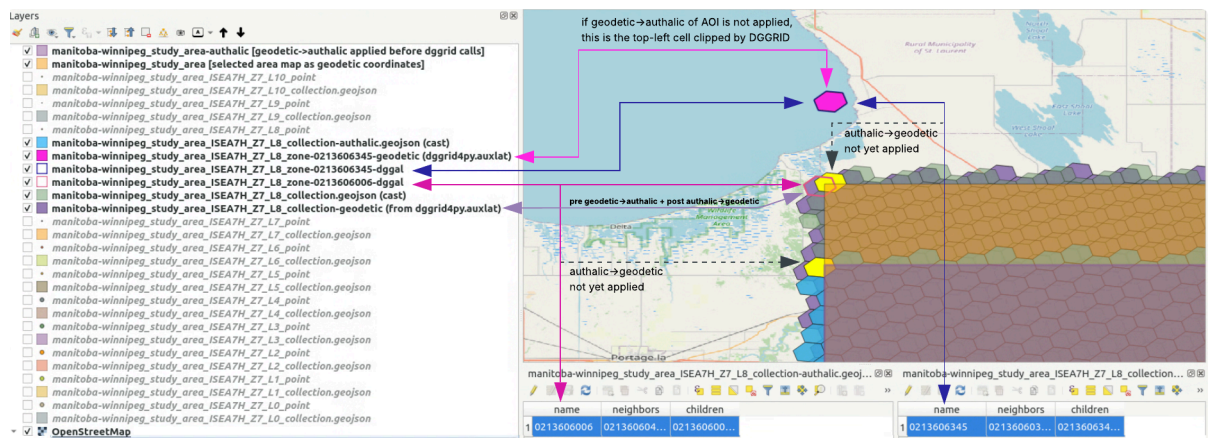


Figure G.1 — Example of Geodetic vs Authalic Latitude Zone Shift by different DGGS libraries without pre/post-processing correction

Initial testing of Parquet and DGGS-UBJSON as storage formats alternatives were also evaluated, to take into account different dataset considerations with regard to DGGS resolution and sparsity, their relevant compression mechanisms, and their potential format-specific encoding optimizations.

Finally, tests of DGGS H3-based zone information queries were performed with CS-Group, although further developments were not performed due to time constraints and the temporal dimension complexities.

All experiments and explorations results performed by CRIM are openly available on its [CRIM OGC DGGS Pilot GitHub Repository](#), which contains Python Jupyter Notebooks and DGGRID meta scripts for evaluated data preparation, quantization strategies and storage tests.

G.4. Performance and Challenges

One of the main challenges encountered during the pilot was related to the quantization strategies to represent the source data (RCM-ARD) into DGGS zones, especially when considering the temporal dimension. While the reference OGC DGGS API standard documents provide guidelines to implement the access mechanisms of the zonal data, few details are provided regarding best practices to prepare and structure the data efficiently. It has been observed during the pilot that efficient use of DGGS API largely depends on how the data is accessed, both from the API internal storage and HTTP request side, which strongly depends on its initial preparation and structure. Failing to apply the adequate access patterns can sufficiently degrade the performance of the API to the point where it becomes impractical or loses its advantages over other traditional geospatial APIs. There is also a lack of recommendations and reference implementations to perform the zone quantization calculation themselves, leading to disparate approaches and results between different server implementations. Even once integrated in the DGGS API, the data structure and metadata information about its quantization remain fairly limited, which can lead to misinterpretation by clients accessing the data for further

analysis and statistics against other DGGs collections. It is therefore recommended that future reference implementation and best practices provide guidance regarding these concerns to ensure consistent and efficient use of DGGs APIs are employed.

Although multiple DGGs definitions are provided in the OGC DGGs standards and reference documents, it has been observed that different implementations (i.e.: `DGGRID` and `dgga1`) may lead to slight variations depending on the initial configuration parameters selected to align the DGGs grid, although they nominally refer to equivalent DGGs. Similarly, distinct authalic sphere and geodetic ellipsoid definitions are used between the core libraries, which led to different zone geometry results (see latitude zone shift) when performing zone index calculations, unless adequate post-processing adjustments are applied. Since results from the libraries do not directly embedded projection information, nor can rely on an explicit DGGs registry reference to identify them, this causes ambiguity in the interpretability of the results. Notably because DGGs relies on servers and clients to agree on the understanding of a common DGGs to access data efficiently and without geospatial coordinates (using the established zone indices instead), these discrepancies contribute to interoperability issues, are harder to detect, and may lead to inconsistent analyses by AI or traditional geospatial systems. It is strongly recommended that OGC establishes a DGGs registry to help annotate the fined-grained definitions of DGGs used by different implementations, and that this registry remains easily extensible to accommodate future DGGs definitions while avoiding locking its entries to older definitions predating its creation.

Finally, it has been observed that the default zone data retrieval format of DGGs-JSON can lead to inefficient response payload, notably when retrieving multiple deeply nested sub-zones over a wide temporal extent that can contain many “holes”. Because this response structure relies on dense 1D arrays per property respecting a specific ordering of the zones according to the reference DGGs employed, multiple `null` empty values must be included to represent missing zones. It is therefore recommended that systems support alternative formats such as DGGs-UBJSON with GZIP compression to compact redundant `null` values, or directly leverage columnar-based formats like (Geo)Parquet including the zone ID, which can mitigate the empty-zones issue entirely by omitting missing values and by compressing the data more efficiently.

G.5. Outlook and Resources

Next steps for the CRIM DGGs API implementation include further optimizations of the data storage back-end and adding support for on-the-fly processing of zonal statistics through OGC API – Processes integration as *Collection Input* and *Collection Output*.

Ideally, STAC metadata definitions for DGGs collections would be further developed, as proposed on corresponding issues of the the [STAC Specification](#) and the [STAC Grid Extension](#), therefore allowing better discoverability and seamless integration between OGC API –DGGs and existing STAC Catalogues.

Finally, CRIM plans to pursue exploring the AI potential of DGGs data representations as well as Provenance metadata integration to DGGs responses, notably to offer better spatiotemporal explainability and lineage of data analysis.

Readers interested in using the open-source implementations or working on extending their capabilities are welcomed to submit issues or contribute directly to the referenced repositories.



ANNEX H (INFORMATIVE) ECERE — DGGS API IMPLEMENTATION



ANNEX H

(INFORMATIVE)

ECERE – DGGG API IMPLEMENTATION

Contributor: Jérôme Jacovella-St-Louis (Ecere)

H.1. Architecture and Design

Ecere served as the primary **Standards Architect** and technical backbone for the pilot, providing the core mathematical foundations and grid geometry logic used across the federated ecosystem. The implementation is centered on two main components: the **GNOSIS Map Server** and the **Discrete Global Grid Abstraction Library (DGGAL)**. Ecere also provided DGGAL “High Vibes”, a set of high-level Python vibe-coded tools providing an implementation of a compact and scalable DGGG-UBJSON(-FG) data store capable of storing both non-rasterized vector and raster data quantized to multiple DGGRSs, with command-line utilities to import or export common raster formats or GeoJSON, clone a collection from a remote OGC API – DGGG implementation and serve the data store through OGC API – DGGG. DGGAL High Vibes thus includes a second OGC API – DGGG implementation by Ecere which is fully open-source (BSD-3 license).

- **Server Framework:** The GNOSIS Map Server is a high-performance, multi-protocol engine written in the **eC programming language**. It provides a native implementation of the OGC API family, including DGGG, Features, and Processes. DGGAL High Vibes includes a second fully open-source OGC API – DGGG implementation.
- **DGGG Library:** DGGAL is the foundational DGGG library developed by Ecere. It provides a common interface for multiple Discrete Global Grid Reference Systems (DGGRS). During the pilot, DGGAL was validated across multiple programming languages (C, eC, Java, Python, Rust, and JavaScript/WebAssembly) and runtime environments to ensure cross-platform integrity (Friction 4).
- **Data Pipeline:** The architecture supports a “Chameleon” data approach, where source data (e.g., GeoPackages or COGs) are re-quantized on-the-fly into the requested DGGRS and refinement level.
- **Deployment:** The server implementation was deployed as a Web API accessible from Ecere’s servers, while DGGAL was provided as a library with bindings for multiple languages, as well as packages for Python and node.js, and DGGAL High Vibes was provided as a set of Python scripts.

H.2. Implemented Capabilities

Ecere provided a full implementation of the **OGC API – DGGS** (Part 1: Core) Standard, extending it with advanced filtering and processing capabilities:

- **Multi-DGGRS Support:** Implemented eighteen standard DGGRSs including **IVEA3H/RTEA3H/ISEA3H**, **IVEA7H/RTEA7H/ISEA7H** (including support for the **Z7** hierarchical indexing for interoperability with DGGRID and IGEO7), **IVEA/RTEA/ISEA 4R/9R**, **HEALPix/rHEALPix**, and the **GNOSIS Global Grid** (a projection-free grid based on a 2D tile matrix set). Ecere also initiated the development of support for the **A5** DGGRS in DGGAL
- **Sub-zones for H3:** Although DGGAL does not yet support H3, the work on similar aperture 7 (7H) grids paved the way towards its support. During the pilot, Ecere clarified some aspects of the H3 grid that are important for an OGC API – DGGS implementation. These aspects include correctly defining the exact zone boundaries as great circle arcs (one or two per edge) on the internal H3 authalic sphere (requiring the inverse projection of multiple points along those arcs to generate the correct refined zone boundaries), converting between WGS84 geodetic and authalic latitude when interfacing with the H3 library, as well as the ability to identify the exact topological sub-zones of a root zone. Ecere provided an annex-h3-subzone-algorithm of the correct sub-zone logic leveraging the existing H3 functionality, which Geomatys also ported to Java and submitted as a contribution to the H3 project (<https://github.com/uber/h3/issues/1095>). Taking these aspects into consideration allows the widely used H3 grid to be used in an OGC-compliant manner while ensuring mathematical rigor (see also Annex Q).
- **Advanced CQL2 Filtering:** Supported complex attribute and spatial filtering at both the `../dggs/{dggrsId}/zones` and `../dggs/{dggrsId}/zones/{zoneID}/data` endpoints, allowing AI agents to perform targeted retrieval.
- **Virtual Collections:** Leveraged the **OGC API – Processes** integration to generate derived collections on-demand. This enabled the calculation of the **Flood Impact Index (FII)** directly within the grid environment.
- **Scanline-Based Iteration:** Developed a deterministic, scanline-based sub-zone ordering for 7H grids, which is essential for efficient data encoding in **DGGS-(UB)JSON** and **DGGS-(UB)JSON-FG** formats.
- **DGGS ViT/MAE:** Ecere also researched the concept of a Vision Transformer (ViT) / Masked Auto-Encoder (MAE) built and trained with a native understanding of a DGGRS.

H.3. Interoperability and Testing

As a lead for the **Server Backbone**, Ecere's implementation was the primary reference for interoperability testing with AI clients (**Compusult**, **CS Group**, and **Hartis**).

- **H3 Alignment (Friction 1):** Ecere worked closely with **CRIM** and **GeoLynx** to resolve discrepancies in coordinate resolution for ISEA / IVEA DGGRSs. Ecere also clarified how to use the H3 library in an interoperable manner consistent with preserving the DGGS topology and in line with the documentation. This includes clarifying that WGS84 geodetic coordinates should be converted to authalic latitude, ensuring a consistent mapping to physical terrain across different softwares.
- **Conformance Validation:** The implementation served as a testbed for the Draft OGC API — DGGS specification. Ecere suggested integrating the discovery endpoint proposed as roadmap enhancements as part of the existing Zone Query functionality (`.../dggs/{dggrsId}/zones`).

H.4. Performance and Challenges

- **Area under the fractal curve (Friction 3):** A major challenge addressed was the non-congruence of Aperture 7 hexagons. Ecere identified and quantified the **~6.52% area misassignment** that occurs when using simple logical indexing. This was mitigated by implementing intersection-based sub-zone discovery (identifying 13 intersecting sub-zones rather than just 7).
- **Standardization Gaps:** The pilot highlighted the urgent need for a formal **DGGRS Registry**. Without a "Source of Truth" for grid definitions, independent implementations will continue to produce slight geometric offsets, undermining the trust required for disaster management.
- **Quantization Extension:** Ecere Identified the potential for a quantization extension to DGGS-JSON so that each data value required as few bytes as possible. For example, quantizing data values to 16-bit integers takes up half the space of a 32-bit single precision floating point number, or a quarter for 64-bit double precision (which might be the default UBJSON encoding for some some conversion tools or libraries). Additionally, leveraging the UBJSON Optimized Containers might also significantly reduce the size of UBJSON blobs. While this is advantageous for reducing bandwidth usage and accelerating data access for clients, this is particularly important for data stores as implemented in DGGAL High Vibes aiming to store a very large number of blobs in the most compact way possible.

H.5. Outlook and Resources

Next steps include refining the **Standardization Roadmap** for an authoritative DGGRS register, evolving the pilot's H3 integration findings into a formal **OGC Best Practice** document, support for additional grids to DGGAL such as A5, IVEA4H, HEALPix_NUNIQ and H3, continued refinement of a DGGS-JSON-FG vector pipeline, improvements to the DGGAL High Vibes tools and continued research on the topic of DGGS ViTs and their integration in AI-DGGS workflows.

Technical Resources:

- **DGGAL Library Repository:** <https://github.com/ecere/dggal>
- **DGGAL “High Vibes”:** <https://github.com/ecere/dggal/tree/main/high-vibes>
- **GNOSIS Map Server Endpoints:** <https://maps.gnosis.earth/ogcapi/>
- **DGGAL Python Package:** <https://pypi.org/project/dggal/>
- **DGGAL Web Demo (using OpenLayers):** <https://dggal.org/web/>
- **DGGAL (WASM) for Web Clients:** [NPM Package](#)

Zone 02136621530
for IVEA7H_Z7 DGGRS (1m)

(View [JSON](#), [ECON](#), [GeoJSON](#) / on [geojson.io](#) representation)

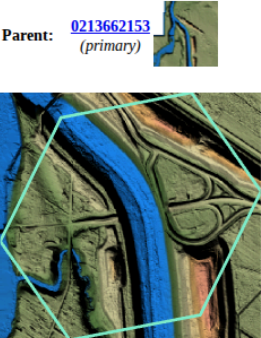
[Back to IVEA7H_Z7 DGGRS zones](#)

Refinement Level: 9
Corresponding 7H Zone: [J0-372966-B](#)
64-bit integer representation: 2519278771878494207 (0x22F646B1FFFFFFFF)
Zone Surface Area: 1.26 kilometers square (0% more than reference mean zone for level 9)
Centroid: { lat: 50.085116723766, lon: -96.928100335478 }
Extent: { { lat: 50.0791488817672, lon: -96.93789254864 }, { lat: 50.0910842174774, lon: -96.9183074297635 } }

[Download data as GeoTIFF](#)

[Download data as DGGS-JSON](#)

Parent: [0213662153](#)
(primary)



Child	Zone ID	Map
Centroid	021366215300	
Vertex	021366215303	
Vertex	021366215301	
Vertex	021366215305	
Vertex	021366215304	
Vertex	021366215306	
Vertex	021366215302	
Secondary	021366215334	
Secondary	021366215316	
Secondary	021366215352	
Secondary	021366215343	
Secondary	021366215361	
Secondary	021366215325	

Neighbor	Sibling	Zone ID	Map
First	Yes	02136621533	
Second	Yes	02136621531	
Third	Yes	02136621535	
Fourth	Yes	02136621534	
Fifth	Yes	02136621536	
Sixth	Yes	02136621532	

Figure H.1 — GNOSIS Map Server serving NRCan’s High Resolution DEM for Red River region through OGC API - DGGS leveraging DGGAL’s new IVEA7H Z7 support



ANNEX I (INFORMATIVE) GEOINSIGHT – DGGS API IMPLEMENTATION



ANNEX I

(INFORMATIVE)

GEOINSIGHT – DGGS API IMPLEMENTATION

Contributors: João Manuel, Michael Jendryke (GeoInsight)

I.1. Architecture and Design

GeoInsight developed a high-concurrency, cloud-native DGGS API designed to unlock global-scale GeoAI. The implementation focuses on turning fragmented “Big Geodata” into harmonized, machine-readable **Spatial Tokens**.

- **Server Framework:** The implementation is built from the ground up using **Rust**, ensuring memory safety, high performance, and efficient data processing.
- **DGGS Engine:** The core functionality is powered by the open-source **GeoPlegma** library, which provides an abstraction layer across multiple DGGRs and robust tools for multi-resolution grid generation and spatial indexing.
- **Storage and Compute (Cloud-Native Pointer Architecture):** The architecture utilizes **DuckDB** as the primary storage access engine for managing metadata and executing lightweight analytical queries. This is complemented by a flexible data access layer that integrates **S3 object storage** for hosted datasets via **Parquet** files and **STAC catalogs** for on-the-fly access to external resources.
- **Quantization Logic:** The system implements an “Any Data, Any Grid” philosophy, performing on-the-fly data transformation for both vector and raster datasets (e.g., Cloud Optimized GeoTIFFs, COGs), enabling dynamic processing without the need for upfront transformation storage.

On-the-fly quantization: any grid, any refinement level, any dataset* - deploy API anywhere

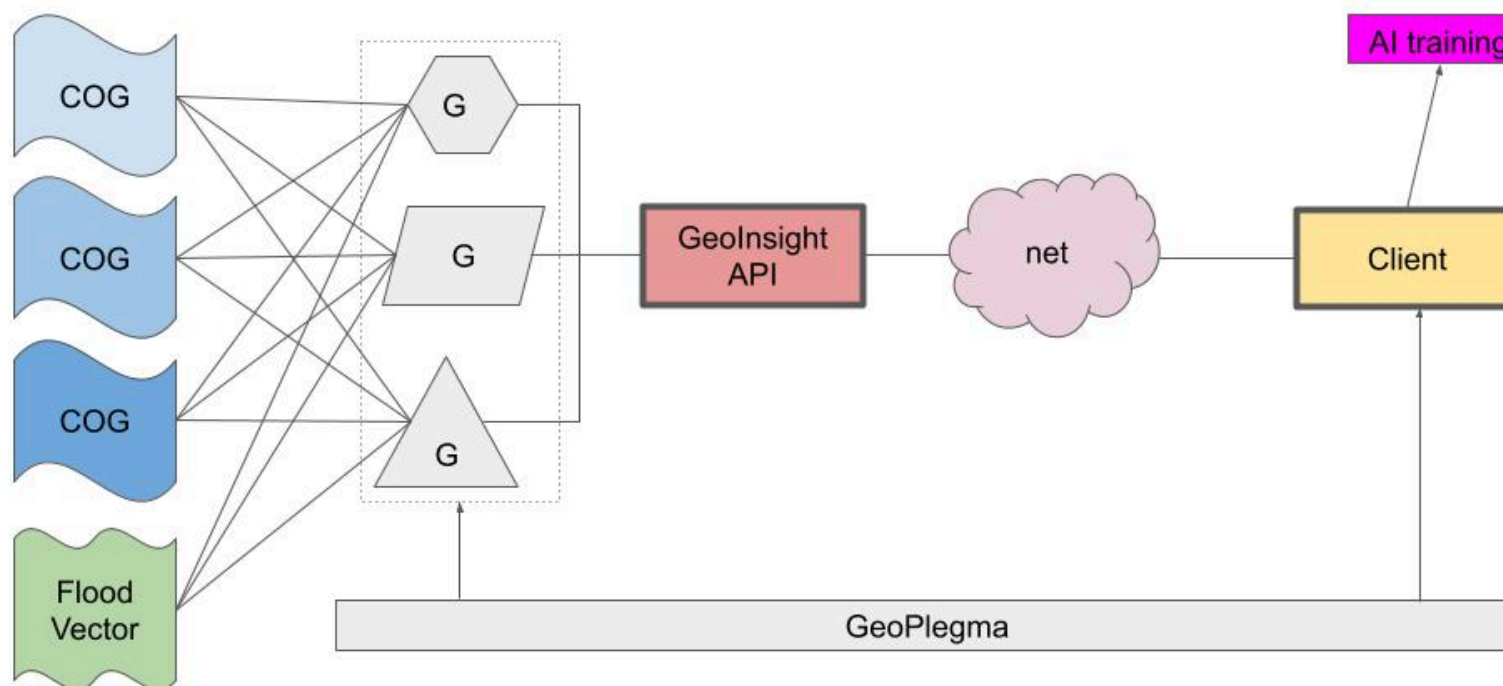


Figure I.1 – Geolnsight: DGGS API Architecture and Spatial Tokenizer Logic

I.2. Implemented Capabilities

The Geolnsight implementation follows a grid-agnostic approach, supporting a wide variety of Discrete Global Grid Reference Systems (DGGRS).

- **Standardized Endpoints:** Full implementation of the OGC API – DGGS standard responses for accessing and exchanging zones and data via HTTP.
- **Supported DGGRSs:** The system enables clients to query and analyze data across multiple hierarchies, including **H3**, **ISEA3H**, **ISEA9R**, **IVEA3H**, **IVEA9R**, **RTEA9R**, and **IGEO7**.
- **Spatial Tokenizer:** The server aggregates heterogeneous sources—including satellite imagery, environmental indicators, and administrative statistics—into discrete zones. These zones act as unique spatial keys or “tokens” that can be plugged directly into AI models, databases, or data platforms.

- **AI-Ready Features:** Beyond simple value retrieval, the system is designed to serve reusable **spatial embeddings**, providing a standardized geospatial representation for institutional and enterprise GeoAI workflows.

The following operations and/or classes are implemented:

I.2.1. DGGs Core class:

We have implemented all Core resources.

- /dggrs endpoint which list the following DGGRS's:
 - ISEA3H (from DGGRID)
 - IGEO7 (from DGGRID)
 - H3
 - ISEA3H (from DGGAL)
 - ISEA9R (from DGGAL)
 - IVEA9R (from DGGAL)
 - RTEA3H (from DGGAL)
 - RTEA9R (from DGGAL)
 - IVEA7H (from DGGAL)
 - IVEA7H_Z7 (from DGGAL)
- .../dggs/dggrsId
- .../dggs/dggrsId/zones/zoneId

We are using the following query parameters:

- **f:** it returns either a json (default) or html format at the moment.

I.2.2. Zone Query:

The zones are implemented on the resource .../dggs/{dggrsId}/zones, as stipulated by the standards. Things to consider regarding the endpoint:

- The default response is a list of all the zoneld's for the respective dggrsId, for a default depth of 2.

- If the endpoint called has a collectionId as a path parameter, we get the collection metadata and if the bbox of that collection exists, then the list of zoneld is filtered and the output is the zoneld's that exist within that bbox.
- It uses the following query parameters:
 - **f**: it returns either a json (default), html, or geojson format at the moment.
 - **parent-zone**: zoneld when used, it will return its children (zoneld) for a default relative depth (which is 4 for GeoPlegma).
 - **zone-level**: is also called as query parameter but it only work with conjunction with the **parent-zone**. It returns the children of the parentId for that level (as long as relative depth doesn't exceeds max relative depth).
 - **bbox**: If the bbox query parameter is used, then it will override the collection bbox as long as this bbox (the query one) is smaller.

I.2.3. Data Retrieval:

We have implemented Zone Data retrieval. For the data retrieval we get the collection metadata and the collection fields metadata. These last one comes from an implemented Queryables resources (collections/{collectionId}/queryables). It then fetches the data based on the query parameters. There are three data sources that it can choose from and fetch or process from:

- Duckdb: a duckdb database that contains the stored zones
- An external bucket stored Cloud Optimized Geotiff (COG) that then get converted to DGGs zones on the fly.*
- Parquet file stored that gets processed*

The response then gets converted by default into a DGGs-JSON. The option for a GeoJson response is also available.

It uses the following query parameters:

- **f**: it returns either a dggs-json (default) or geojson format at the moment.
- **zone-depth**: It's the relative depth that will be used for the parent zoneld. This will get the children which then will be used to get the data (depending on the data source)
- **properties**: We use this parameter on the collection fields metadata, to get the field we want.
- **exclude-properties**: We use this parameter on the collection fields metadata, to exclude the non-desirable fields.
- the file is coming from an url, this url is in the collection metadata.

I.3. Interoperability and Testing

During the pilot, GeoInsight successfully demonstrated cross-server compatibility and multi-vendor interoperability.

- **On-the-Fly Quantization:** Validated the ability to consume any Cloud Optimized GeoTIFF (COG) registered in a collection catalogue and expose it on demand in multiple DGGRSs and resolutions.
- **STAC Integration:** The API alignment with OGC interoperability tests confirmed that it could decouple storage from grid choice, integrate with heterogeneous DGGS implementations, and serve analysis-ready gridded data transparently.
- **Federated Access:** The hybrid storage approach allows the API to federate data from diverse sources while maintaining consistent access through standardized OGC API DGGS endpoints.

I.4. Performance and Challenges

- **Scalability:** By leveraging Rust and DuckDB, the implementation successfully managed lightweight analytical queries over massive datasets without the overhead of upfront grid re-sampling.
- **Caching Strategy:** To optimize performance, a stable caching layer was developed to store resolved grid and data requests as Parquet files. This reduces recomputation costs and enables the efficient re-use of AI-ready outputs.
- **Standardization Gaps:** The pilot revealed ambiguities regarding how grid selection and transformation behaviors should be expressed in a standardized way. GeoInsight is actively contributing proposals to the OGC community to resolve these ambiguities for future API versions.
 - Number of zones in the browser map view or other client
 - refined vs. non-refined geometry
 - Query speed on-the-fly and caching/storage
 - Data quantization
 - cross collection queries
 - large STAC archives (extra query parameter)

I.5. Outlook and Resources

Future developments for the GeoInsight implementation will focus on refining cache invalidation strategies and benchmarking end-to-end performance for continental-scale workloads. Key planned enhancements include:

- **Universal Ingestion:** Adding an optional query parameter to accept arbitrary COG URLs in addition to collection-registered assets.
- **Discovery Automation:** Integrating a STAC-based discovery mechanism to allow any compliant catalogue to serve as a source for gridded inputs.
- **GPU Acceleration:** Further exploring the use of GPU-accelerated pipelines for the generation of high-resolution spatial tokens and embeddings.

Technical Resources:

- **GeoInsight Pilot Hub:** <https://aidggs-pilot.hartis.org/#geoinsight>
- **GeoPlegma Project:** <https://github.com/GeoPlegma/GeoPlegma>
- **API Documentation:** <https://dggs.geoinsight.ai>



ANNEX J (INFORMATIVE) GEOLYNX – DGGS API IMPLEMENTATION



ANNEX J

(INFORMATIVE)

GEOLYNX – DGGS API IMPLEMENTATION

Contributors: Wai Tik Chan(Landscape Geoinformatics Lab), Alexander Knoch (GeoLynx)

J.1. Architecture and Design

GeoLynx, in collaboration with the Landscape Geoinformatics Lab at the University of Tartu and CRIM, developed the **pydggsapi**—an open-source web API designed to provide an easy-to-use discovery and access service for DGGS data.

- **Server Framework:** The implementation is built on **Python 3** using the **FastAPI** framework, chosen for its high performance and native support for asynchronous operations.
- **DGGS Libraries:** To support a wide array of grids, pydggsapi acts as a wrapper for multiple low-level libraries:
 - **DGGRID:** Interfaced via the dggrid4py wrapper for IGEO7 and ISEA7H support.
 - **DGGAL (Ecere):** Utilized for HEALPix, rHEALPix, and IVEA7H hierarchies.
- **Modular Design:** The system uses a “Provider” architecture based on Python abstract base classes (`AbstractDGGRSProvider` and `AbstractCollectionProvider`). This modularity allows developers to add new grid systems or data storage backends (e.g., PostGIS, ClickHouse, or Zarr) by simply inheriting from the base classes and implementing standardized methods.
- **Storage Backends:** The pilot instance validated two primary storage strategies:
 - **Database-driven:** Using **ClickHouse** for high-speed analytical queries.
 - **File-driven:** Using **Zarr** and **Parquet**, which are optimized for chunked access in scientific research and cloud-native environments.

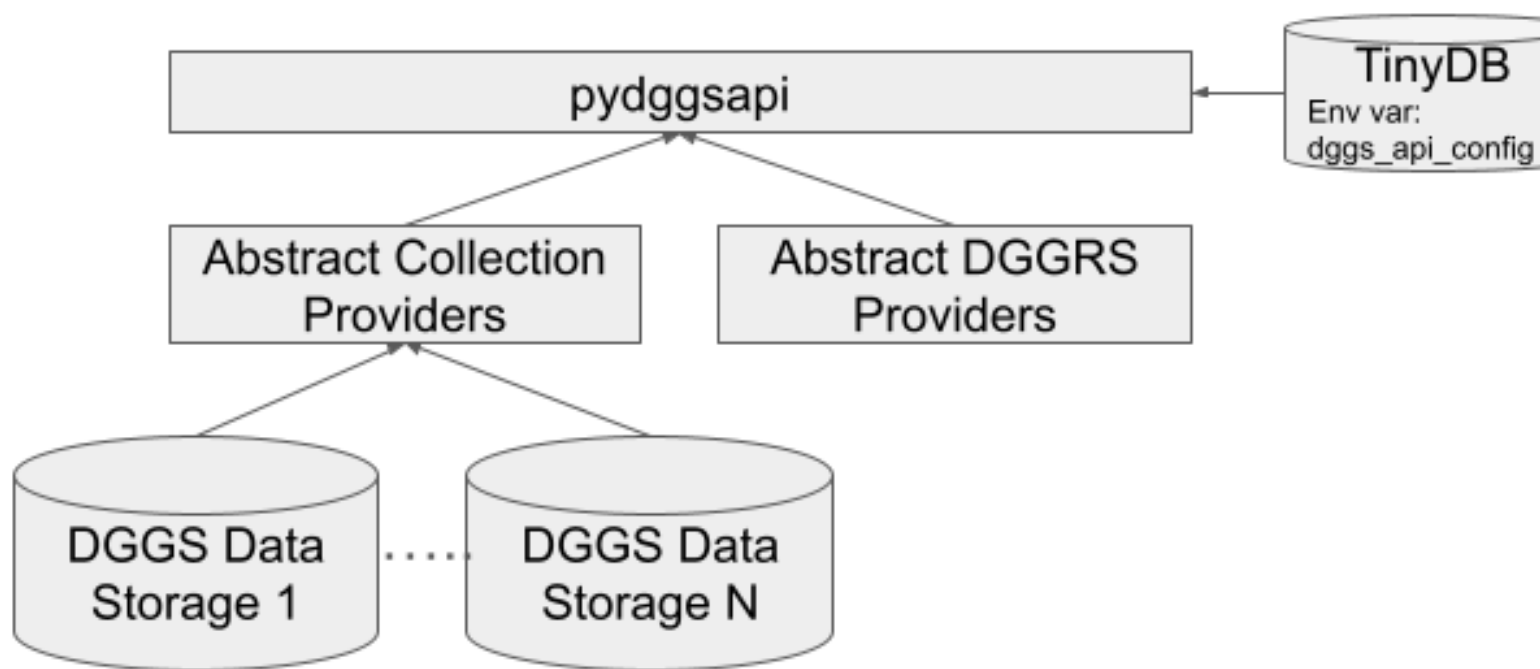


Figure J.1 — pydggsapi Architecture: Modular Provider Design

J.2. 2. Implemented Capabilities

OGC API-DGGS conformance classes

The API implemented the dggrs-core, zone-query and zone data retrieval conformal class. The following list shows the conformance classes supported by the API. Users can also refer to the [example notebook](#) from the pydggsapi documentation for examples of the query endpoints.

Table J.1

CONFORMANCE CLASS	SUPPORTED BY THE API
http://www.opengis.net/spec/ogcapi-common-1/1.0/conf/core	#
http://www.opengis.net/spec/ogcapi-common-1/1.0/conf/landing-page	#
http://www.opengis.net/spec/ogcapi-common-1/1.0/conf/html	#
http://www.opengis.net/spec/ogcapi-common-1/1.0/conf/json	#

CONFORMANCE CLASS	SUPPORTED BY THE API
http://www.opengis.net/spec/ogcapi-common-2/1.0/conf/collections	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/data-retrieval	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/data-subsetting	partially: datetime, properties and exclude-properties
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/data-custom-depths	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/data-cql2-filter	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/zone-query	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/zone-query-cql2-filter	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/root-dggs	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/collection-dggs	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/data-json	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/data-ubjson	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/data-dggs-jsonfg	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/data-dggs-ubjsonfg	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/data-geojson	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/data-zarr	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/zone-uint64	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/zone-geojson	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/zone-geotiff	#
https://www.opengis.net/spec/ogcapi-dggs-1/1.0/conf/operation-ids	#

Mapbox Vector Tiles API

The mapbox vector tiles API endpoint aims to provide an easy way to visualise DGGRS datasets. Users can easily explore DGGRS datasets using GIS software such as QGIS by adding vector layers to the map. This API endpoint works on the collection level.

- (/tiles-api/<collection_id>.json) : It returns the tiles json of the collection
- (/tiles-api/<collection_id>/{z}/{x}/{y}): The tiles source url

The following screenshots demonstrate the use of the tiles API with QGIS.

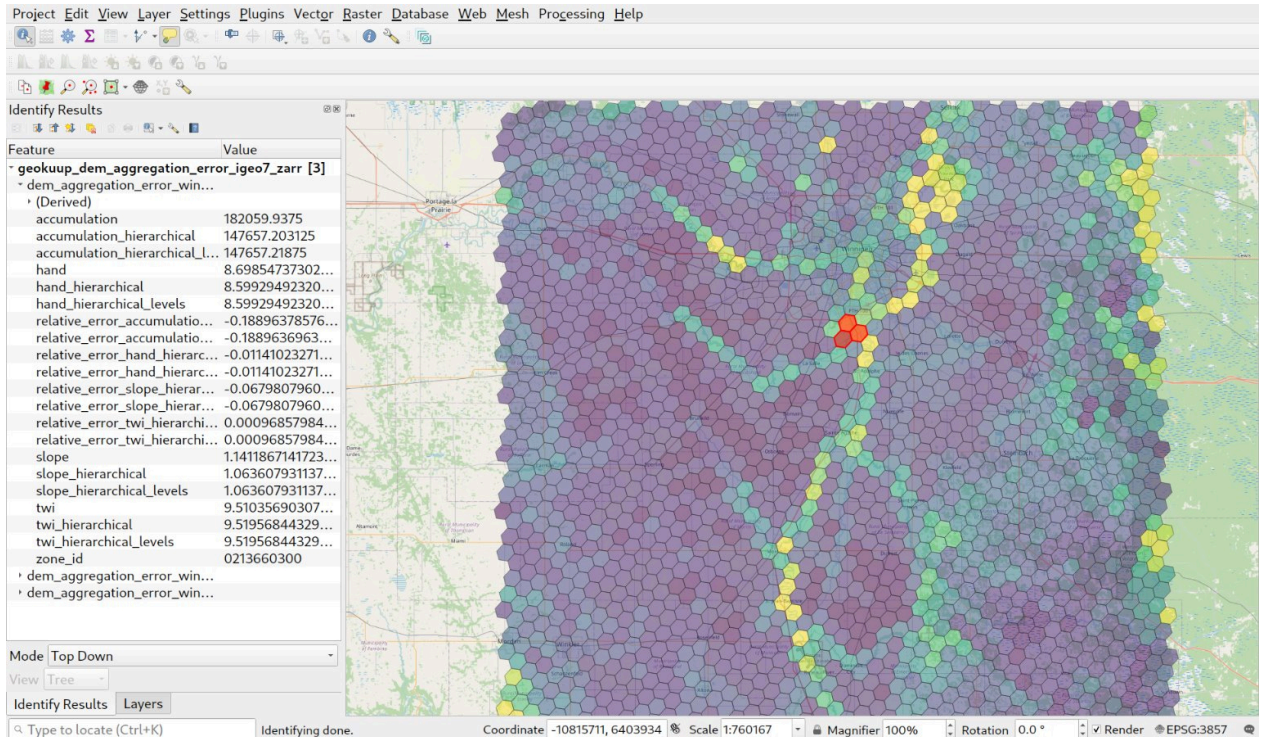


Figure J.2 – Using Mapbox vector tiles API with QGIS to explore a DGGS collection

J.3. 3. Interoperability and Testing

There are two main aspects for testing. The first aspect concerns DGGRS functionality and storage access, and the second concerns validation of conformance classes. We adopted `pytest` to implement test cases to verify the implementation.

Each DGGRS test case consists of tests from each conformance class in the API standard. The following list shows the test cases for a DGGRS :

- test_collection_list

- test_core_dggrs_description
- test_core_dggrs_list
- test_core_dggrs_zoneinfo
- test_data-retrieval_query_zone_data
- test_zone-query_dggrs_zones

We prepared the expected returns for each query and compared them with the actual returns to verify that the implementation worked correctly and conforms to the API standard. The structure minimises testing effort for new DGGRS and storage implementation, as each test case is almost identical, differing only in the dggrs_id query parameter.

One point of discussion was the zone data retrieval, which can return both temporal and non-temporal collections together, creating the illusion of a temporal dimension for the non-temporal data. This might affect data quality in downstream applications. As a result, we refined the API logic to segregate the query into temporal and non-temporal queries, based on the datetime filter parameter.

J.4. 4. Performance and Challenges

During the project, we converted several raster datasets into IGEO7 DGGRS-indexed collections in Zarr format. The sizes of the DGGRS collections, in terms of the number of zones, range from 60 million to 200 million at refinement level 14 (~75 square meter per zone). The most extensive collection comprises 900 million zones at a refinement level of 18 (~0.03 square meter per zone).

We published most of the collections via the pydggsapi instance running on a VM with 16 cores and 32GB of RAM; however, due to insufficient memory, the collection with 900 million zones could not be published. This issue can be traced back to the labelled axes capabilities of Xarray, which is the library that facilitates DGGS Zarr operations through XDGS. Xarray needs to eagerly load the full index dimension, in our case this is the full array of all zone_ids of a Zarr-derialised DGGS dataset.

The performance of pydggsapi started to degrade with finer refinement level (13-14). The FastAPI workers started experiencing timeouts due to the Zarr collection provider being slow to respond to data queries. In the figure below, we performed a simple test to query 1000 zones' data using xarray with the default pandas index on the 200 million zones collection; it took around 49s to complete the initial query. On the other hand, we found that the memory usage of the pandas index increased significantly after the data query.

At the time of writing the Xarray team has proposed a more flexible lazy-loaded index. This could translate to XDGS to two possibilities of improvement: either only load chunks of the index dimension at request time, or to create a virtual index that only describes ranges of zone_ids to calculate the array locations at runtime. Neither of these options could be tested

so far. However, the second option seems to be the direction to move towards to, as global datasets have their index order known because it is a complete global grid, akin to a known sub-zone ordering. In this case, no index need to be materialised (neither in memory nor in the Zarr archive).

```
%memit pandas_index_ds = xr.open_zarr('./pydggsapi_testing_dataset/refinement_level_14', chunks='auto')
peak memory: 4809.15 MiB, increment: 1746.70 MiB

%%time
pandas_index_ds.sel({'zone_id': target_zone_ids})

CPU times: user 46.5 s, sys: 1.29 s, total: 47.8 s
Wall time: 47.8 s

xarray.Dataset

Dimensions:          (zone_id: 1000)
Coordinates:
  zone_id            (zone_id)  int64  2519238812669837311 ... 25192035...
  spatial_ref        ()          int64  ...
Data variables:
  accumulation       (zone_id)  float32  dask.array<chunksize=(1000,), meta=np.ndarray>
  hand               (zone_id)  float64  dask.array<chunksize=(1000,), meta=np.ndarray>
  slope              (zone_id)  float32  dask.array<chunksize=(1000,), meta=np.ndarray>
  twi                (zone_id)  float32  dask.array<chunksize=(1000,), meta=np.ndarray>

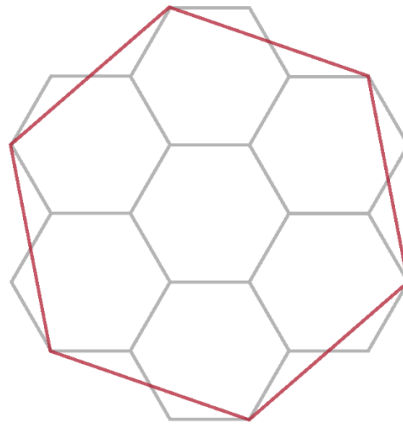
pandas_index_ds.indexes['zone_id'].memory_usage()/1024**3

9.768143765628338
```

Figure J.3 – [Geolynx]A simple test to query 1000 zones with xarray using pandas index

The initial slow responses and the heavy reliance on memory prevent publishing collections with a large number of zones and finer refinement levels. The above findings underscore the need for an indexing method for DGGs-indexed data in Zarr that is both memory- and computation-efficient.

Apart from the challenges arising from computation, aggregating data from a finer to a coarser refinement level also highlights the structure to follow when performing aggregation. The aggregation follows the hierarchical structure of DGGRS (e.g. 1 to 7 hierarchical structure of IGEO7), which provides easy implementation and fast performance for aggregating data. However, it can occur a maximum of 6.5% statistical error (cf. Jérôme Jacovella-St-Louis, Ecere) due to the configuration (not complete congruence of 7H) of refinement ratio 7 hexagonal grids, in which the parent zone does not fully cover all the children and touches children from neighbouring logical parent zones. The following diagram illustrates the incongruence of IGEO7 (similar to H3 in this case).



REFINEMENT RATIO 7 HEXAGONAL GRID
(ISEA7H, IGEO7)

Figure J.4 – [Geolynx] DGGRS IGEO7 hierarchy

The above discrepancy should be studied quantitatively using actual data to identify the conditions under which aggregation method (7 logical vs 14 spatial children overlay) each method should be used, as well as the advantages and disadvantages of each method based on the data.

J.5. 5. Outlook and Resources

Given the challenges mentioned in the previous section, we are working to make use of the improved indexing methods for Zarr data and the xarray/xdggs libraries for large-scale datasets with over 200 million zones. This includes range-based indexing schemes.

Resources:

- Github repo: <https://github.com/LandscapeGeoinformatics/pydggsapi>
- pydggsapi documentation: <https://pydggsapi.readthedocs.io/en/latest/>
- pydggsapi live instance: <https://dggs.geokuup.ee/dggs-api/>
- Xarray's data model: <https://xarray-indexes.readthedocs.io/>
- Xdggs accessor for Xarray: <https://xdggs.readthedocs.io/en/latest/>



ANNEX K (INFORMATIVE) GEOMATYS – DGGS API IMPLEMENTATION



ANNEX K

(INFORMATIVE)

GEOMATYS – DGGS API IMPLEMENTATION

Contributor: Johann Sorel (Geomatys)

K.1. Architecture and Design

Geomatys focused on the challenges of scaling DGGS to handle five-dimensional data (Space, Time, and Vertical dimensions). The implementation is built into the **Examind Server**, a robust Java-based engine designed for high-performance geospatial data management.

- **Server Framework:** The implementation utilizes the **Examind** platform, written in **Java**.
- **Core Library:** Geospatial processing is powered by **Apache SIS** (Spatial Information System). Apache SIS provides the referencing and coverage geometry engine required to map traditional data structures to the DGGS grid.
- **DGGS Backends:** To provide a comprehensive suite of grid systems, the server integrates several specialized libraries:
 - **DGGAL:** Primary abstraction library for most hexagonal and rhombic grids.
 - **CDS-Healpix:** Specific support for the HEALPix grid used in astronomy and global Earth observations.
 - **H3geo & S2geometry:** Support for the industry-standard H3 and S2 grid systems.
- **Resampling Logic:** A key architectural component is the **Resampler**, which allows the server to ingest disparate formats (Shapefiles, GeoTIFFs, GRIB, NetCDF) and resample them into any requested DGGS on-the-fly.

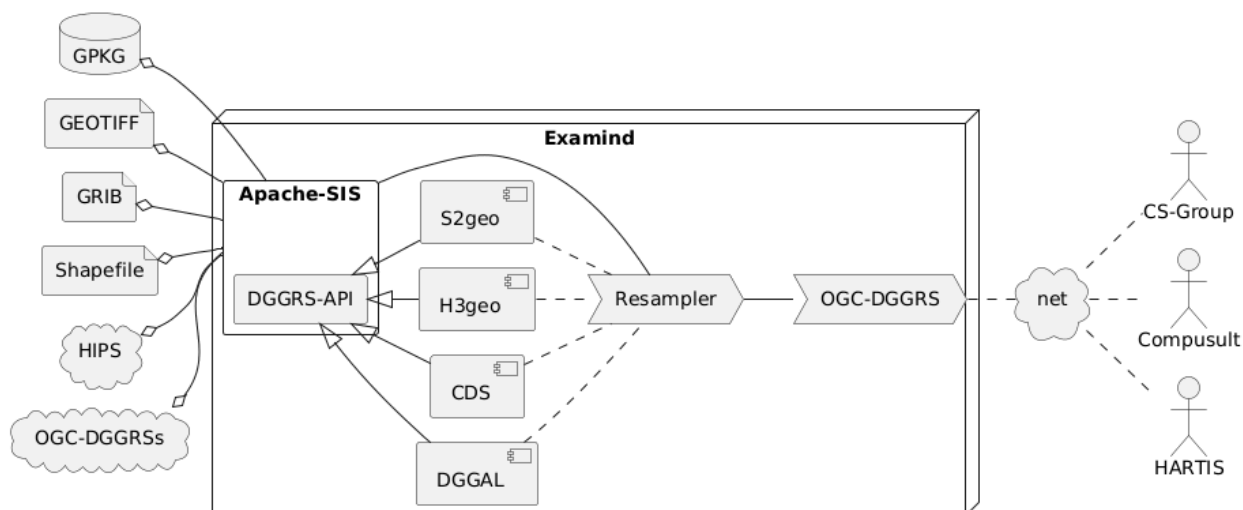


Figure K.1 — Geomatys Examind Architecture: Multi-Source 5D DGGS Serving

K.2. Implemented Capabilities

The Geomatys implementation achieved a high level of compliance, passing **21 out of 25** conformance classes for the OGC API – DGGS.

- **Multi-Dimensional Support:** Demonstrated retrieval of data across 2D (Spatial), 3D (Vertical), and 4D/5D (Temporal) dimensions, allowing AI agents to query flood dynamics over time and height.
- **The /zones Discovery Endpoint:** Geomatys proposed and implemented a prototype endpoint that allows clients to query by geometry (BBOX or Polygon) and receive a list of matching zones. This significantly simplifies the navigation of grid hierarchies for AI agents that may not have prior knowledge of specific Zone IDs.
- **Output Formats:** Supported a wide array of encodings including **JSON**, **UBJSON**, **GeoJSON**, **GeoTIFF**, **NetCDF**, **PNG**, and the cloud-native **Zarr3**.

K.2.1. Produced Instances and Data Examples

The following table showcases the variety of data sources successfully served through the DGGS interface during the pilot:

Table K.1 — Data examples provided by the Examind server

NAPL photo in H3

Bluemarble satellite image in A5

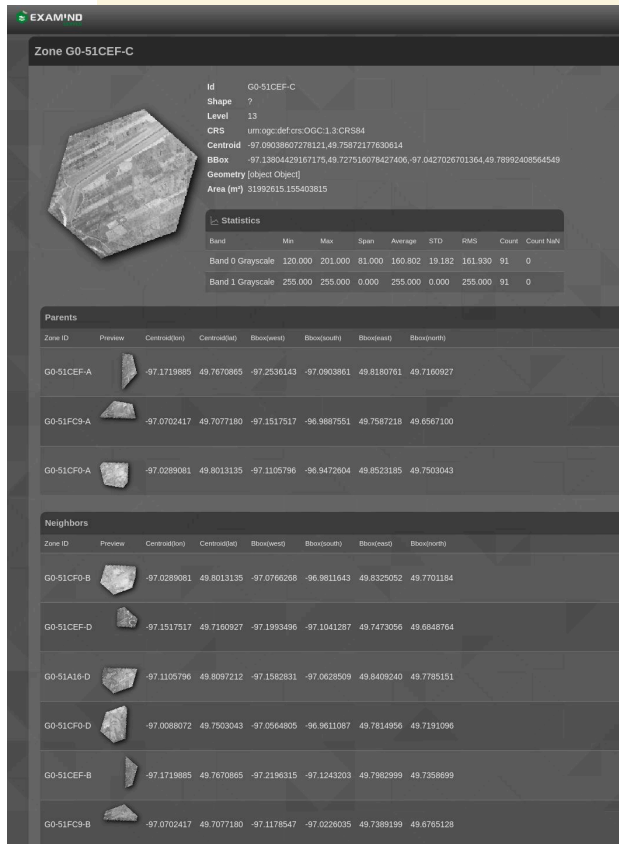


Figure K.2

Roads shapefile in S2

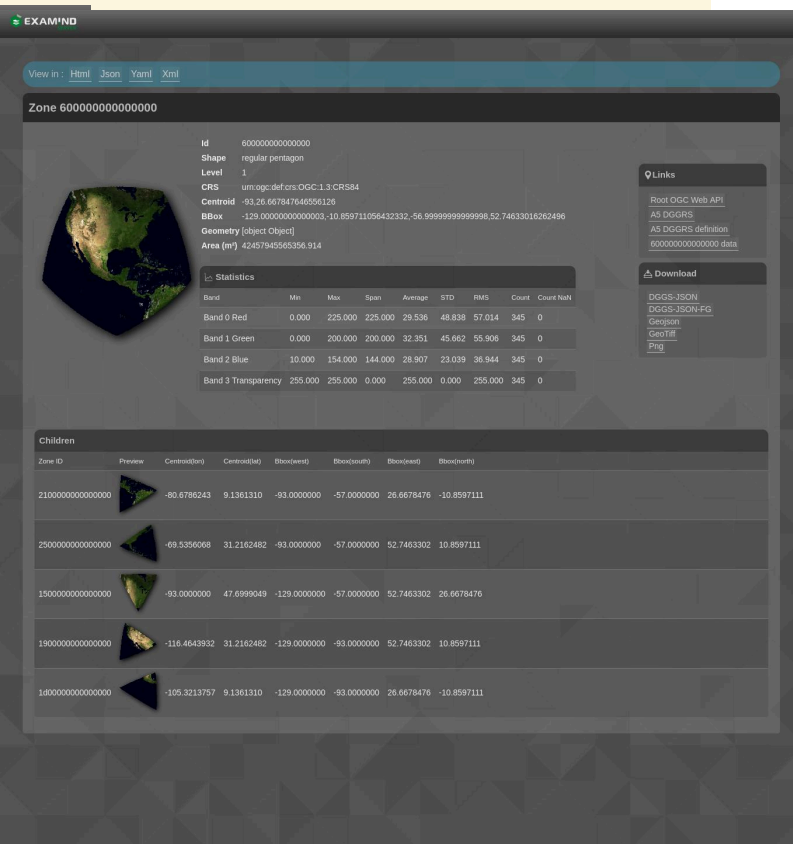


Figure K.3

Fermi telescope in Healpix

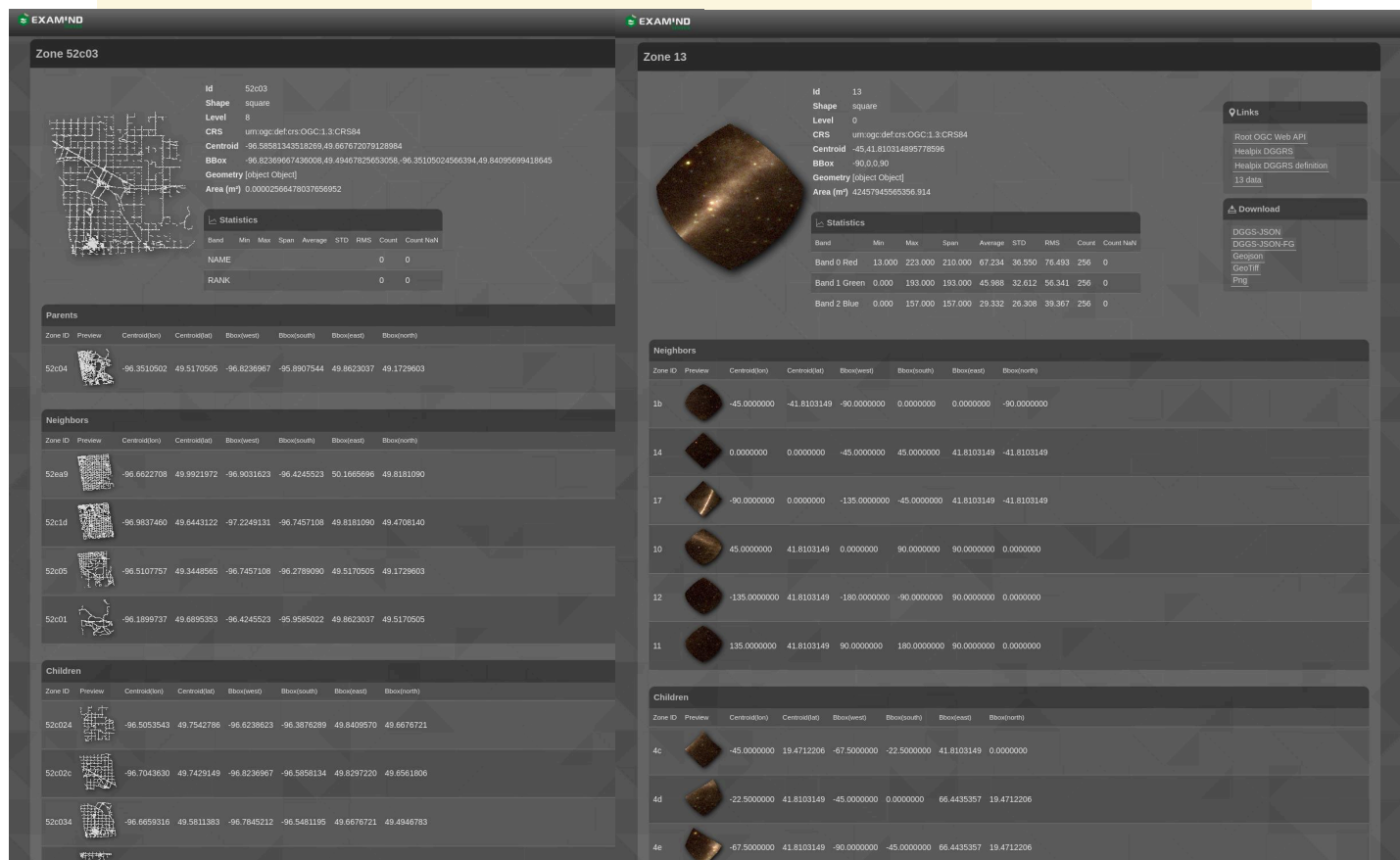


Figure K.4

Figure K.5

K.3. Interoperability and Testing

Geomatys provided the **Server Backbone** for several **Technology Integration Experiments (TIEs)** with AI clients including **CS Group**, **Compusult**, and **Hartis**.

- **Standardization Contributions:** The implementation provided critical feedback to the OGC DGGRS SWG, resulting in a technical proposal for the /zones endpoint and refinements to the **Zarr DGGRS metadata** profile (documented in Annex F).
- **Cross-Community Interoperability:** Developed a connector for **Hierarchical Progressive Surveys (HIPS)**, a standard from the astronomy community, proving that OGC API – DGGRS can serve as a bridge to vast datasets already encoded in Healpix format.
- **Apache SIS Integration:** Created a reference programming API and client for DGGRS as part of the official Apache SIS project.

K.4. Performance and Challenges

- **The Stacking Paradox and Sub-zones:** Initial testing relied on the S2 library for sub-zone computation, which led to performance bottlenecks and inaccurate containment results (Friction 3). Geomatys successfully replaced these generic geometric intersections with faster, grid-specific logic for H3, S2, and Healpix, though challenges remain for the A5 (pentagonal) grid.
- **Data Type Constraints:** The implementation highlighted a limitation in the current DGGS JSON format, which assumes numeric value types. Geomatys successfully relaxed this constraint to support categorical data (strings, booleans) necessary for land-cover and status-based attributes.
- **Image API Limitations:** Java's standard image APIs lack native support for Int64 types. Geomatys mitigated this by utilizing Float64 (double) to handle raw long bits, a key takeaway for future high-precision data handling in DGGS.
- **Scalability:** Performance tests indicated that while simple queries were nearly instantaneous, complex sub-zone discovery at high refinement levels (Level 7+) requires rigorous topological optimization to maintain response times under one second.

K.5. Outlook and Resources

Geomatys will continue to refine the multi-dimensional query capabilities of Examind and advocate for the formal adoption of the /zones endpoint within the OGC API – DGGS standard.

- **API Endpoint:** <https://server.examinind.com/examinind/WS/dggs/ogcaipilot/>
- **Apache SIS Project:** <https://sis.apache.org/>
- **DGGAL Java Bindings:** [Source Repository](#)



ANNEX L (INFORMATIVE) SAFE SOFTWARE – DGGS API IMPLEMENTATION



ANNEX L

(INFORMATIVE)

SAFE SOFTWARE – DGGS API IMPLEMENTATION

Contributors: Dave Campanas, Dean Hintz, Kailin Opaleychuk (Safe Software)

L.1. Architecture and Design

Safe Software demonstrated how enterprise ETL (Extract, Transform, Load) platforms can function as a “universal translator” for OGC DGGS standards. The implementation utilized the **FME (Feature Manipulation Engine)** platform to serve DGGS data via no-code automation.

The architecture consists of **FME Flow Hosted** (deployed on Amazon AWS) running an instance of **FME Flow** in a Linux environment. Workflow authoring was conducted using **FME Form** on both MacOS and Windows. To support the mathematical requirements of the grid, Safe Software developed custom **FME Custom Transformers** (encoders, decoders, indexers, and queriers) that wrap the open-source **DGGAL** (Discrete Global Grid Abstraction Library) for use within FME. These DGGS custom transformers were published on [FME Hub](#) to make it easy for any FME user to access.

FME Flow’s data virtualization environment was utilized to publish these workflows as RESTful services conforming to OGC API standards. This setup allowed for the dynamic conversion of traditional geospatial formats into grid-native spatial-temporal representations at multiple resolutions.

L.2. Implemented Capabilities

The implementation provides a compliant **OGC API – DGGS** interface supporting grid zone discovery, metadata access, and geometry delivery (including GeoJSON and DGGSJSON formats). A key feature is the support for **CQL2 (Common Query Language)**, allowing AI agents and clients to perform attribute, spatial, and temporal range queries.

FME's Data Virtualization (DV) platform was used to implement the OGC API DGGS. This was entirely a no code process based on OpenAPI.

OGC API creation steps:

1. Created a new Data Virtualization API definition.
2. Imported DGGS OpenAPI description. This creates a full API description with all possible endpoints. It also creates empty FME workspaces with required inputs and outputs predefined.
3. Edited FME workspaces to produce the appropriate result based on the relevant API input parameters.
4. Configured the API landing page and any other static responses like capabilities documents in order to improve performance.
5. Configured caching

The resulting Open API configuration for DGGS can be seen in the FME Flow screenshots below which include the API configuration page and the associated autogenerated swagger page.

OGC API - Discrete Global Grid Systems 1.0

dggs

Endpoints Workspaces Schemas Security API Details

DGGS Description

- GET /collections/{collectionId}/dggs
- GET /collections/{collectionId}/dggs/...
- GET /collections/{collectionId}/dggs/...
- GET /dggs
- GET /dggs/{dggId}
- GET /dggs/{dggId}/zones/{zoneId}

DGGS Data Retrieval

- GET /collections/{collectionId}/dggs/...
- GET /dggs/{dggId}/zones/{zoneId}/...

DGGS Zone Query

- GET /collections/{collectionId}/dggs/...
- GET /dggs/{dggId}/zones

Landing Page

GET /collections/{collectionId}/dggs/{dggId}/zones Cancel Save

Details Parameters Response Caching

Path Parameters
Path parameters added via { } in the Details tab will appear here

NAME	TYPE	DESCRIPTION
collectionId	Choose a type (optional)	Local identifier of a collection
dggId	Choose a type (optional)	Identifier for a supported Discrete Global Grid System

Query Parameters Add

NAME	TYPE	DESCRIPTION	REQUIRED
bbox	Array X v Number v	Only resources that have a geometry that interse	☐
bbox-crs	String X v	crs for the specified bbox	☐
compact-zones	Boolean X v	If set to true (default), when the list of DGGS zon	☐
parent-zone	String X v	The optional parent zone parameter restricts a zi	☐

Figure L.1 – OGC DGGS API Configuration on FME Data Virtualization

GET

/collections/{collectionId}/dggs/{dggsId}/zones/{zoneId}

Retrieve information about a DGGRS Zone, such as geometry and data availability, in the context of a specific collection.

Parameters

Cancel

Name	Description
dggsId * required (path)	Identifier for a supported Discrete Global Grid System dggsId
zoneId * required string (path)	Identifier for a specific zone of a Discrete Global Grid Systems. This identifier usually includes a component corresponding to a hierarchy level / scale / resolution, components identifying a spatial region, and a optionally a temporal component. zoneId
collectionId * required (path)	Local identifier of a collection collectionId
f string (query)	The format of the response. If no value is provided, the accept header is used to determine the format. Accepted values are 'json' or 'html'. f

Execute

Responses

Code	Description	Links
200	Example Value <pre>"string"</pre>	No links

Figure L.2 – OGC DGGRS API Data Virtualization Swagger Page

L.2.1. Supported DGGRS

The implementation supports on-the-fly re-quantization to several reference systems via DGGAL, including GNOSISGrid, ISEA9R, ISEA3H, IVEA9R, IVEA3H, RTEA9R, RTEA3H, rHEALPix. More details related to the supported DGGRS and associated methods can be found in the DGGRS transformers' documentation such as the [DGGIndexer](#).

Table L.1 – Supported Discrete Global Grid Reference Systems (DGGRS)

GNOSIS GRID	ISEA9R	ISEA3H	IVEA9R	IVEA3H	RTEA9R	RTEA3H	RHEALPIX
✓	✓	✓	✓	✓	✓	✓	✓

L.2.2. Data Quantization and Ingestion

A variety of source datasets were consumed by the FME platform and made available via the DGGRS:

- **Manitoba DEM:** 5m Digital Elevation Model (DEM) for southern Manitoba streamed on-the-fly as DGGS zones from Cloud Optimized GeoTIFF (COG).
- **Climate Projection Time Series:** Climate model derived temperature and precipitation, monthly time series for 2020–2060. The climate data is based on the MIP5 RCP4.5 model runs downscaled to southern Manitoba. These NetCDF data cubes were made available via DGGS layers.
- **2011 Flood Model:** Daily time-series flood levels from spring 2011 for an area of interest south of Winnipeg. This was based on results from a MatLab model run conducted by RSS Hydro and Dartmouth Flood Observatory (DFO) for OGC Disaster Pilot 2021 to cover daily timesteps from the flood of 2011 from Mar 1 to May 30.

For the DEM and Climate datasets, the ETL workflows provided **on-the-fly data access** directly from the source datasets (DEM Cloud Optimized GeoTIFF and Climate data NetCDF). Even though the dataset volumes were considerable, the on-the-fly translations scaled efficiently within the FME environment.

In contrast, the flood time-series data cube was read from the model GeoTIFF results and loaded into a **PostGIS** database as DGGS-indexed points to optimize complex filtering. Client CQL2 queries are translated into SQL and executed against the database, allowing for high-performance streaming of DGGS data limited to specific zones and values. This allows clients to perform complex reasoning, such as asking: **“Which land areas had flooding greater than 1.5 meters in the first week of April?”**

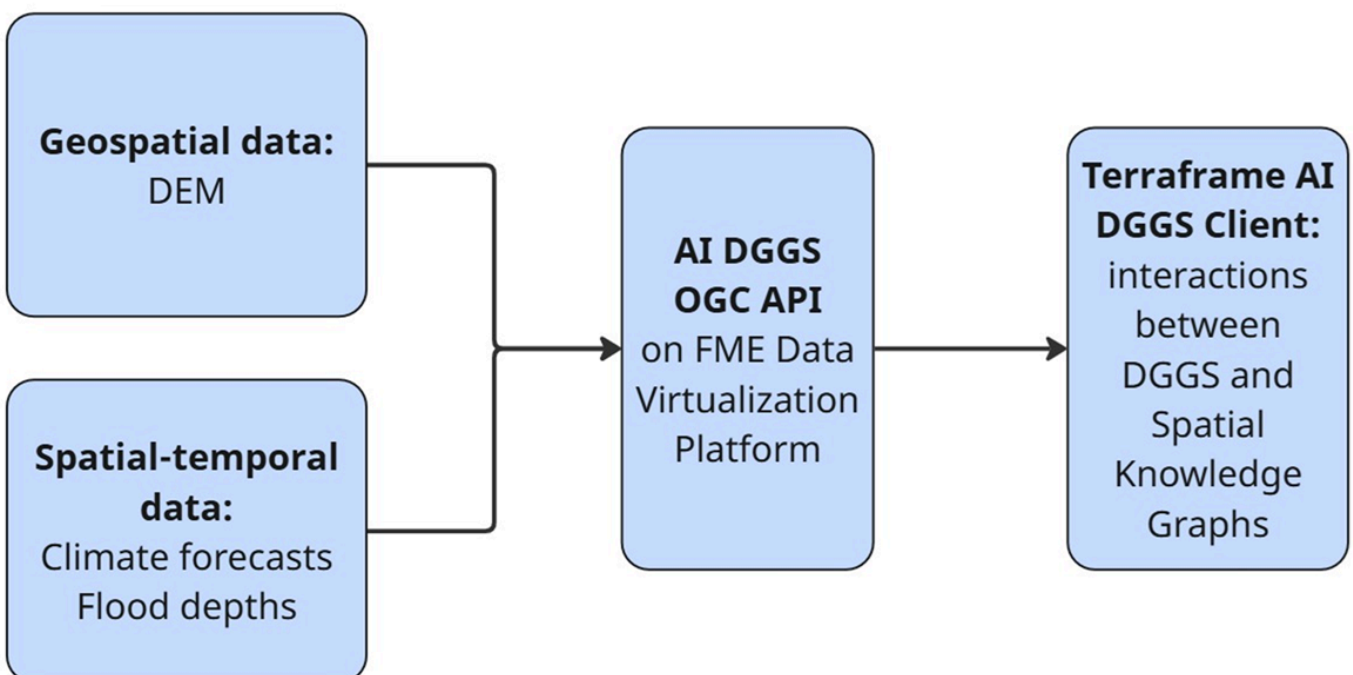


Figure L.3 — DGGS Service Data Ingestion and Provision

L.2.3. Real-time Data Streaming Simulation

An experiment was conducted to test the utility of DGGS for delivering real-time data streams. The “Current flood level” layer was simulated based on the 2011 flood time-series stored in PostGIS. In this configuration, the flood level returned is a function of the current system time, meaning sequential requests return sequential time slices of the historical data.

This experiment demonstrated the potential for DGGS to deliver natural hazard risk information in a real-time manner, which is essential for time-sensitive disaster response. Future applications could replace modeled data with live IoT sensor feeds to provide immediate situational awareness.

L.2.4. Sample Queries

FME’s OGC DGGS implemented a full range of service methods as defined in OGC’s DGGS Open API definition. This includes queries for collection metadata, zone id retrieval such as retrieval of zone ids for a given bbox, as well as data retrieval for a given set of zone ids. In addition, various filter query and selection facilities are supported such as CQL2, zone subsetting.

The following live endpoints demonstrate the API’s capabilities:

- **Data Collection Overview:** [Collection Metadata](#)
- **Zone Discovery (ISEA3H, Zoom 12):** [Zone Query Example](#)
- **Zone Data Retrieval (ISEA3H, Depth 7):** [Data Retrieval Example](#)

The following table lists all the configured DGGS API endpoints and associated FME workflows published to support them.

☐	ENDPOINT	TAGS	LAST UPDATED	WORKSPACE STATUS	WORKSPACE ^
☐	GET /	Landing Page	2026-1-20 16:31:19	Assigned	DV_OGC API - Discrete Global Grid Systems_Workspace_5f6801c12.fmw
☐	GET /api	API	2025-10-22 13:57:11	Assigned	DV_OGC API - Discrete Global Grid Systems_Workspace_90e2c3598.fmw
☐	GET /api/all-collections	API	2025-11-28 11:11:34	Assigned	DV_OGC API - Discrete Global Grid Systems_Workspace_47e90e32c.fmw
☐	GET /api/dggs	API	2025-10-8 15:24:47	Assigned	DV_OGC API - Discrete Global Grid Systems_Workspace_a8ac7d14f.fmw
☐	GET /collections	Data Collections	2025-11-27 17:49:38	Assigned	DV_OGC API - Discrete Global Grid Systems_Workspace_8c1dd3965.fmw
☐	GET /collections/{collectionId}	Data Collections	2025-11-28 11:07:21	Assigned	DV_OGC API - Discrete Global Grid Systems_Workspace_10f37f099.fmw
☐	GET /collections/{collectionId}/dggs	DGGRS Description	2025-11-28 11:13:07	Assigned	DV_OGC API - Discrete Global Grid Systems_Workspace_aaf79e6c3.fmw
☐	GET /collections/{collectionId}/dggs/{dggrsId}	DGGRS Description	2025-11-28 11:15:36	Assigned	DV_OGC API - Discrete Global Grid Systems_Workspace_bf554170.fmw
☐	GET /collections/{collectionId}/dggs/{dggrsId}/zones	DGGS Zone Query	2025-11-27 17:23:35	Assigned	DV_OGC API - Discrete Global Grid Systems_Workspace_0a5b2114d.fmw
☐	GET /collections/{collectionId}/dggs/{dggrsId}/zones/{zoneId}	DGGRS Description	2025-11-27 17:25:46	Assigned	DV_OGC API - Discrete Global Grid Systems_Workspace_4cd794772.fmw
☐	GET /collections/{collectionId}/dggs/{dggrsId}/zones/{zoneId}/data	DGGS Data Retrieval	2025-11-28 13:47:16	Assigned	DV_OGC API - Discrete Global Grid Systems_Workspace_81631144c.fmw
☐	GET /collections/{collectionId}/queryables	Data Collections	2025-11-28 11:19:02	Assigned	DV_OGC API - Discrete Global Grid Systems_Workspace_d1e7c0fc3.fmw
☐	GET /conformance	Conformance	2025-10-8 15:38:15	Assigned	DV_OGC API - Discrete Global Grid Systems_Workspace_6a5b2114d.fmw
☐	GET /dggs	DGGRS Description	2025-10-8 15:26:28	Assigned	DV_OGC API - Discrete Global Grid Systems_Workspace_9a08a344b.fmw
☐	GET /dggs/{dggrsId}	DGGRS Description	2025-10-8 16:18:16	Assigned	DV_OGC API - Discrete Global Grid Systems_Workspace_e199adc82.fmw
☐	GET /dggs/{dggrsId}/zones	DGGS Zone Query	2025-10-22 16:27:06	Assigned	DV_OGC API - Discrete Global Grid Systems_Workspace_b98565571.fmw
☐	GET /dggs/{dggrsId}/zones/{zoneId}	DGGRS Description	2025-10-8 16:21:25	Assigned	DV_OGC API - Discrete Global Grid Systems_Workspace_c26222b5f.fmw
☐	GET /dggs/{dggrsId}/zones/{zoneId}/data	DGGS Data Retrieval	2025-11-28 13:55:20	Assigned	DV_OGC API - Discrete Global Grid Systems_Workspace_d945a1588.fmw

Figure L.4 – DGGS API endpoints in Data Virtualization configuration

To support Data Virtualization, once the DGGS openAPI endpoints were defined, the next task was to define the FME transformation models necessary to produce the appropriate result based on the relevant API input parameters.

The workspace below defines the workflow used to generate the response JSON for a DGGRS ID query.

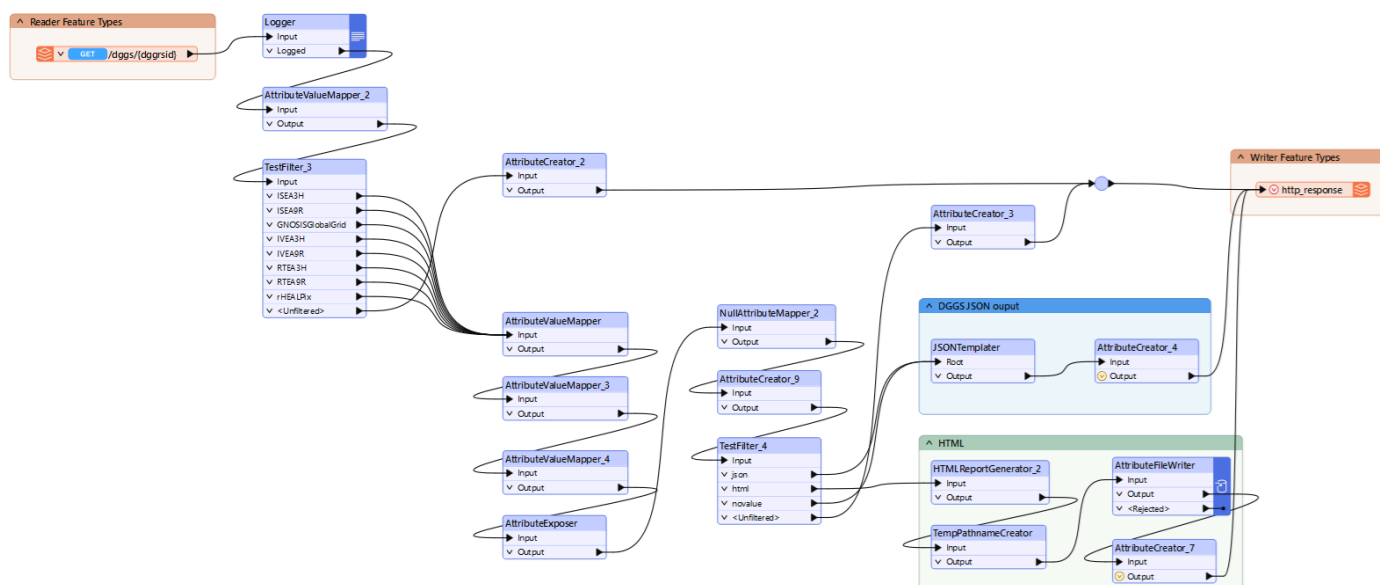


Figure L.5 – FME DGGs API Workspace for DGGRS ID query / response workflow

This next workspace defines the workflow used to generate the response JSON for a Zone ID query.

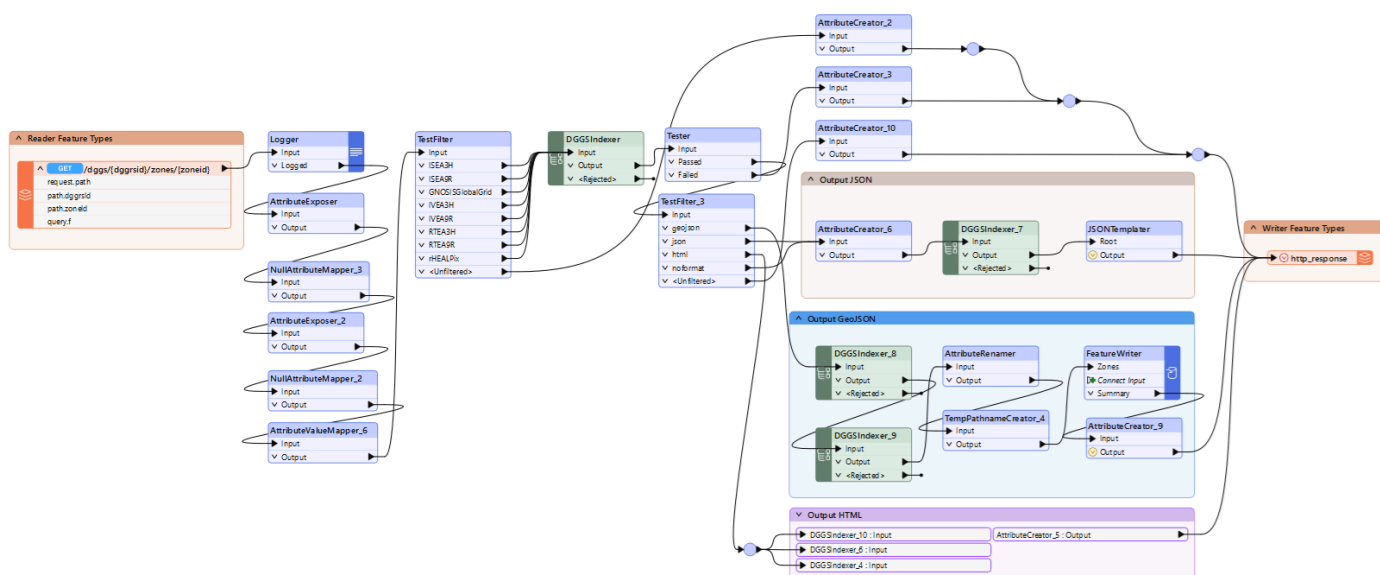


Figure L.6 – FME DGGs API Workspace for zone ID query / response workflow

In both cases above, the dynamic data stream was configured in FME's no code visual development environment, making it rapid to prototype, self document, and easy to maintain. Where responses are static, such as Landing page or conformance documents, they can be configured with predefined response documents (JSON, HTML).

L.3. Interoperability and Testing

Technology Integration Experiments (TIEs) were performed with several AI-enabled DGGS clients to validate the function of the service and demonstrate the value of DGGS as a spatial enabler for AI process chains:

- **TerraFrame Integration:** Extended the hazard area estimates associated with Safe Software's flood risk cells using **Spatial Knowledge Graphs (SKG)**. This allowed the AI to identify cascading impacts where flooded zones compromised road or power networks. In this way the downstream effects of the network interruptions to be assessed in terms of cumulative impacts.
- **Hartis Integration:** Provided the dynamic flood depth data used to ground the **Flood Impact Index (FII)**.
- **Format Validation:** Testing confirmed that the FME-based service could successfully deliver grid data in **GeoTIFF**, **GeoJSON** and **DGGSJSON** formats, ensuring compatibility with both traditional GIS and modern AI tools.

L.4. Performance and Challenges

One technical challenge was the efficient management of high-resolution "Data Holes". By utilizing FME's native ability to handle null values and sparse datasets, the implementation optimized the delivery of **DGGS-JSON** payloads.

Another challenge was scaling rapid delivery of high resolution elevation data. By using Cloud Optimized Geotiff as the source data structure for the DEM DGGS layer, client requests could be rapidly supplied by the cloud optimized 'read only what you need' web hosted data source. FME was able to translate the DGGS client requests into the extents and zoom level requests that the source cloud native data structure is optimized to supply, and then transform this into the appropriate DGGS grid response.

In terms of general performance, the DGGS services were sufficiently responsive with 16 cores running on AWS. When this was scaled back to 4 cores there was some lag observed depending on traffic levels. Evaluating queries on the server side (CQL2 etc) certainly helped improve performance, as opposed to sending too much data to the client to have it evaluated client side. In the future, more testing should be done assessing the effects of volume and velocity of queries — both the number of queries and the data amounts requested.

A significant outcome was the validation of the "Universal Translator" concept. Because FME supports over 30 OGC standards, hundreds of data formats, it was possible to bridge the gap between legacy CAD/GIS data and grid-native AI reasoning without writing custom integration

code. The use of AWS hosting ensured that the service could scale to handle concurrent requests from multiple agentic clients during simulated crisis scenarios.

L.5. Outlook and Resources

Future development will focus on expanding DGGS functionality, improving overall OGC API support, developing and testing a variety of DGGS clients and processing options, and evaluating more capabilities related to real time data streams. In addition, more deployment and integration options with AI tools will be developed, including integration with MCP, and evolving more approaches for AI and DGGS geo-enablement and orchestration.

To begin with, the library of available **FME Hub Custom Transformers** will be enhanced to support more advanced OGC API DGGS features. This could include adding more grid reference systems. More caching options will be configured and tested to improve performance further. Also zone level spatial temporal range links can be added to cloud optimized grid formats such as COG and Zarr. This would help address limitations related to the exponential increase in the number of zones required to support the very high resolutions needed for detailed flood modelling, and the data volumes involved when working with climate and weather model data cubes. This approach was explored by a number of other participants (**GeoInsight**, **GeoLynx**, **GeoMatys** and others) and is discussed in **Section 2.5 Spatial Tokenizer 4.6.**, **Friction 5: Spatial Temporal Resolution and Range Queries** and **Appendix F DGGS-ZARR Draft Encoding Specification**.

Further tests will be carried out to evaluate approaches for publishing real time data via DGGS from live stream sources such as IoT sensors like stream flow gauges. In the case of dynamic risk estimation, values between sensors could be interpolated across both space and time before being indexed onto a DGGS grid to support live monitoring via status dashboards.

Other OGC APIs will also be developed, refined and made available via FME Hub. This is expected to include updates to OGC API Features as well as other OGC APIs such as Coverages, Processes and Tiles, so a full range of OGC APIs are available to users developing with the FME platform.

In this pilot, a basic FME DGGS client was developed to assist with testing of the DGGS service. Going forward, the next step would be to add a DGGS reader to make it easier for any FME workflow to connect directly to any DGGS service, and for users to make ad hoc DGGS queries using Data Inspector. This would also allow for experiments on client side vs server side processing, analysis, and development of mediation services where one DGGS service mediates a query by passing on sub queries to other DGGS services before composing its response. This could also support experiments around server and client side grid algebra. These additional capabilities could provide more flexibility around multifactor analysis related to climate change hazard impact scenarios.

In order to support evolving AI requirements, development is currently progressing on FME MCP support. This could benefit DGGS support in several ways. For one, an MCP service could be deployed on top of a DGGS service in order to make it more readily accessible by AI clients. Also, the FME platform itself can be used to support AI agent orchestration with a variety of

AI, DGGs, SKG and other geoenabled services to broaden the spectrum of geoinformation pathways available to multi-AI systems.

In regards to ongoing collaboration with the OGC community, Safe Software has partnered with the OGC for more than 20 years and has participated in a variety of innovation pilots including Disaster pilots (2021,2023), Climate Resilience pilots (2023, 2024) and other pilots, testbeds, and code sprints. Safe Software will continue to engage in a range of DWGs and SWGs, and continue to support open standards efforts at OGC in support of FAIR principles of data sharing and integration across all data types and any AI tools.

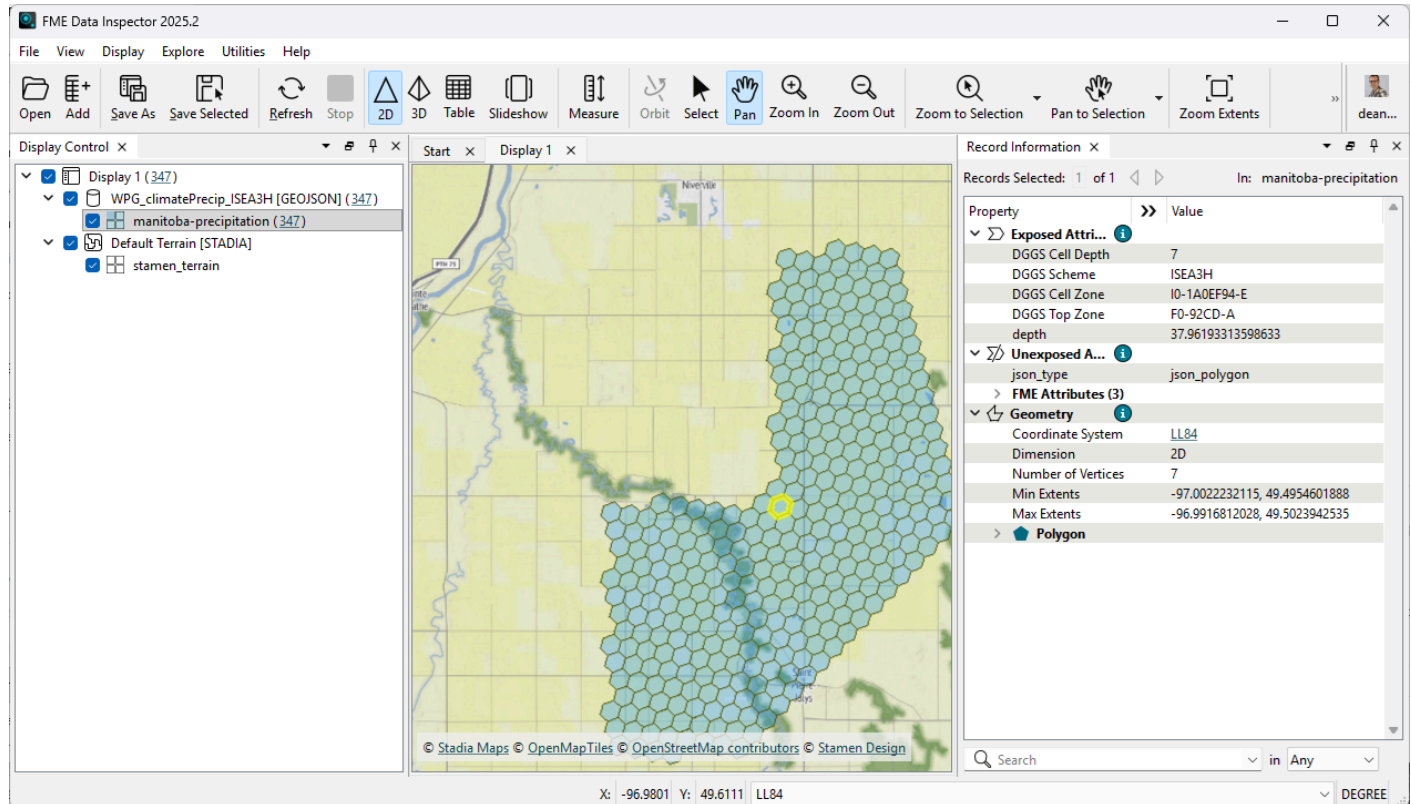


Figure L.7 – FME DGGs test client viewing climate projections for precipitation
> hazard threshold value, as read from DGGs results encoded in GeoJSON

Technical Links:

- **Pilot Instance:** [FME DGGs Server](#)
- **Swagger Page:** [Interactive API Docs](#)
- **FME Hub DGGs Tools:** [Custom Transformers](#)
- **DGGAL Library:** [DGGAL GitHub](#) / [DGGAL Documentation](#)
- **Related OGC Pilots:** [OGC Disaster Pilot 2021](#)
- **Disaster Management FME Examples:** [Situational Awareness Blog](#)
- **FME Climate Resilience Webinar:** [Translating Climate Data into Risk Assessments](#)



ANNEX M (INFORMATIVE) COMPUSULT – AI-ENABLED DGGS CLIENT



ANNEX M (INFORMATIVE) COMPUSULT – AI-ENABLED DGGs CLIENT

Contributor: Craig Coffin (Compusult)

M.1. System Overview

Compusult's AI-enabled DGGs client is a DGGs-aware GenAI chatbot prototype deployed in a microservices architecture. The following overview describes the microservices architecture for the chatbot prototype, including the technologies used to implement this architecture.

M.1.1. Front-End User Interface

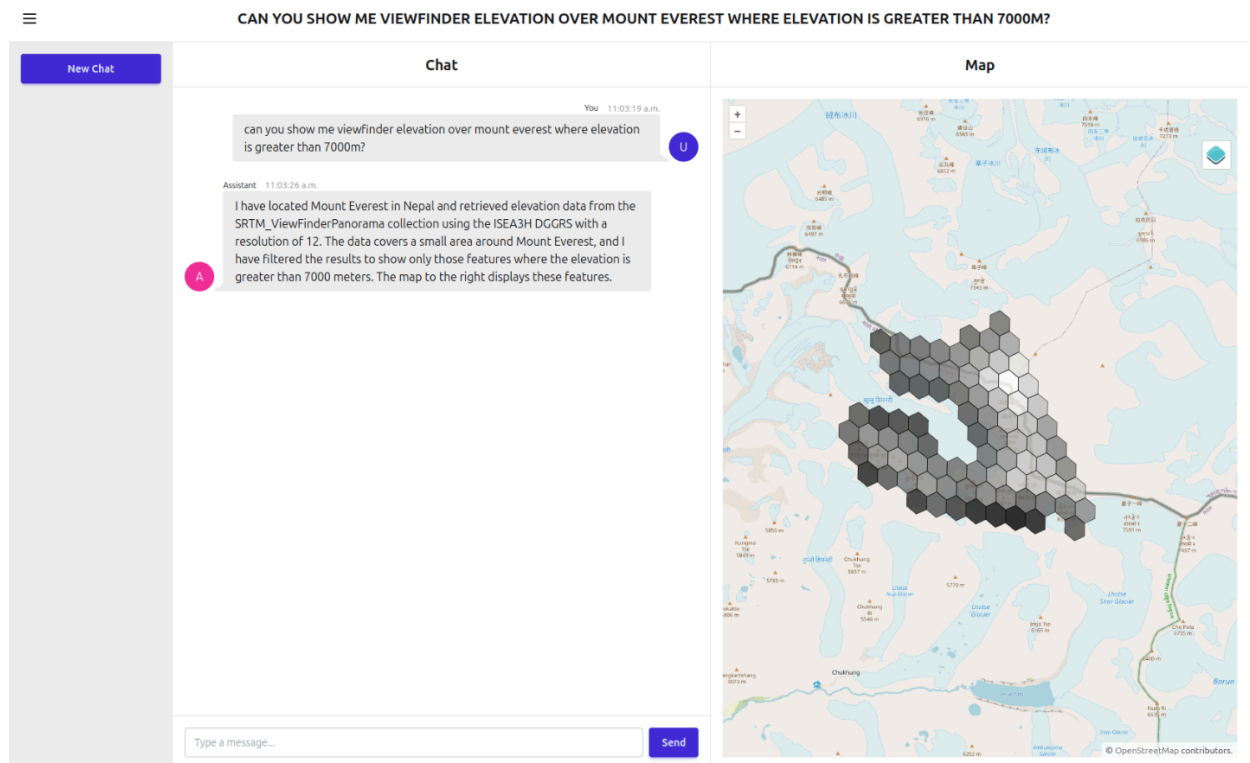


Figure M.1 – Front-End User Interface

The user enters their search queries using the text input field in the bottom-left of the interface. Above the text input, is the area containing the response(s) generated by the GenAI Chatbot. Additionally, an interactive map client is included on the right to allow users to visualize any associated data resources, such as results from DGGs zone data queries. Finally, a pop-out menu is included on the far-left of the screen, allowing the user to create new chat sessions and load previously saved chat sessions.

The front-end communicates with the back-end services via well-defined Representational State Transfer (REST) APIs. Through these APIs, queries entered by the user are submitted for processing, with associated results being returned and displayed in the interface.

M.1.2. AI-enabled DGGs Client Architecture

The back-end microservices have been implemented using a software stack based on the Python programming language. In order to utilize DGGs search results, the back-end microservices will contact the DGGs endpoints via OGC API – DGGs on behalf of the user and then assist the user in interpreting the results returned.

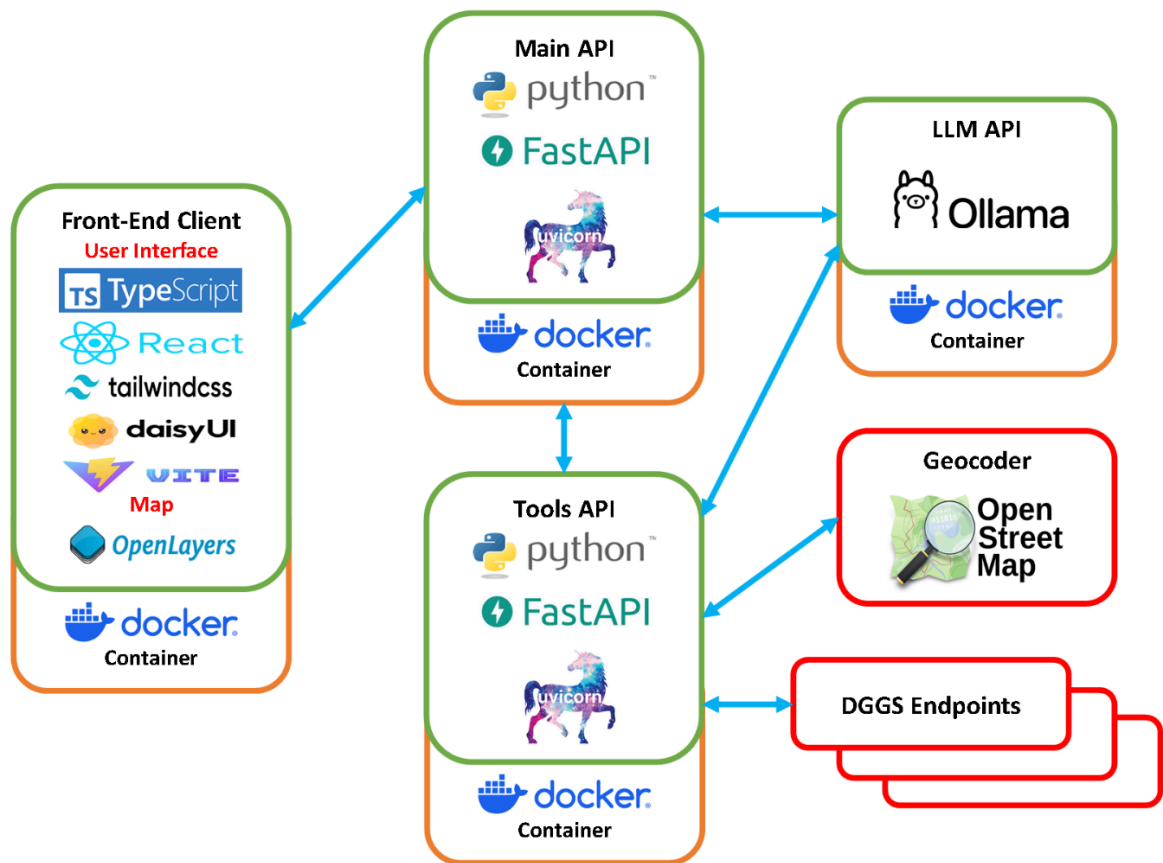


Figure M.2 – Composult AI-DGGS Client Architecture

The chatbot front-end has been developed primarily using ReactJS. Supporting libraries, such as Tailwind Cascading Style Sheets (CSS) and Next.js, have been used to add additional styling, performance, and efficiency.

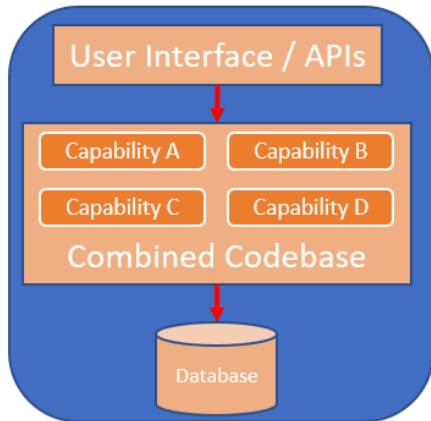
The microservices that comprise the chatbot architecture are:

- **Chat API:** Responsible for accepting queries from the user and coordinating with the other microservices. It has one endpoint named “generate” which returns a generation from an LLM based on conversation history. Written in Python with FastAPI and uvicorn.
- **Tool Invocation API:** Responsible for determining which tools are required to answer a user query or access a DGGS API endpoint. It uses open-source Python-based geospatial libraries including:
 - **GeoPy:** geocoding access.
 - **GeoPandas:** modifying and visualizing geospatial data.
- **Large Language Model (LLM) API:** The core of the response is generated using an LLM designed to process human-like text.

M.1.3. Microservices Architecture

The differences between Legacy Application Architecture and Microservices Architecture are shown in Figure 1-4 below.

Legacy Application Architecture



Microservices Architecture

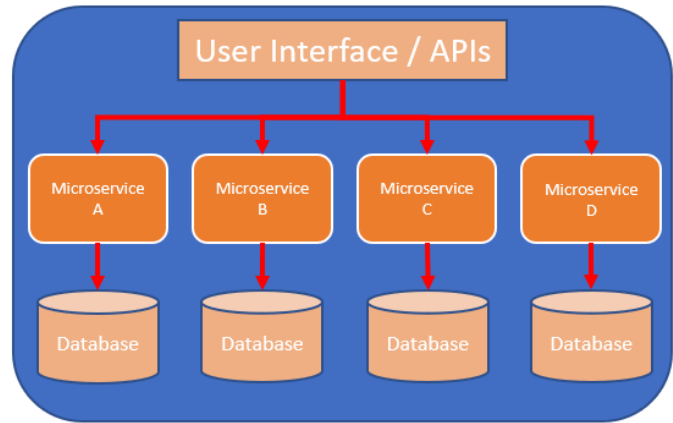


Figure M.3 — Example Microservices Architecture

Advantages include: * **Scalability and Fault-Tolerance:** Each microservice can be scaled independently. If one fails, it doesn't bring down the entire application. * **Continuous Deployment:** Microservices can be deployed and updated independently. * **Agility:** Smaller teams can develop and deploy independently. * **Cost-Effectiveness:** Microservices can be scaled based on demand.

M.1.4. Containerization

In the chatbot architecture, each microservice is hosted in a separate Docker container. Docker Compose is used to define and start multi-container applications via YAML files. This allows the system to be deployed across AWS, Microsoft Azure, or on-premises servers (e.g., Enterprise data centres).

M.2. DGGs Integration Logic

Compusult's AI-enabled DGGs client consumes and interacts with DGGs endpoints via third-party libraries:

- The **Discrete Global Grid Abstraction Library (DGGAL)** is used for HEALPix/rHEALPix, GNOSIS Global Grid, and projections based on ISEA, IVEA, and RTEA.

- The **Python binding for the H3 Core Library (h3-py)** is used for the H3 grid system.
- The client supports OGC Common Query Language (CQL2) for filtering zone and data queries.

M.3. AI and Explainability Functions

Compusult's AI chatbot uses **Large Language Model Meta AI (LLaMA)**. To provide an API to interact with the LLM, **Ollama** software is used to run LLMs locally with optimized performance.

Note on Storage: The size of the LLM files used for this prototype is approximately 7GB, though complex LLaMa models can require up to 800GB.

M.4. User Interaction and Testing

Representative sample queries include discovery of datasets, retrieval and display from DGGS endpoints, and temporal/spatial filtering. Client-server interoperability testing was conducted with: * **GNOSIS Map Server** (Ecere Corporation) * **Examind Server** (Geomatys)

M.4.1. AI Hallucinations

We noticed several occurrences of “hallucinations” where the LLM generated entirely fabricated but plausible information. This occurs because AI models generate responses based on learned patterns rather than verifying facts.

M.4.2. Example Queries and Visualizations

- **Rideau Street Imagery:** Resulting GeoJSON layer quantized to H3 cells. (Figure 4)
- **Rideau Street Contours:** GeoJSON layer consisting of contour lines overlapping an ISEA3H zone. (Figure 5)
- **St Jean Baptiste Elevation:** GeoTIFF map layer consisting of Manitoba Digital Elevation Model data, clipped to an H3 cell. (Figure 6)
- **Mount Everest Filtered Elevation:** GeoJSON layer quantized to ISEA3H zones, containing only sub-zones where elevation > 7000m. (Figure 7)

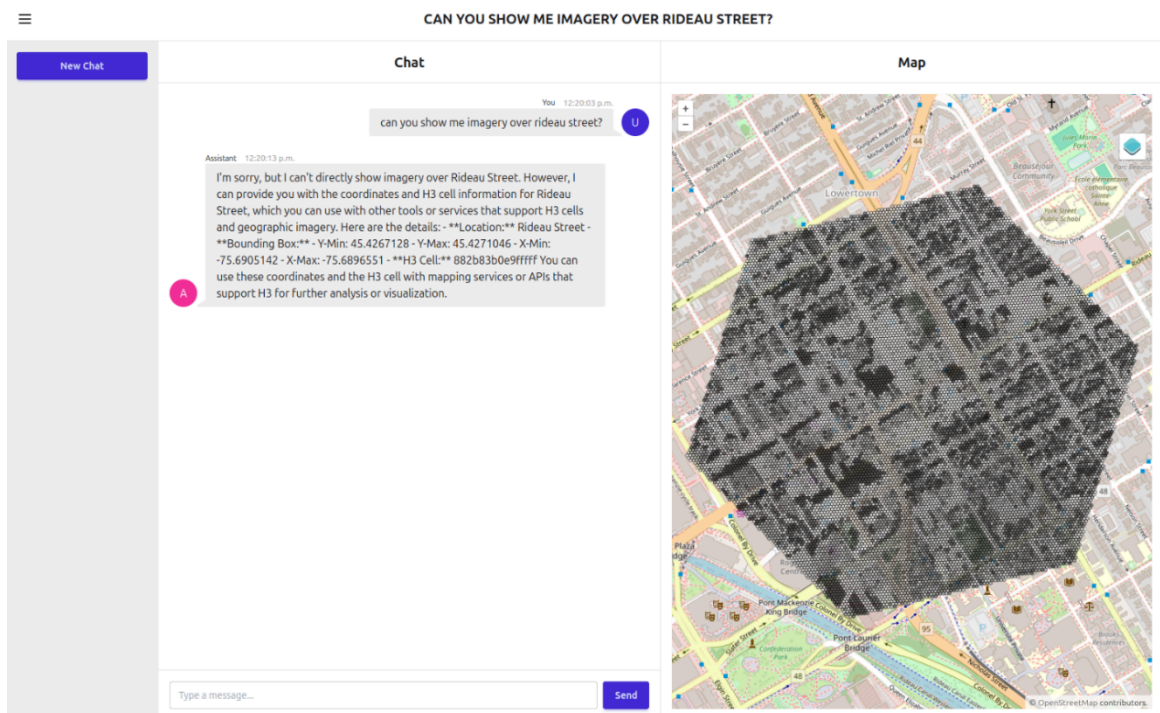


Figure M.4 — Example Query: “Can you show me imagery over Rideau Street?”

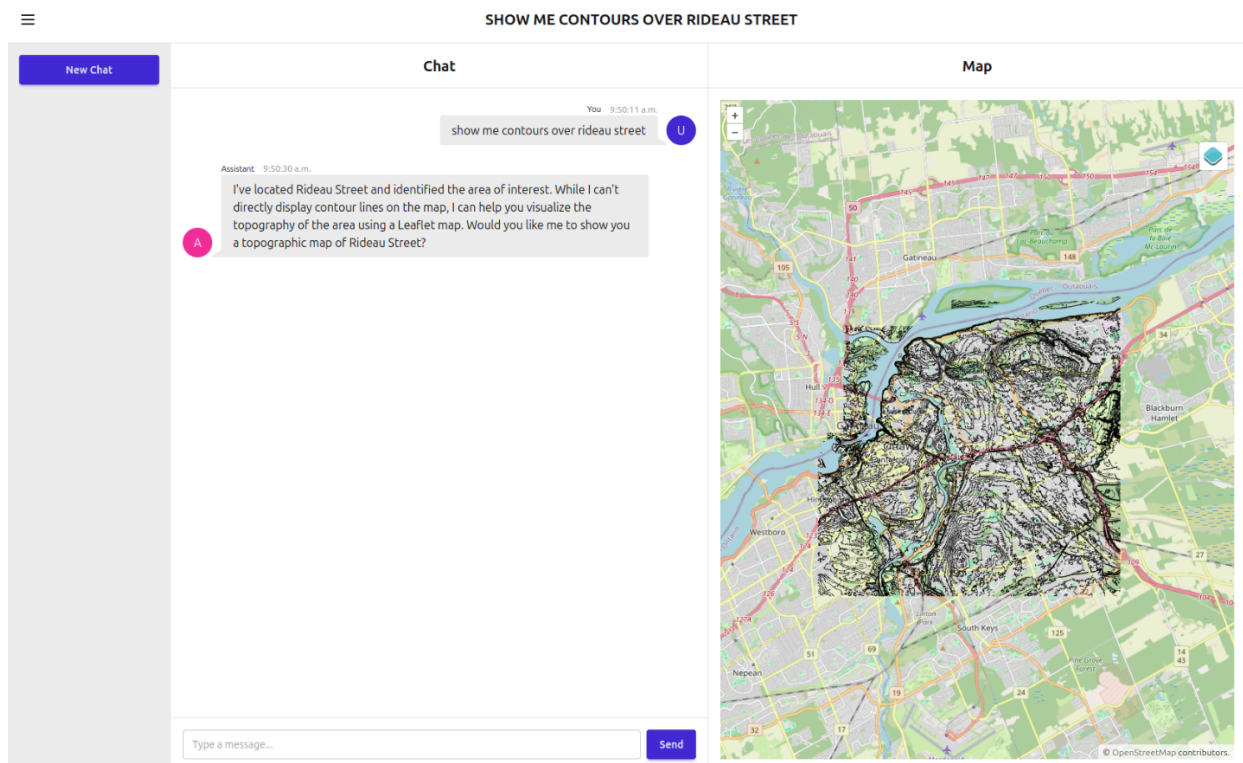


Figure M.5 — Example Query: “Show me contours over Rideau Street”

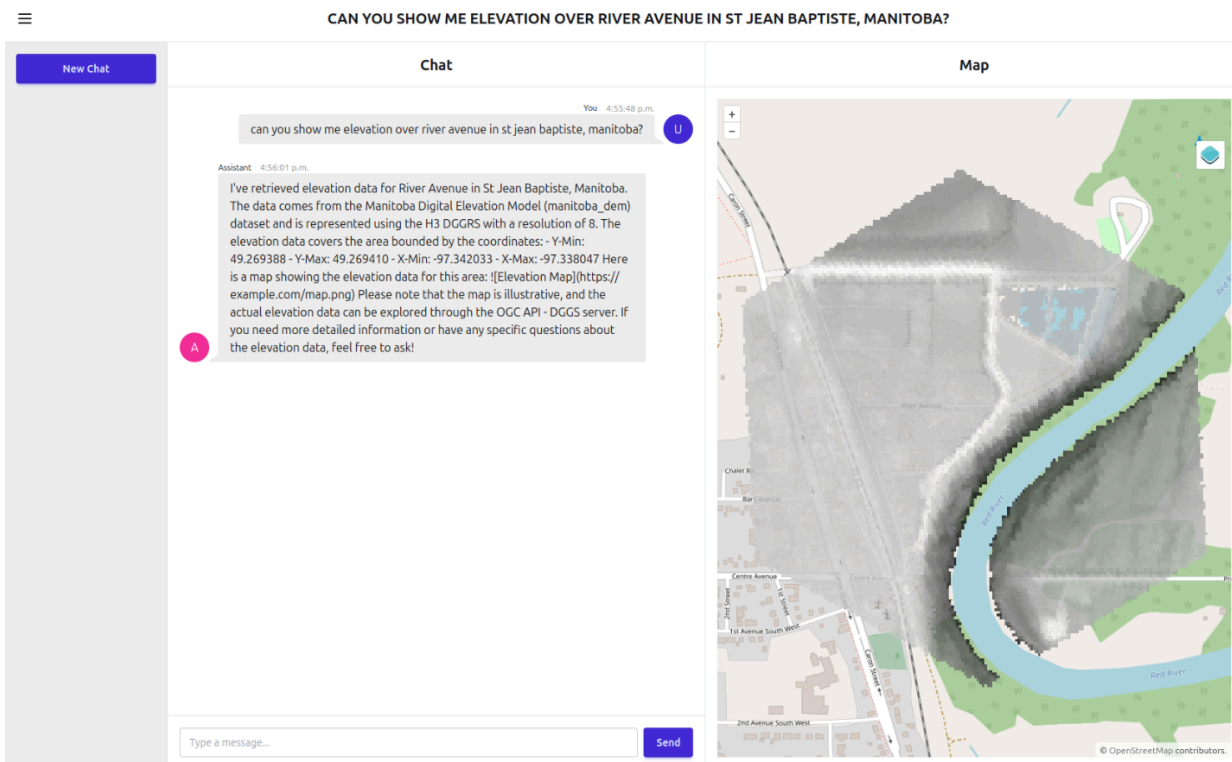


Figure M.6 — Example Query: “Can you show me elevation over River Avenue in St Jean Baptiste, Manitoba?”

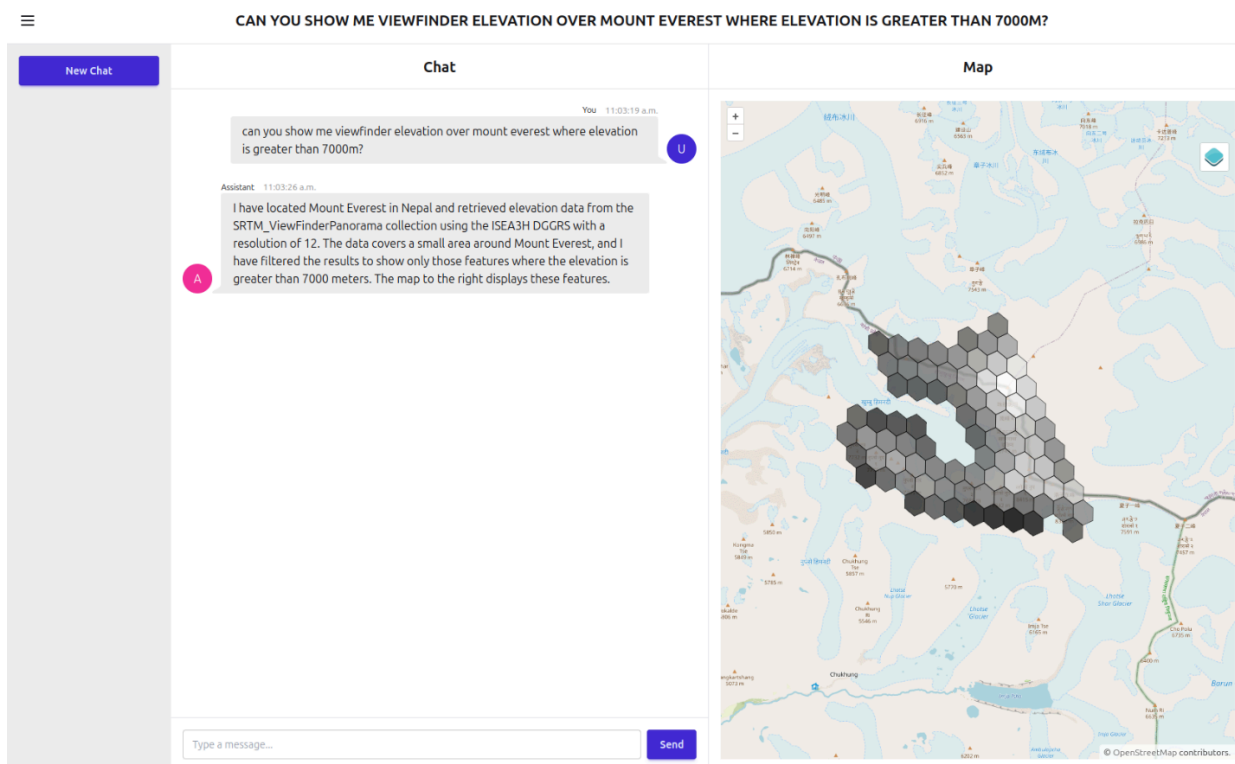


Figure M.7 — Example Query: “Can you show me viewfinder elevation over Mount Everest where elevation is greater than 7000m?”

M.5. Conclusions and Future Work

The client is currently able to respond to:

- **Data retrieval:** Imagery (Kanata, ON), Elevation (Red River Valley), Population (Winnipeg, 2020).
- **Filtering:** Southern Manitoba < 250m elevation; Red River < 250m with density > 10 people/sq km.
- **Data discovery:** Available collections over specific cities or collection descriptions (SRTM).

M.5.1. Query Analysis Questions

- How many and which data collections are involved?
- Do collections reside on a single endpoint?

- Do collections support a common DGGRS?
- Are external tools (geocoders) required?
- Should the client display a direct representation (grayscale) or an analyzed result?

M.5.2. Future Development

- **Data aggregation:** “Average rainfall in southern Manitoba over the past 10 years.”
- **AI Insights:** “Identify areas requiring evacuation assistance” (analysis of roads and population).
- **Predictive Analysis:** “Greatest flood risk near the Red River in the next five years” based on historical levels and projected rainfall.



ANNEX N (INFORMATIVE) TERRAFRAME – AI-ENABLED DGGS CLIENT



ANNEX N

(INFORMATIVE)

TERRAFRAME – AI-ENABLED DGGS CLIENT

Contributors: Justin Smethie, Nathan McEachen (TerraFrame)

N.1. System Overview

The TerraFrame client uses the Angular front-end framework. Additionally, for the front-end mapping components the client uses MapLibre and DGGAL.js to display map features and DGGS zone data.

The front-end communicates with a back-end server which orchestrates communication between the LLM hosted on AWS bedrock, the DGGS servers, and feature data. All communication is passed through the orchestration server. A user cannot directly interact with the LLM, DDGS server, or any other service. The orchestration server is a SpringBoot server which uses basic Spring security to authenticate users.

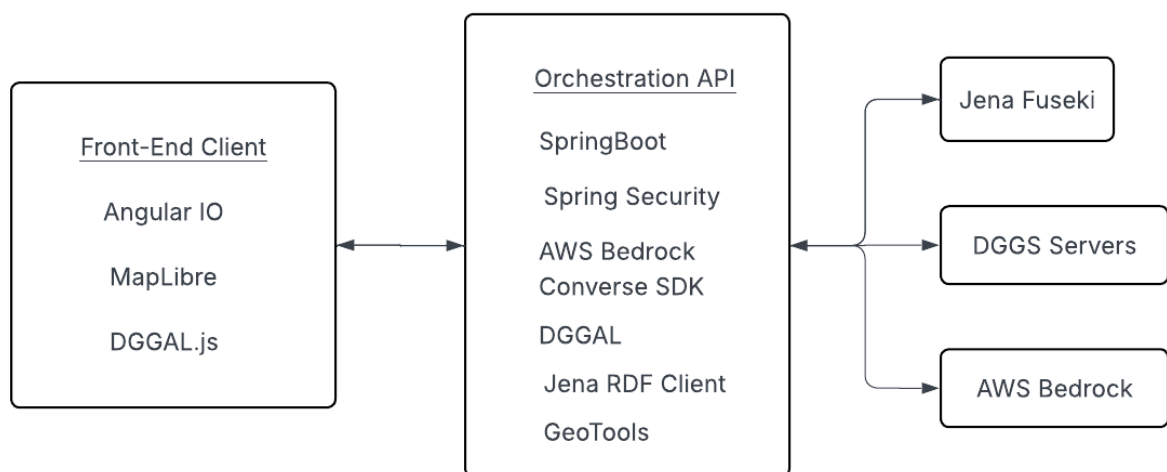


Figure N.1 – TerraFrame AI-enabled DGGS Client Architecture

N.2. DGGs Integration Logic

The TerraFrame client interacts with DGGs servers to both configure its agent and query data on behalf of its users. On startup the back-end server consumes collection information for all the DGGs servers it has registered. The collections are filtered to only include collections in the applications area of interest. For instance, collections with data in Portugal are filtered out because the application area of interest is Manitoba. The server then retrieves the DGGs information available for each collection. The collection is then assigned a DGGs to use when querying data for that collection. This information is then provided to the LLM in a prompt so that the LLM knows what collections it has available to it.

When performing data queries to the DGGs server the client first queries the /zone endpoint with a bounding box of the area of interest and a zone depth in order to get the parent zone ids. The bounding box is determined by the location resolution tool which converts a location name to a bounding box. The zone depth is defaulted to 9 unless the user specifies a different zone depth in their question. Once the client has the parent zone ids it then makes a /data request for each parent zone. By default the client gets the data in DGGs-JSON format from the server and converts that information into GeoJSON in the browser.

N.3. AI and Explainability Functions

The TerraFrame client uses Anthropic Claude 3.7 hosted on AWS Bedrock. In order to facilitate communication between the LLM and the other information sources the system uses tool specifications. A tool specification tells the LLM what tools are available for it to use, describes the purpose of the tool and how to use it, as well as what parameters are available and required by the tool. When a user asks a question the LLM can select the tool it needs to use in order to answer the question. If the LLM determines that it needs to use a tool to answer the question it makes a tool request response to the orchestration server. The orchestration server then makes a request to the appropriate service and returns the results back to the LLM or user.

N.4. User Interaction and Testing

TIEs performed with:

- Safe Software — elevation, temperature, precipitation, and flood DGGs data in the ISEA3H DGGs integrated with spatial knowledge graph.
- Geol Insight – elevation, NALP and Landsat data in the IVEA3H DGGs.

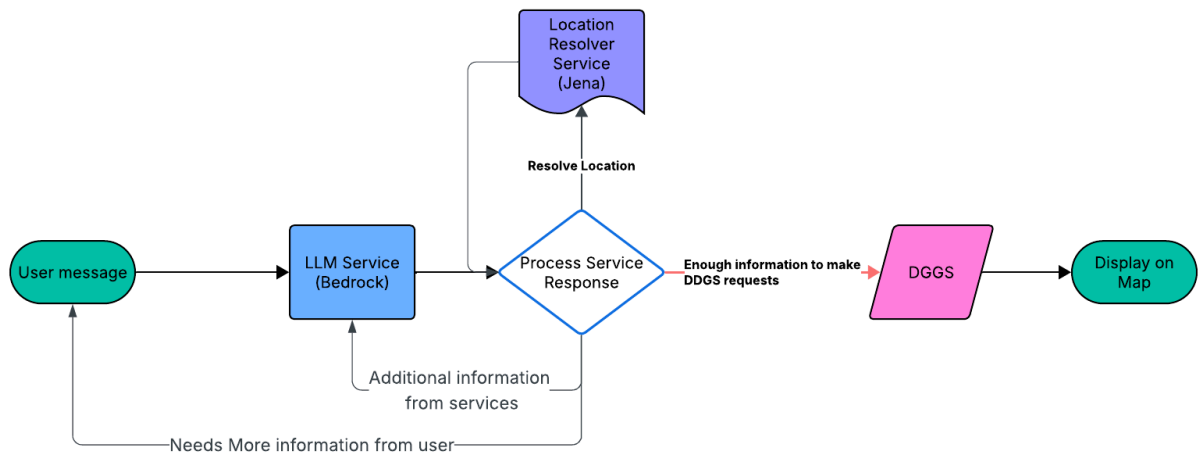


Figure N.2 – TerraFrame AI-enabled DGGS Client Workflow

A typical conversation workflow is as follows: When a user asks a question the orchestration server passes that question to the LLM. The LLM processes the question and determines if it needs to execute a remote service to answer the question. If the LLM needs to make a DGGS request then it must first provide an area of interest. As such it strips the location name out of the question. That location name is then passed onto the location resolver tool which converts the location name into a bounding box which represents the area of interest. Once the area of interest is provided to the LLM it then determines what DGGS collection or analysis it needs to perform in order to answer the question. The LLM then passes that tool request response to the orchestration server which then performs the analysis or makes a call to the DGGS server to get the data required. That data is then return to the user.

Currently, the TerraFrame client communicates with the Safe Software and GeoInsight servers. In general, the interoperability between the two DGGS servers is very good. The slight exception is the Safe Software server provides a /queryables endpoint in which the client can get metadata about what attributes can be filtered on for a collection. The TerraFrame client uses this metadata to configure the LLM such that it know what attributes it can use for filtering and the description of each attribute. The GeoInsight server does not provide that same endpoint. However, that endpoint is not part of the official DGGS server specification, so it is not expected that all servers provide that functionality.

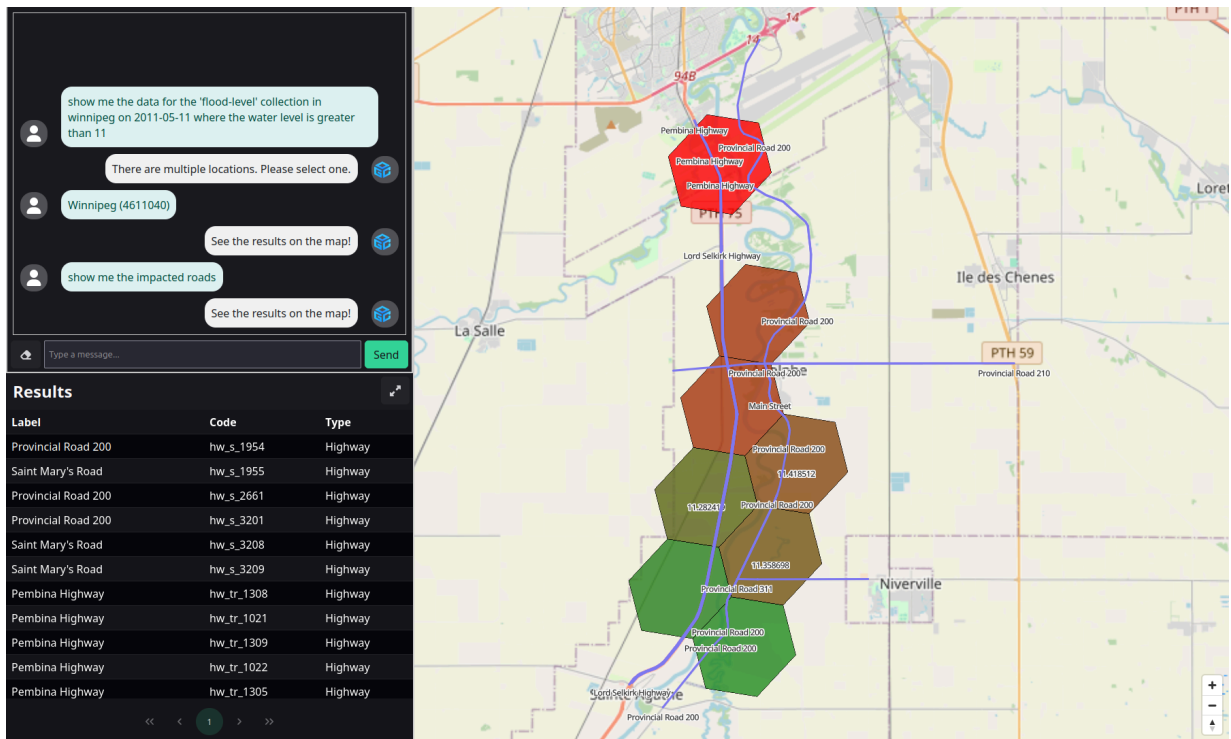


Figure N.3 – TerraFrame AI-enabled Traffic Analysis

Beyond normal DGGs requests the TerraFrame client supports multiple different types of analysis using the DGGs data. The first type of analysis it can perform is to determine what power infrastructure features overlap with the current DGGs data. The power infrastructure analysis combined with restrictive CLQ2 DGGs queries can answer questions like “What power infrastructure will be impacted if the water level is greater than 4 meters?”. Leveraging the power of a spatial knowledge graphs the TerraFrame client can then perform further analysis to calculate which dissemination areas will be affected based on the impacted power infrastructure. Finally, the TerraFrame client can also perform basic road analysis to see what highways overlap with the DGGs data. Similar to the power infrastructure analysis this can be used to determine what highways are impacted based on varying water levels.

N.5. Outlook and Resources

One of the key issues stated for emergency management is the routing of vehicles during a flood or wild fire. We believe that parts of this issue can be addressed through the use of DGGs and AI. Some future work we think will be helpful is to integrate the TerraFrame client with the Google Routing API and the Google Geocoding API in order to give a user the ability to generate routes from point A to point B. We can then use AI and DGGs to determine if the given route will be impacted due to a flood or some other data analysis on the DGGs server. Additionally, we believe that it is possible to give alternatives to a route if it is determined that the given route will be impacted. This workflow can be useful during an emergency because a user will be able to quickly find alternative routes based on current conditions as well as simulate

flood conditions in the DGGS to plan future routes. The solution can further be enhanced by incorporating additional data sources to include other road information such as weight limits and drainage capabilities.

A demo of the TerraFrame client can be found at <https://ai-dggs.geoprism.net>. Please request credentials at info@terraframe.com



ANNEX O (INFORMATIVE) CS GROUP – AI-ENABLED DGGS CLIENT



ANNEX O (INFORMATIVE) CS GROUP – AI-ENABLED DGGS CLIENT

Contributor: Nicolas Vila (CS Group)

O.1. System Overview

Acting as the **Autonomous Agent Architects** for the pilot, CS Group developed an AI-enabled client designed to move beyond simple chatbot interactions toward autonomous **Agentic AI workflows**. The primary purpose of the system is to demonstrate how conversational agents can discover, query, and interpret geospatial data through standardized protocols.

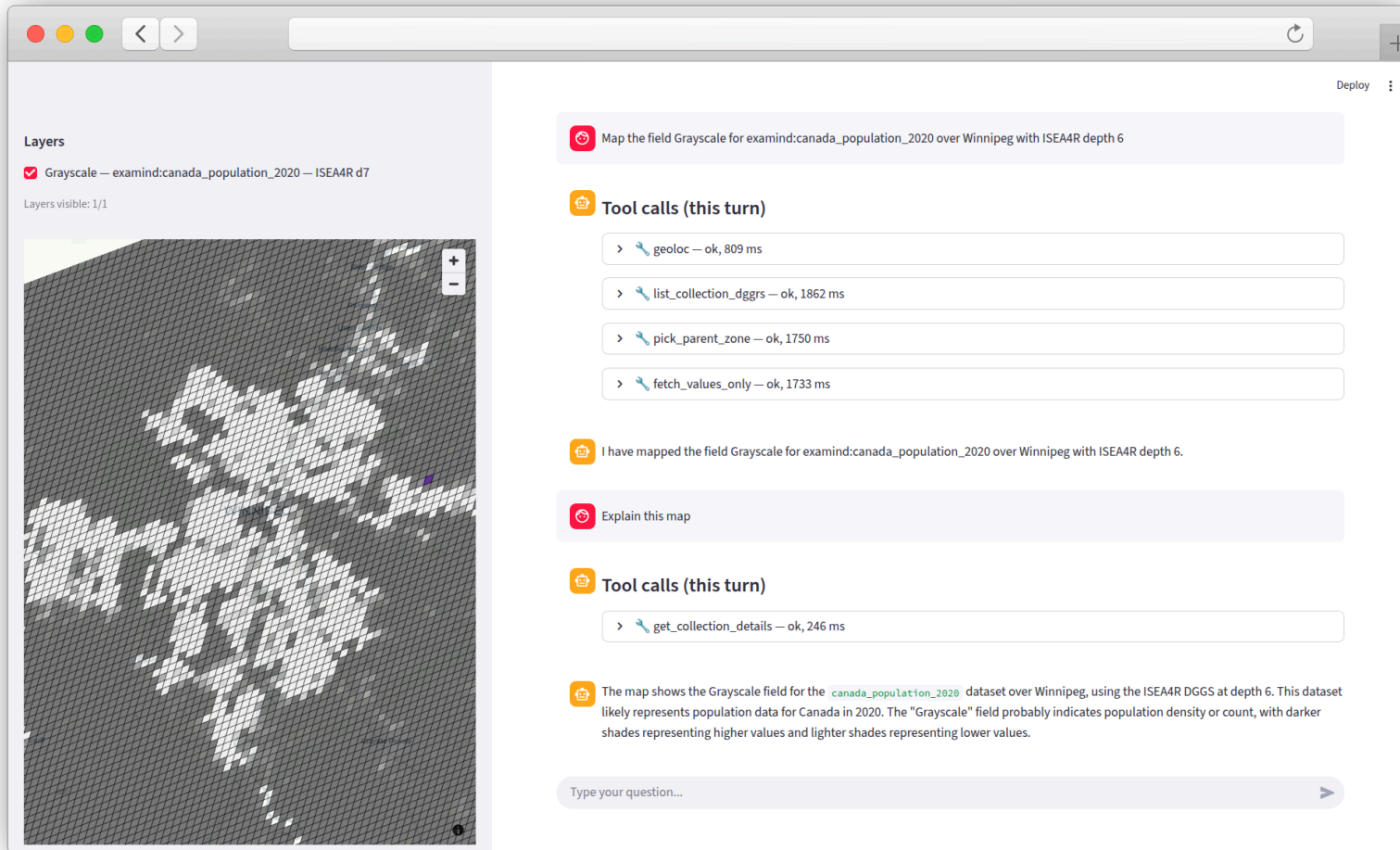


Figure O.1 — CS Group AI-DGGS client Front-End User Interface

The client is a web application structured into two core modules:

- **Frontend:** Developed using **Streamlit (Python)**, providing a conversational user interface integrated with interactive map visualization components. For the map visualization, the client uses **PyDeck** based on **DeckGL** to display geospatial data retrieved from DGGS servers and processed on the client side (geometry reconstruction).
- **Middleware Backend:** Implemented as a **Model Context Protocol (MCP)** server using the **FastMCP** framework

The system is **LLM-agnostic**, successfully integrating with both local models (served via **Ollama** like **Minstral**, **Qwen**, etc.) and cloud-based commercial models (such as **Google Gemini Flash**). This flexibility ensures the client can operate in varied environments, from high-performance cloud infrastructures to secured local nodes.

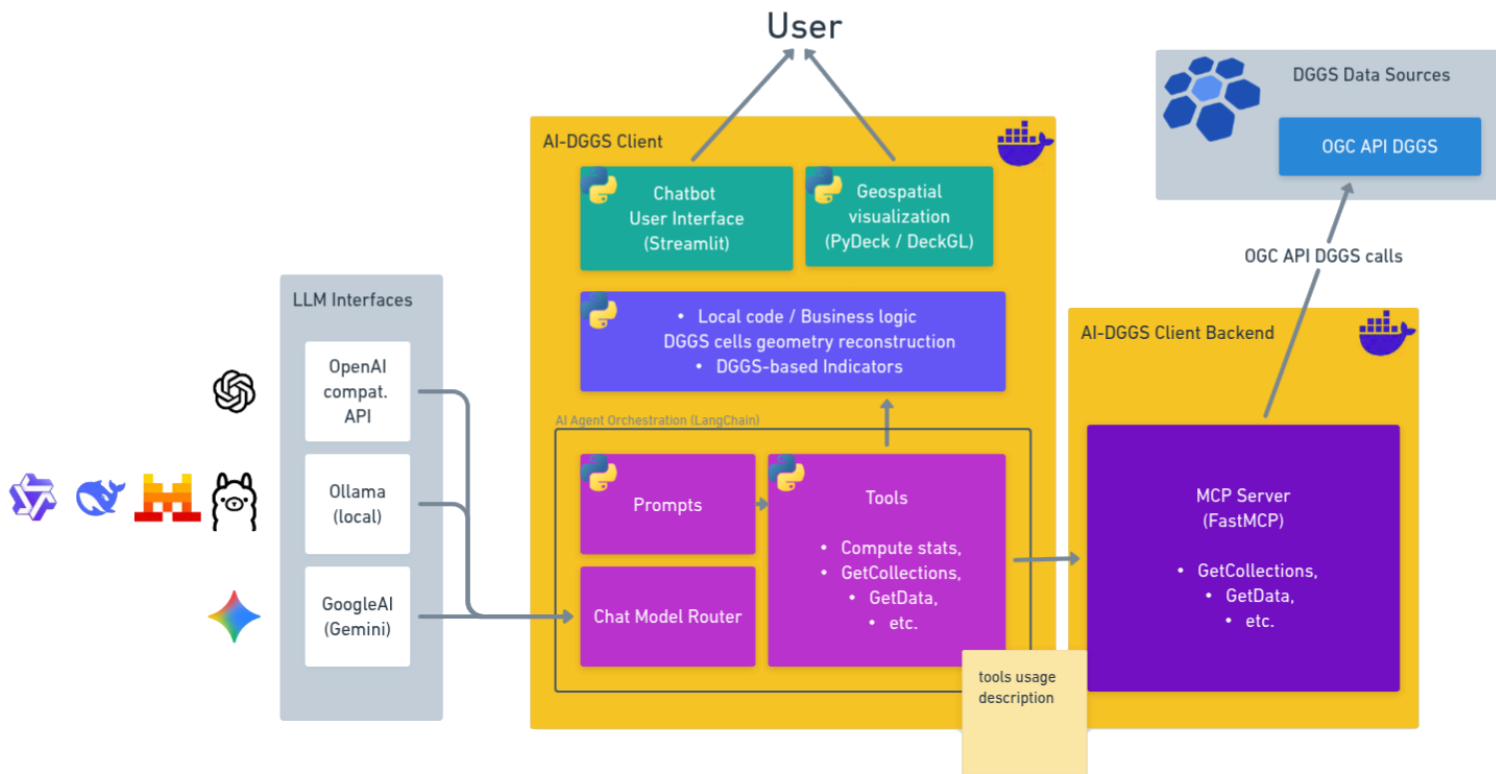


Figure O.2 – CS Group AI-enabled DGGs Client architecture diagram

O.2. DGGs Integration Logic

A fundamental design choice of the CS Group client was the separation of AI reasoning from geospatial data access. This was achieved through the adoption of the **Model Context Protocol (MCP)**. Instead of hard-coding the logic for specific grid systems into the AI, the backend provides a suite of self-documented Python-based MCP tools. The client embeds a pre-prompt which gives recommendations to the LLM on how to use the tools, and the LLM can then autonomously select which tools to use and in what sequence. This design allows the client to interact with multiple DGGs servers without needing custom code for each server, as the MCP tools abstract away the specifics of each server's API.

The integration logic focuses on:

- **Standardized Discovery:** AI agents autonomously “read” server capabilities, identifying available collections and query parameters without custom vendor code.

- **Stepwise Reasoning:** When resolving a query (e.g., “Analyze flood risk in Winnipeg”), the agent executes a sequence of tools: first geolocating the area, then searching for relevant DGGs collections, and finally constructing valid OGC API – DGGs queries.
- **Multi-Vendor Interoperability:** The client demonstrated the ability to consume disparate endpoints provided by pilot participants (D100), synchronizing metadata to ensure correct interpretation of zone identifiers and refinement levels across different grid designs.

Typical workflows, use the following sequence of steps:

1. The user submits a **natural language query** through the interface (in english or french).
2. The query is analyzed to determine the **Area of Interest (AoI)**
3. The agent retrieve the location and the bounding box of the AOI using a **geocoding** tool ([Nominatim](#) geocoding API) in our case.
4. The agent then queries the registered DGGs servers to **discover relevant collections and fields** for the AOI (as well as the target DGGs).
5. The agent selects the appropriate collection and field, and **constructs a valid OGC API -**
6. DGGs query to **retrieve the data**.
7. The retrieved data is then **processed and visualized in the interface**, with the agent providing explanations of the reasoning and tools used to arrive at the results.

O.3. AI and Explainability Functions

The client utilizes **Retrieval-Augmented Generation (RAG)** to ground AI responses in “Grid-Truth.” By referencing specific DGGs query results and associated metadata, the agent avoids hallucinations and provides responses based strictly on authoritative data. The user can ask the agent to explain its reasoning, which includes: why specific tools were selected, what data was retrieved, and how conclusions were drawn. This is facilitated by the explicit tool invocation model of MCP, which allows for detailed tracing of the agent’s decision-making process. The user can check each tool calls, the inputs provided, and the outputs received, ensuring transparency and trust in the AI’s conclusions.

Explainability is built into the tool-based interaction model:

- **Traceable Reasoning:** Each step of the reasoning process—from tool selection to data subsetting—is made explicit.

- **Metadata Alignment:** AI-generated summaries reference the underlying collections and grid structures (e.g., explaining why a specific population count was associated with a risk zone).
- **Machine-Readable Documentation:** MCP tools expose their purpose and expected outputs in a format that the LLM can sequence autonomously, supporting FAIR (Findable, Accessible, Interoperable, Reusable) data principles.

O.4. User Interaction and Testing

The typical user workflow centers on a conversational interface that lowers the barrier to entry for non-expert users. The agent interprets natural language input to select the Area of Interest (AOI) and refinement levels. The agent can process either unitary queries or complex scenarios. This allows the user to explore available collections, fields, and DGGRS from the different API providers.

- List available providers and collections
- Map the field Grayscale for examind:canada_population_2020 over Winnipeg with ISEA4R depth 4
- Compute the mean, min and max of population in this area
- Any flood collections available?
- Map flood cause in examind:historical_flood_event_0 over Winnipeg with ISEA4R depth 4
- Combine both into a risk index, put 0.8 weights to flood presence and 0.2 to population
- Map gnosis:HRDEM-RedRiver:DTM:2m over Winnipeg with HEALPix
- Increase detail / Reduce detail / Zoom in / Zoom out

"Look at the available collections, on the different providers. I want to analyse the situation over Winnipeg.
Find one concerning flood and map it with one field related to the "flood cause" with depth 5.
Next, find one collection concerning population and map it with the same parameters.
Give me details about the collection and the field you mapped.
If the 2 collections are found and mapped, then combine both into a risk index, put 0.8 weights to flood presence and 0.2 to population.
When done, explain the steps you followed, the tools you used and why you executed them in that order.
Finally, write a summary in a paragraph."

Listing O.1 — Example of "complex" user scenario

On next screenshot, we can see the result map created, the summary of the actions done by the agent, and the explainability of the reasoning. It appears that the answers of the agent are

consistent and are reproducible. Details and presentation in the **explanations can slightly vary** with the same scenario but **the business logic is preserved**, and the same tools are used in the same order (sometimes explanation is more detailed, sometimes less). The user can ask for more details if needed. This phenomenon was observed during testing, and is a known behaviour of LLMs, which can produce different outputs for the same input (with same or different languages models). However, the consistency in the logic and tool usage across different runs provides confidence in the reliability of the agent's reasoning process.

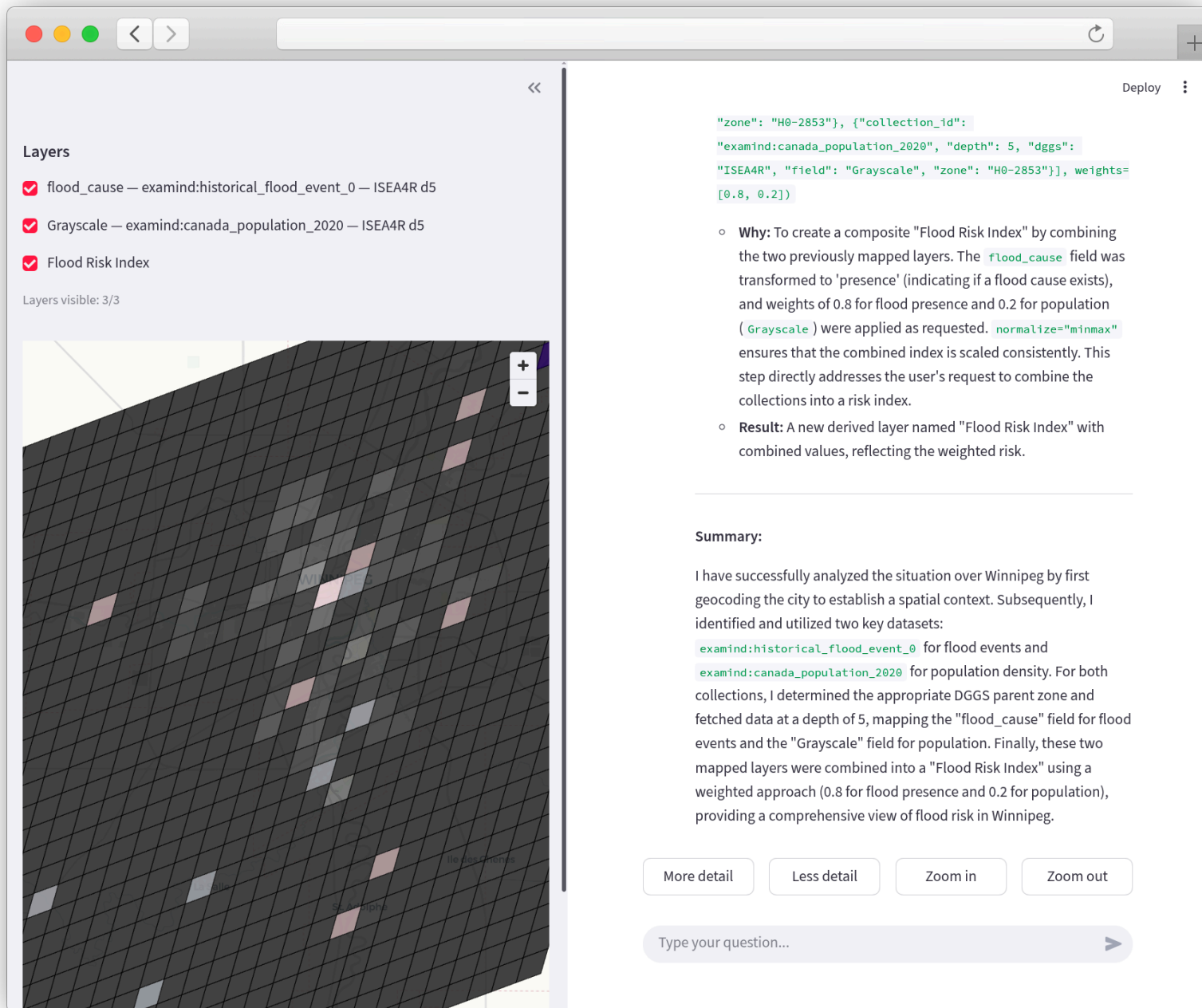


Figure O.3 — CS Group AI-enabled DGGs Client: Multi-Server Tool Chaining via MCP using Geomatys Examind data, with ISEA4R DGGs

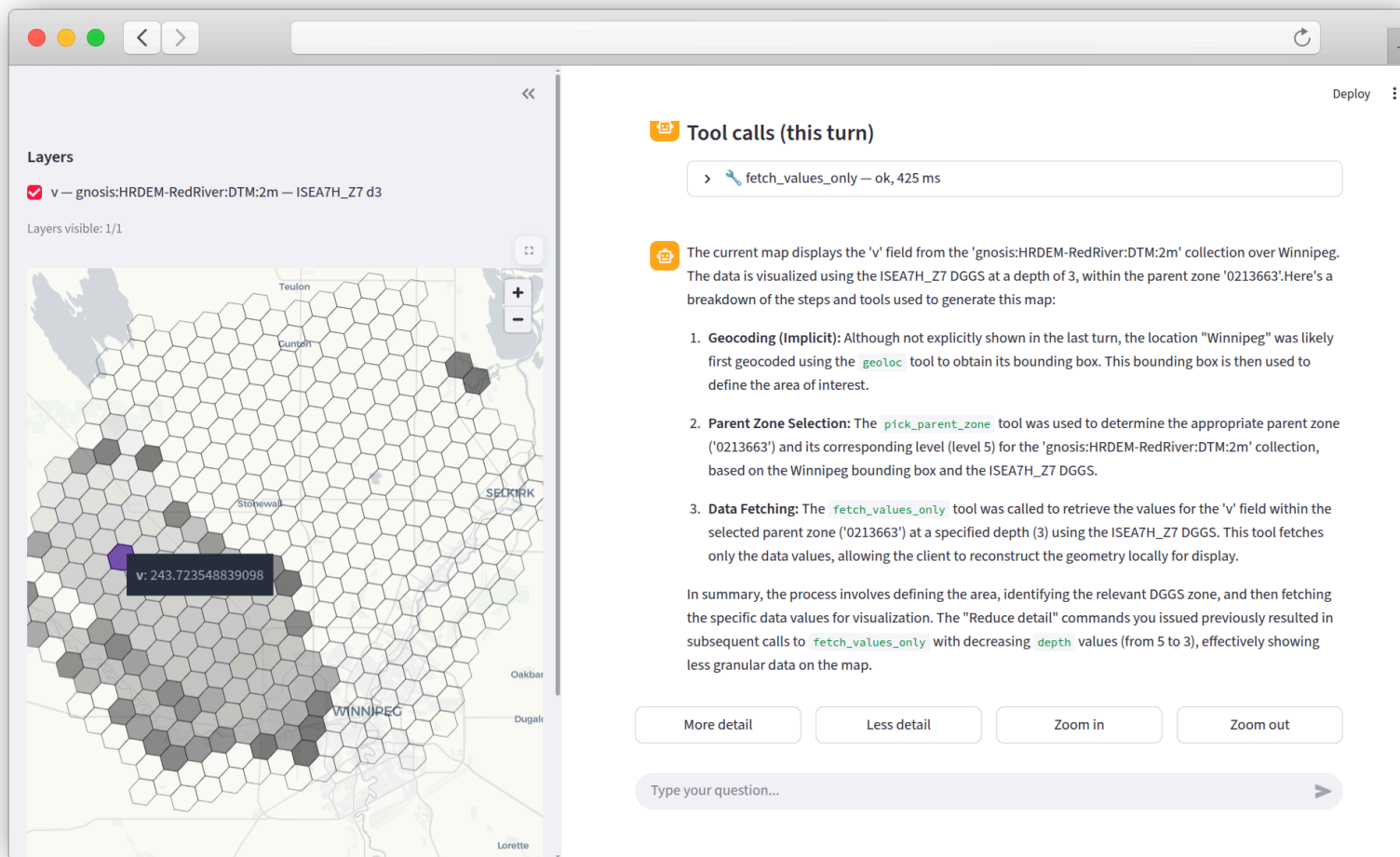


Figure O.4 — CS Group AI-enabled DGGS Client: Query and display of DTM data from Ecere Gnosis server, with ISEA7H_Z7 DGGRS

O.5. Technology Integration Experiments (TIEs)

During **Technology Integration Experiments (TIEs)**, the client was tested against multiple DGGS server implementations (Geomatys Examind, Ecere Gnosis, Geolynx, Geolinsight, etc.)



List available providers and collections



Tool calls (this turn)

>  list_collections — ok, 30015 ms



I found 805 collections from the following providers: crim, examind, geoinsight, geolynx, gnosis, safe.

Here are a few examples:

- crim:manitoba-sample
- crim:manitoba-sample-coordinates
- crim:manitoba-rcm-ard
- crim:canada-population
- examind:OIS-Reech_A26207_184_PlanarRectified

Figure O.5 — List of available providers and collections in the CS Group AI-enabled DGGs Client.

Key findings from these tests included:

- **Handling API Verbosity:** To manage the large response payloads typical of high-resolution DGGs queries, the agent was configured to work with mid/low refinement levels initially, progressively drilling down into higher resolutions only when necessary.
- **Interoperability Validation:** The tests demonstrated that while small variations exist in how different servers expose metadata, the MCP-based abstraction successfully mediated these differences, allowing the agent to function consistently across the federated ecosystem.

O.6. DGGs Metadata and Naming Conventions

CS Group additionally observed that, in the absence of sufficiently rich and machine-readable metadata, **naming conventions effectively become a proxy for semantic description**, introducing fragility into client implementations. During client development, it was noted that layer and field names were often not self-describing (e.g., a field labeled “**Grayscale**” within a population-related collection, or a generic “v” field within an elevation collection), requiring interpretation or adjustment during data import. In such cases, the client agent was **required to make assumptions about field semantics**, providing a practical stress test of real-world datasets and highlighting limitations for autonomous use. This issue could be mitigated directly on the client,

by providing **additional text-based documentation about the collections and fields to the LLM** (from web pages, pdf docs), or by improving the metadata provided by the servers. The latter is more sustainable in the long term, as it would allow any client to interpret the data without needing custom adjustments or assumptions.

Furthermore, variations in **DGGRS** naming conventions across servers (e.g., “**HEALPix**” versus “**Healpix**”) were identified as an additional interoperability concern, necessitating tolerance for naming variants on the client side and reinforcing the need for standardized, self-describing semantic metadata rather than reliance on naming conventions alone. To solve this issue from the client perspective would have been to use:

- The **uri** property from `.../dggs/{dggrsId}` which should link to the **authoritative entry** (if one exists) on the **OGC DGGRS register** (once it comes alive),
- The **link** in the **links** property with the rel `http://www.opengis.net/def/rel/ogc/1.0/dggrs-definition` which is required and should point to the DGGRS definition (whose schema is still informative, but a first draft of it exists in OGC API – DGGS Annex B)

O.7. Outlook and Resources

Future developments for the CS Group client will focus on enhancing multi-server federation capabilities and providing even more granular explainability features, such as explicit query traces and “human-in-the-loop” reasoning checkpoints. The team plans to further explore the alignment of DGGS with **other OGC API standards**, potentially making DGGS a universal interface for querying both raster and vector data through agentic systems. Indeed **OGC API Processes**, **OGC API Features**, and **OGC API Records** could be interfaced with the same MCP-based architecture, allowing the agent to discover and chain together a wider variety of geospatial services beyond just DGGS.

Technical Resources:

- **Pilot Website Detail:** [CS Group Overview](#)
- **Demonstrator** [CS Group AI-DGGS client demonstrator](#)
- **Standardization Action:** Adoption of MCP for OGC API interfacing.
- **Source Repositories:** Links to demonstration instances and code will be updated in the final repository as they become available.



ANNEX P (INFORMATIVE) HARTIS – AI-ENABLED DGGS CLIENT



ANNEX P

(INFORMATIVE)

HARTIS – AI-ENABLED DGGS CLIENT

Contributor: Loukas Katikas (Hartis)

P.1. System Overview

The AI-enabled DGGS Client developed by Hartis is a reference implementation demonstrating how agentic Artificial Intelligence can operate natively on Discrete Global Grid Systems (DGGS) to support disaster risk assessment and decision-making.

The system is designed to address a key limitation identified during the AI-DGGS Pilot: the difficulty for AI systems to autonomously discover, interpret, and reason over geospatial data exposed through OGC APIs. To this end, the Hartis client provides a natural-language interface that enables users to query complex, multi-source geospatial datasets without requiring prior knowledge of collection identifiers, coordinate reference systems, or grid indexing schemes.

Architecturally, the system is composed of: * a high-performance interactive frontend for visualization and user interaction, and * an orchestration backend responsible for AI reasoning, tool execution, and DGGS-native data access.

The frontend is implemented using **Next.js (React)** and provides interactive map rendering through **MapLibre GL**. The backend is implemented in **Django (Python)** with **PostGIS**, serving as the integration layer between AI components and OGC API – DGGS servers. Authentication and access control mechanisms are designed to be compatible with OGC identity and security practices.

The intelligence layer is built around Large Language Models (LLMs) hosted on **Amazon Bedrock**, integrated via the **Model Context Protocol (MCP)**. MCP enables the translation of high-level user intent into structured, machine-executable tool calls that interact directly with DGGS-enabled OGC APIs.

P.2. System Workflow

The end-to-end workflow implemented by the Hartis client follows a consistent and auditable sequence:

1. **Intent Interpretation:** The user submits a natural-language query describing a geographic area, time horizon, or operational question.
2. **Spatial Scoping:** The system resolves the query into one or more Areas of Interest (AOIs) and identifies relevant DGGs resolutions.
3. **Tool Orchestration:** The AI orchestrator invokes MCP tools to discover available DGGs collections and retrieve cell-level data.
4. **DGGs-native Analysis:** Retrieved variables are aggregated, joined, and analyzed directly in DGGs space.
5. **Explanation and Visualization:** Results are presented as both map layers and natural-language explanations grounded in DGGs cell values.

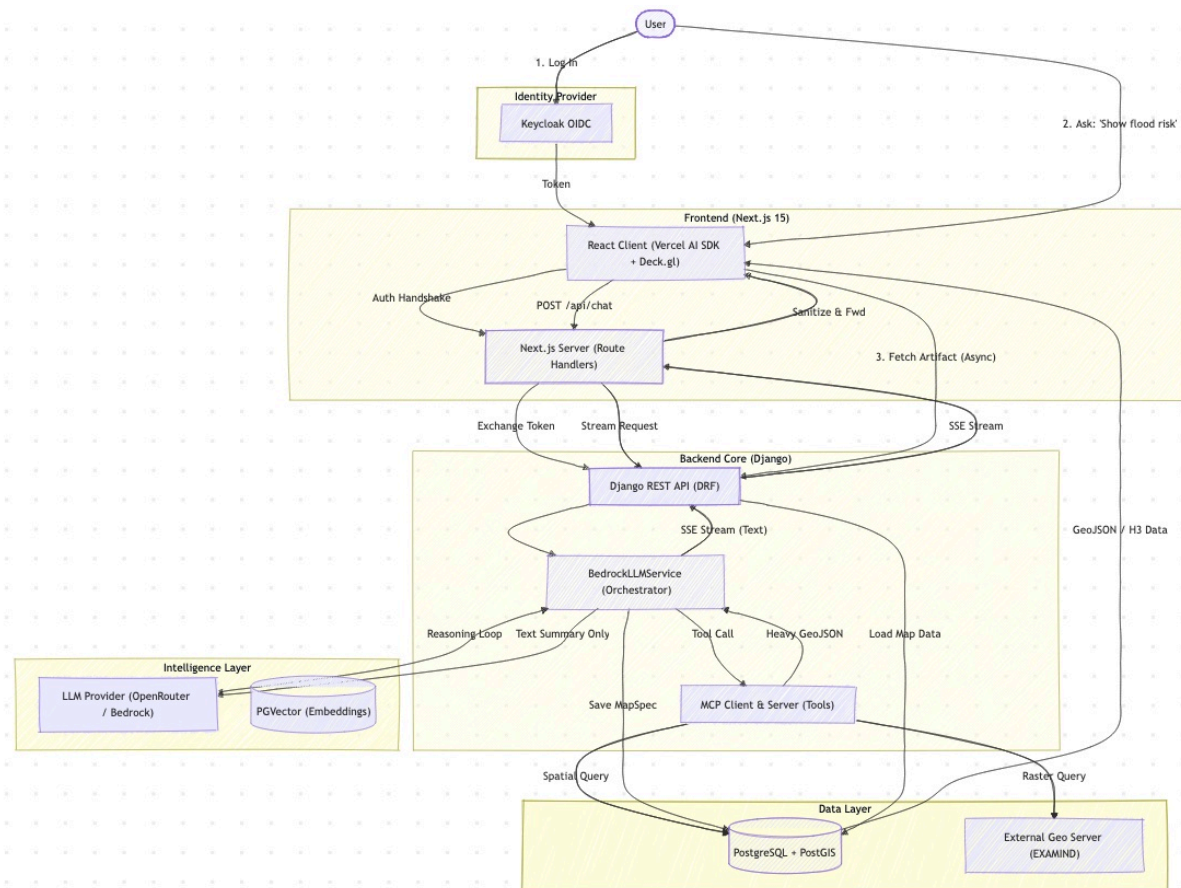


Figure P.1 – Hartis AI-DGGS Client – Reference Workflow

P.3. DGGs Integration and Spatial Processing

The Hartis client treats the DGGs grid as the primary spatial abstraction and uses it consistently as a **spatial tokenization layer** across all datasets and processing steps. This approach ensures spatial coherence when combining heterogeneous data sources.

The system supports **H3 (levels 8–12)** and **ISEA3H** grids, using the **DGGAL** library for both frontend rendering (via WebAssembly) and backend analysis (Python bindings). DGGs Zone IDs are used as the sole spatial join key, eliminating the need for repeated coordinate reprojection.

To mitigate the risk of **AI scale blindness**, the client implements a **progressive refinement strategy**: * coarse-resolution queries are used for initial discovery and screening, * finer-resolution DGGs queries are automatically triggered only for areas exhibiting significant signals or user interest.

Resolution-dependent effects, such as smoothing at coarser resolutions and noise at finer resolutions, were explicitly evaluated during development and are exposed as configurable parameters in the analysis workflow.

Interoperability is achieved through a federated access model in which DGGs collections from multiple providers are combined transparently using common DGGs identifiers.

P.4. DGGs Playground

The **DGGs Playground** is a lightweight inspection and verification interface that exposes DGGs query behavior independently of the AI orchestration layer. It allows implementers to explore DGGs tiling, resolution selection, and aggregation behavior in a transparent and reproducible manner.

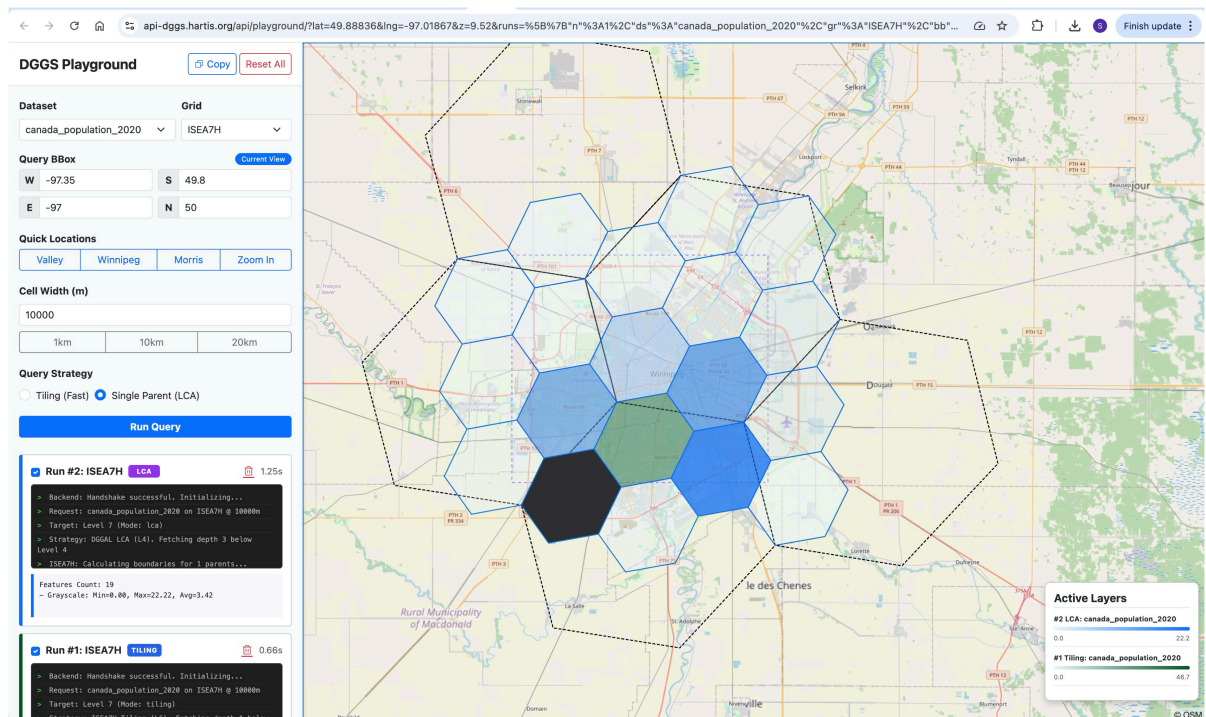


Figure P.2 – Hartis DGGS Playground – Interactive DGGS Inspection Interface

The DGGS Playground exposes the following capabilities:

- **Interactive DGGS Tiling:** Users can generate DGGS cells over a specified bounding box using selectable grid systems (e.g. ISEA3H) and target cell widths, enabling direct comparison of resolution and tiling strategies.
- **Dataset and Grid Selection:** The interface allows explicit selection of datasets and DGGS grids, making the spatial and semantic assumptions of each query visible and adjustable.
- **Bounding Box and AOI Control:** Queries can be scoped via numeric bounding boxes or predefined locations, with real-time feedback on spatial extent and coverage.
- **Aggregation and Optimization Inspection:** The Playground visualizes the results of server-side tiling and aggregation logic, allowing users to assess effects such as cell clipping, coverage density, and resolution trade-offs.
- **Execution Trace Feedback:** Each query returns structured execution metadata (e.g., tiling strategy, parent-child relationships, cell counts), supporting validation of DGGS behavior and performance characteristics.

P.5. AI Reasoning and Explainability

The AI reasoning framework is based on **Retrieval-Augmented Generation (RAG)** anchored in DGGs cell values. Rather than relying on latent knowledge embedded in the LLM, all analytical conclusions are derived from explicitly retrieved geospatial variables.

For each analytical response, the system: * retrieves DGGs cell-level indicators (e.g., HAND, TWI, elevation, flood fraction), * computes composite risk metrics using the **Flood Impact Index (FII)** model, * generates a natural-language explanation that explicitly references the contributing indicators and thresholds.

This approach ensures that all AI outputs are: * grounded in observable data, * reproducible by human analysts, and * suitable for audit in operational contexts.

To support transparency and FAIR principles, the client maintains explicit references to the DGGs cells, collections, and aggregation methods used in each reasoning step.

P.6. User Interaction and Operational Scenarios

The Hartis client supports a structured operational workflow aligned with disaster management practices:

1. **Risk Discovery:** Identification of areas exhibiting elevated hazard or susceptibility.
2. **Impact Assessment:** Intersection of risk layers with exposure datasets such as population, roads, and critical infrastructure.
3. **Operational Reasoning:** Generation of prioritized summaries and recommendations based on combined hazard, exposure, and vulnerability indicators.

The client presents its operational capabilities through a combined map-and-dialog interface, allowing users to explore DGGs-based risk layers, infrastructure exposure, and AI-generated analytical summaries in a single, coherent view.

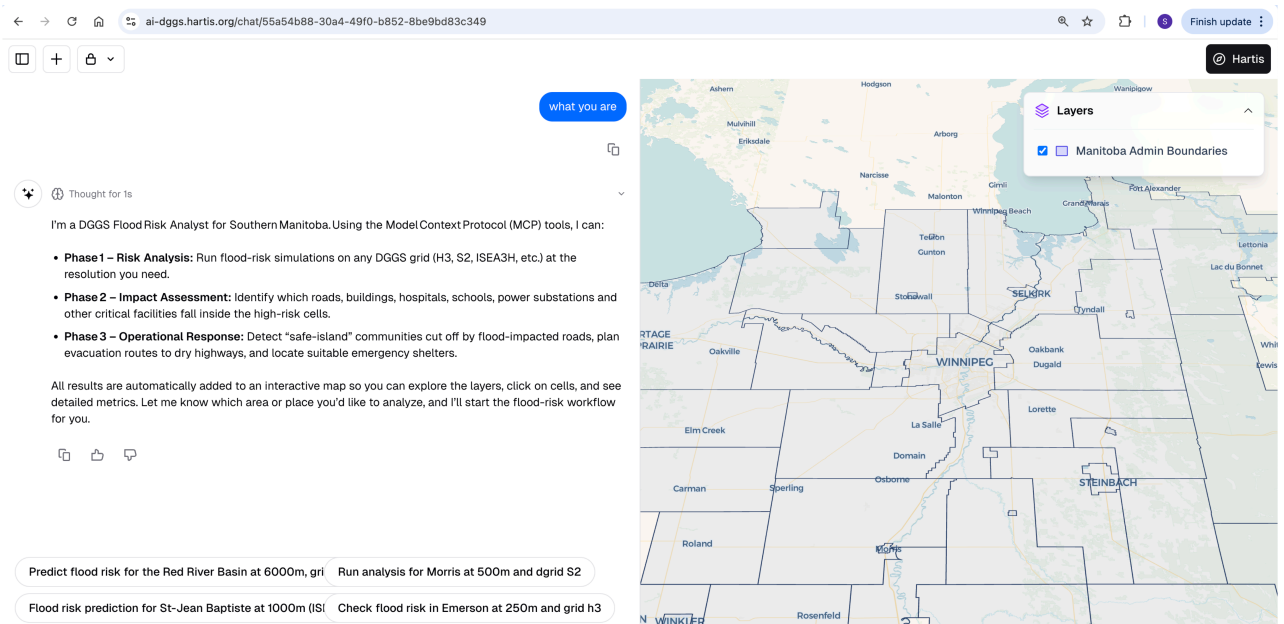


Figure P.3 — Hartis AI-DGGS Client — Operational Capabilities View

The figure below illustrates AI-assisted transportation impact analysis performed on DGGS-derived flood risk outputs for the Morris area. Flood probability values aggregated at DGGS resolution are intersected with road network data to classify road segments into operational risk categories (blocked, hazardous, cautionary, passable). The AI client presents both a quantitative summary and a spatial visualization, enabling users to identify affected segments, inspect per-segment risk attributes, and trigger follow-up operational actions such as evacuation routing or infrastructure prioritization.

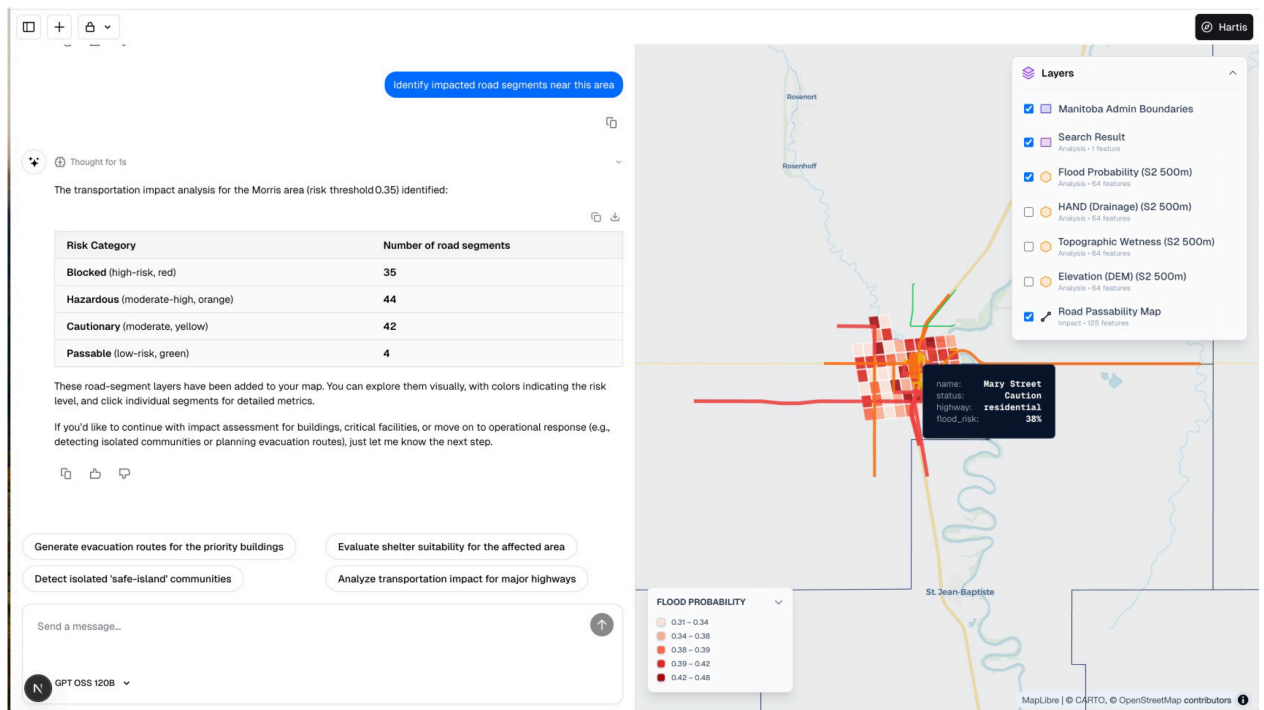


Figure P.4 – AI-enabled transportation impact assessment using DGGS-derived flood risk

Usability testing demonstrated that AI-driven DGGS discovery significantly lowers the technical barrier for end users, enabling effective use of DGGS-based analytics without specialized geospatial expertise.

P.7. Outlook and Relevance to OGC Standardization

The Hartis AI-enabled DGGS Client demonstrates how existing and emerging OGC standards can be combined to support **agentic geospatial reasoning**. The implementation provides concrete evidence for several standardization needs identified in the AI-DGGS Pilot, including:

- * AI-ready metadata for DGGS collections,
- * standardized mechanisms for DGGS-native aggregation and quantization,
- * auditable representations of AI reasoning processes.

The system also provides a practical foundation for future experimentation with **Scientific Geospatial Foundation Models (GeoFMs)**, as it already enforces DGGS-native data representations, physically meaningful predictors, and provenance-aware workflows.

Planned extensions include tighter integration with real-time sensor streams, DGGS-native probability-of-exceedance products, and volumetric (3D/4D) reasoning within Common Operating Picture (COP) frameworks.

P.8. Demonstrators and Resources

- **Live AI-DGGS Demonstrator:** <https://ai-dggs.hartis.org/>
- **DGGS Playground (tool execution and verification environment):** <https://api-dggs.hartis.org/api/playground/>
- **Flood Impact Index (FII) – Methodology and Reference Annex:** Annex Y



ANNEX Q (INFORMATIVE) H3 OGC API – DGGs IMPLEMENTATION AND INTEROPERABILITY CONSIDERATIONS



ANNEX Q

(INFORMATIVE)

H3 OGC API – DGGS IMPLEMENTATION AND INTEROPERABILITY CONSIDERATIONS

Contributor: Jérôme Jacovella-St-Louis (Ecere)

Q.1. Scope and Rationale

This annex documents clarifications and implementation considerations related to the use of the H3 Discrete Global Grid Reference System (DGGRS) within OGC API – DGGS.

There are three aspects relating to how the H3 library is commonly used which poses some challenges for use as a DGGRS:

- the mapping of data referenced to the WGS84 ellipsoid,
- the exact definition of zone boundaries,
- the ability to identify the exact topological sub-zones of a root zone.

Q.2. Earth reference model

As of March 2026, the Overview section of the H3 library documentation's "H3 internals" ambiguously mentions:

The coordinate reference system (CRS) is spherical coordinates with the WGS84/EPSG:4326 authalic radius. It is common to use WGS84 CRS data with the H3 library.

Internally, the library assumes the globe is a perfect sphere (an authalic sphere, i.e., a sphere with the same area as the WGS84 ellipsoid).

What this was intended to mean in terms of how to use the library to map most geospatial data which is referenced to the WGS84 ellipsoid is not entirely clear, with two possible interpretations:

- A) The geodetic latitude of geographic coordinates passed to the library should first be converted to authalic latitude,
- B) The geodetic latitude of geographic coordinates passed to the library should be passed verbatim, without performing any conversion.

When mapping geographic coordinates returned by the library, interpretation A) would also require converting authalic latitude back to geodetic latitude. The library itself does not automatically perform this geodetic / authalic latitude conversion.

Although H3 is not an equal-area DGGRS, when an authalic sphere is mentioned in the context of an equal-area DGGRS (such as ISEA3H or ISEA7H), it usually implies a conversion between geodetic and authalic latitude. This conversion ensures that zones of an equal-area DGGRS are truly equal on the surface of the ellipsoid (with the geodetic latitude), even though the equal-area projection are performed on the authalic sphere (where the zones with the corresponding authalic latitude are also of equal area), which simplifies calculations. While most H3 users likely in practice use interpretation B) because it is the simplest, the more correct interpretation which should be described as part of a registered H3 DGGRS definition should probably be interpretation A). However, a separate H3 DGGRS might need to be defined as well for the commonly used interpretation B). The ambiguity between these two interpretations is a major source of potential interoperability issues between different implementations for use cases requiring accurate locations. See Figure G.1 as a related example of shifted zones based on whether the conversion between authalic and geodetic latitude is performed or not in the context of the similar ISEA7H DGGRS.

This issue is related to the second challenging aspect described in the next sub-section, since the great circle arcs of the zone boundaries are on the internal authalic sphere, so that whether conversion between authalic and geodetic conversion is performed or not, the great circle arcs are never the true geodesics (the shortest path, which is close to but not exactly great ellipse arcs) on the WGS84 ellipsoid.

Adding a build-time or runtime flag to the H3 library to automatically convert between geodetic/authalic latitude would greatly help all users transitioning to interpretation A).

The conversion between geodetic and authalic latitude is relatively easy to implement, with several open-source libraries implementing this capability, often using an efficient and accurate algorithm described in Charles Karney's [On Auxiliary Latitudes](#). For example, [Geographiclib](#) implements this capability, and the following Python code snippet uses the functionality implemented in DGGAL.

```
# Import and initialize DGGAL
from dggal import *
app = Application(appGlobals=globals())
pydggal_setup(app)

# Calculate the coefficients once based on the WGS84 ellipsoid
# (DGGAL defines wgs84Major and wgs84Minor based on the official
# 6378137 meters major radius and 298.257223563 inverse flattening)
cp = authalicSetup(wgs84Major, wgs84Minor)
```

```

lat = Degrees(45.0)

# Perform a round-trip conversion using the coefficients
authalic = latGeodeticToAuthalic(cp, lat)
geodetic = latAuthalicToGeodetic(cp, authalic)

print("Authalic latitude:", Degrees(authalic))
print("Geodetic latitude:", Degrees(geodetic))

```

Listing Q.1 — Example conversion between authalic and geodetic latitude using DGGAL in Python

Q.3. Zone Boundaries and Topology

Another common misconception is the true nature of the H3 zone boundaries. The H3 library's `cellToBoundary()` function will only return either one or two intermediate points per each of the five or six edges for any zone, depending on whether that edge crosses between two icosahedron faces or not. The true boundary of H3 zones are, for each edge, the one or two great circle arcs on the internal authalic sphere between these points. H3 uses a different gnomonic projection for each face of the spherical icosahedron, which maps all great circle arcs (within that face) to straight lines. However, a straight line on a Cartesian plate carrée projection is not the proper boundary, and neither is a geodesic on the WGS84 ellipsoid.

In order to present a more accurate approximation of the exact zone boundary, intermediate refinement points must be interpolated along those great circle arcs of the authalic sphere, and the authalic latitude of those points should be converted to geodetic latitude to determine points on the WGS84 ellipsoid (assuming interpretation A) as described in previous sub-section).

If this approach is not used, the topological relationship (as defined in [OGC Topic 21 DGGs Topological Query Functions](#) “ZoneQuery”) between the hexagonal grids at the different refinement levels is not maintained, and the alignment of points along zone edges, centroids and vertices is broken. Implementing this approach ensures that the sub-zones are perfectly aligned with the ancestral coarser zones, with the interior sub-zones fully inside, and the sub-zones straddling the coarse zone boundaries at the exact proportions of $\frac{1}{12}$, $\frac{5}{18}$, $\frac{1}{2}$, $\frac{13}{18}$, $\frac{11}{12}$.

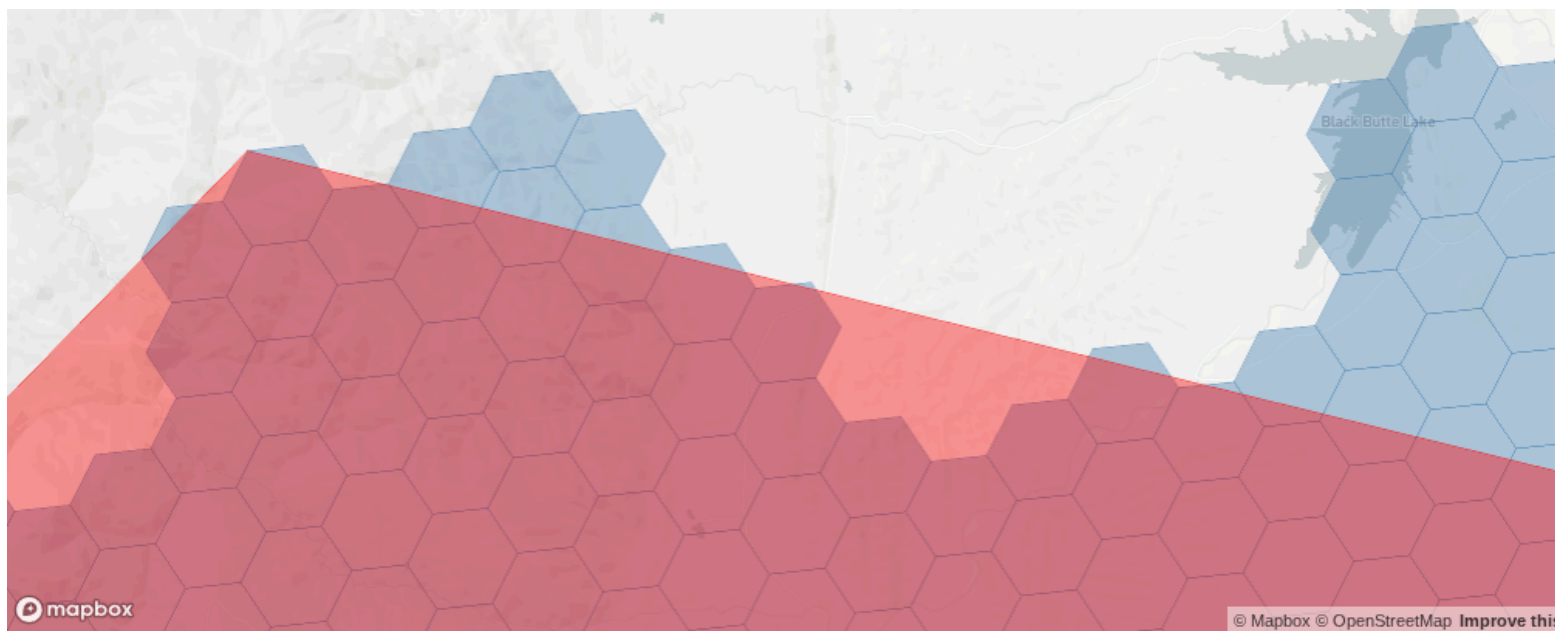


Figure Q.1 — Example of wrong topology resulting from drawing straight lines between vertices returned by `cellToBoundary()`. Notice the tiny proportion of small blue hexagons intersected by the red overlay which is not one of the five ratios. The red overlay is a hexagon which is 5 levels coarser. (from <https://observablehq.com/@nrabinowitz/h3-hierarchical-non-containment>)

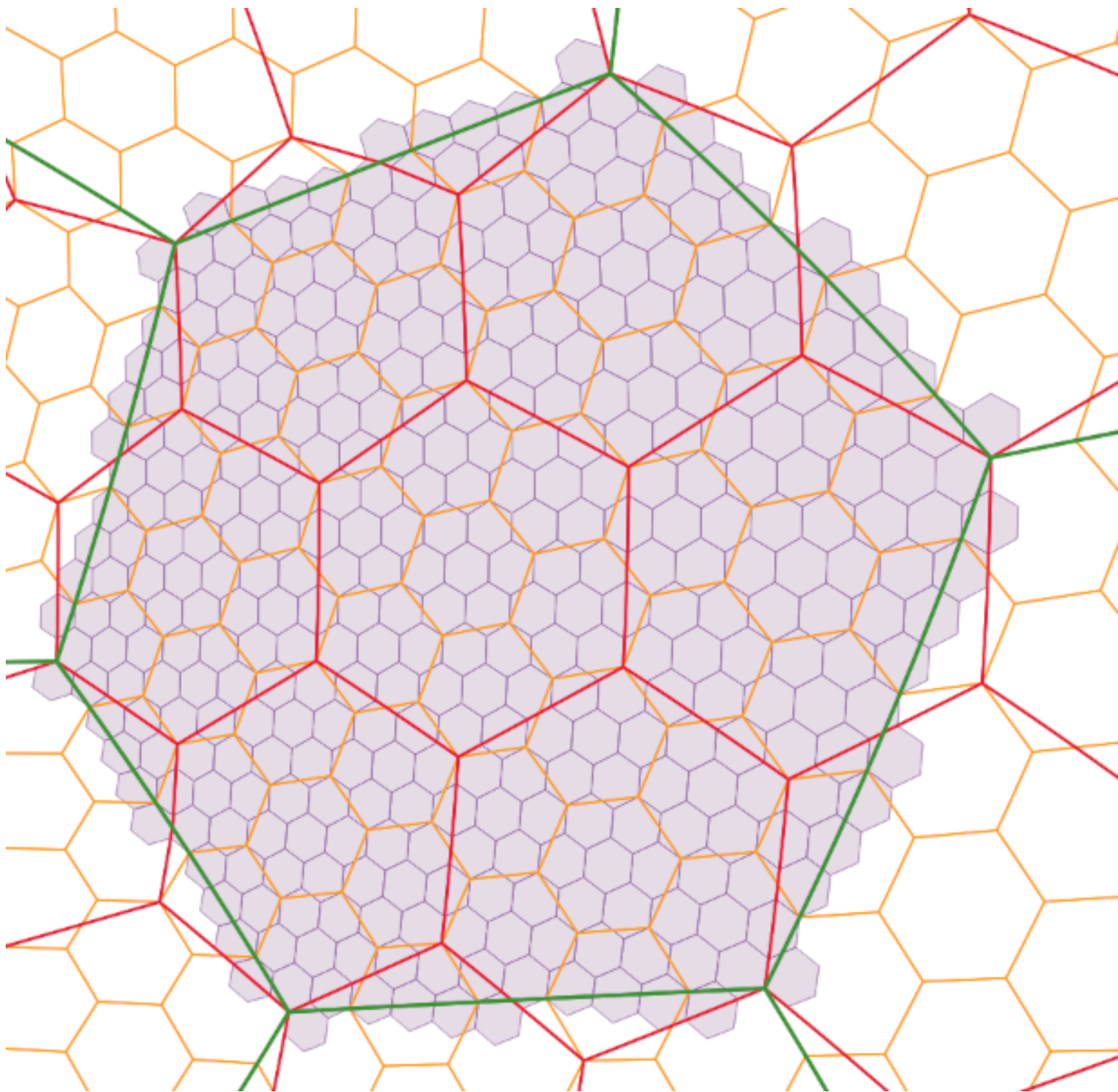


Figure Q.2 — Example of correct topology showing the five valid proportions of bordering sub-zones and the perfect alignment of edges and vertices at relative depths 1, 2 and 3. This example is not using an actual H3 inverse projection, but illustrates the similar topology of ISEA7H.

Adding a function to the official H3 library which returns such a refined zone boundary would greatly facilitate using this proper zone geometry preserving the topological consistency. As part of the planned H3 implementation in DGGAL, the `getZoneRefinedWGS84Vertices()` function would serve this purpose.

Implementing this functionality is relatively straightforward, using spherical linear interpolation (SLERP) between the two or three points of each edge to create additional points on the authalic sphere. Enough intermediate points should be used to achieve the desired accuracy. An implementation of SLERP is available as `geometric_slerp` from SciPy, and a custom implementation using only the standard Python math library is also presented in Annex R.

Q.4. Topological sub-zones

The official library does not yet support the “sub-zone” concept used by DGGS-JSON(-FG) encoding, and the related DGGS-quantized profiles of Zarr, netCDF, CoverageJSON encodings. That is, one can only query the seven logical hierarchical indexing children, not which finer zones have their interior intersecting a larger zone’s interior (13 sub-zones at relative depth of 1).

While the official H3 library did not yet include built-in support at the time of writing this report, it is still possible to leverage the existing functionality of the library to calculate sub-zones. The Annex R section presents such an algorithm in both Python and Java. However, the performance of these sub-zone calculations could be significantly improved by avoiding the re-projection and remaining in the icosahedral space.

Q.5. Towards an H3 DGGRS definition and OGC Best Practice

This annex could be useful for the formal definition of a DGGRS definition for use with OGC API – DGGS to be registered in the [OGC DGGRS register](#), as well as possibly as a starting point to evolve this guidance into an OGC Best Practice on **Using H3 with OGC API – DGGS**.



ANNEX R (INFORMATIVE) H3 SUBZONE ALGORITHM



ANNEX R

(INFORMATIVE)

H3 SUBZONE ALGORITHM

Contributors: Jérôme Jacovella-St-Louis (Ecere), Johann Sorel (Geomatys)

R.1. H3 subzone, the missing feature of H3

H3 DGGRS has been available for several years and is used in analysis and processing in various situations. This DGGRS, being a precursor, did not have all the required functions to be used in the OGC DGGRS API.

The **subzone*** function which was missing from H3 has been implemented during this pilot to ensure clients and servers obtain the same zone ids when requesting subzones.

Algorithm has been provided to the H3 project. (<https://github.com/uber/h3/issues/1095>)

R.2. Computing H3 subzones

```
#!/usr/bin/python3
import h3
import math
import time
from multiprocessing import Pool, cpu_count

def slerp(start_lat, start_lng, end_lat, end_lng, t):
    """
    Performs spherical linear interpolation between two geographic points
    with optimizations to compute trigonometric values only once.
    Assumes a perfect sphere model for simplicity.
    """
    # Convert to radians once
    start_lat_rad = math.radians(start_lat)
    start_lng_rad = math.radians(start_lng)
    end_lat_rad = math.radians(end_lat)
    end_lng_rad = math.radians(end_lng)

    # Pre-calculate trigonometric values
    cos_start_lat = math.cos(start_lat_rad)
```

```

sin_start_lat = math.sin(start_lat_rad)
cos_end_lat = math.cos(end_lat_rad)
sin_end_lat = math.sin(end_lat_rad)
delta_lng = end_lng_rad - start_lng_rad

# Calculate omega (angular distance)
cos_omega = sin_start_lat * sin_end_lat + cos_start_lat * cos_end_lat *
math.cos(delta_lng)

# Clamp cos_omega to [-1, 1] due to floating point inaccuracies
cos_omega = max(-1.0, min(1.0, cos_omega))
omega = math.acos(cos_omega)

# Handle the case where the two points are the same
if omega == 0.0:
    return (start_lat, start_lng)

# Calculate sin(omega) only once
sin_omega = math.sin(omega)

# Calculate the blend factors
a = math.sin((1.0 - t) * omega) / sin_omega
b = math.sin(t * omega) / sin_omega

# Calculate interpolated coordinates
x = a * cos_start_lat * math.cos(start_lng_rad) + b * cos_end_lat *
math.cos(end_lng_rad)
y = a * cos_start_lat * math.sin(start_lng_rad) + b * cos_end_lat *
math.sin(end_lng_rad)
z = a * sin_start_lat + b * sin_end_lat

lat = math.atan2(z, math.sqrt(x*x + y*y))
lng = math.atan2(y, x)

return (math.degrees(lat), math.degrees(lng))

def zoneHasSubZone(ancestor_zone, candidate_subzone):
    """
    Checks if a candidate H3 index is geometrically contained within an
    ancestral H3 index using a vertex-based boundary check with SLERP.
    """
    if h3.get_resolution(candidate_subzone) <= h3.get_resolution(ancestor_
zone):
        return False

    ancestor_resolution = h3.get_resolution(ancestor_zone)
    centroid_lat, centroid_lng = h3.cell_to_latlng(candidate_subzone)
    vertex_indices = h3.cell_to_vertexes(candidate_subzone)

    for vertex_index in vertex_indices:
        if vertex_index == 0: continue

        vertex_lat, vertex_lng = h3.vertex_to_latlng(vertex_index)

        interpolated_lat, interpolated_lng = slerp(
            vertex_lat, vertex_lng, centroid_lat, centroid_lng, 0.01
        )

        coarser_zone = h3.latlng_to_cell(interpolated_lat, interpolated_lng,
ancestor_resolution)

        if coarser_zone == ancestor_zone:
            return True

```

```

    return False

def _process_neighbor(args):
    """
    Helper function for multiprocessing. It processes a single neighbor
    and returns its valid descendants.
    """
    ancestor_zone, neighbor, target_resolution = args
    valid_descendants = []
    descendants = h3.cell_to_children(neighbor, target_resolution)
    for descendant in descendants:
        if zoneHasSubZone(ancestor_zone, descendant):
            valid_descendants.append(descendant)
    return valid_descendants

def getGeometricSubZones(ancestor_zone, relative_depth):
    """
    Calculates all geometrically contained sub-zones within an ancestral zone
    at a specific relative depth, with parallel processing for rings.
    """
    ancestor_resolution = h3.get_resolution(ancestor_zone)
    target_resolution = ancestor_resolution + relative_depth

    if not 0 <= target_resolution <= 15:
        raise ValueError("Relative depth results in an invalid resolution.")

    # Get the centroid child
    centroid_lat, centroid_lng = h3.cell_to_latlng(ancestor_zone)
    centroid_child = h3.latlng_to_cell(centroid_lat, centroid_lng, ancestor_
resolution + 1)

    candidate_cells = []

    # 1. Add all logical descendants of the centroid child (sequential)
    if centroid_child is not None:
        centroid_descendants = h3.cell_to_children(centroid_child, target_
resolution)
        candidate_cells.extend(centroid_descendants)

    # 2. Get ring 1 neighbors of the centroid child
    ring1_neighbors = list(h3.grid_ring(centroid_child, 1))

    # 3. Get ring 2 neighbors of the centroid child
    ring2_neighbors = list(h3.grid_ring(centroid_child, 2))

    # Prepare arguments for parallel processing
    ring_neighbors_to_process = []
    for neighbor in ring1_neighbors:
        ring_neighbors_to_process.append((ancestor_zone, neighbor, target_
resolution))
    for neighbor in ring2_neighbors:
        ring_neighbors_to_process.append((ancestor_zone, neighbor, target_
resolution))

    # Parallelize processing of the rings
    # Use 'map' to maintain the order of the results
    if ring_neighbors_to_process:
        with Pool(cpu_count()) as pool:
            # The 'map' function ensures the results list has the same order
            # as the input list, preserving the ring order.

```

```

        parallel_results = pool.map(_process_neighbor, ring_neighbors_to_
process)

    # Collect results in the correct order
    for result_list in parallel_results:
        candidate_cells.extend(result_list)

    return candidate_cells

# --- Expanded Test Suite ---
if __name__ == "__main__":
    relative_depths = [1, 2, 3, 4, 5, 6]

    # Select one specific pentagon from the list for testing
    pentagon_res3 = list(h3.get_pentagons(3))
    pentagon_res2 = list(h3.get_pentagons(2))

    test_cases = [
        ("Hexagonal_NYC_Res5", h3.latlng_to_cell(40.7128, -74.0060, 5)),
        ("Hexagonal_London_Res4", h3.latlng_to_cell(51.5074, -0.1278, 4)),
        ("Pentagonal_SouthPole_Res3", pentagon_res3[0]),
        ("Pentagonal_Atlantic_Res2", pentagon_res2[0]),
        ("Hexagonal_HighRes_Res10", h3.latlng_to_cell(34.0522, -118.2437, 10)),
        ("Hexagonal_Res3_at_-80_0", h3.latlng_to_cell(-80.0, 0.0, 3)),
    ]

    print("Starting comprehensive test suite...\n")
    for name, parent_h3 in test_cases:
        parent_resolution = h3.get_resolution(parent_h3)
        parent_type = "Pentagonal" if h3.is_pentagon(parent_h3) else
"Hexagonal"

        print(f"--- Testing {name} ({parent_type}, Res={parent_resolution}, H3=
{parent_h3}) ---")

        for relative_depth in relative_depths:
            target_resolution = parent_resolution + relative_depth

            if target_resolution > 15:
                print(f"    Skipping depth {relative_depth}: Target resolution
{target_resolution} exceeds max (15).")
                continue

            try:
                start_time = time.perf_counter()
                geometric_subzones = getGeometricSubZones(parent_h3, relative_
depth)
                end_time = time.perf_counter()
                duration = end_time - start_time
                print(f"    Found {len(geometric_subzones)} geometric sub-zones
at resolution {target_resolution} (Depth {relative_depth}). Took {duration:.4f}
seconds.")
            except ValueError as e:
                print(f"    Error testing depth {relative_depth}: {e}")

        print("-" * 30 + "\n")

```

Listing R.1 — Python algorithm

```

/**
 * Python code provided by Jerome Saint-Louis to ensure H3 subchild results
match DGGRS requirements.
 */

```



```

*
* @author Jerome Saint-Louis (Ecere), original code in python
* @author Johann Sorel (Geomatys), java port
*/
public final class H3Ext {

    /**
     * Performs spherical linear interpolation between two geographic points.
     * Assumes a perfect sphere model for simplicity.
     */
    public static LatLng slerp(LatLng start, LatLng end, double ratio) {
        final double startLatRad = Math.toRadians(start.lat);
        final double startLngRad = Math.toRadians(start.lng);
        final double endLatRad = Math.toRadians(end.lat);
        final double endLngRad = Math.toRadians(end.lng);

        final double sinStartLatRad = Math.sin(startLatRad);
        final double cosStartLatRad = Math.cos(startLatRad);
        final double sinEndLatRad = Math.sin(endLatRad);
        final double cosEndLatRad = Math.cos(endLatRad);

        final double cosOmega = sinStartLatRad * sinEndLatRad
            + cosStartLatRad * cosEndLatRad *
Math.cos(endLngRad - startLngRad);
        final double omega = Math.acos(cosOmega);

        if (omega == 0.0) {
            return new LatLng(start.lat, start.lng);
        }

        final double sinOmega = Math.sin(omega);
        final double a = Math.sin((1.0 - ratio) * omega) / sinOmega;
        final double b = Math.sin(ratio * omega) / sinOmega;

        final double x = a * cosStartLatRad * Math.cos(startLngRad)
            + b * cosEndLatRad * Math.cos(endLngRad);
        final double y = a * cosStartLatRad * Math.sin(startLngRad)
            + b * cosEndLatRad * Math.sin(endLngRad);
        final double z = a * sinStartLatRad + b * sinEndLatRad;

        final double lat = Math.atan2(z, Math.sqrt(x*x + y*y));
        final double lng = Math.atan2(y, x);

        return new LatLng(Math.toDegrees(lat), Math.toDegrees(lng));
    }

    /**
     * Checks if a candidate H3 index is geometrically contained within an
     * ancestral H3 index using a vertex-based boundary check with SLERP.
     */
    public static boolean zoneHasSubZone(long ancestorZone, long
candidateSubzone) {
        final int candidateRes = H3Index.getResolution(candidateSubzone);
        final int ancestorRes = H3Index.getResolution(ancestorZone);
        if (candidateRes <= ancestorRes) {
            return false;
        }

        final LatLng centroid = H3Dggrs.H3.cellToLatLng(candidateSubzone);
        final List<Long> vertexIndices =
H3Dggrs.H3.cellToVertexes(candidateSubzone);

        for (Long vertexIndex : vertexIndices) {

```

```

        if (vertexIndex == 0) continue;

        final LatLng vertex = H3Dggrs.H3.vertexToLatLng(vertexIndex);
        final LatLng interpolated = slerp(vertex, centroid, 0.01);
        final long coarser_zone = H3Dggrs.H3.latLngToCell(interpolated.lat,
interpolated.lng, ancestorRes);

        if (coarser_zone == ancestorZone) {
            return true;
        }
    }

    return false;
}

/**
 * Calculates all geometrically contained sub-zones within an ancestral
zone
 * at a specific relative depth, using a rigorous topological search and
 * the specified optimization.
 */
public static LongStream getGeometricSubZones(long ancestorZone, int
relativeDepth) {
    final int ancestorRes = H3Index.getResolution(ancestorZone);
    final int targetRes = ancestorRes + relativeDepth;
    if (targetRes < 0 || targetRes > 15) {
        throw new IllegalArgumentException("Relative depth results in an
invalid resolution.");
    }

    final long centerChild = H3Dggrs.H3.cellToCenterChild(ancestorZone,
ancestorRes+1);
    final List<Long> rings = new ArrayList<>();
    rings.addAll(H3Dggrs.H3.gridRing(centerChild, 1));
    rings.addAll(H3Dggrs.H3.gridRing(centerChild, 2));

    LongStream stream = LongStream.empty();
    stream = LongStream.concat(stream,
H3Dggrs.H3.cellToChildren(centerChild,
targetRes).stream().mapToLong(Long::longValue));
    for (Long r : rings) {
        LongStream s = H3Dggrs.H3.cellToChildren(r, targetRes)
            .parallelStream()
            .filter((Long l) -> zoneHasSubZone(ancestorZone, l))
            .mapToLong(Long::longValue);
        stream = LongStream.concat(stream, s);
    }

    return stream;
}
}

```

Listing R.2 — Java algorithm



ANNEX S (INFORMATIVE) APACHE-SIS JAVA DGGRS API



ANNEX S

(INFORMATIVE)

APACHE-SIS JAVA DGGRS API

Contributor: Johann Sorel (Geomatys)

The following UML describe the API created for DGGRS in [Apache SIS](#).

After reading and processing the different DGGRS documents and existing ISO Reference System specifications. We noted one major change with the [Topic 21 – Discrete Global Grid Systems](#) document. The decision was made to make DGGRS extent `ReferenceByIdentifiers`(ISO 19112) instead of `CoordinateReferenceSystem`(ISO 19111). This choice do not have any impact on the exposed OGC DGGRS API.

The reasons behind this choice are:

- Zone ID are Integer 64bits, not decimal values. All CRS API (SIS, PROJ, ...) expect decimal values, at best 64bits, so there is not a direct mapping between float64 and int64;
- Zone ID may extent 64Bits (for 3D, 4D or more) or may be texts;
- Zones have a 'depth' or 'level' information, this property is correctly defined in `ReferenceByIdentifiers`, but nonexistent in CRS; and
- Zones are areas/volumes, not exact positions, which matches the `ReferenceByIdentifiers` locations.

The API and code developed here is a preliminary work for a futur integration in [OGC GeoAPI](#)

S.1. Overview

The created API is composed of two new packages.

- RS : which contain a new API for Compound Reference Systems. Allowing to aggregate DGGRS with additional dimensions, such as temporal and vertical.
- DGGRS : which contain the new DGGRS API.

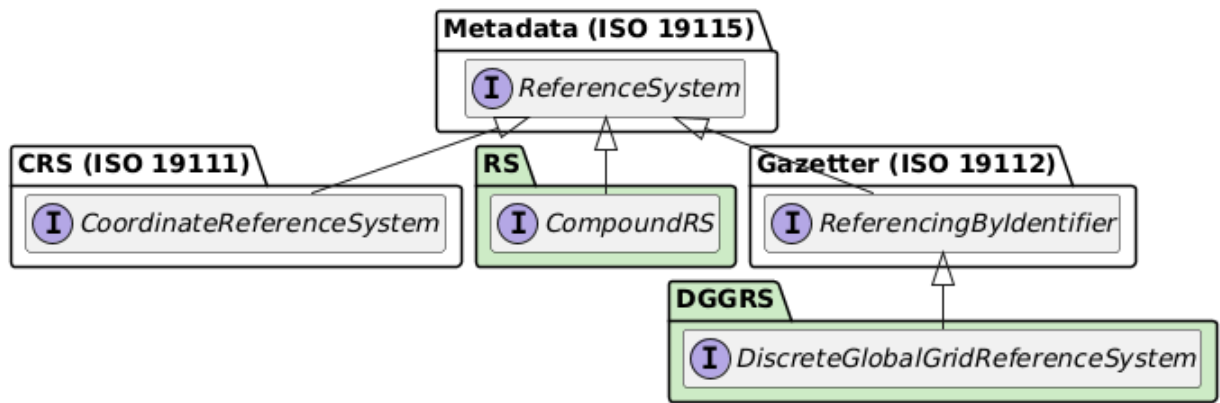


Figure S.1

S.2. Discrete Global Grid Reference System API

The UML below is the result from aggregating and factorisation of the different API which exist in : [OGC DGGRS API](#), [Topic 21 – Discrete Global Grid Systems](#), [H3geo](#), [S2geometry](#), [DGGAL](#) and [Healpix CDS](#)

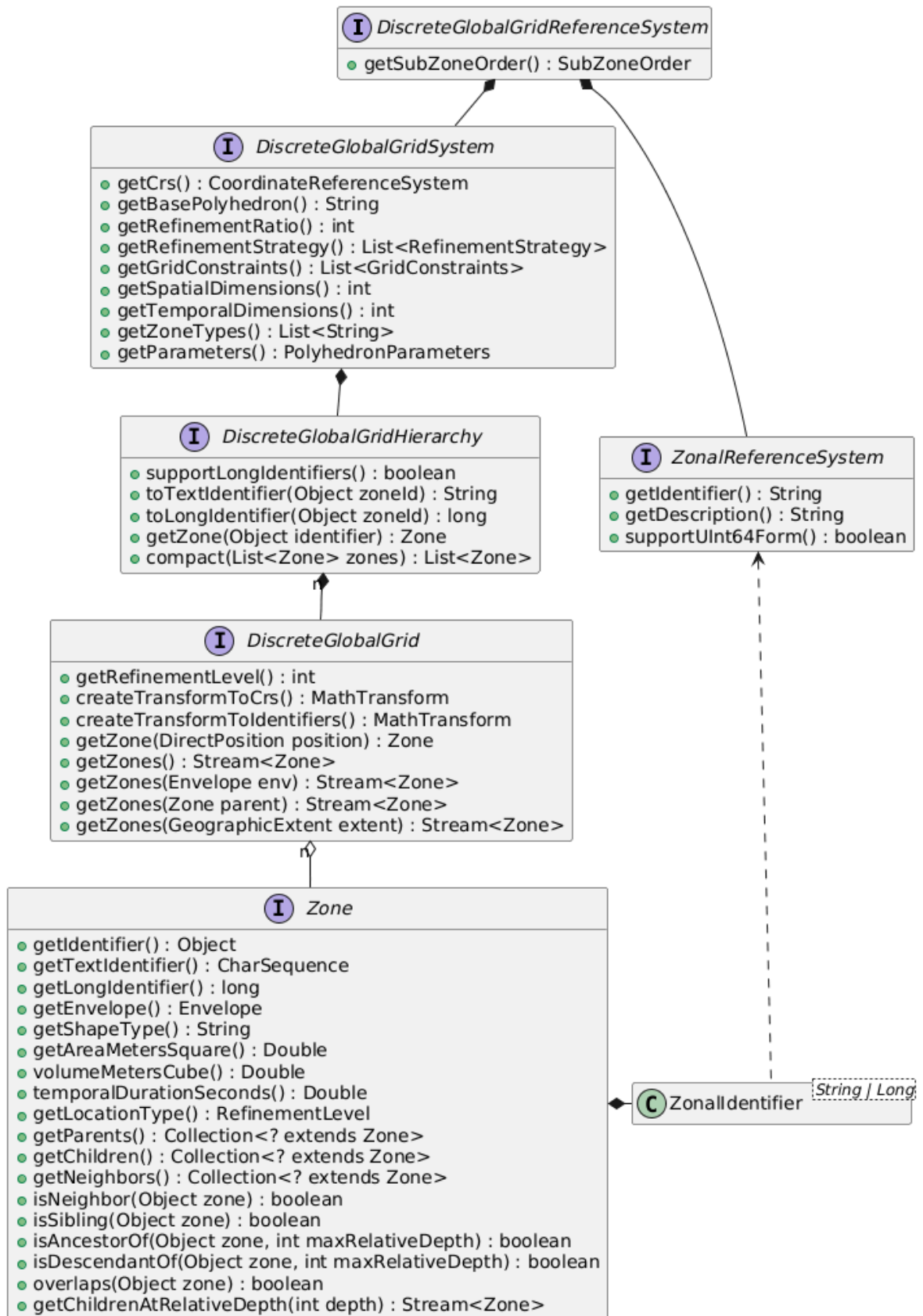


Figure S.2

S.3. Reference System

Since all the DGGRS which were used in the pilot were limited to 2D, we needed a solution for combining it with additional dimensions to support time and elevation. The resulting API composed of the three classes **Code**, **CompoundRS** and **CodeOperation** share the same organisation as **DirectPosition**, **CoordinateReferenceSystem** and **Operation**.

They extend the scope of transformation to all kind of Reference systems, may they be using numerical or textual. Allowing transformation for DGGRS, MGRS, GeoHash, OLS or any kind of code base systems.

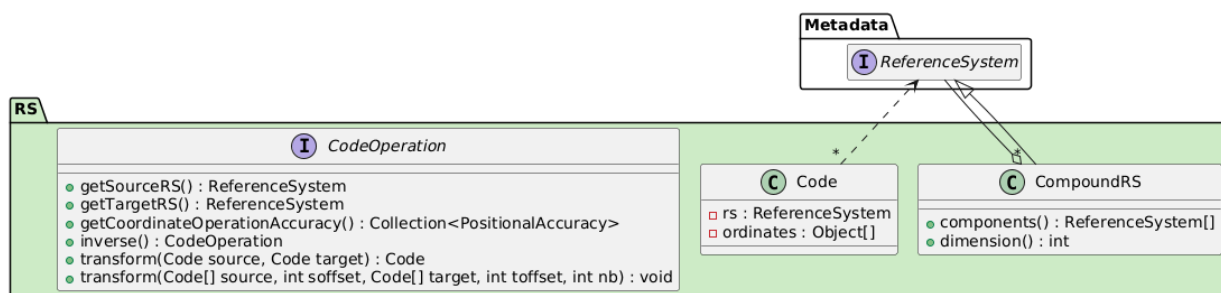


Figure S.3

S.4. Resource and Data API

The diagram below shows where the new DGGRS resource classes are located in Apache-SIS Resource API.

We do not document those API since they are still evolving at this time.

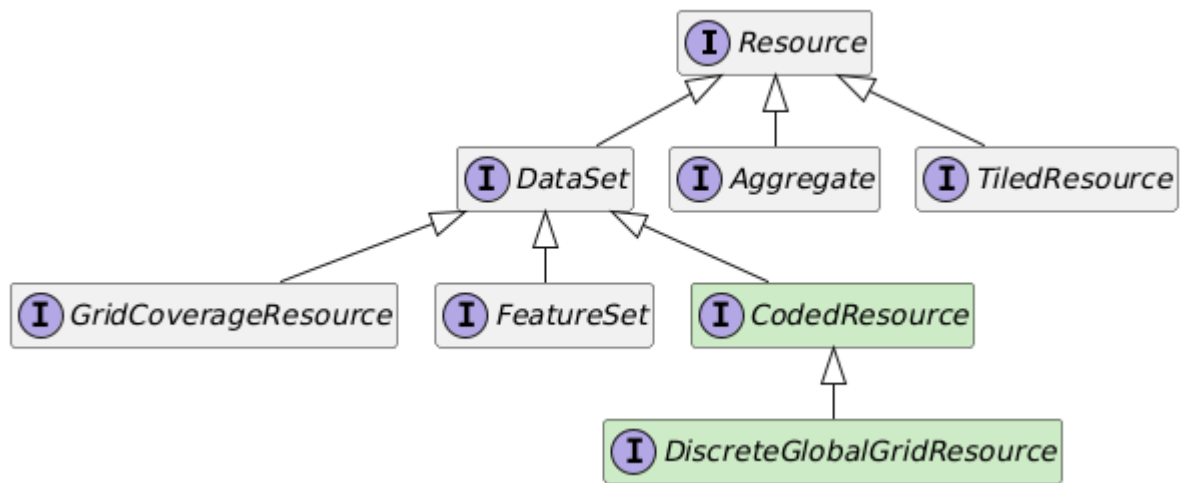


Figure S.4

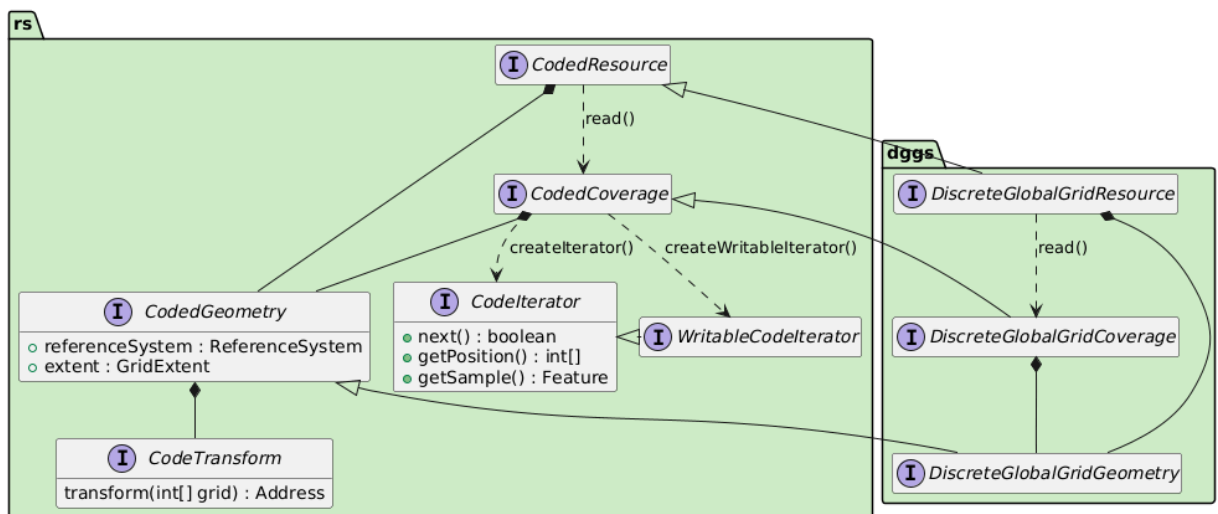


Figure S.5



ANNEX T (INFORMATIVE) DGGAL JAVA BINDING



ANNEX T (INFORMATIVE) DGGAL JAVA BINDING

Contributor: Johann Sorel (Geomatys)

T.1. Panama binding

DGGAL Java bindings were made available in the Geotoolkit project at : <https://github.com/Geomatys/geotoolkit/tree/main/geotk-dggal>

This binding will move in Apache-SIS after they have passed all reviews and quality checks.

T.2. Relationship between DGGAL API and Apache SIS API

The relationship between DGGAL and Apache SIS API is not one-to-one for several reasons.

- DGGAL C API is flat (only functions), while Java API is object-oriented
- Apache SIS API tries to provide a general purpose DGGRS API which can be implemented by all other libraries (S2,H3,Healpix,A5). It is a common denominator API.
- Apache SIS already has a large API for any referencing needs, many utility functions from DGGAL are therefore duplicates of what can be done elsewhere in Apache SIS

Table T.1

DGGAL	APACHE SIS
DGGAL_DGGRS_getZoneFromTextID	DiscreteGlobalGridHierarchy.getZone(zid)

DGGAL	APACHE SIS
DGGAL_DGGRS_getZoneLevel	Zone.getLocationType().getRefinementLevel()
DGGAL_DGGRS_getRefinementRatio	DiscreteGlobalGridSystem.getRefinementRatio()
DGGAL_DGGRS_getMaxDGGRSZoneLevel	DiscreteGlobalGridHierarchy.getGrids().size()
DGGAL_DGGRS_getZoneWGS84Centroid	Zone.getPosition()
DGGAL_DGGRS_getZoneWGS84Vertices	Zone.getGeographicExtent()
DGGAL_DGGRS_getZoneArea	Zone.getAreaMetersSquare()
DGGAL_DGGRS_getZoneTextID	Zone.getTextIdentifier()
DGGAL_DGGRS_getZoneParents	Zone.getParents()
DGGAL_DGGRS_getZoneChildren	Zone.getChildren()
DGGAL_DGGRS_getZoneNeighbors	Zone.getNeighbors()
DGGAL_DGGRS_getZoneWGS84Extent	Zone.getEnvelope()
DGGAL_DGGRS_listZones	DiscreteGlobalGrid.getZones(envelope)
DGGAL_DGGRS_getSubZones	DiscreteGlobalGrid.getZones(parent)
DGGAL_DGGRS_getZoneFromWGS84Centroid	DiscreteGlobalGrid.getZone(position)
DGGAL_DGGRS_getSubZoneAtIndex	DiscreteGlobalGrid.getZones(parent).skip(index)
DGGAL_DGGRS_getSubZoneCRSCentroids	DiscreteGlobalGrid.getZones(parent) + Zone.getPosition() + any transform
DGGAL_DGGRS_getSubZoneWGS84Centroids	DiscreteGlobalGrid.getZones(parent) + Zone.getPosition() + any transform
DGGAL_DGGRS_getZoneCRSCentroid	Zone.getPosition() + any transform
DGGAL_DGGRS_getZoneCRSExtent	Zone.getEnvelope() + any transform
DGGAL_DGGRS_compactZones	DiscreteGlobalGridHierarchy.compact(zones)
DGGAL_DGGRS_areZonesNeighbors	Zone.isNeighbor(other)

DGGAL	APACHE SIS
DGGAL_DGGRS_areZonesSiblings	Zone.isSibling(other)
DGGAL_DGGRS_doZonesOverlap	Zone.overlaps(other)
DGGAL_DGGRS_isZoneAncestorOf	Zone.isAncestorOf(other, maxRelDepth)
DGGAL_DGGRS_isZoneDescendantOf	Zone.isDescendantOf(other, maxRelDepth)
DGGAL_DGGRS_doesZoneContain	-
DGGAL_DGGRS_isZoneContainedIn	-
DGGAL_DGGRS_zoneHasSubZone	-
Operation is handled by SIS in a different API or should not be in a DGGRS API.	
DGGAL_DGGRS_getZoneRefinedWGS84Vertices	-
DGGAL_DGGRS_getZoneRefinedCRSVertices	-
DGGAL_DGGRS_getLevelFromMetersPerSubZone	-
DGGAL_DGGRS_getLevelFromPixelsAndExtent	-
DGGAL_DGGRS_getLevelFromRefZoneArea	-
DGGAL_DGGRS_getLevelFromScaleDenominator	-
DGGAL_DGGRS_getMetersPerSubZoneFromLevel	-
DGGAL_DGGRS_getRefZoneArea	-
DGGAL_DGGRS_getScaleDenominatorFromLevel	-
Redundant methods.	

DGGAL	APACHE SIS
DGGAL_DGGRS_isZoneImmediateChildOf	Zone.isDescendantOf(other, 1)
DGGAL_DGGRS_isZoneImmediateParentOf	Zone.isAncestorOf(other, 1)
DGGAL_DGGRS_getFirstSubZone	Zone.getChildren().limit(1)
DGGAL_DGGRS_getSubZoneIndex	Zone.getChildren().filter(Objects::equals)
Methods are needed for C-like programming to allocate array sizes, which is irrevelant in java.	
DGGAL_DGGRS_countZones	-
DGGAL_DGGRS_countZoneEdges	-
DGGAL_DGGRS_countSubZones	-
DGGAL_DGGRS_getMaxChildren	-
DGGAL_DGGRS_getMaxNeighbors	-
DGGAL_DGGRS_getMaxParents	-
Candidate methods for the API, not sure yet.	
DGGAL_DGGRS_getZoneCentroidParent	-
DGGAL_DGGRS_getZoneCentroidChild	-
DGGAL_DGGRS_isZoneCentroidChild	-
DGGAL_DGGRS_getIndexMaxDepth	-

DGGAL	APACHE SIS
DGGAL_DGGRS_get64KDepth	-
DGGAL_DGGRS_getMaxDepth	-

This mapping is not final, and until the DGGRS Java API reaches the GeoAPI project it will evolve.



ANNEX U (INFORMATIVE) CLIENT/SERVER DGGRS AND FORMATS BENCHMARKS



ANNEX U

(INFORMATIVE)

CLIENT/SERVER DGGRS AND FORMATS BENCHMARKS

Contributor: Johann Sorel (Geomatys)

U.1. Method

The benchmark has been made with Examind server and Apache-SIS client, on a local computer with the following configuration.

- Intel® Core™ i7-8750H CPU @ 2.20GHz 12 threads
- 32GB ram

The measured operations are for a standard case scenario using Apache SIS DGGRS API. Executed by a single client in a single thread. The queries data is a small version of BlueMarble in geotiff.

- Client : Java DGGRS API Query to OGC DGGRS API Query based on service capabilities
- Client : Establish connection
- Client : Sending request
- Server : Parsing request
- Server : validate request
- Server : Compute DGGRS zones
- Server : Fetch coverage data
- Server : Resample to DGGRS zones
- Server : Encode to DGGS-Json
- Server : Send response

- Client : Parse DGGRS-Json
- Client : DGGRS-Json to Java DGGRS API coverage

U.2. Formats performances

U.3. Response size by format and zone depth

Table U.1 — Response size by format and zone depth, values are in bytes.

Zone-Depth	Zones	JSON	UBJSON	CBOR	JSON+GZIP	UBJSON+GZIP	CBOR+GZIP
1	4	343	303	253	249	245	213
2	16	568	411	361	363	305	272
3	64	1457	843	796	755	498	474
4	256	4879	2491	2447	2139	1160	1131
5	1024	18437	9019	8975	6885	3631	3594
6	4096	72192	34851	34807	23642	13368	13335
7	16384	286820	137835	137791	87786	50921	50884
8	65536	1143576	548971	548933	337873	196995	196960

Figure U.1

U.4. Response size ratio compared with JSON

Table U.2 — Response size ratio compared with JSON

Zones	UBJSON	CBOR	JSON+GZIP	UBJSON+GZIP	CBOR+GZIP
4	0.8833819241982	0.7376093294460	0.7259475218658	0.7142857142857	0.620991253644315
16	0.7235915492957	0.6355633802816	0.6390845070422	0.5369718309859	0.47887323943662
64	0.5785861358956	0.5463280713795	0.5181880576527	0.3417982155113	0.325326012354152
256	0.5105554416888	0.5015372002459	0.4384095101455	0.2377536380405	0.231809797089568
1024	0.4891793675760	0.4867928621793	0.3734338558333	0.1969409339914	0.194934099907794
4096	0.4827543218085	0.4821448359929	0.3274878102836	0.1851728723404	0.184715757978723
16384	0.4805627222648	0.4804093159472	0.3060665225577	0.1775364340004	0.177407433233387
65536	0.4800476750124	0.4800144459135	0.2954530350409	0.1722622720308	0.172231666281909

Figure U.2

U.5. Response time by format and zone depth

Table U.3 — Response time by format and zone depth for S2 DGGRS, values are in milliseconds

Zones	4	16	64	256	1024	4096	16384	65536
JSON	139.5	137	138	137.5	140	146	172.5	291
UBJON	137.5	139	139	147	138.5	146	173.5	276.5
CBOR	141.5	149.5	139.5	152.5	139.5	146.5	170	276

JSON +GZIP	136	136	137	137.5	139.5	147	176.5	295
UBJSON +GZIP	136.5	136	139	140	141	146	171	279.5
CBOR +GZIP	137	142	138.5	139	138.5	145.5	172	275

Figure U.3

U.6. DGGRS performances

Table U.4 — Response time by DGGRS and zone depth

DGGRS	Format	Depth	Zone-Depth	Zoneld	Count	Min(ms)	Max(ms)	Mean (ms)	Rms	t/z
H3	ubjson+gzip	8	1	88271c06	13	144	146	145	145.003	11.154
H3	ubjson+gzip	8	2	88271c06	55	140	144	142	142.014	2.582
H3	ubjson+gzip	8	3	88271c06	379	147	150	148.5	148.508	0.392
H3	ubjson+gzip	8	4	88271c06	2449	183	186	184.5	184.506	0.075
H3	ubjson+gzip	8	5	88271c06	17053	457	465	461	461.017	0.027
H3	ubjson+gzip	8	6	88271c06	117991	2395	2400	2397.5	2397.501	0.02
S2	ubjson+gzip	8	1	52c19	4	137	137	137	137	34.25
S2	ubjson+gzip	8	2	52c19	16	138	139	138.5	138.501	8.656
S2	ubjson+gzip	8	3	52c19	64	137	140	138.5	138.508	2.164
S2	ubjson+gzip	8	4	52c19	256	137	141	139	139.014	0.543
S2	ubjson+gzip	8	5	52c19	1024	138	139	138.5	138.501	0.135

S2	ubjson +gzip	8	6	52c19	4096	145	146	145.5	145.501	0.036
S2	ubjson +gzip	8	7	52c19	16384	169	173	171	171.012	0.01
S2	ubjson +gzip	8	8	52c19	65536	281	289	285	285.028	0.004
Healpix	ubjson +gzip	8	1	125e7e	4	137	138	137.5	137.501	34.375
Healpix	ubjson +gzip	8	2	125e7e	16	144	150	147	147.031	9.188
Healpix	ubjson +gzip	8	3	125e7e	64	136	137	136.5	136.501	2.133
Healpix	ubjson +gzip	8	4	125e7e	256	137	138	137.5	137.501	0.537
Healpix	ubjson +gzip	8	5	125e7e	1024	139	142	140.5	140.508	0.137
Healpix	ubjson +gzip	8	6	125e7e	4096	149	153	151	151.013	0.037
Healpix	ubjson +gzip	8	7	125e7e	16384	187	191	189	189.011	0.012
Healpix	ubjson +gzip	8	8	125e7e	65536	336	351	343.5	343.582	0.005
ISEA3H	ubjson +gzip	8	1	E0- 10A9-A	7	136	137	136.5	136.501	19.5
ISEA3H	ubjson +gzip	8	2	E0- 10A9-A	13	135	136	135.5	135.501	10.423
ISEA3H	ubjson +gzip	8	3	E0- 10A9-A	37	138	138	138	138	3.73
ISEA3H	ubjson +gzip	8	4	E0- 10A9-A	91	137	137	137	137	1.505
ISEA3H	ubjson +gzip	8	5	E0- 10A9-A	271	138	139	138.5	138.501	0.511
ISEA3H	ubjson +gzip	8	6	E0- 10A9-A	757	141	143	142	142.004	0.188
ISEA3H	ubjson +gzip	8	7	E0- 10A9-A	2269	148	153	150.5	150.521	0.066
ISEA3H	ubjson +gzip	8	8	E0- 10A9-A	6643	172	175	173.5	173.506	0.026
ISEA3H	ubjson +gzip	8	9	E0- 10A9-A	19927	246	247	246.5	246.501	0.012

ISEA3H	ubjson +gzip	8	10	E0- 10A9-A	59293	459	474	466.5	466.56	0.008
IVEA3H	ubjson +gzip	8	1	E0- 10A9-A	7	140	142	141	141.004	20.143
IVEA3H	ubjson +gzip	8	2	E0- 10A9-A	13	140	148	144	144.056	11.077
IVEA3H	ubjson +gzip	8	3	E0- 10A9-A	37	137	137	137	137	3.703
IVEA3H	ubjson +gzip	8	4	E0- 10A9-A	91	142	142	142	142	1.56
IVEA3H	ubjson +gzip	8	5	E0- 10A9-A	271	145	171	158	158.534	0.583
IVEA3H	ubjson +gzip	8	6	E0- 10A9-A	757	145	146	145.5	145.501	0.192
IVEA3H	ubjson +gzip	8	7	E0- 10A9-A	2269	153	157	155	155.013	0.068
IVEA3H	ubjson +gzip	8	8	E0- 10A9-A	6643	180	180	180	180	0.027
IVEA3H	ubjson +gzip	8	9	E0- 10A9-A	19927	253	253	253	253	0.013
IVEA3H	ubjson +gzip	8	10	E0- 10A9-A	59293	481	482	481.5	481.5	0.008
ISEA7H	ubjson +gzip	8	1	I0- 37EE56- A	13	137	137	137	137	10.538
ISEA7H	ubjson +gzip	8	2	I0- 37EE56- A	55	137	140	138.5	138.508	2.518
ISEA7H	ubjson +gzip	8	3	I0- 37EE56- A	379	141	142	141.5	141.501	0.373
ISEA7H	ubjson +gzip	8	4	I0- 37EE56- A	2449	148	151	149.5	149.508	0.061
ISEA7H	ubjson +gzip	8	5	I0- 37EE56- A	17053	247	247	247	247	0.014
ISEA7H	ubjson +gzip	8	6	I0- 37EE56- A	117991	669	671	670	670.001	0.006

IVEA7H	ubjson +gzip	8	1	IO- 3813DD- 13 A	137	147	142	142.088	10.923
IVEA7H	ubjson +gzip	8	2	IO- 3813DD- 55 A	135	137	136	136.004	2.473
IVEA7H	ubjson +gzip	8	3	IO- 3813DD- 379 A	139	139	139	139	0.367
IVEA7H	ubjson +gzip	8	4	IO- 3813DD- 2449 A	146	147	146.5	146.501	0.06
IVEA7H	ubjson +gzip	8	5	IO- 3813DD- 17053 A	237	241	239	239.008	0.014
IVEA7H	ubjson +gzip	8	6	IO- 3813DD- 117991 A	646	656	651	651.019	0.006

Figure U.4

U.7. Conclusions

The provided results are specific to the Examind server application and Apache SIS library, in an unpolished state, with known ways for further improvements. The tests were made by a single client without parallel queries and all the applications running on a single machine. So it does not reflect a real scenario. But still, some global behavior can be extracted from those results.

- All DGGRS : performances are always better in time per zone when querying at greater sub-zone levels. Meaning it is more efficient to make less request with a greater zone-depth rather than a multitude requests with a small zone-depth.
- H3 : is the slowest DGGRS among those tested.
- S2 : is the fastest DGGRS, closely followed by Healpix.
- ISEA7H or IVEA7H : should be used instead of H3 if they are available.
- All excluding H3 : can provide the equivalent of a 256×256 tile (65000 zones) in less than 500ms.
- CBOR : is the most efficient encoding, but closely followed by UBJSON.

- GZIP : should be enable when possible, providing further compression at nearly no cost for the server.



ANNEX V (INFORMATIVE) PROTOTYPING AN EXTENSION FOR SIMPLER DATA ACCESS



ANNEX V

(INFORMATIVE)

PROTOTYPING AN EXTENSION FOR SIMPLER DATA ACCESS

Contributors: Johann Sorel (Geomatys), Nicolas Vila (CS Group), Jérôme Jacovella-St-Louis (Ecere), Francis Charette-Migneault (CRIM), Michael Jendryke (GeoInsight)

V.1. Why a new extension ?

The current API allows to return data using a request at location: `/collections/{collectionId}/dggs/{dggsId}/zones/{zoneId}/data`

While the data retrieval endpoint is useful for clients retrieving data values for tens of thousands of zones, for client wishing to retrieve values for only a few hundred zones for a specific area at a given resolution, this introduces unnecessary complexity. With the current API, the client needs to:

- find/compute the parent zones which will be needed to retrieve the data,
- ensure it complies with the maximum subzone depth allowed,
- make multiple requests to download the zones,
- remove all duplicated zones and zones outside the area of interest.

Of course, those steps can be simplified if an advanced DGGs client library exists.

V.2. Summary zone values extension

The proposed new extension enhances the Zone Query end-point (`/collections/{collectionId}/dggs/{dggsId}/zones`) with the ability to include one summary value for each zone returned. This provides a simple mechanism for a client to request data, including specifying an area of interest as geometry if the server also supports CQL2 spatial filtering

on zone queries. A conceptually equivalent capability was successfully implemented and demonstrated in Geomatys Examind server.

V.2.1. Examples



Figure V.1 — GET /collections/BlueMarble/dggs/H3/zones?zone-level=5&compact-zones=false&crs=[ogc:crs84]&f=geojson&filter-crs=[ogc:crs84]&filter=S_INTERSECTS(geometry,POLYGON%2044.604492%20-25.244696,%2045.483398%20-25.601902,%2047.241211%20-24.726875,%2049.87793%20-16.130262,%2050.493164%20-14.98724,%2048.955078%20-11.652236,%2047.724609%20-13.710035,%2046.40625%20-15.496032,%2044.165039%20-16.341226,%2043.725586%20-17.602139,%2044.472656%20-19.932041,%2042.978516%20-21.943046,%2043.59375%20-24.567108,%2044.604492%20-25.244696)



Figure V.2 — GET /collections/BlueMarble/dggs/Healpix/zones?zone-level=8&compact-zones=false&crs=[ogc:crs84]&f=geojson&filter-crs=[ogc:crs84]&filter=S_INTERSECTS(geometry,POLYGON 44.604492 -25.244696, 45.483398 -25.601902, 47.241211 -24.726875, 49.87793 -16.130262, 50.493164 -14.98724, 48.955078 -11.652236, 47.724609 -13.710035, 46.40625 -15.496032, 44.165039 -16.341226, 43.725586 -17.602139, 44.472656 -19.932041, 42.978516 -21.943046, 43.59375 -24.567108, 44.604492 -25.244696)



Figure V.3 — GET /collections/BlueMarble/dggs/ISEA3H/zones?zone-level=7&compact-zones=false&crs=[ogc:crs84]&f=geojson&filter-crs=[ogc:crs84]&filter=S_INTERSECTS(geometry,POLYGON 44.604492 -25.244696, 45.483398 -25.601902, 47.241211 -24.726875, 49.87793 -16.130262, 50.493164 -14.98724, 48.955078 -11.652236, 47.724609 -13.710035, 46.40625 -15.496032, 44.165039 -16.341226, 43.725586 -17.602139, 44.472656 -19.932041, 42.978516 -21.943046, 43.59375 -24.567108, 44.604492 -25.244696)

It is still possible to return the zone query results in formats that can readily be visualized such as png, geotiff, geojson or zarr. For the required JSON encoding of the zone query results, an additional values property is included in the response.

```
`json { "zones": [ "C5-F-D", "C5-F-C", "C5-10-D", "C5-F-B", "C5-6-D", "C5-10-B", "C5-10-C", "C5-7-D", "C5-11-D", "C5-11-B" ], "values": { "Band 0 Red": [ 38, 112, 91, 11, 15, 24, 76, 16, 6, 9 ] }, "links": [ { "rel": "https://www.opengis.net/def/rel/ogc/1.0/dggrs", "href": "/collections/BlueMarble/dggs/IVEA7H" }, { "rel": "https://www.opengis.net/def/rel/ogc/1.0/dggrs-definition", "href": "/dggrs/IVEA7H" }, { "rel": "https://www.opengis.net/def/rel/ogc/1.0/geodata", "href": "/collections/BlueMarble" } ], "linkTemplates": [ { "rel": "https://www.opengis.net/def/rel/ogc/1.0/dggrs-zone-data", "uriTemplate": "/collections/BlueMarble/dggs/IVEA7H/zones/{zoneld}/data" } ] }
```

V.3. Zone resolution extension

Another suggested extension is the introduction of a `zone-resolution` query parameter as an alternative to `zone-level` for clients which are not aware of the correspondence between the refinement levels of the DGGRS and the size of the zones. This would allow to instead specify the desired resolution in the output CRS (if one is selected), or in the native (`storageCrs`) of the collection otherwise. Additional discussion is needed on the name of this query parameter whether it should be specified in terms of area, square root of the area, or Characteristic Length Scale (CLS).

V.3.1. Example

In this following example, zones whose area smaller or equal to 0.01 x 0.01 degrees squared would be returned, corresponding approximately to `zone-level=9` for IVEA7H.

```
GET /collections/BlueMarble/dggs/IVEA7H/zones?crs=[ogc:crs84]&compact-zones=false&zone-resolution=0.01&f=geojson&bbox=10,20,30,40
```

These extensions could be further discussed within the DGGS Standard Working Group to see whether there is a need to support additional capabilities such as statistics (minimum, maximum, average, standard deviation) or additional dimensions beyond those of the DGGRS.



ANNEX W (INFORMATIVE) HIERARCHICAL PROGRESSIVE SURVEYS INTEROPERABILITY



ANNEX W

(INFORMATIVE)

HIERARCHICAL PROGRESSIVE SURVEYS

INTEROPERABILITY

Contributor: Johann Sorel (Geomatys)

W.1. What is HIPS ?

Hierarchical Progressive Surveys (HIPS) is a DeFacto web standard for sharing astronomic data made accessible using Healpix DGGRS.

The standard has been elaborated by the **International Virtual Observatory Alliance (IOVA)** to share data in a common structure. <https://www.ivoa.net/documents/HiPS/>

As of today thousands of catalogs made by space observation agencies exist and ExaBytes of data are already encoded this way.

W.2. Examind connector

To test the OGC DGGRS API and ensure it is possible to provide HIPS data with the new OGC API, a connector has been develop in the Examind server.

The resulting service of the experiment is available at : <https://server.examind.com/examind/WS/dggs/hips/>

W.3. Results

The HIPS standard is very close to a Cloud Native format, it is a hierarchical folder structure to access the DGGRS zones with a defined zone-depth.

On the other side the OGC DGGRS API support query parameters to select the zone-depth and output formats. The connector had to make a little processing to aggregate and sometime recompute some zones to match the client requests.

But once done, the service works and can provide the data from any existing HIPS service.

So this experiment is a success, OGC DGGRS API is a viable alternative to HIPS.

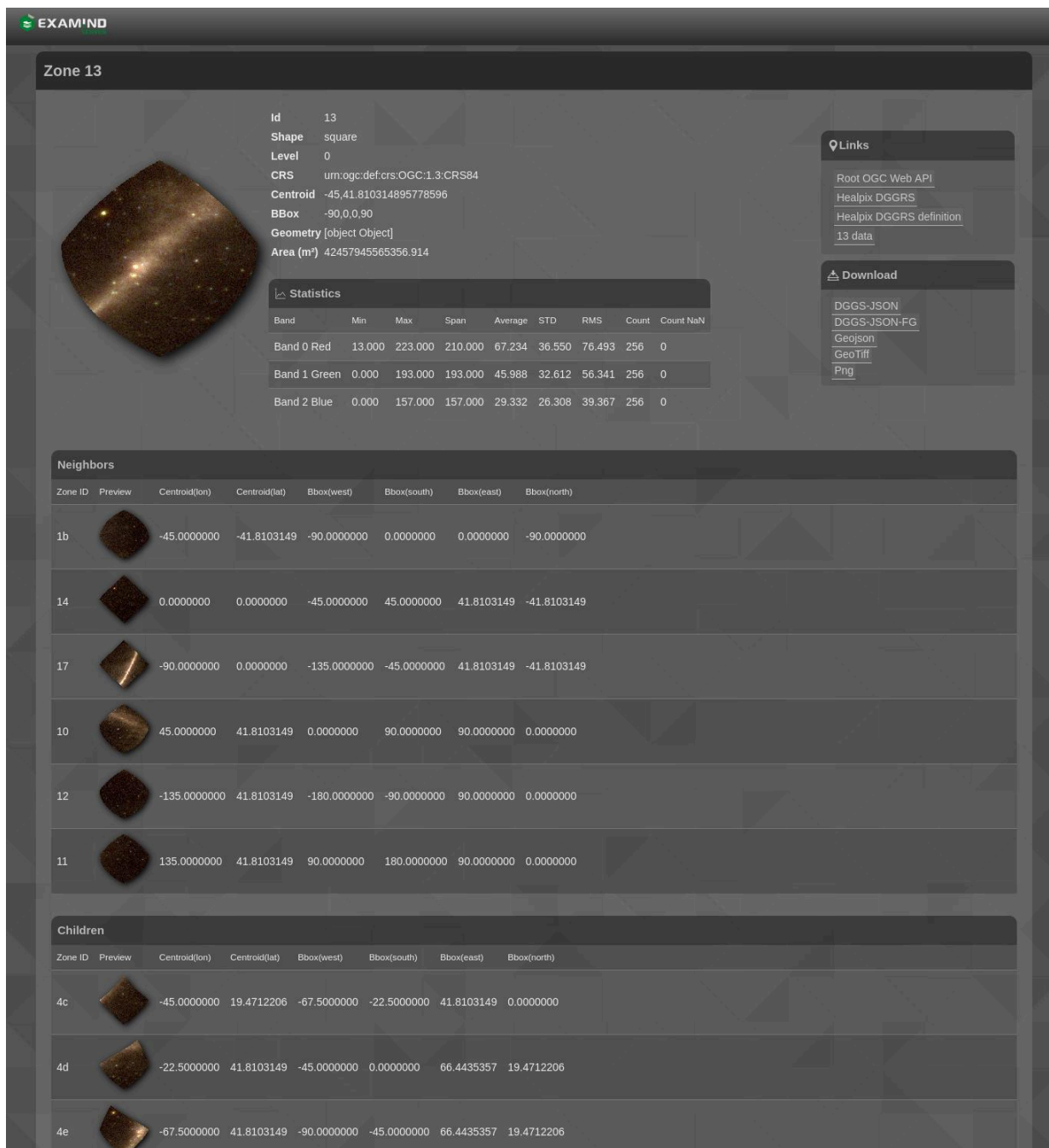


Figure W.1



ANNEX X (INFORMATIVE) COMMON OPERATING PICTURE (COP)



ANNEX X

(INFORMATIVE)

COMMON OPERATING PICTURE (COP)

Contributor: Lucio Colaiacono (4113 Engineering)

X.1. Context and Rationale

This Annex specifies an extended framework for Common Operating Picture (COP) implementations that incorporates Identity, Provenance, and Trust (IPT) verification mechanisms, agent-based orchestration capabilities, and Small Local Language Model (SLLM) integration. The specification builds upon the STAC (SpatioTemporal Asset Catalog) COP Extension v1.0.0 and OGC OWS Context (12-080r2) to enable secure, trusted information sharing and AI-enhanced decision support in emergency management and defense scenarios.

The COP framework defined herein serves dual purposes.

1. **Information Alignment:** Ensuring unified situational awareness across multiple actors with varying security clearances.
2. **Orchestration Platform:** Enabling automated coordination of geospatial simulations, data integration services, and trustworthy AI inference through Model Context Protocol (MCP) agents.

X.2. Keywords

Common operating picture, COP, STAC, OWS Context, identity provenance trust, IPT, emergency management, geospatial orchestration, MCP, SLLM, DGGS, security classification, smart certificates.

X.3. Background

Emergency management and defense operations require real-time coordination among multiple actors (first responders, command centers, supporting agencies, allied forces) operating under different security constraints. The Common Operating Picture (COP) has emerged as a critical capability for aligning situational awareness across these stakeholders.

Traditional COP implementations face several challenges.

- **Security fragmentation:** Data sharing is inhibited by inconsistent classification markings.
- **Provenance opacity:** Chain of custody information is often lost during data exchange.
- **Trust uncertainty:** Decision-makers lack quantitative metrics for assessing reliability.
- **AI integration gaps:** AI/ML capabilities cannot be safely incorporated without trust verification.
- **Service reproducibility:** Results cannot be validated due to missing processing parameters.

X.4. Evolution of COP Standards

The OGC OWS Context specification (12-080r2) established a foundation for encoding collections of geospatial resources. The STAC COP Extension v1.0.0 added mission-specific metadata including security classification and Discrete Global Grid System (DGGS) spatial references.

Recent developments in Identity, Provenance, and Trust (IPT) frameworks (OGC Testbed-19/20) and the Model Context Protocol (MCP) create opportunities to extend COP capabilities significantly.

X.4.1. Purpose and Objectives

This work extends the COP conceptual model to address identifying challenges by:

1. Integrating IPT smart certificates for verifiable data provenance;
2. Defining an orchestration framework for agent-based simulations;
3. Establishing trust-based decision thresholds for safety-critical applications;
4. Specifying requirements for trustworthy SLLM integration with geospatial constraints; and

5. Providing RDF/JSON-LD schemas for semantic interoperability.

X.4.2. Information Alignment Objectives

During emergency response, the COP enables the following.

- **Unified Situational Awareness:** A synchronized view for all actors involved.
- **Security-Controlled Dissemination:** Data tagging (Public, Internal, Confidential, Restricted, Classified).
- **Temporal Consistency:** Timestamped snapshots to ensure “when” information was valid.
- **Geospatial Contextualization:** DGGs identifiers provide unambiguous references across CRS systems.
- **Provenance Tracking:** Full chain of custody from acquisition to dissemination.

X.4.3. QGIS Plugin Implementation

To support the creation and exchange of these enhanced Common Operating Pictures within existing analyst workflows, a dedicated **QGIS Plugin** was developed as part of the pilot <https://github.com/luciocola/share-common-operating-picture>.

The plugin allows a user to take the current map view—including multiple data layers from various OGC API providers—and export it as a machine-readable COP package. The exporter enables the selection of target DGGs coordinate reference systems (CRSs) and the injection of IPT metadata, effectively bridging the gap between desktop GIS and the AI-ready ecosystem.

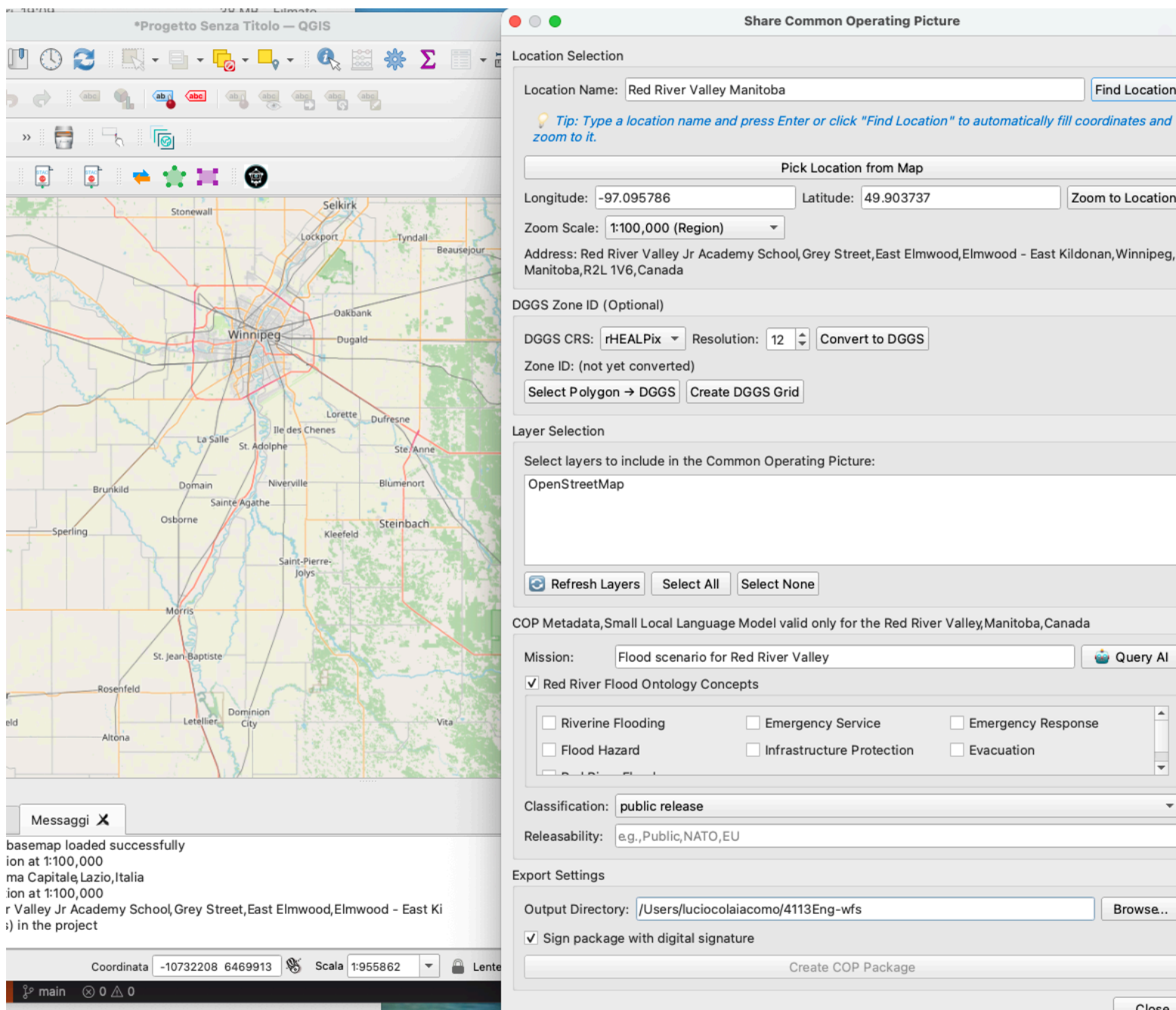


Figure X.1 – Shared Common Operating Picture Interface - Red River Pilot

X.5. COP as an Information Sharing Mechanism

X.5.1. Multi-Actor Information Distribution

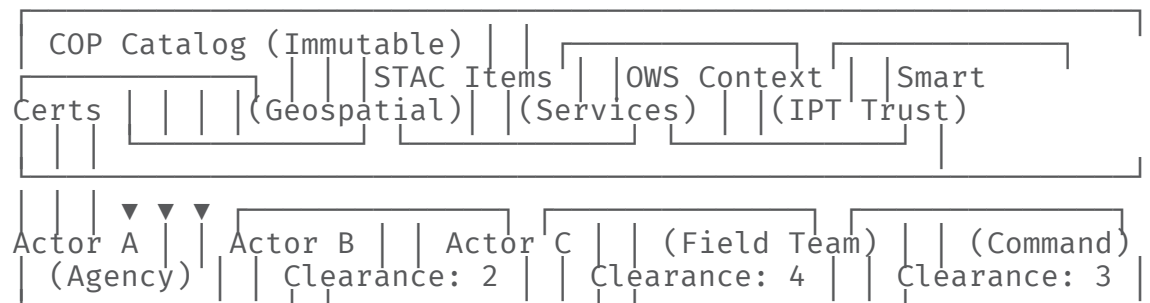


Figure X.2 – COP Immutable Catalog Distribution

X.5.2. Security-Aware Filtering

REQUIREMENT X.1

Each actor SHALL receive a filtered view of the COP based on: Security clearance level; Releasability constraints (e.g., “Public”, “Limited”, “National”); Need-to-know mission context; and Temporal validity windows.

X.5.3. Reproducibility Through Service References

The COP SHALL capture not just data snapshots, but executable service chains.

```
{
  "type": "Feature",
  "stac_version": "1.0.0",
  "stac_extensions": ["https://github.com/luciocola/common-operating-picture/
blob/main/json-schema/schema.json/"],
  "properties": {
    "cop:mission": "flood-response-2026-01",
    "cop:classification": "confidential",
    "cop:releasability": "national-emergency-services",
    "cop:dggs_crs": "rHEALPix-R12",
    "cop:dggs_zone_id": "N3456",
    "cop:service_chain": [
      {
        "service_type": "WPS",
        "endpoint": "https://geoprocessing.example.org/wps",
        "process_id": "flood-extent-model",
        "parameters": {
          "dem_source": "asset://dem_lidar_2025",
          "water_level": 3.5,

```

```

    "resolution": 5
  }
}
]
}
}

```

Listing X.1 — Example service chain in COP properties

NOTE: Any authorized actor can reproduce analytical results by re-executing the service chain.

X.6. COP as an Orchestrator Platform

X.6.1. Agent-Based Orchestration Architecture

The COP MAY be extended to an active orchestrator by integrating with MCP agents.

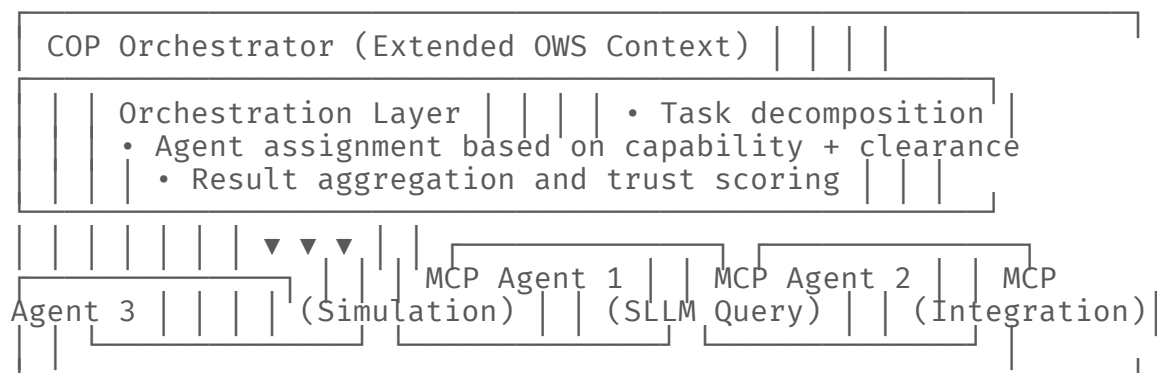


Figure X.3 — Agent-Based Orchestration Architecture

X.6.2. Use Cases for Orchestration (Informative)

X.6.2.1. Simulation Activation

Example: “Model flood extent for Red River at 4.2m water level.” The orchestrator identifies the flood modeling MCP, retrieves DEMs from STAC, verifies clearance, launches the agent with DGGs zone context, and publishes a new STAC item with provenance.

X.6.2.2. Collateral Information Enrichment

Example: “What critical infrastructure is affected in zone N3456?” The orchestrator queries an SLLM trained on infrastructure, retrieves OSM data, generates a summary, cross-validates with a registry, and returns an enriched COP layer.

X.6.2.3. Multi-Agent Coordination

Example: “Generate evacuation routes avoiding flooded areas.” The orchestrator chains Agent 1 (Flood Simulation), Agent 2 (Road Network/Traffic), Agent 3 (Shelter Capacity), and Agent 4 (Route Optimization) to produce a multi-source evacuation plan.

X.7. Identity, Provenance, and Trust (IPT) Integration

X.7.1. Smart Certificates for COP Components

REQUIREMENT X.2

STATEMENT Every component in the COP SHALL carry a smart certificate conforming to the OGC Testbed-19/20 IPT framework.

```
{
  "smart_certificate": {
    "cert_id": "urn:uuid:a1b2c3d4-e5f6-7890-abcd-ef1234567890",
    "issuer": {
      "organization": "National Geospatial Agency",
      "clearance_level": 4
    },
    "subject": {
      "asset_id": "flood-dem-lidar-2025-01-10",
      "asset_type": "raster"
    },
    "provenance_chain": [
      {
        "timestamp": "2025-01-10T08:30:00Z",
        "actor": "LiDAR Survey Team Alpha",
        "action": "acquisition"
      }
    ],
    "trust_score": 0.94
  }
}
```

}

Listing X.2 — Example smart certificate structure

X.7.2. Trust Scoring for Agent Results

REQUIREMENT X.3

STATEMENT The COP orchestrator SHALL calculate a composite trust score using the weighted factors defined in the methodology.

Table X.1 — Trust Score Weighting Factors

FACTOR	WEIGHT	SCORING CRITERIA
Input Data Trust	30%	Average trust score of all input assets
Agent Identity	20%	Verified certificate and developer
Process Transparency	20%	Documented algorithm and logged parameters
Validation Results	20%	Cross-validation with ground truth or peer review
Temporal Currency	10%	Recency of data vs. event timeline

Composite Trust Score Calculation Example: $(0.94 \times 0.3) + (0.90 \times 0.2) + (1.00 \times 0.2) + (0.50 \times 0.2) + (0.85 \times 0.1) = 0.847$

X.7.3. Trust-Based Decision Thresholds

REQUIREMENT X.4

STATEMENT The COP orchestrator SHALL enforce minimum trust levels: * Life-safety decisions: ≥ 0.90 * Resource allocation: ≥ 0.80 * Situational awareness: ≥ 0.70 * Exploratory analysis: ≥ 0.50

REQUIREMENT X.5

STATEMENT Results below the applicable threshold SHALL be flagged for human review before dissemination.

X.8. Small Local Language Models (SLLM) Integration

X.8.1. Constraints and Capabilities

SLLMs in the COP context are locally deployed (air-gapped), domain-specific, spatially indexed via DGGs embeddings, and temporally aware. List of documents used for trianing:

12-080r2_OWS_Context_Conceptual_Model.pdf 12-084r2_OWS_Context_Atom_Encoding.pdf
13_Risk-Reduction-Coping-and-Adaptation-to-Flood-Hazards-in-Manitoba-Canada-Evidence-from-Communities-in-the-Red-River-Valley.pdf
14-055r2_OGC_OWS_Context_GeoJSON_Encoding_Standard.pdf 1A-Red-River-Flood-Hazard-Mapping-Andrew-Weiss.pdf 2000_September_FridayForum.pdf Commissioned-Reports-2004-2005-Red_River_Floodway_Expansion.pdf cs_canada_sum.pdf csw_10entries.txt
Emergency Management Plans – additional guidance final.pdf Gippsland Regional Emergency Management Plan Earthquake Sub-Plan.pdf gml_road.txt gmlcov.txt GS-30.pdf meris_borders_users.txt meris.txt merisl2.txt MNRM Thesis Zaman, Jobaed.pdf OGC API – Discrete Global Grid Systems – Part 1: Core.pdf rrvdfa_boundary.pdf sea_ice_extent_01.txt Victorias Critical Infrastructure All Sectors Resilience Report 2024.pdf wcs_kml.txt wfs_100entries.txt wms_gml_hurricane_tomas.txt wms_meris.txt wms_scale.txt wmts.txt wps_52north.txt

X.8.2. Retrieval-Augmented Generation (RAG) Workflow

1. **Query Router:** Maps “Hospitals in N3456” to DGGs zone N3456 and security clearance.
2. **STAC Query:** Retrieves infrastructure assets for N3456 with Trust Score ≥ 0.75 .
3. **SLLM Inference:** Injects retrieved context (Hospital status) into the model.
4. **Output Generation:** SLLM generates summary and attaches provenance metadata.

X.8.3. SLLM Trust Requirements

SLLM outputs must include:

1. **Model Provenance:** Training sources, version, and validation metrics.
2. **Query Context:** DGGs zones queried, temporal range, and input classification.
3. **Confidence Metrics:** Per-fact scores and hallucination detection flags.

4. **Reproducibility:** Query hash, random seeds, and retrieval ranking methods.

X.9. Extended Schema Proposal

X.9.1. COP-RDF Schema (Turtle)

```
@prefix cop: <https://github.com/luciocola/common-operating-picture/> .
@prefix ipt: <https://testbed20.ogc.org/ipt/> .
@prefix prov: <http://www.w3.org/ns/prov#> .
@prefix dggs: <https://www.opengis.net/def/dggs/> .

cop:CommonOperatingPicture a rdfs:Class ;
  rdfs:label "Common Operating Picture" ;
  rdfs:comment "Extended OWS Context with IPT and ML metadata" .

cop:mission a rdf:Property ;
  rdfs:domain cop:CommonOperatingPicture ;
  rdfs:range xsd:string .

cop:dggsContext a rdf:Property ;
  rdfs:domain cop:CommonOperatingPicture ;
  rdfs:range dggs:ZoneIdentifier .

ipt:trustScore a rdf:Property ;
  rdfs:domain cop:Resource ;
  rdfs:range xsd:decimal .

cop:AgentCapability a rdfs:Class ;
  rdfs:label "Agent Capability" ;
  rdfs:comment "Describes an MCP server or SLLM endpoint" .
```

Listing X.3

X.9.2. Example COP Instance (JSON-LD)

```
{
  "@context": {
    "@vocab": "https://github.com/luciocola/common-operating-picture/blob/main/
context.jsonld/",
    "ipt": "https://testbed20.ogc.org/ipt/",
    "dggs": "https://www.opengis.net/def/dggs/"
  },
  "@type": "CommonOperatingPicture",
  "id": "cop-flood-red-river-2026",
  "dggsContext": {
    "@type": "dggs:ZoneSet",
    "crs": "rHEALPix-R12",
    "zones": ["N3456", "N3457"]
  },
  "resources": [
    {
```

```

    "id": "flood-extent-dem",
    "href": "https://stac.example.org/items/dem-2026",
    "trustScore": 0.94
  }
]
}

```

Listing X.4

X.10. Benefits and Expected Outcomes

- **For Emergency Responders:** Unified information view, trust indicators for prioritization, and natural language query access.
- **For Information Managers:** Automated security enforcement, complete audit trails via provenance, and efficient STAC-based dissemination.
- **For Decision Makers:** Rapid scenario exploration, risk-aware decisions with transparent scoring, and cross-agency collaboration via standard formats.
- **Further Developments:** Revision of OWS Context Standard, Geospatial Foundation Models based on SLLM, and Smart Contract integration via OpenEO.

X.11. Security and Ethical Considerations

- **Data Protection:** TLS 1.3/AES-256 encryption, granular access control, and audit logs for all agent invocations.
- **AI Transparency:** SLLM model cards, hallucination alerts, and clear labeling of AI-generated vs. observed data.
- **Privacy:** Anonymization of PII in shared COPs, differential privacy for statistics, and compliance with GDPR/PIPEDA.

X.12. Annex References

1. OGC OWS Context Conceptual Model (12-080r2)

2. STAC Specification v1.0.0 (<https://stacspec.org/>)
3. COP STAC Extension (<https://github.com/luciocola/common-operating-picture>)
4. OGC Testbed-19: Smart Certificates for Geospatial Data
5. Model Context Protocol Specification (Anthropic 2024)
6. DGGs Abstract Specification (OGC 15-104r5)



ANNEX Y (INFORMATIVE) FOUNDATION OF THE FLOOD IMPACT INDEX (FII)



ANNEX Y (INFORMATIVE) FOUNDATION OF THE FLOOD IMPACT INDEX (FII)

Contributor: Vassilis Andronis (Hartis)

Y.1. DGGs-Native Framing

The Flood Impact Index (FII) defines a formal, engineering-oriented methodology for combining dynamic flood hazard signals with static susceptibility and exposure information. The index is expressed as a bounded scalar in the range $[0, 1]$, computed per DGGs cell and time step.

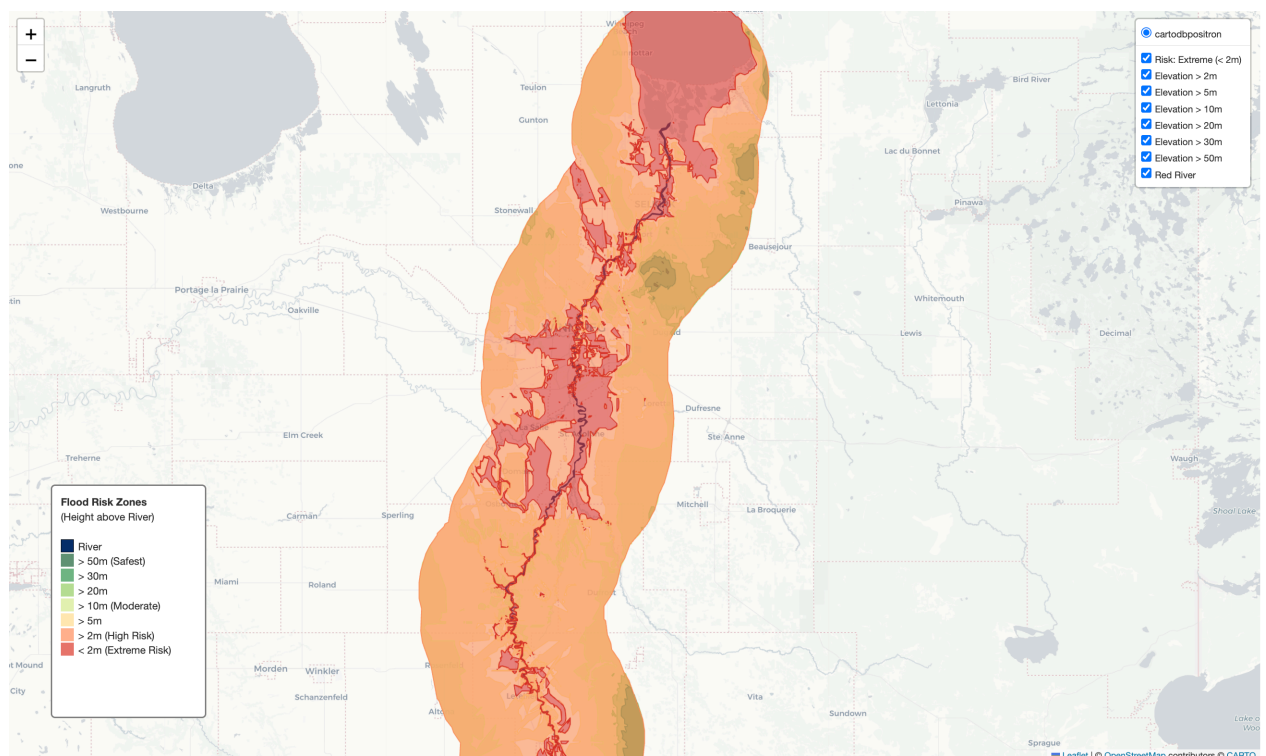


Figure Y.1 — Flood risk zones analysis (illustrative example)

Figure Figure Y.1 provides an illustrative example of flood risk zoning, where terrain-based susceptibility (height above river) is used to delineate extreme, high, and moderate risk corridors along a river network.

A key technical characteristic of the methodology is the adoption of a **DGGS-native framing**, in which a unique DGGS cell identifier is used as the sole spatial join key across heterogeneous data sources. Flood evidence derived from SAR, meteorological reanalysis variables, DEM-based topographic indices, and population datasets are aggregated into a unified per-cell, per-time-step feature space. This approach avoids coordinate-based spatial joins and supports consistent, auditable analytics. Resulting indicators are persisted in a relational data model suitable for database-backed processing and downstream decision support.

Y.2. Pixel to DGGS Transfer for Flood Evidence

Observational flood evidence derived from SAR backscatter is transferred to the DGGS grid by computing the per-cell flood fraction $f_{h,t}$. This operation summarizes high-resolution binary flood masks into continuous areal fractions associated with each DGGS cell.

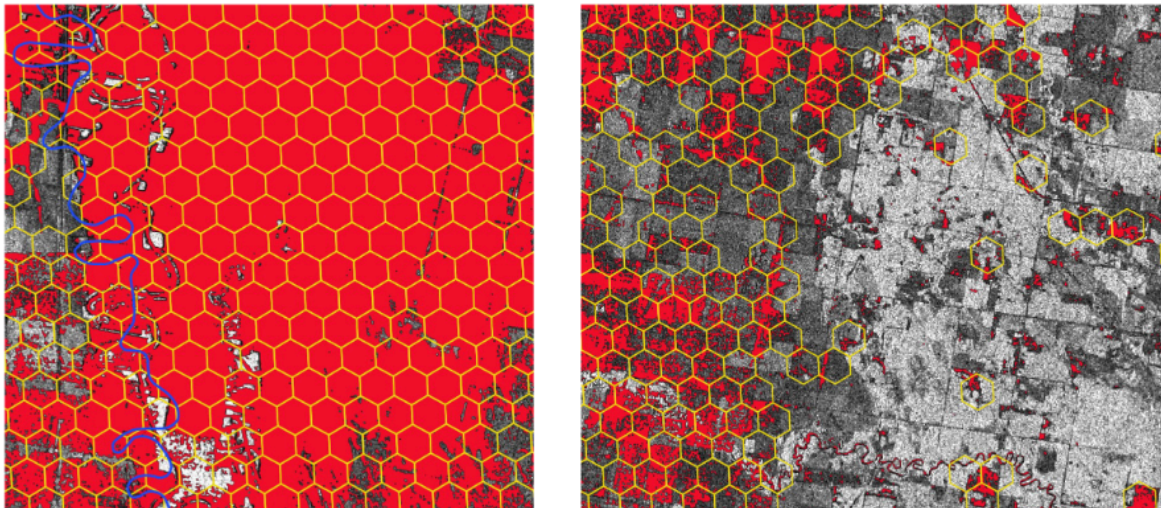


Figure Y.2 — Pixel-based flood mask to DGGS transfer (illustrative example). Flood pixels (red) are aggregated to DGGS cells (yellow outlines). Left: interior flood patches; right: flood fringe (mixed land and water).

Y.3. Data Classes and Indicators

The FII prototype combines multiple data classes to derive per-cell indicators. For each DGGS cell, a coverage statistic (valid fraction, $q_{h,t}$) is maintained to quantify usable data support and to reduce boundary effects near the edges of an Area of Interest (AOI).

Table Y.1 — Representative FII indicators within the DGGS feature space

COMPONENT	DATA CLASS	INDICATORS	QA AND NOTES
Hazard	Flood Evidence (SAR)	$f_{h,t}$; binary status	Threshold-based; gated by coverage $q_{h,t}$
Hazard	Meteorological / Cryosphere	Total precipitation; melt proxy; ROS tier	Unit checks and overlap-weighted aggregation
Vulnerability	Terrain / Hydrology	HAND; slope; TWI; D8 accumulation	Morphometric descriptors for susceptibility mapping
Exposure	Population	Population count; density	Standardized using GHSL (GHS-POP)
Exposure	Land Cover	Class fractions	Categorical vulnerability weighting

Y.4. Normalization and Synthesis Logic

The FII formulation is designed to remain stable under partial data availability and to be robust to statistical outliers commonly encountered in environmental datasets.

Y.4.1. Robust Quantile Normalization

Continuous environmental variables (e.g. precipitation) often exhibit heavy-tailed distributions. To address this, the FII applies **quantile normalization**, anchored at the 5th and 95th percentiles, to map variables into the $[0, 1]$ interval. This approach preserves sensitivity within the operational range of interest while limiting the influence of extreme outliers through clipping.

Y.4.2. Evidence Reinforcement (Noisy-OR)

Multiple hazard drivers are combined using a monotone product-of-complements operator (commonly referred to as **Noisy-OR**). This operator functions as a continuous logical OR: a single dominant signal (e.g. extreme rainfall or snowmelt) is sufficient to elevate the overall hazard score, while multiple moderate signals may reinforce one another. This behavior is better aligned with physical flood processes than simple additive or arithmetic averaging schemes.

Y.5. Hybrid Risk Products and AI-Enabled Serving

Derived FII metrics and their contributing indicators are stored as per-cell records indexed by a composite key consisting of **Cell ID, Run Date, Lead Day, and Valid Date**. This structure supports a range of automated decision-support workflows, including:

- **Spatial prioritization:** Ranking DGGs cells by relative risk magnitude.
- **Model interpretability:** Decomposition of index values into contributing hazard, vulnerability, and exposure components.
- **Infrastructure exposure analysis:** Intersection of high-risk DGGs cells with vector datasets (e.g. OpenStreetMap infrastructure).
- **GIS interoperability:** Export of filtered DGGs results in standard formats (e.g. GeoJSON, CSV) for use in external systems.

Y.6. Core Data Sources

The FII methodology integrates established scientific datasets to ensure that flood risk characterization is grounded in widely accepted reference data.

- **ERA5-Land:** Daily precipitation and snow-related variables used to characterize hydrometeorological forcing.
- **Height Above Nearest Drainage (HAND):** Terrain-derived indicator of relative inundation potential.
- **Topographic Wetness Index (TWI):** TOPMODEL-based metric representing geomorphological wetness propensity.

- **Global Human Settlement Layer (GHSL):** Versioned population datasets used to quantify human exposure.

Y.7. Operational Use in the AI-DGGS Client

The Flood Impact Index (FII) defined in this annex is implemented as a DGGS-native analytical product and is consumed by the Hartis AI-enabled DGGS Client as an input to higher-level decision-support workflows. In this context, FII outputs support automated risk prioritization, infrastructure impact assessment, and the generation of explainable operational summaries.

The methodological separation between FII computation and AI orchestration preserves transparency, reproducibility, and independent verification of the index, while enabling its integration into agentic geospatial reasoning systems.

- **Illustrative implementation:** <https://ai-dggs.hartis.org/>



INDEX



Index

..., Table E.1

44.604492 -25.244696, 45.483398 -25.601902, 47.241211 -24.726875, 49.87793
-16.130262, 50.493164 -14.98724, 48.955078 -11.652236, 47.724609 -13.710035,
46.40625 -15.496032, 44.165039 -16.341226, 43.725586 -17.602139, 44.472656
-19.932041, 42.978516 -21.943046, 43.59375 -24.567108, 44.604492 -25.244696,

Figure V.2, Figure V.3

44.604492%20-25.244696,%2045.483398%20-25.601902,%2047.241211%20-24.726875,
%2049.87793%20-16.130262,%2050.493164%20-14.98724,
%2048.955078%20-11.652236,%2047.724609%20-13.710035,
%2046.40625%20-15.496032,%2044.165039%20-16.341226,
%2043.725586%20-17.602139,%2044.472656%20-19.932041,
%2042.978516%20-21.943046,%2043.59375%20-24.567108,
%2044.604492%20-25.244696, Figure V.1