OGC® DOCUMENT: 25-044

External identifier of this OGC® document: http://www.opengis.net/doc/PER/OSPD-2025-BB

Open Geospatial Consortium

OPEN SCIENCE PERSISTENT DEMONSTRATOR (OSPD) 2025 BUILDING BLOCKS REPORT

ENGINEERING REPORT

PUBLISHED

Submission Date: 2026-02-12
Approval Date: 2026-03-04
Publication Date: 2026-08-12
Editor: Pedro Pereira Gonçalves, Rob Atkinson, Sina Taghavikish

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

The OSPD 2025 Building Blocks Report provides an overview of how specification artefacts can be used to support open and reproducible scientific workflows. Rather than focusing on individual platform implementations, the report abstracts common specification patterns observed across multiple platforms, capturing reusable concepts such as workflow intent, execution packaging, configuration parameters, and execution environments.

By applying a modular, schema-driven approach, the Building Blocks enable workflows to be described once and reused across multiple platforms. This approach supports transparency, improves comparability between workflows, and reduces the effort required to share scientific methods.

The report consolidates the current state of these Building Blocks as developed within OSPD 2025 and documents their scope, relationships, and validation status.

II EXECUTIVE SUMMARY

The **Open Science Persistent Demonstrator (OSPD) 2025 Building Blocks Report** documents the specification artefacts developed to support interoperable and reproducible scientific workflows. The report focuses on a set of **Open Science Building Blocks** that describe how workflows, their configuration, and their execution environments can be represented in a consistent and reusable manner.

Scientific workflows are increasingly executed across diverse platforms and infrastructures. However, differences in how workflows, parameters, and execution environments are described often limit reuse and reproducibility. OSPD 2025 addresses this challenge by identifying common specification patterns and expressing them as modular Building Blocks aligned with the OGC Specification Building Blocks framework.

The Building Blocks documented in this report are developed and validated within the OSPD 2025 initiative, primarily through WP4, and informed by workflow discovery, semantic modelling, and cross-platform reuse experiments. They are published in the OGC Incubator environment and are considered experimental. The results presented here provide a practical foundation for future OGC Best Practice documents or Standards supporting open science interoperability.

III KEYWORDS

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

Open Science, Building Blocks, Reproducibility, Interoperability, OGC API – Processes, Provenance, Metadata, FAIR Principles

IV CONTRIBUTORS

All questions regarding this document should be directed to the editor or the contributors:

NAME	ORGANIZATION	ROLE
Pedro Goncalves	Terradue Srl	Editor
Rob Atkinson	OGC	Editor
Sina Taghavikish	OGC	Editor
Alex Hayward	Telespazio VEGA UK Ltd	Contributor
Deyan Samardzhiev	Lampata	Contributor
Cameron Sajedi	Starling Foundries	Contributor
Gérald Fenoy	GeoLabs	Contributor
Nenad Radosevic	RMIT University	Contributor

V FUTURE OUTLOOK

The Open Science Building Blocks documented in this report will continue to evolve beyond the lifetime of OSPD 2025. Further refinement is expected as feedback from workflow reuse experiments and platform validation activities is consolidated and incorporated.

In the near term, the focus will be on strengthening semantic alignment, improving schema consistency, and expanding validation examples. In the longer term, selected Building Blocks may be transitioned into OGC Best Practice documents or considered as candidates for formal standardization.

The Building Blocks approach adopted in OSPD 2025 provides a sustainable pathway for evolving open science specifications while remaining responsive to community needs and technological change.

VI VALUE PROPOSITION

The OSPD 2025 Building Blocks offer clear benefits to the open science community,

- **Interoperability**: workflows can be described and reused across heterogeneous platforms using shared specification components.
- **Reproducibility**: configuration, execution context, and relationships are expressed in a transparent, traceable, and machine-readable way.
- **Efficiency**: modular Building Blocks reduce duplication and simplify the development of interoperable systems.
- **Scalability**: the approach supports a wide range of platforms, from research prototypes to operational services.
- **Sustainability**: registry-based Building Blocks provide a foundation for long-term maintenance and evolution.

By documenting and validating these Building Blocks, OSPD 2025 contributes to a more open, collaborative, and standards-based scientific ecosystem.

1 INTRODUCTION

1 INTRODUCTION

The **Open Science Persistent Demonstrator (OSPD) 2025 Building Blocks Report** documents the specification artefacts developed, refined or tested within the OSPD 2025 initiative to support interoperable and reproducible scientific workflows. This report focuses on the definition, structure, and validation of **Open Science Building Blocks** that can be reused across platforms, domains, and future OGC specifications. It also discusses the experiences and lessons learned in application of this powerful new approach to handling rich, extensible descriptions of complex capabilities.

The need for such Building Blocks arises from the increasing complexity and heterogeneity of scientific computing environments. While many platforms support workflow execution, differences in metadata models, interfaces, and execution semantics often limit the portability and reuse of scientific methods. There is however significant commonality of scientific methodology, data management and infrastructure provisioning for research and data supply chains. Whilst the complete description of science workflows will be unique for each circumstance, the opportunity exists for much of the required description to be supported by common, interoperable standards. The key challenge is a way to exploit this commonality without limiting the ability to adapt standards for specific circumstances. OSPD 2025 addresses this challenge by identifying common specification patterns and expressing them as modular, registry-based Building Blocks aligned with OGC standards and practices.

This report is produced as part of **WP4 OGC Workflow Building Blocks**, and builds on inputs from:

- workflow discovery and metadata analysis (WP1),
- semantic modelling and ontology alignment (WP2), and
- cross-platform reuse experiments (WP3).

The Building Blocks documented here are **experimental** and are maintained within the OGC Incubator environment. They are intended to serve as candidate inputs for future OGC Best Practice documents or Standards, rather than as normative specifications.

2 AIMS

2 AIMS

The primary aim of this report is to document and consolidate the Open Science Building Blocks developed during OSPD 2025. These Building Blocks provide a structured and reusable way to describe scientific workflows, their configuration, and their execution environments.

More specifically, this report aims to:

- capture common specification patterns emerging from real open-science workflows,
- describe these patterns using modular, schema-driven Building Blocks,
- demonstrate how the Building Blocks can be composed to support interoperable workflow execution,
- capture key experiences, insights and lessons learned by the participants, and
- provide a shared reference for further validation and refinement activities.

By doing so, the report supports the longer-term objective of improving interoperability and reproducibility in scientific research through open standards.

3 OBJECTIVES

3 OBJECTIVES

The objectives of this Building Blocks Report are to:

- document the scope, structure, and intent of each Open Science Building Block developed in OSPD 2025;
- describe the relationships between Building Blocks and how they can be composed;
- present the current maturity and validation status of each Building Block;
- show how the Building Blocks are applied and tested through reuse experiments;
- identify areas requiring further refinement or extension; and
- provide guidance for future standardization and adoption within the OGC framework.

The objectives are pursued in an incremental manner, recognizing that the Building Blocks will continue to evolve as additional workflows, platforms, and use cases are incorporated into the demonstrator.

4 TOPICS

4 TOPICS

This section presents the **Open Science Building Blocks** developed within the framework of the **Open Science Persistent Demonstrator (OSPD) 2025**. The Building Blocks documented here are intended to support the description, execution, and reuse of scientific workflows in an open, transparent, and interoperable manner.

The Building Blocks constitute the main technical output of **WP4 OGC Workflow Building Blocks**.

They are informed by workflow discovery activities (WP1), semantic modelling efforts (WP2), and validation through reuse experiments conducted across multiple platforms (WP3). At the current stage, the Building Blocks are considered **experimental** and are maintained within the OGC Incubator environment as candidate components for future OGC Best Practice or Standard specifications.

4.1. Summary

The Open Science domain requires highly detailed metadata for multiple aspects, and these in turn require common patterns to be extended with relevant details. This leads to **highly complex schemas**, irrespective of how these are defined and managed.

The use of Building Blocks to decompose this complexity and create **encapsulated, reusable solutions** that can be combined to meet the descriptive power was introduced to a set of participants and explored. Each participant was able to make some progress towards understanding and utilizing this approach, and some significant success was achieved in developing new Building Blocks for the emerging decision to support testing the WF4ever design to meet OSPD needs, integrating with the extensive libraries based on existing OGC standards.

In addition, the OSPD project supported deeper testing and refinement of the STAC Building Blocks in particular, and has already raised awareness of the potential for this approach to solve the same types of problems in other communities.

Finally, the use of Building Blocks to support semantically enriched descriptions of schemas has been shown to provide significant advantages in the use of AI approaches to extract and translate reports, code and other metadata formats into a common interoperable workflow description model.

These results are quite preliminary in nature, but provide a qualitative assessment that refactoring complex requirements for workflow and data descriptions into a set of simplified Building Blocks provides a powerful enabler to address the long term challenges around effective interoperability in scientific and other complex information processing contexts.

4.2. Building Blocks Principles

The Open Science Building Blocks developed in OSPD 2025 follow the principles defined by the OGC Specification Building Blocks framework. These principles aim to ensure that specification artefacts remain reusable, extensible, and applicable across different domains and implementation contexts.

In particular, the following principles are applied.

- **Normalisation of specifications**: Building Blocks provide a common way to integrate and describe relationships between different forms or components of specifications, managing dependencies in a normalised, technology-neutral way.
- **Specification-oriented design**: the Building Blocks describe interoperability requirements for APIs, data exchange models, schemas, and relationships between elements of these, rather than executable software implementations.
- **Modularity**: each Building Block addresses a clearly delimited concern that can be reused whenever that concern is relevant.
- **Composability**: Building Blocks can be combined to form more complex descriptions, such as comprehensive descriptions of data structures and semantics, and common aspects of workflow execution models.
- **Registry-based governance**: Building Blocks are published, versioned, and managed through shared registers.
- **Validation through practical usage**: refinement of Building Blocks is driven by their application in real workflow scenarios, through the use of examples and test cases.
- **Integrated testing and validation**: complexity is handled by CI/CT (continuous integration/continuous testing) of simple components and compositions of these to address more complex needs.

These principles allow the Building Blocks to evolve incrementally while remaining compatible with broader OGC standardization activities.

4.3. Registers and Identification

The Building Blocks described in this report are published through the OGC Incubator Building Blocks infrastructure. Each Building Block is intended to be uniquely identifiable within a register, allowing it to be referenced by specifications, documentation, and validation artefacts.

The use of registers enables:

- stable identification of Building Blocks over time,
- independent evolution and versioning, and
- traceability between Building Blocks and their usage examples.

At the time of writing, the identifiers and versions of the Building Blocks are provisional. They may be refined as the demonstrator progresses and as feedback is collected from implementation and validation activities.

4.4. Overview of Open Science Building Blocks

OSPD 2025 focuses on a core set of Building Blocks that are considered essential for representing scientific workflows and their execution environments in an open-science context. These Building Blocks reflect recurring specification patterns observed across different platforms and scientific domains.

The project team decide to base the OSPD solution around the OGC API Processes model, as the most viable and relevant option for describing interoperable processes. Scientific workflows can be described as chains of such processes.

The initial focus was describing workflows through use of standard metadata records for data inputs, outputs, the workflows themselves and "experiments" — or workflow execution records.

The pragmatic choice for a short term project was to leverage the ESA EarthCODE model, and the specific metadata profiles for "Product", "Workflow" and "Experiment."

By binding these to the OGC API Process model an abstract profile can be defined that specifies interoperability of all these descriptive sub-components of the OSPD.

This architecture is enabled by the OGC Building Blocks that uses a common packaging for available standards for each aspect, enabling the **building** of a comprehensive interoperability specification from standardized components.

Two key advantages accrue:

1. The resulting profile is itself a Building Block that can be used in combination with additional Building Blocks to specialize the solution for particular applications where additional aspects can be described in an interoperable fashion.
2. Schemas and Ontologies can be mapped to each other to increase the semantic expressivity of specifications — linking the earthCODE implementation to standardized ontology components.

At the highest level therefore the “Open Science Profile of OGC API Processes” consists of specifications for:

- **Workflows as Processes**;
- **Workflow description metadata records**;
- **Data Product**: metadata records based on a common set of STAC extensions;
- **Experiments**: metadata records for workflow executions based on the PROV ontology; and
- **OS Ontology**: a modular ontology using standard ontology components to describe the all the elements in the schemas of the descriptive records.

The generic concept of a “Process” is refined with the following Building Blocks to standardize how workflows are described:

- **ApplicationPackage**: describing how a workflow is technically implemented and made executable; and
- **Parameter**: describing configurable inputs and outputs used during execution.

Additional Building Blocks may be introduced to describe additional aspects as required, such as:

- **Application**: describing the logical intent and structure of a workflow; and
- **Platform**: describing the environment in which workflows are executed.

The following diagram (generated automatically from the OpenScience Profile of OGC API Processes shows some of the key relationships between such elements as well as the registers of reusable Building Blocks that are leveraged to create this model.

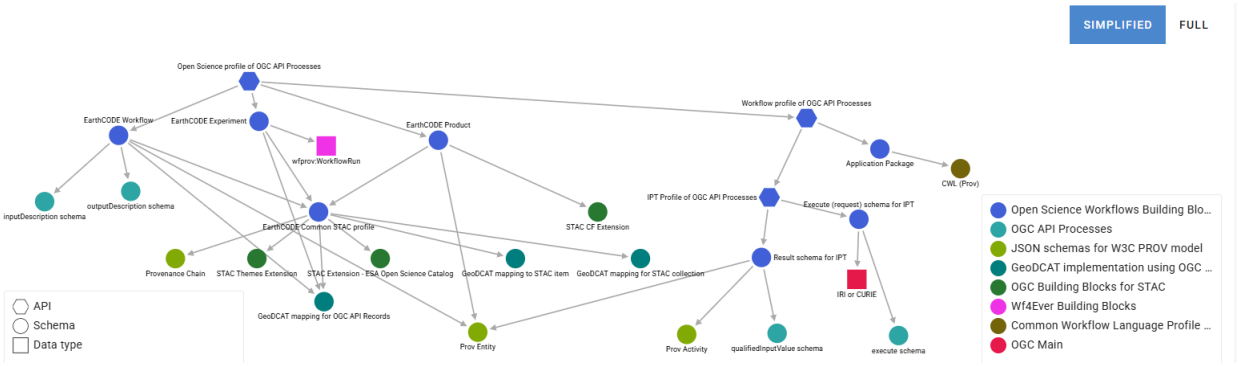

Figure 1

4.5. Building Block registers being re-used

- OGC Main: Reusable Sub-schemas for data exchange from published OGC specifications, and utilities and standardised patterns supporting these.
- OGC API Processes: schemas from the OGC API Processes Open API specification
- OGC Building Blocks for STAC: Building Blocks for STAC core and extensions
- Common Workflow Language Profile: schemas from the CWL specification
- WF4Ever: Wf4Ever ontologies suite (wfdesc, wfprov, ro, WF4Ever) enabling workflow description, provenance tracking, research object aggregation, and specialized artifact types.
- JSON schemas for the W3C PROV model
- GeoDCAT Building Blocks: schemas for Geo profile of DCAT, including mappings from OGC API Records and STAC to DCAT ontology
- Cross Domain Model: register of standard ontology Building Blocks published by OGC or defined by other standrads bodies and used to semantically describe OGC concepts

The full suite of components is thus extensive, and OSPD is able to create a comprehensive interoperable descriptive framework in a short project by reusing all these elements.

4.6. OSPD contribution and scope

The particular contribution of OSPD is thus to explore how these various existing standards can be effectively combined to describe the complexities of workflow descriptions. The degree of "granularity" of workflow description has been explored at different levels of detail:

- As a "black box" Process, with profiling of input and outputs and standardized metadata to describe it;
- Using CWL to describe in detail how the workflow operates (high level complexity);
- Using WF4Ever generalised model of workflow steps (intermediate level complexity);
- As a "ApplicationPackage" describing the execution environment; and
- Using standardised provenance models to describe workflow execution steps.

In addition to architectural exploration, OSPD participants conducted hands-on validation exercises using the Building Blocks to describe existing workflows, executions, and resulting

data products. These exercises demonstrated that while baseline provenance representations based on W3C PROV are useful, they are insufficient on their own to fully describe workflow inputs, outputs, and their relationships in a workflow-engine-agnostic manner. As a result, participants combined PROV with the WF4Ever family of ontologies (including WFDesc and WFProv), as well as OGC API – Processes 'inputDescription' and 'outputDescription' schemas, to achieve a more complete and navigable representation of workflows, executions, and products.

The OSPD phase 2 scope has shown that all these levels of description are possible using a common mechanism, however further work remains to explore the feasibility of a high level of semantic description for common concepts for highly detailed description of workflow elements.

Preliminary investigations of use of LLMs and AI to assist in generating highly detailed description suggests that the use of modular, semantically described Building Blocks can substantially increase the potential to use AI to generate candidate descriptions and support interpretation of legacy scientific workflows and outputs. Further work is required to establish the optimum level of detail in workflow description to be supported by standardized Building Blocks and the level of effort required to validate these models as detail increases.

4.7. Relationships Between Workflow Description Building Blocks

The Building Blocks are designed to interact through explicit relationships rather than through hierarchical containment. This approach supports flexible composition while preserving a clear separation of concerns.

Typical relationships include:

- an **Application** can be realised by one or more **ApplicationPackages**;
- an **ApplicationPackage** declares and consumes **Parameters**;
- an **ApplicationPackage** is executed on a **Platform**; and
- a **Platform** provides capabilities and constraints that influence execution behavior.

These relationships allow different Building Blocks to evolve independently while remaining interoperable.

4.8. Application Building Block

4.8.1. Scope and Purpose

The **Application** Building Block describes the logical purpose and structure of a scientific workflow. It represents what the workflow is intended to achieve, without making assumptions about execution technologies or deployment environments.

This Building Block is particularly important for workflow discovery and comparison, as it allows workflows with similar intent to be identified even when they are implemented differently.

4.8.2. Structure and Pattern

The Application Building Block follows a descriptive pattern that typically includes:

- identifiers and descriptive metadata,
- declared input and output dataset types, and
- a high-level description of processing steps.

Detailed execution logic is intentionally excluded and delegated to other Building Blocks.

4.8.3. Current Status

An initial version of the Application schema has been defined and populated using workflows identified under WP1. Further refinement is ongoing as additional workflows are documented and analyzed.

4.9. ApplicationPackage Building Block

4.9.1. Scope and Purpose

The **ApplicationPackage** Building Block describes how an Application is implemented and prepared for execution. It represents the technical realisation of a workflow, including references to executable artefacts.

This separation allows a single Application to have multiple implementation variants.

4.9.2. Structure and Pattern

The ApplicationPackage Building Block typically includes:

- references to container images or workflow definitions,
- runtime requirements and constraints, and
- a link to the corresponding Application.

This pattern supports portability across platforms with different execution environments.

4.9.3. Current Status

Prototype ApplicationPackage schemas have been tested using containerized workflows executed on participating OSPD platforms. Work is ongoing to further harmonize runtime-related metadata.

4.10. Parameter Building Block

4.10.1. Scope and Purpose

The **Parameter** Building Block defines the configurable inputs and outputs used during workflow execution. It supports reproducibility by making configuration choices explicit and machine-readable.

4.10.2. Structure and Pattern

The Parameter Building Block includes:

- parameter identifiers and descriptions,
- data types, units, and constraints, and
- optional semantic references to ontology terms.

Where domain-specific schemas are not yet available, auxiliary representations are used as an intermediate solution.

4.10.3. Current Status

Initial Parameter definitions have been derived from the workflows documented in WP1 and are being aligned with the Open Science Ontology developed in WP2.

4.11. Platform Building Block

4.11.1. Scope and Purpose

The **Platform** Building Block describes the computational environment in which workflows are executed. It provides contextual information required to understand execution conditions and limitations.

4.11.2. Structure and Pattern

The Platform Building Block captures:

- platform identifiers and metadata,
- supported interfaces and services, and
- execution and data access characteristics.

This information is essential for transparent and reproducible workflow execution across infrastructures.

4.11.3. Current Status

Platform descriptions have been created for the participating OSPD platforms. These descriptions will be further refined based on feedback from reuse experiments.

4.12. Profiles and Composition

Profiles represent constrained combinations of Building Blocks tailored to specific contexts or communities of practice. The OSPD scope is based on a profile supporting the EarthCODE environment. The Building Blocks presented here are designed to support future profiles using

the same pattern, or for extensions to support richer descriptions needed for sub-domains for the EarthCODE environment

4.13. Validation Through Use Cases

The Open Science Building Blocks are validated through their application in workflow reuse experiments conducted as part of OSPD 2025. These use cases provide evidence of:

- cross-platform applicability,
- adequacy of schema definitions, and
- support for provenance and reproducibility.

Validation activities also highlighted practical constraints in existing schema definitions. In several cases, schemas were found to be overly permissive, limiting their effectiveness as validation mechanisms. Participants therefore used the Building Blocks not only as descriptive components but as a syntactic and semantic “ground truth” layer against which generated or transformed metadata could be checked. This approach proved effective for detecting ambiguity and noise in workflow metadata and for identifying areas requiring tighter definitions.

Validation activities also included partial end-to-end demonstrations of workflow lifecycles, starting from exploratory notebook environments, through workflow formalization and deployment, and concluding with access to execution provenance via standardized APIs. These exercises highlighted both the feasibility of such lifecycles using existing Building Blocks and the remaining manual steps required for full automation.

The results of these validation activities will inform future revisions of the Building Blocks and guide their potential transition towards more formal OGC specifications.

4.14. Participant Contributions and Feedback

Participant contributions were delivered through concrete implementations and updates to Building Block repositories, including submitted and merged pull requests, example records, and tooling enhancements. These artefacts informed the validation findings summarized in the preceding sections and provide traceable evidence of participant engagement beyond conceptual design. The collected feedback will be used to inform refinements of the Building Blocks and to guide future standardization activities.

Participants from **Kurrawong AI**, **GeoLabs**, **Starling Foundries**, **Lampata**, **RMIT University**, and **Telespazio VEGA UK Ltd** contributed technical inputs to the OSPD 2025 Building Blocks Report through practical application, validation, and extension of the Open Science Building Blocks.

These contributions were provided through the use of GeoDCAT, STAC, EarthCODE, OGC API – Processes, and provenance-related Building Blocks to describe existing workflows, workflow executions, and resulting data products. The activities included the preparation of concrete example records and validation exercises used to assess the suitability and limitations of the current Building Block definitions.

A significant focus of participant contributions concerned provenance modeling. **GeoLabs** reported that while W3C PROV-based Building Blocks were useful, they were insufficient on their own to represent workflow inputs, outputs, and their relationships in a workflow-engine-agnostic manner. To address these limitations, PROV was combined with the WF4Ever family of ontologies, including WFDesc and WFProv, and supplemented with OGC API – Processes inputDescription and outputDescription schemas. This combined approach enabled more complete representations of workflows, workflow executions, and produced datasets, while preserving compatibility with clients supporting only PROV or PROV together with WF4Ever.

Participants also identified concrete interoperability and validation issues arising from the interaction of provenance Building Blocks with Records, STAC, and EarthCODE profiles. In particular, conflicts related to required identifier and type fields were observed, preventing simultaneous validation against multiple specifications. These findings highlighted the need for clearer guidance on integrating provenance-oriented Building Blocks with Records-based metadata models and for the possible use of alternative patterns or extensions to resolve these conflicts.

Kurrawong AI assessed the use of Building Blocks as a validation layer for both manually created and automatically generated metadata. In this work, the Building Blocks were treated as a "ground truth" against which workflow metadata could be checked. While the validation mechanisms were found to be effective for syntactic compliance, several schemas were reported to be overly permissive, limiting their effectiveness for automated discovery and reasoning. This feedback emphasized the need for clearer definitions and stricter constraints, particularly for AI-assisted metadata generation scenarios.

Starling Foundries contributed validation experience focused on workflow reuse and composability. Their work demonstrated that the adoption of standardized Building Blocks for common parameters, particularly bounding boxes, enabled workflows to be chained and reused across different spatial and temporal contexts without custom adapters. This validation showed that consistent use of simple, shared Building Blocks can significantly improve interoperability and reuse across workflows and platforms .

Additional contributions from **Lampata**, **RMIT University**, and **Telespazio VEGA UK Ltd** were provided through example workflows, validation feedback, and engagement with Building Block repositories. Across all participating organizations, contributions included submitted and merged pull requests, development of example workflows and metadata records, enhancements to visualization and discovery tooling (including ontology and provenance visualization), and partial end-to-end demonstrations of workflow lifecycles from notebook-based development through deployment via OGC API – Processes and access to execution provenance endpoints. These artefacts informed the validation findings summarized in this report and supported the identification of remaining gaps and future work .

5

BUILDING BLOCKS AND FAIR PRINCIPALS

5 BUILDING BLOCKS AND FAIR PRINCIPALS

During this phase of the OSPD project, extensive research and hands-on experimentation have been carried out on the functionality, maturity, and supporting tooling of the OGC Blocks. This work has gone beyond conceptual evaluation, focusing instead on the practical adoption of OGC Blocks in real project workflows and demonstrators.

The intensive use of OGC Blocks by OSPD participants, combined with the contributions submitted and the technical discussions held within the community, has directly driven the evolution of the framework. This process has resulted in the implementation of several new features, the resolution of multiple defects, and meaningful improvements to existing components. At the same time, it has provided valuable insights into best practices for using OGC Blocks effectively, revealed new and emerging use cases that require support, and highlighted key areas for future development.

The following sections explore the work and experimentation undertaken during the OSPD for each of the FAIR aspects (Findability, Accessibility, Interoperability, and Reusability).

5.1. Findability

OGC Blocks and related schema artifacts benefit from strong structural transparency, which improves their discoverability by both humans and machines. The use of modular JSON Schema constructs such as `allOf`, `oneOf`, and `anyOf` enables complex definitions to be decomposed into smaller, well-defined blocks that can be independently identified, referenced, and reasoned about. Within the OGC Blocks ecosystem, this “jigsaw puzzle” style of schema assembly is actively encouraged and supports the clear identification of individual components and their roles.

Even in the absence of full compositional automation, OGC Blocks implementations can already declare conformance to specific profiles. These declarations do not enforce composition at the API or schema level, but they provide valuable metadata that improves discoverability and interpretability. By explicitly signaling profile conformance, implementations enable both users and tooling to more easily identify relevant standards, profiles, and extensions, thereby improving semantic clarity across the ecosystem. OGC Blocks implementations can already declare that they implement specific profiles, and even in the absence of full composition support, these declarations provide valuable information to both human readers and automated tooling, by making profile conformance explicit and improving the discoverability, interpretability, and semantic clarity of the definitions.

As part of the OSPD work, significant effort has focused on demonstrating OGC Blocks not only as abstract building blocks, but as practical enablers of semantic interoperability in operational, end-to-end solutions. By integrating OGC Blocks into running services and user-facing applications, this work makes semantically enriched resources visible and discoverable in practice, rather than confining them to documentation or theoretical models. This approach

is illustrated by a running pygeoapi instance in which examples from multiple OGC Blocks repositories are compiled and re-published as part of a single deployment. By surfacing blocks authored and maintained in different registers through one operational API, the demonstration shows how distributed semantic assets can be discovered and explored coherently by both human users and client applications.

At the governance level, the OGC Blocks framework deliberately avoids imposing a centralized model for OGC Blocks registers. Any organization may create and maintain its own registers, allowing adoption to scale organically across different institutional and project contexts. Registers may be publicly accessible or privately maintained, and within a single organization, registers can also be subdivided or modularized to enable fine-grained control over ownership, review responsibilities, and release management. This decentralized approach supports independent publication while still enabling blocks to be discovered and referenced across organizational boundaries.

At the same time, the proliferation of independently governed registers introduces a clear risk of fragmentation. Without additional mechanisms, it may become increasingly difficult for users to discover existing blocks, assess their maturity, or determine which registers are authoritative or widely adopted. Improving findability is therefore a key governance challenge. OGC Blocks can be automatically published in OGC RAINBOW definition services, which provides a partial solution to discoverability. However, this approach requires the existence of a suitable service capable of accepting and publishing externally provided definitions (e.g., from community-led or project-specific registers) and proper interlinking between services to enable traversal and discovery across registers. In the absence of such links, standalone registers containing otherwise reusable blocks are likely to remain difficult to find.

In OSPD 2025, in parallel with specification work, contributors implemented tooling improvements to enhance the visualization and exploration of Building Blocks and associated ontologies. These efforts focused on improving ontology display and dependency visualization in user-facing interfaces, and on extending OGC RAINBOW capabilities to better surface semantic relationships and provenance information.

Addressing these limitations will likely require the development of a dedicated, register-agnostic findability mechanism. One possible source of inspiration is the software package ecosystem, where repositories aggregate contributions from multiple authors and organizations and provide rich search and filtering capabilities based on descriptive metadata. Comparable mechanisms could enable discovery of OGC Blocks based on textual search, block type, author or owning organization, maturity level, declared profiles, or dependencies. A potential technical and governance model is provided by services such as w3id.org, which issues persistent identifiers for the semantic web using a lightweight redirection mechanism, with definitions maintained in a public GitHub repository that accepts third-party pull requests. Each contributor controls its own namespace while adhering to shared governance rules. A similar approach for OGC Blocks could involve a curated index of metadata records pointing to distributed registers, combined with a web application that consumes this metadata, retrieves the referenced registers, and offers unified query and discovery capabilities across a federated OGC Blocks environment.

In addition to findability, there remains the unresolved issue of official adoption or endorsement of externally developed blocks. During OSPD, participants created individual registers containing blocks that may be of broader relevance and interest to the OGC community. At present, no formal mechanism exists to support the transition of such registers or individual blocks into an officially recognized or OGC-maintained status. In practice, the closest available option has

been to fork selected repositories under one of the official OGC GitHub organizations (such as `opengeospatial` or `ogcincubator`). Establishing a clearer pathway for official adoption or endorsement—analogous to processes used for OGC Community Standards—would improve trust, visibility, and findability of blocks that demonstrate value beyond their original project context.

5.2. Accessibility

JSON Schema's widespread adoption and mature tooling make schema definitions broadly accessible across platforms and communities. Its well-established composition mechanisms allow schemas to be shared and reused using standard tooling, without requiring specialized infrastructure. As a result, base definitions and extensions remain readable, processable, and usable by a wide range of existing validators, editors, and development environments.

An additional accessibility challenge emerged during OSPD in the context of Denied, Degraded, and Disrupted Space Operational Environments (D3SOE). OGC Blocks make extensive use of references, both internally and across registers, and in typical deployments these references—such as JSON Schema documents, JSON-LD contexts, and linked data URIs—are resolved dynamically at runtime. In D3SOE scenarios, network connectivity may be limited, intermittent, or entirely unavailable, making such runtime resolution unreliable or impossible. As a result, schemas, contexts, or linked definitions that are normally accessible through standard network mechanisms may become unreachable, directly affecting an application's ability to access the resources required for validation, interpretation, or semantic enrichment. Several mitigation strategies can be considered: eager or early resolution of references during build or deployment time, caching of schemas, contexts, and linked resources, and the use of edge-node or local storage to ensure that all required dependencies are available within the operational environment. More broadly, these considerations highlight the importance of deployment-aware tooling and guidance for OGC Blocks, particularly for use cases in which assumptions about continuous connectivity do not hold.

5.3. Interoperability

Interoperability within the OGC Blocks ecosystem is fundamentally supported by the modular design of JSON Schema, which promotes consistent reuse of shared components across domains. Composition mechanisms such as `allOf`, `oneOf`, and `anyOf` allow base schemas to expose extension points that can be specialized in a controlled and semantically consistent manner. In principle, dynamic references further extend this model by enabling late binding of constraints, allowing different communities to build interoperable profiles on top of common foundations without hard-coding domain-specific assumptions.

In practice, however, interoperability is constrained by uneven tool support and by limitations at the API layer. Dynamic references, while powerful, are not currently supported by OGC Blocks

tooling, nor by OpenAPI at the document level. These limitations complicate the extension of complex blocks such as the Provenance Schema, where repeated references to `Entity`, `Activity`, and `Agent` definitions and the intentional use of cyclical references to support directed graph–like structures make profiling difficult. As a result, extensions are often applied manually and inconsistently, reducing interoperability across profiles and implementations.

For example, interoperability challenges were observed at the schema level when combining provenance-related Building Blocks with Records, STAC, and EarthCODE profiles. In particular, conflicting requirements on type and identifier fields limited simultaneous validation against multiple specifications. These conflicts indicate a need for clearer guidance on the interaction between PROV-based Building Blocks and Records-oriented schemas, potentially through the exclusive use of 'prov:type' or dedicated provenance extensions.

To address some of these challenges, an extension mechanism for OGC Blocks was developed as part of the OSPD project. This mechanism allows extension points to be declared for a base block in a purely declarative manner. For JSON Schema–based blocks, the mechanism operates by compiling an annotated schema in which declared extension points are resolved and combined with their respective targets using `allOf`. As a result, the generated schema transparently incorporates the constraints of both the base and extension blocks. An important consequence of this approach is that the JSON-LD context generated from the compiled schema automatically includes the bindings contributed by the target blocks involved in the extension. Conceptually, this provides functionality similar to what could be achieved using dynamic references, but without requiring native tool support for them.

Beyond schema-level mechanisms, OGC Blocks act as practical enablers of semantic interoperability by relying on widely adopted standards such as RDF, JSON-LD, and SPARQL. In alignment with the core principles of the OGC RAINBOW methodologies, this semantic layer builds on established technologies and practices rather than introducing bespoke or isolated solutions, facilitating integration with existing tools, services, and data infrastructures. Concrete integrations developed within OSPD have also stimulated broader discussions within the `pygeoapi` community on how JSON-LD resources and semantics could be supported more generally. Although these discussions are ongoing, they illustrate how applied use of OGC Blocks can act as a catalyst for upstream improvements, contributing to improved interoperability across the open-source ecosystem.

A concrete outcome of the OSPD effort has been the development of enhancements to the pygeoapi server that improve support for JSON-LD-enabled resources. These enhancements include, for example, the semantic annotation of the properties table rendered in HTML pages for GeoJSON features and STAC items, allowing property semantics to be explicitly exposed to human users as well as to machine clients (Figure 2). By integrating semantics directly into familiar representations, this work makes semantic information easier to access and understand in operational services. These enhancements illustrate how existing service implementations can be augmented to expose semantic information derived from OGC Blocks without fundamentally altering their core architecture. By embedding semantics into user-facing interfaces that developers and users already interact with, the approach significantly lowers the barrier to accessing and interpreting semantic metadata.

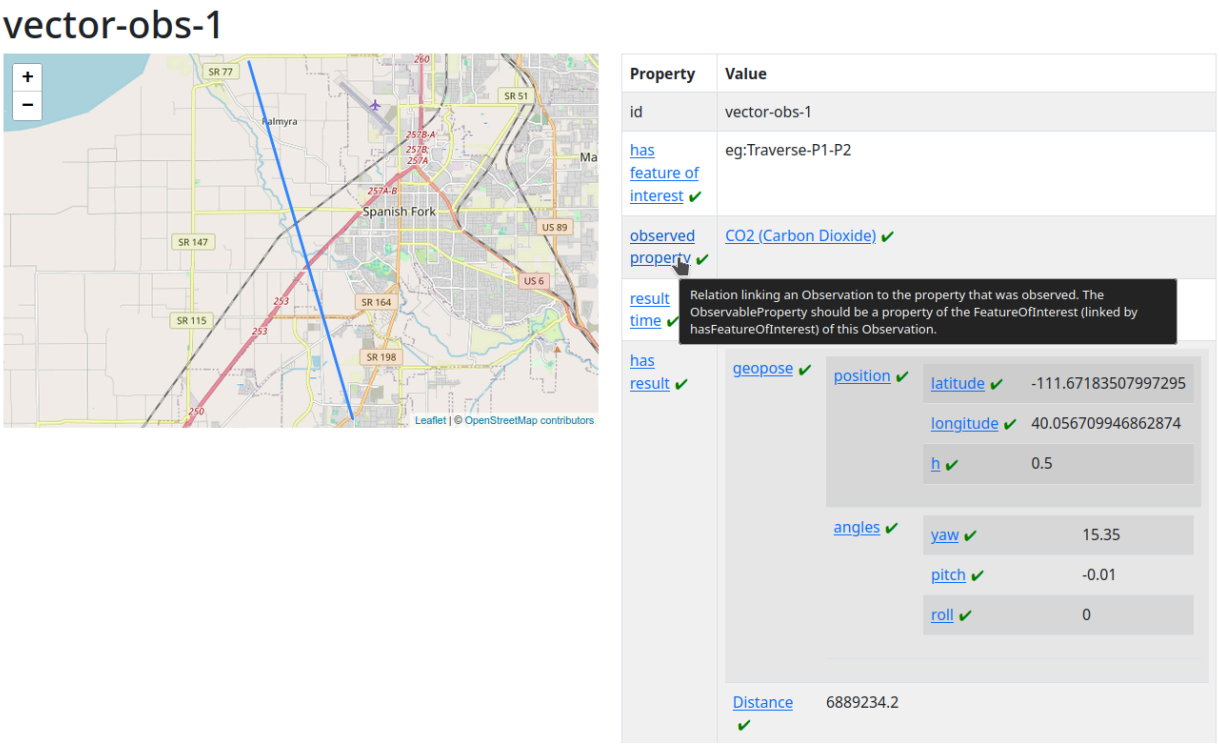

Figure 2 — Semantically enabled pygeoapi instance with property and object lookups

Interoperability is further supported at the governance level by the fact that the OGC Blocks framework does not impose a central authority over registers. Multiple governance regimes—ranging from simple, lightweight setups to more formal, multi-level workflows—can coexist while still producing blocks that are compatible and combinable within the broader ecosystem. This is illustrated by the OSPD project itself, where several partners have established and maintained their own registers alongside OGC-maintained blocks. Shared conventions and tooling enable blocks originating from different governance contexts to interoperate across organizational boundaries. In addition, the use of Git-based repositories as the primary technical foundation allows existing version control practices, access controls, and contribution workflows to be reused without requiring specialized infrastructure.

At the same time, inconsistent or underspecified versioning practices pose a significant risk to interoperability. OGC Blocks include a `version` metadata property, which allows individual blocks to declare their version identifier. However, at present there is no standardized mechanism for publishing, referencing, or discovering multiple versions of the same logical block within or across registers. This complicates evolution over time, particularly when downstream users or profiles depend on different versions, and creates uncertainty around compatibility, deprecation, and parallel maintenance of older versions. These observations highlight the need for an official specification of the OGC Blocks format itself—one that clearly defines the expected structure, metadata properties, and generated outputs of blocks, and explicitly addresses versioning concepts and workflows. Providing a standardized approach to version identification, compatibility signaling, and version coexistence would significantly improve interoperability across registers and implementations.

Finally, the reliance on external reference resolution introduces additional interoperability challenges in operational environments where robustness and autonomy are critical. OGC

Blocks frequently depend on externally resolved resources, including schemas, JSON-LD contexts, and linked data URIs. If these resources cannot be resolved consistently at runtime, validation, interpretation, or semantic enrichment processes may fail or behave unpredictably. While mitigation is discussed under Accessibility (D3SOE), from an interoperability standpoint, divergent runtime resolution behavior can undermine predictable and interoperable operation across platforms and deployments. Looking ahead, future versions of OpenAPI are expected to introduce improved support for modularity and composition, which may address some of these shortcomings; however, further research is necessary to better understand how extension and profiling mechanisms can be improved in practice, and how schema and API composition can be more effectively automated or supported within the OGC Blocks ecosystem.

5.4. Reusability

Reusability is a core strength of JSON Schema and a guiding principle of the OGC Blocks ecosystem. By enabling schemas to be composed from smaller, reusable building blocks, JSON Schema supports the creation of extensible base components that can be adapted to different domains and use cases.

In OSPD 2025, reusability benefits were particularly evident where standardized spatial parameters were adopted. The use of a shared bounding-box Building Block enabled workflows to be composed and chained without custom adapters, allowing the same workflow implementation to be reused across multiple geographic locations and temporal scenarios. This demonstrated that even simple, well-defined Building Blocks can significantly enhance composability and reuse when consistently applied.

Dynamic references in JSON Schema are also highly relevant from a reusability perspective. By allowing constraints to be bound late, they enable base schemas to be designed as genuinely reusable foundations rather than as artifacts tied to a specific profile or domain. A base block can expose well-defined semantic and structural “hooks” without anticipating how or where it will be reused, making it possible for downstream profiles to impose additional constraints while preserving a single, shared definition. This significantly increases the long-term value and applicability of base schemas, even though limited tooling support currently constrains practical adoption. The extension mechanism developed within the OSPD project enhances reusability by allowing base blocks to be extended without modification. Declared extension points are resolved automatically at compile time, producing schemas that transparently incorporate both base and extension constraints. An important consequence of this approach is that derived artifacts, such as JSON-LD contexts, also inherit the semantic bindings contributed by extension blocks, further increasing the value of reuse across applications and domains.

Despite these strengths at the schema level, reusability at the API level remains limited. OpenAPI does not support composition of complete API definitions, making it difficult to reuse or combine APIs and profiles without duplicating paths and manually integrating components. Two concrete examples illustrate the current limitations and challenges. In the first example (Figure 3), a custom OGC API – Processes instance defines its own schemas for process inputs and outputs. Integrating these schemas into the API responses is cumbersome and requires manual wiring into the relevant OpenAPI paths. Multiple distinct blocks must be defined to

represent a single process definition, and although extension points can be declared, there is no tooling support for automatically producing a bundled OpenAPI document that combines all required components.

OSPD 2025 Contributors also assessed the suitability of Building Blocks for automated metadata generation and reasoning, including AI-assisted approaches. These experiments indicated that while Building Blocks provide a strong structural foundation, insufficiently constrained schemas can introduce ambiguity and noise, reducing their effectiveness for automated interpretation. This reinforces the need for clearer definitions and stricter constraints in future iterations.

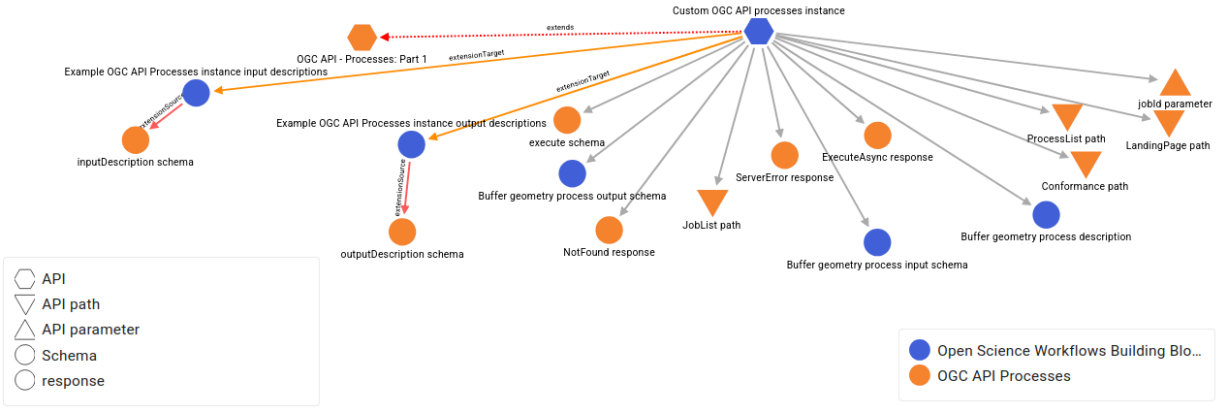

Figure 3 — Custom OGC API - Processes instance with inputs and outputs schemas

In the second example (Figure 4), an IPT-enabled profile of OGC API – Processes is defined. If this profile were to be combined with the custom process definition from the previous example, the integration would again have to be performed manually, highlighting the current lack of automation and limited reuse across profiles. While future versions of OpenAPI may address some of these shortcomings, additional research and tooling are required to fully realize reusable and composable APIs within the OGC Blocks ecosystem.

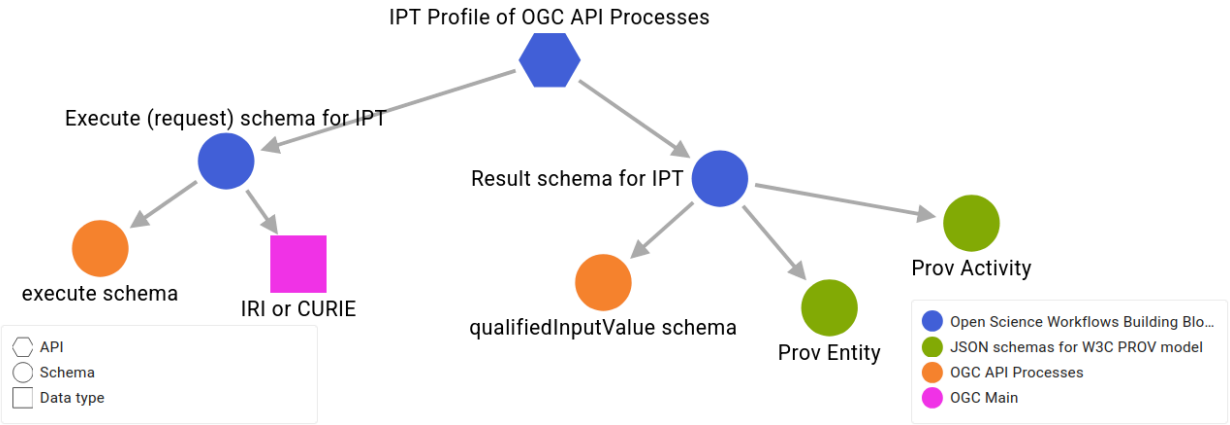

Figure 4 — IPT-enabled profile of OGC API - Processes

The proof-of-concept `pygeoapi` deployment developed during OSPD demonstrates how OGC Blocks originating from different repositories can nonetheless be composed, processed, and reused within a single operational service. This reuse extends beyond schemas themselves to

include JSON-LD contexts and semantic mappings, enabling consistent interpretation of data across multiple applications and deployments. In practice, integration effort can be minimal: in some cases it is sufficient to add mappings to relevant OGC Blocks, or even to reuse only their JSON-LD contexts, to make schemas, relationships, and metadata machine-interpretable. More advanced integrations can build on this foundation incrementally, demonstrating that meaningful reuse does not require large-scale reengineering of existing systems.

Governance flexibility further contributes to reusability. Registers can be subdivided or modularized within a single organization, enabling fine-grained control over ownership, review responsibilities, and release management. By building on Git-based repositories and established development practices, blocks can be versioned, reviewed, and reused using familiar workflows. At the same time, reusability is affected by the lack of a clear pathway for official adoption or endorsement of externally developed blocks. In practice, the closest available option has been to fork selected repositories under official OGC GitHub organizations such as `opengeospatial` or `ogcincubator`. Establishing a clearer adoption or endorsement pathway—analogous to processes used for OGC Community Standards—would improve trust, visibility, and long-term sustainability of blocks that prove broadly useful beyond their original project context.

Versioning emerged as a recurring concern during OSPD, reflecting its importance for reuse and long-term maintenance. While OGC Blocks include a `version` metadata property, there is currently no standardized mechanism for publishing, referencing, or discovering multiple versions of the same logical block within or across registers. This complicates evolution over time, particularly when downstream users or profiles depend on different versions, and creates uncertainty around compatibility, deprecation, and parallel maintenance of older versions. These challenges highlight the need for an official OGC Blocks format specification that explicitly defines versioning concepts and workflows (including compatibility signaling and version coexistence), as well as expected structures, metadata properties, and generated outputs.

Finally, reuse in operational environments must also account for deployment constraints such as Denied, Degraded, and Disrupted Space Operational Environments (D3SOE). Several mitigation strategies can support reuse in such contexts, including eager resolution of references at build or deployment time, caching of schemas, JSON-LD contexts, and linked resources, and the use of edge-node or local storage to ensure required dependencies remain available. More broadly, these considerations underscore the importance of deployment-aware guidance and tooling. When assumptions about continuous connectivity do not hold, reusable blocks must be supported by practices that ensure they remain reliable and dependable across a wide range of operational scenarios.

6 OUTLOOK

6 OUTLOOK

The Open Science Building Blocks documented in this report represent an initial, experimental step toward a more interoperable and reproducible open-science ecosystem. This experiment addressed multiple aspects of the underlying OGC Blocks methodology, as well as the applicability of a range of resources developed using that methodology — in short the extent to which they realize the FAIR principles.

As the OSPD 2025 initiative progresses, these Building Blocks will continue to evolve based on feedback from implementation and validation activities.

The findings and recommendations can be organized under the individual FAIR principles, however the key outcomes are best assessed in "reverse order" — i.e., OSPD has explicitly focussed on the hardest part of this paradigm — **Reusability**.

The body of this report has shown that the **Reuse** Building Blocks has provided a highly efficient means to rapidly develop rich models required to describe significant aspects of open science, and reusable workflows. This has led to improvements in the **Interoperability** of such workflows, however in a short project only a few examples have been addressed, and significant further benefit can be gained by extending this work to many more workflows across different science domains, and identification of common patterns that can be further standardized, to support both **Interoperability** and **Reusability**.

In the short term, further work will focus on refining the Building Block schemas and their relationships, informed by the results of cross-platform workflow reuse experiments. This includes improving the consistency of metadata definitions, strengthening semantic alignment with the Open Science Ontology, and extending validation examples derived from real workflow executions.

In parallel, the use of registers and versioned identifiers will be further consolidated to support stable referencing and long-term maintenance of the Building Blocks. As maturity increases, selected Building Blocks may be profiled for specific application contexts, enabling more constrained and targeted usage without modifying the underlying specification components.

With the applicability of the approach established and refined, the potential value of this approach can then be characterized by the other FAIR principles: **Findability** and **Accessibility**.

Accessibility of standards is improved by the OGC Blocks standardized way of bundling and integrating a range of resources and examples needed to have semantically rich versions of schemas and APIs. Dependency management is well-supported, however further work on versioning has been identified as necessary to operate at scale over long time frames where implementations use and drive evolution of open science metadata and code.

Findability has been marginally enhanced through capture in a shared knowledge base (OGC Rainbow) and improved navigation in the User Interface — however significant improvements can and will need to be made as increased volumes of content emerge.

Such improvements should include the following.

- Establishment of governance linking tested Building Blocks to specific standards development processes and communications with OGC.
- Patterns and tools for infrastructure providers to specify interoperability requirements for hosted applications and data.
- Integration of capabilities into common shared software tools and libraries relevant to open science and geoinformatics.
- Creating of systematic links from activities and discussions **to the specific Blocks and case studies that can help implement solutions**.
- Integration and federation of catalogs of resources between OGC and activities such as EarthCODE and various Data Spaces.
- Use of AI (as demonstrated in OSPD, but also in other modes to harvest and organize catalogued content) to enrich available metadata with the Open Science model.
- Policy frameworks to support and require interoperable **provenance** so that data and workflows can seamless integrate into richer metadata required to improve the transparency and repeatability of science.
- AI exploitation of provenance and workflow process descriptions, combined with metadata catalogues, to suggest **Reuse** opportunities for data and workflows.

In the medium term, the outcomes of OSPD 2025 are expected to inform the development of OGC Best Practice documents and, where appropriate, candidate Standards. The Building Blocks approach adopted in this report provides a flexible pathway for transitioning experimental artefacts into more formal specifications, while preserving compatibility with existing OGC standards.

More broadly, the Building Blocks described here are intended to support continued collaboration between research institutions, platform operators, and standards developers. By capturing common specification patterns derived from practical open-science workflows, OSPD 2025 contributes to a shared foundation that can be extended and reused in future initiatives addressing environmental monitoring, climate resilience, and other data-intensive scientific domains.

7

SECURITY, PRIVACY AND ETHICAL CONSIDERATIONS

7 SECURITY, PRIVACY AND ETHICAL CONSIDERATIONS

An extensive assessment was carried out to detect possible issues related to security, privacy, and ethics. Upon thorough review, it was concluded that these aspects were not pertinent to the report's scope and subject matter. Consequently, no particular actions or safeguards were deemed necessary.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] OGC: OGC 25-043: Open Science Persistent Demonstrator (OSPD) 2025 Report, 2026

A

ANNEX A (NORMATIVE) ABBREVIATIONS/ACRONYMS

A ANNEX A (NORMATIVE) ABBREVIATIONS/ACRONYMS

CI/CT	continuous integration/continuous testing
CSV	Simple Comma Separated Values
CVI	Coastal Vulnerability Index
CWL	Common Workflow Language
D3SOE	Denied, Degraded, and Disrupted Space Operational Environments
FAIR	Findable, Accessibile, Interoperabile and Reusabile
GISTAM	Geographical Information Systems Theory, Applications and Management
OGC	Open Geospatial Consortium
OSPD	Open Science Persistent Demonstrator

B

ANNEX B (NORMATIVE) LAMPATA CONTRIBUTIONS

B ANNEX B (NORMATIVE) LAMPATA CONTRIBUTIONS

B.1. Introduction

Reproducing geospatial workflows across different platforms has the potential to increase the adoption, and FAIRness — Findability, Accessibility, Interoperability and Reproducibility — of scientific results, aiding in the generation and dissemination of knowledge.

However, reproducing a scientific workflow is currently a challenging and cumbersome process. First, doing this requires the creation of additional data about the workflow — metadata — about its execution environment, input output data, for example. Second, there is no standard definition or ontology of what is required.

In this report we investigate the use of Agentic workflows to automate the creation of workflow metadata and extraction of workflow execution logic, in order to help address the above challenges and contribute to the reproducibility work. We carry out three experiments designed to show how Agentic workflows can help users generate progressively more descriptive workflow metadata for six diverse real-world workflows.

The metadata definitions that we target are the OSPD OGC Building blocks for:

- Workflow Descriptions (metadata); and
- Data products (as inputs and outputs).

Additionally, we are using the CWL specification as a stand in for the Workflow contents BB, assuming that both will be at least partially compliant with each other. Lastly, we directly use the more restrictive Open Science Catalog validator, which also checks for relationships and existing criteria.

The outcomes show promising results that by using lightweight LLMs (gemma3/gemini-flash) with robust verification loops and static code analysis, it is possible to automate the generation of high-quality, interoperable workflow metadata with minimal human intervention.

B.1.1. Data

We use the below six workflows, or a subset of them, in the experiments:

Coastal Vulnerability Index (CVI) (contributed by Hartis)	A reproducible workflow that calculates transect-based vulnerability values using the USGS method by integrating open datasets to analyze land cover, slope, erosion, and elevation. CWL Application package
ML4Floods (contributed by EarthCode / Terradue Srl)	An end-to-end machine learning pipeline for estimating flood extent using pre-trained models on optical satellite data from Sentinel-2 or Landsat-8/9.CWL Application package
Mangrove Detection and Biomass Estimation (contributed by Starling Foundries)	A baseline workflow that utilizes Sentinel-2 imagery and vegetation indices to detect mangrove extent and estimate above-ground biomass and carbon stocks using IPCC Tier 2 methodology. CWL Application package
POLARIS Algorithm (contributed by EOX)	A risk assessment methodology that uses WMO sea ice charts to calculate the operational limits and safety risks for ships based on their assigned ice class. CWL Application package
Polar Warp Algorithm (contributed by EOX)	A processing algorithm that aligns and warps polar satellite SAR images by applying drift vectors from wind, tide, and ice models to correct for motion over time. Argo workflow
Water bodies detection (contributed by Telespazio UK)	A Common Workflow Language (CWL) application that identifies water bodies in satellite imagery using NDWI and Otsu thresholds, designed to process multiple STAC items in parallel. CWL Application package

B.2. Methodology

The general approach for this analysis is to establish an iterative, self-correcting AI process designed to produce strictly formatted workflow metadata, validated against a target OGC Building Blocks profile. The inputs to this process are: the target schema, an annotated description of that schema to guide the LLM, the workflow/product context (including source code, product metadata, and unstructured documentation such as READMEs or academic papers).

The methodology evolved in three stages, growing in sophistication:

Schema-Driven Agentic Loop	The first experiment established a straightforward agentic loop. We used annotated schemas and repository context to generate valid metadata, validated against the OGC Building Blocks. This was later augmented by using the AI to first generate the annotated schema itself, optimizing it for the specific project context.
CWL Generation	The second experiment focused on generating executable Common Workflow Language (CWL) packages directly from the context. This introduced a stricter validation loop, using the cwltool compiler to check for logical consistency and execution errors.
Bottom-Up Code Analysis	The third experiment addressed the limitations of pure LLM reasoning. We combined static code analysis (parsing Abstract Syntax Trees) with generative AI to create a detailed dependency graph of the workflow. This graph—capturing steps, inputs, and outputs—served as the ground truth to guide the metadata generation.

For all methods, we intentionally used the simplest, lightweight LLMs available (e.g., Gemini 2.5 Flash) to ensure that the methodology relies on architectural robustness rather than the raw intelligence of the model. This strategy ensures the process is scalable and allows for significant performance gains when upgraded to state-of-the-art reasoning models (such as GPT-5, Claude, Opus, or Gemini 3 Pro).

B.2.1. Experiment 1: Metadata Schema-Driven Creation

The first experiment involved generating metadata against the building blocks profile using two approaches.

First, by using the examples for EarthCODE Products and Workflows building blocks profiles, we task the LLM to generate a generic marked up example of the profile with annotations. We used the following examples submitted as part of the OSPD as inputs: Coastal Vulnerability Index (CVI), POLARIS Algorithm, and Polar Warp Algorithm. This annotated target metadata was shared across all process runs and passed to the LLM alongside the context. The key idea was to build a more controlled one-shot prompt for the LLM to be guided by when generating metadata. For example, the resulting annotated JSON metadata was generated for EarthCODE Workflows:

```
{
  "id": "Extract unique identifier (e.g., 'worldcereal-workflow2')",
  "type": "Feature",
  "geometry": null,
  "conformsTo": [
    "http://www.opengis.net/spec/ogcapi-records-1/1.0/req/record-core",
    "Include other conformance URIs found (e.g., stac-extensions, metadata-
profiles)"
  ],
  "properties": {
    "title": "Extract human-readable title",
    "description": "Extract full text description",
    "osc:type": "workflow",
    "osc:project": "Extract project slug (e.g. 'worldcereal')",
    "osc:status": "Extract status if available (e.g. 'completed'), default to
'completed'",
```

```
    "application:type": "Extract application type (e.g. 'argo-workflow') based
on information and context provided.",
    "application:container": "Extract boolean (true/false) based on information
and context provided.",
    "application:language": "Extract programming language (e.g. 'Python')",
    "updated": "Generate current UTC timestamp YYYY-MM-DDTHH:MM:SSZ",
    "created": "Generate current UTC timestamp YYYY-MM-DDTHH:MM:SSZ",
    "keywords": [
      "Extract string. Max 5 keywords."
    ],
    "license": "Extract license or 'proprietary'",
    "version": "1",
    "themes": [
      {
        "scheme": "https://github.com/stac-extensions/osc#theme",
        "concepts": [
          {
            "id": "Extract one: atmosphere, cryosphere, land, magnetosphere-
ionosphere, oceans, solid-earth."
          }
        ]
      }
    ],
    "contacts": [
      {
        "name": "Extract contact name",
        "roles": [
          "Extract role (e.g. 'consortium_member')"
        ],
        "emails": [
          {
            "value": "Extract email address"
          }
        ]
      }
    ],
    "extent": "Extract spatial/temporal object if present, else null"
  },
  "linkTemplates": [],
  "links": [
    {
      "rel": "root",
      "href": "../../catalog.json",
      "type": "application/json",
      "title": "Open Science Catalog"
    },
    {
      "rel": "parent",
      "href": "../catalog.json",
      "type": "application/json",
      "title": "Workflows"
    },
    {
      "rel": "self",
      "href": "Construct: .../workflows/{id}/record.json",
      "type": "application/json"
    },
    {
      "rel": "related",
      "href": "Construct: ../../projects/{osc:project}/collection.json",
      "type": "application/json",
      "title": "Construct: Project: {project_title}"
    },
```

```
    {
      "rel": "related",
      "href": "Construct: ../../themes/{theme_id}/catalog.json (Create one
object per theme found)",
      "type": "application/json",
      "title": "Construct: Theme: {Theme_ID_Capitalized}"
    },
    {
      "rel": "vcs",
      "title": "Git source repository",
      "href": "Extract git URL from the provided information",
      "vcs:type": "git",
      "vcs:branch": "Extract branch name (e.g. 'main')"
    },
    {
      "rel": "application",
      "title": "Extract workflow title",
      "href": "Extract URL to specific workflow file (e.g. .yml)",
      "type": "Extract MIME type (e.g. 'application/x-argo-workflow-yaml')",
      "application:type": "Extract application type",
      "application:container": "Extract boolean",
      "application:language": "Extract language",
      "argo-workflow:": "Extract specific requirements object if present"
    }
  ]
}
```

Listing B.1

With this metadata we then build a one-shot prompt that contains all the context (source code, paper, and extracted metadata for the data outputs) with the task to create valid metadata. Then a validation loop starts:

1) Metadata gets created, 2) The metadata is validated against the relevant Building Block,and 3) Errors and past attempts are fed to the initial step, until the result is valid.

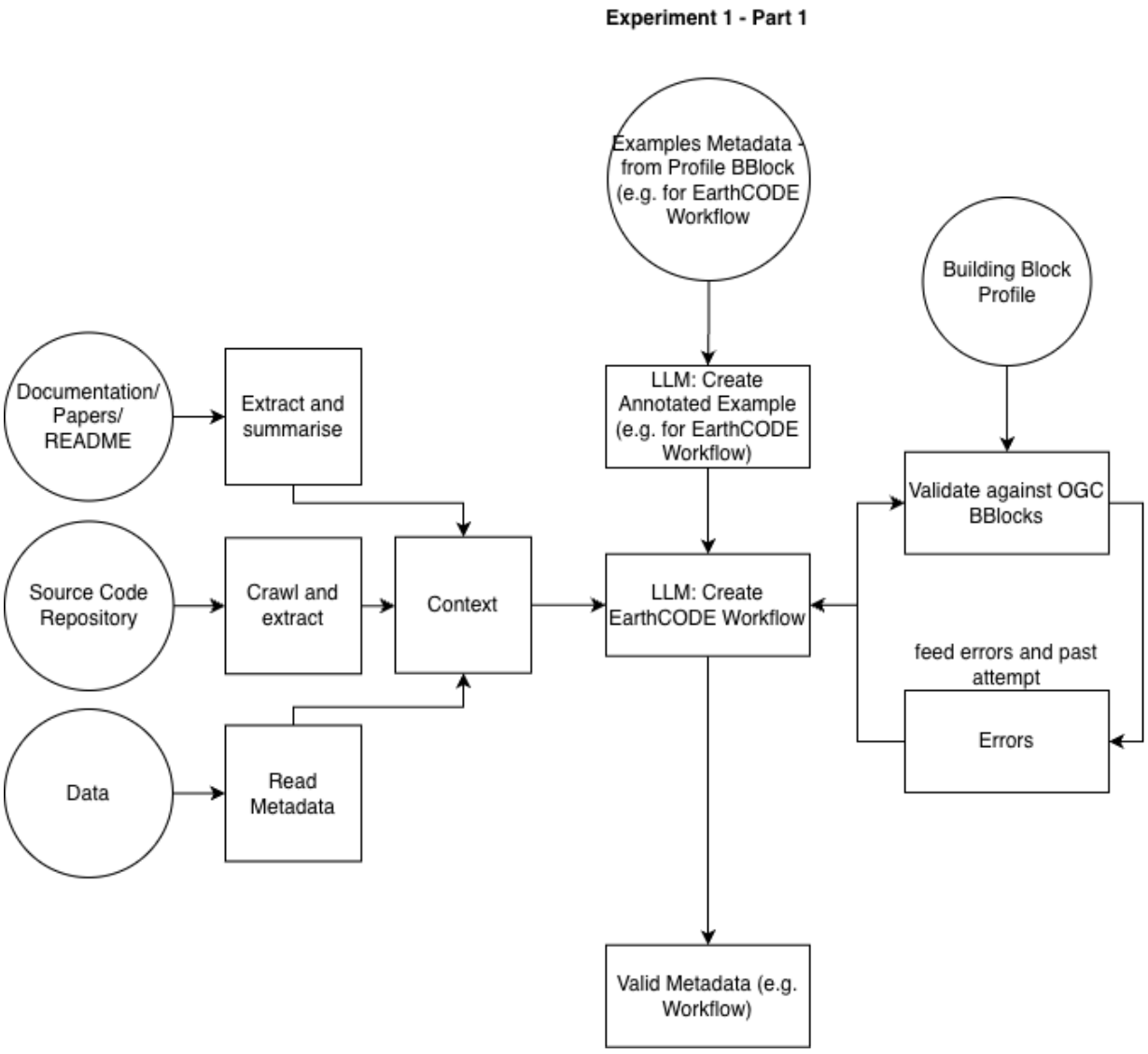

Figure B.1

In the second approach, we aimed to first create the annotated example for Products and Workflows directly from the Building Blocks profile ontology, using the project context rather than from the examples. The idea was to create an optimized, extensive JSON example that would contain relevant fields which richly describe the specific context.

As a first step all possible fields are extracted from a building block profile (e.g. for EarthCODE Workflows), by crawling each dependency. It identifies syntax rules, shacl rules, descriptions and notes. For example for Workflows, it crawls the Open Science BBlock, GeoDCAT, OGC API Records, OGC Main, and other inherited building block definitions, to define all possible fields a valid metadata item might have, which results in approximately 1300 possible fields. The strictly required fields are taken and extracted in a schema.

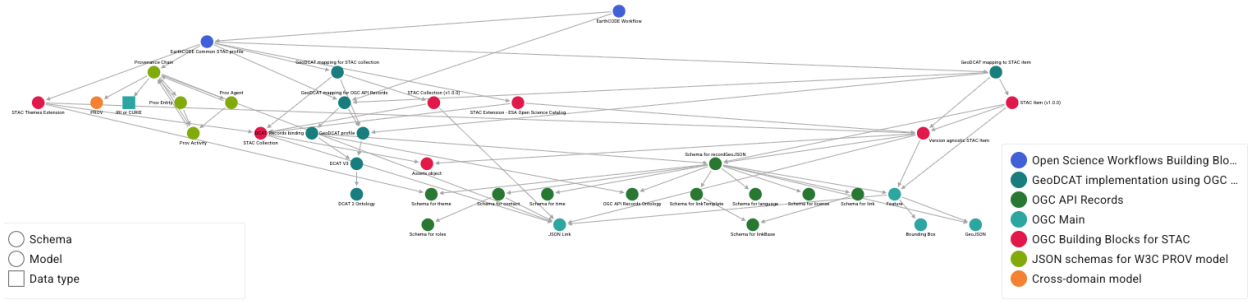

Figure B.2

Then, an LLM workflow does a keyword search that executes three distinct passes to extract, merge, and deduplicate terms from the context, which is used as a critical filter for the subsequent field selection stage.

Using these keywords the system then prioritizes building block fields either through LLM-based reasoning or a deterministic heuristic score based on token overlap to decide which optional fields to include. This selection is passed to a specialized ReAct Agent where it utilizes tools to validate fields and look up specifications, for example https://github.com/radiantearth/stac-spec/tree/master for best practices on STAC metadata. During this phase the Agent syntheses a rigid JSON skeleton keyed exactly to the ontology of the Building Block profile. For example, for the Mangrove detection algorithm, the following product schema was dynamically created for the PolarWarp workflow:

```
 "template_json": {
   "<def:core>": {
     "id": "Extract the repository name from the `Repository URL` (e.g.,
'polarwarp').",
     "type": "Static value 'Workflow' indicating the primary type of content.",
     "bbox": [
       "Extract the geographic bounding box (min_lon, min_lat, max_lon, max_lat)
from the `lon_min`, `lat_min`, `lon_max`, `lat_max` input parameters of the
`execute` workflow in `workflow.yml`."
     ],
     "geometry": "Construct a GeoJSON Polygon from the coordinates specified in
the `bbox` field.",
     "properties": {
       "datetime": "Extract the `date` input parameter from the `execute`
workflow template in `workflow.yml`, representing the start time of the forecast.
"
     }
   },
   "<def:collection>": {
     "title": "Extract the workflow name from `workflow.yml` metadata (`metadata.
name`) or the main title from the top-level `README.md` (`# polarwarp`).",
     "description": "Combine the primary description from the top-level `README.
md` and `polarwarp_code/README.md`, detailing the workflow's purpose, inputs
(Sentinel-1, Copernicus, DESP), and output (sea ice forecasting).",
     "keywords": [
       "List provided keywords: 'Sea ice forecasting', 'Sentinel-1 satellite
data', 'SAR imagery', 'Copernicus Marine Service data', 'NeXtSIM ice drift
forecast', 'ICON weather model', 'Wind-to-ice-drift conversion', 'Nansen
rule', 'Geographic bounding box', 'Ground Control Points (GCPs)', 'Polar
Stereographic projection', 'EPSG:3413 projection', 'EPSG:3031 projection',
'Data download', 'Data resampling', 'Region of Interest (ROI) subsetting',
```

```
'Temporal alignment', 'Drift vector calculation', 'Thin Plate Warping', 'Image
manipulation', 'Landmask filtering', 'Displacement calculation', 'NetCDF files',
'GRIB2 format', 'Shapefiles', 'GeoTiff format', 'Cloud-Optimized GeoTiff (COG)',
'Video generation', 'Argo Workflows orchestration', 'Destination Earth (DESP)
platform'."
      ],
      "extent": {
        "temporal": {
          "interval": [
            [
              "Define the temporal interval using the `date` input parameter
from `workflow.yml` for the start, and calculate the end by adding the `forecast-
duration` parameter defined in `priima/config.py` (or `workflow.yml`)."
            ]
          ]
        },
        "spatial": {
          "bbox": [
            [
              "Extract the typical geographic bounding box of operations,
encompassing both Arctic (EPSG:3413) and Antarctic (EPSG:3031) regions as
implied by the code's projection handling."
            ]
          ]
        }
      },
      "providers": [
        {
          "name": "Extract the name of the organization providing this
collection; e.g., 'Drift+Noise GmbH' as developer/licensor, or 'Copernicus
Marine Service' / 'Destination Earth (DESP) platform' as data providers.",
          "roles": [
            "Assign appropriate roles for the provider; e.g., `producer` or
`licensor` for the developer, and `licensor`, `host`, or `processor` for data
providers/platform operators."
          ]
        }
      ],
      "license": "Extract the software license URI from `polarwarp_code/README.
md` (e.g., `https://www.gnu.org/licenses/gpl-3.0.html`)."
    },
    "<def:assets>": {
      "*": {
        "href": "Construct the URI for output assets: for GeoTiff images, `\"
{output_dir}/{original_image_stem}_forecast_{forecast_step}_{data_source}.
tif\"`; for AVI videos, `\"{output_dir}/{output_subdirectory_name}.avi\"`; for
zipped shapefiles, `\"{output_dir}/trajectories_{sar_datestamp}_{sar_timestamp}.
zip\"`.",
        "type": "Specify the MIME type corresponding to the asset: `image/tiff`
for GeoTiffs, `video/x-msvideo` for AVI videos, `application/zip` for zipped
shapefiles.",
        "roles": [
          "Assign roles based on the asset's function: `['result', 'forecast']`
for GeoTiff images, `['visualisation', 'animation']` for videos, `['vector-
data', 'trajectory']` for shapefiles."
        ]
      }
    },
    "<def:links>": [
      "This is a placeholder for a list of link objects. Populate with objects
following the structure of `<def:link>` for the repository URL, documentation,
and external services (Copernicus, DESP)."
    ],
```

```
   "<def:link>": {
     "rel": "Define the relationship of the link (e.g., `self`, `documentation`,
`related`).",
     "href": "Provide the concrete URL for the linked resource (e.g., repository
URL, specific documentation file, or external service URL)."
   },
   "<def:Entity>": {
     "id": "Generate a unique identifier for each generated output data product
by combining the workflow name, the initial `date` input, and a hash of the
spatial extent (`lon_min`, `lat_min`, `lon_max`, `lat_max`).",
     "type": "Identify the type of data product generated, such as
`IceForecastImage` for GeoTiffs, `IceDriftTrajectory` for shapefiles, or
`IceForecastVideo` for animations.",
     "wasGeneratedBy": "Reference the unique identifier of the `polarwarp`
processing `Activity` that generated this specific `Entity`.",
     "wasDerivedFrom": "Reference the unique identifiers of the primary input
`Entities` (e.g., Sentinel-1 SAR image, NeXtSIM/ICON forecast data) from which
this output was derived."
   },
   "<def:Activity>": {
     "id": "Generate a unique identifier for each distinct activity or workflow
step execution, such as `polarwarp-processing-{timestamp}` or `fetch-s1-data-
{timestamp}`.",
     "activityType": "Describe the type of activity, e.g., `DataProcessing` for
the `polarwarp` step, `DataFetching` for `fetch-predictions` and `fetch-s1`, and
`ForecastGeneration` for the overall `execute` workflow.",
     "qualifiedStart": "Record the start timestamp of the activity's execution,
in ISO 8601 format.",
     "endedAtTime": "Record the end timestamp of the activity's execution, in
ISO 8601 format.",
     "used": "Reference the unique identifiers of the `Entities` (e.g., input
data, configuration parameters, Docker images specified in `workflow.yml`) that
were used as inputs for this activity."
   },
   "<def:Agent>": {
     "name": "Identify the name of the agent, such as 'Drift+Noise GmbH'
(developer), 'Argo Workflows' (orchestrator), 'Copernicus Marine Service' (data
provider), 'DESP Platform' (data provider), or 'PRIIMA' (core software).",
     "type": "Describe the type of agent: `Organization` for Drift+Noise GmbH,
Copernicus Marine Service, DESP Platform; `Software` for Argo Workflows, PRIIMA."
   }
 }
```

Listing B.2

This second approach provided useful information about the generalizability of the process to other building blocks and highlighted potential areas of improvement and importance in the expanded building block schema. Following this initial step to create the example JSON, the methodology continues with the validation loop to extract the metadata in the correct format.

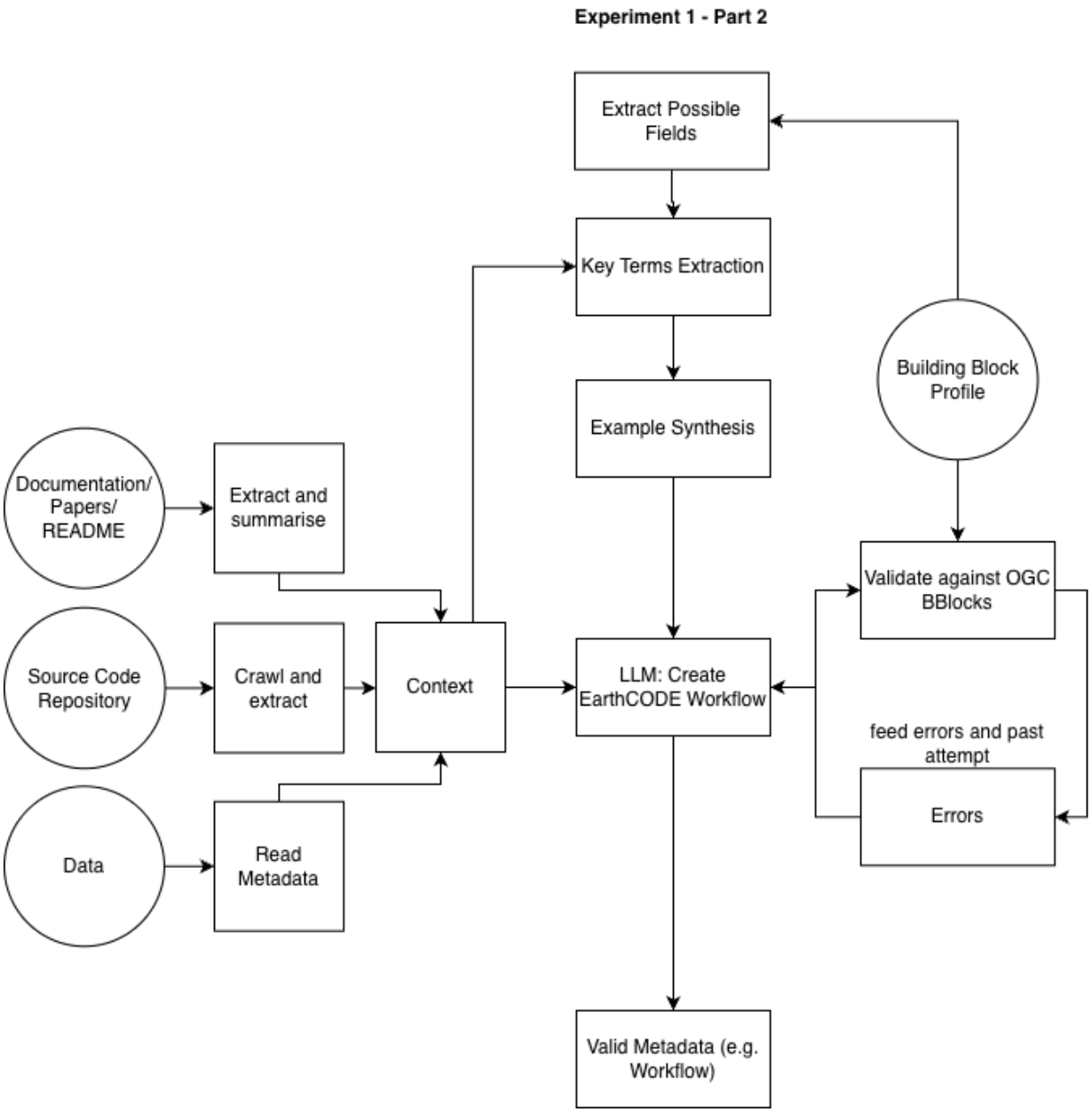

Figure B.3

B.2.2. Experiment 2: Generate CWL Application Packages

In the second experiment, we aim to automatically generate CWL application packages directly from the workflow code and documentation. We used the Common Workflow Language (CWL) specification as the target schema. This expanded the scope of the agentic process and aimed to enhance reproducibility by explicitly capturing information about user inputs, program outputs, and entry points. Generating CWL application packages directly from the workflow code and documentation provides a direct entry point into the workflow, making it possible to re-run the process and replicate and validate the results, assuming access to the execution environment is

available. If the experiment is successful, it would mean that it's possible to automatically map parts of the workflow to different CWL Building Blocks.

In addition to the current validation, we added additional validation using the `cwltool` as we found that the resulting metadata generated for CWLs was passing the CWL Building Block Profile, but failing the more strict `cwltool` validation.

With this approach we were able to generate valid CWL from the context. However, generating CWL definitions with an agentic approach is prone to hallucinations. Even with validation mechanisms in place, the AI can at best guarantee a syntactically correct output that follows the schema definition. It cannot consistently produce semantically correct results for a program, such as the correct workflow order or valid input parameters, without a deeper understanding of the underlying logic.

Therefore, we carry out a second stage of manual validation trying to run the workflow manually and produce expected outputs. We still rely on LLMs to fix errors, however we manually control the process and edit the CWL files.

B.2.3. Experiment 3: Bottom Up

The third experiment focused on breaking down workflows into structural components. To address the semantic limitations of the previous experiments—specifically the inability to detect implicit dependencies and hardcoded I/O—we performed a detailed bottom-up analysis to decompose workflows into their constituent parts.

The goal was to uncover outputs and execution stages that were not immediately obvious from a surface-level scan, such as local file reads hidden within functions or distinct execution steps that were not explicitly linked in a main script. Even when entry points are identified, the precise sequence of calls is often difficult to manage and infer purely through LLM reasoning.

The methodology for this experiment was significantly more complex, involving a neuro-symbolic pipeline that anchors generative AI with static code analysis. The process involved four key steps:

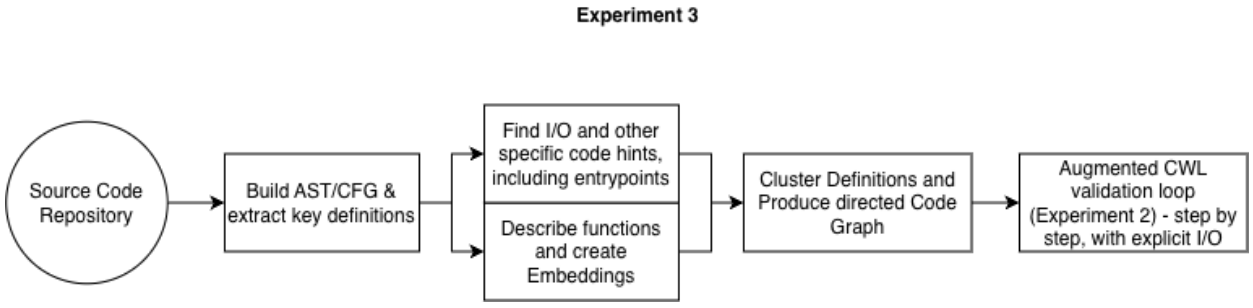

Figure B.4

AST & CFG Construction (Symbolic)	We first built an Abstract Syntax Tree (AST) and Control Flow Graph (CFG) of the code. This deterministic step extracts fully qualified names, literal string values, and specific file path arguments from

	function calls, resolving identifiers within their local and global scopes.
Semantic Embedding (Neuro)	We describe code blocks using LLMs to generate text summaries, which are then converted into numerical vector embeddings. This allows the system to understand the intent of a function, not just its syntax.
I/O Contract Inference	We perform static analysis to identify I/O operations (reads/writes to disk, network calls). These "IO Hints" are passed to an LLM reasoning engine to analyze the larger context, inferring the dataflow contract for each file (e.g., "This script consumes `s2_bands.tif` and outputs `biomass.csv`").
Graph Synthesis & Clustering	Finally, we build clusters of logically grouped functions into "steps" using the Control Flow Graph and the semantic embeddings. These clusters are labeled and assembled into a directed, connected graph representation of the entire repository.

The primary output of this process is a detailed, logical graph of the workflow—a "Code Graph" that describes inputs, outputs, and external dependencies topologically. To illustrate this capability, we ran the algorithm on its own codebase to "self-describe" the analysis pipeline. The system successfully decomposed its own execution logic into distinct, semantically labeled stages, sorting them from inputs to outputs:

```
flowchart TD subgraph "Code AST Analysis & Data Prep"
node_77993718["This workflow step analyzes code structures by parsing
Abstract Syntax Trees to extract fully qualified names, literal
string values, and specific file path arguments from function calls,
simultaneously resolving function identifiers within local and global
scopes. It also standardizes textual information by cleaning code
and converting various inputs into consistent lists of strings,
finally generating sort keys for functions to establish their
processing order."] end subgraph "LLM Thinking & Text Embeddings"
node_95414276["This workflow interacts with Large Language Models to
generate either text responses or numerical vector embeddings based
on input prompts or texts. It extensively uses a robust, thread-safe
caching mechanism, which generates unique keys, stores results in
memory, and persists them to a file, to prevent redundant API calls
and optimize performance."] end subgraph "LLM-driven Code Insight
Extraction" node_77534717["This workflow statically analyzes Python
files to extract function details, identify I/O operations, and build
dependency graphs. It uses large language models to infer dataflow
contracts, summarize functions, cluster them into logical steps, and
generate descriptive labels for these steps, finally exporting all
insights as a combined graph."] end subgraph "Model Codebase Structure
& Workflows" node_57019438["This workflow step meticulously analyzes
a Python codebase to understand its structure, function calls, and I/
O operations, starting from parsing source files and building various
graphs using clustering algorithms on the source code embeddings. It
ultimately groups functions into logical steps, infers their order,
labels them, and exports a comprehensive set of graphs to represent
the codebases architecture and inferred workflows."] end subgraph
"Prepare Allowed Graph Edges" node_99465819["This step filters a graph
```

```
to identify edges that possess a specific kind attribute and share a
common I/O contract. The function then formats these qualifying edges
by prepending fn_ to their node names and returns them as a unique
set."] end
```

Figure B.5

Please note that the preceding chart is a simplified overview, illustrating only the primary, top-level steps and their descriptions.

This resulting graph provides the “ground truth” structure that is often hard to find in scientific code. By explicitly mapping the topological sort of functions and their I/O contracts, this approach produces the precise semantic scaffolding necessary to generate accurate, reproducible CWL application packages.

This structural graph serves as the architectural blueprint for the final generative phase. By traversing the graph in topological order—from initial data ingestion to final product output—the system utilizes the “Code Graph” to orchestrate the incremental construction of the CWL. For each identified step, the LLM is tasked with generating the specific metadata for the Workflow logic, using the inferred I/O contracts to accurately wire inputs and outputs. This approach effectively anchors the stochastic nature of the LLM: the graph dictates the semantic correctness (execution order and dependencies), while the reflexion loop ensures syntactic compliance (schema validation), resulting in a metadata profile that is both logically sound and technically executable.

B.3. Results

The experimental validation was conducted across the six OSPD contributed scientific workflows, assessing generated artifacts for both syntactic compliance (validity against OGC schemas) and semantics (accuracy regarding scientific intent).

B.3.1. Result 1: Metadata Schema-Driven Creation

The results from using the first approach showed that it is possible to automatically generate metadata for all 6 workflows and their related products. Furthermore, the generated metadata items pass the validation checks of both the Workflow and Data Building Blocks, as well as the Open Science Catalog. Three of the existing workflows had human-created metadata that follows the OSC and BB schemas, so we used them to quantitatively verify the quality of our results. We use a f1-score to directly compare the “true” vs “generated” (predicted) JSONs. The results are:

- Waterbodies 0.3272727272727273
- Polaris 0.2603550295857988
- Polarwarp 0.41333333333333333

The low value in the case of the Polaris example is due to two primary reasons. First, the generated record has more information about the argoworkflow structure than the original; and second, the original has external links to other related workflows not present in the metadata. In the case of the Waterbodies workflow, the original does not pass the OSC validation and the generated example has more data.

We used the other three results from the experiments to create three new entries to the open science catalog staging environment — https://osc-staging.earthcode.eox.at/workflows/catalog. The 3 generated examples were manually reviewed by EOX (part of the OSPD) and merged into the staging environment. Areas of improvement during the review were:

- Improve date information;
- Improve the links between the generated items and the existing catalog; and
- Potential semantic problems.

In the review process it became clear that the Kingrove metadata had a semantic issue, since the description was about the notebook version of the workflow, whereas the application specific metadata targeted the CWL version (there are two versions in the same repository). This highlighted the need for a human reviewer. For the other workflows and products we manually validated the results against the data that the workflow creators made, and there were again semantic differences between the description fields and keywords, however the links and the restricted fields such as themes, missions, etc were the same.

Lastly, the generated examples were opened in a PR as examples for the OSPD Workflow building blocks — https://github.com/ogcincubator/bblocks-openscience/pull/3/files. Overall generating and validating this metadata took less than a minute in each case, which would be especially useful for the first-time users of OSC or people that want to generate metadata that follow OGC Building Block specifications.

In the second part of this experiment we attempted to crawl the OSC building block definitions to generate a custom, optimized metadata profile for each workflow. The results pass validation, and the metadata is more richly described (i.e. has more fields), but there is also more noise. The results are:

- polaris 0.011049723756906075
- polarwarp 0.04560260586319218
- waterbodies 0.01662049861495845

While this approach produced syntactically valid JSON, the generated metadata requires more editing to correct. These results can be explained by a combination of the same factors affecting the first part of this experiment and the added noise from ontology selection. During the scanning and field selection process theoretically valid but semantically irrelevant fields get added, resulting in noise. This for example was seen in the Polarwarp generated metadata where:

- There were too many providers / agents fields listed (including Drift+Noise, Copernicus Marine Service, DestinE, and gtif-cerulean);
- There were fields for inputs defined (lon_min, lat_min, lon_max, lat_max, date, num_gcps);
- There were twelve different `Prov:entities` defined;
- There was `Prov:activities` metadata generated; and
- Assets were described in `STAC:Assets` — describing the workflow definition, output tiffs, related zip.

We suspect the problem stems from the ontology fields used, which were extracted from the building block profile. The first issue is the permissive definition of the EarthCODE Workflow building block — for example STAC allows for the definition of Assets under a collection. Although this is valid syntactically, it is not a best practice and is not how EarthCODE Workflows are defined. Restricting the Building Block will greatly reduce the noise as a first step.

The second problem we encountered was the lack of descriptions of the ontology definitions; for example, in the best case scenario a certain field would have a description, type, an enum or clear pattern to indicate what it is expecting, take for example the definition for `STAC.links.type`:

```
"links[].type": {
  "required": false,
  "conditional": true,
  "types": [
    "string"
  ],
  "description": "A hint indicating what the media type of the result of
dereferencing the link should be.",
  "enum": [
    "application/json",
    "application/pdf",
    "application/zip",
    "text/html",
    "text/plain"
  ],
  "pattern": null,
  "format": null,
  "source": "https://schemas.stacspec.org/v1.0.0/collection-spec/json-schema/
collection.json"
},
```

Listing B.3

The LLM can first see a clear description which it can use when "Reasoning", and then take the enum/description to generate an annotated JSON with guidance based on the enum. In contrast to this, the vast majority of fields we see in the derived ontology did not have descriptions, enums, hints or expected patterns; take for example the field for `Prov.wasDerivedFrom`:

```
"<def:Entity>.wasDerivedFrom": {
  "required": false,
  "conditional": false,
  "types": [],
  "description": null,
  "enum": null,
```

```
  "pattern": null,
  "format": null,
  "source": "https://ogcincubator.github.io/bblock-prov-schema/build/annotated/
ogc-utils/prov-entity/schema.json"
},
```

Listing B.4

Adding descriptions to the OGC Building Blocks is expected to significantly improve any Large Language Model (LLM) approach, as LLMs using the ontology will rely on this information. In short, the more explicit and constrained a field's expectations are, the more effectively they can be leveraged by LLM reasoning.

This experiment can be further supplemented by examples to guide and provide additional context. The most powerful approach would be to use a combination of ready-made examples and an enriched, context based approach at the same time — enabling metadata enrichment, as well as generation. This is unfortunately not feasible at the moment because of lack of field descriptions and permissive profiles. In practical terms, the most effective current approach to metadata generation is to use examples directly as guidance and context. Examples within the OGC building blocks are therefore critical not only for human readers and testing, but also for future LLMs attempting to work with these building blocks.

B.3.2. Result 2: Generate CWL Application Packages

The second experiment focused on generating CWL application packages directly from the repository data — code and documentation. We used three workflows of the workflows that already possessed valid CWL definitions to serve as a ground truth baseline for validation — KindGrove, WaterBodies and CVI.

Overall, we managed to produce an executable CWL file for each of the three workflows, using a combination of the previously defined agentic process and manual review and edits, then we ran it to reproduce the data. The required manual edits varied based on the structural complexity of the workflow.

KindGrove	The only change required to generate the KindGrove CWL, was a minor edit of parameter names.
Water Bodies detection	The Water Bodies detection workflow presented another challenge, as the processing logic was defined within the existing CWL structure rather than being explicit in the code base. To resolve this, we had to enrich the repository documentation with explicit Docker environment details and example inputs. Furthermore, specific requirements regarding network access permissions and input scattering needed to be added manually to the CWL.
CVI	Similarly, the CVI workflow required manual steps to define the container and provide example inputs in the README file. Furthermore, the produced CWL varied across runs due to the workflow's multi-step nature and indirectly shared I/O.

These results point to two findings. First, interactively executing the code and fixing runtime errors makes it possible to extract information from workflows that is critical to reproducibility —

input, output, workflow functions, parameters and formats. The second is that there is a heavy dependency on the execution environment, and input and output examples for these types of mapping tasks. Validating a generated CWL file requires the specific runtime environment to be present, as well as example input and output data. If these are not present there is a more complex chicken and egg problem to be solved: the agent needs the environment, input and output to test the generated codes CWL's validity, while at the same time using the generated CWL's (which is uncertain) to validate the generate environment. For example, this was exemplified in the KindGrove workflow, which initially failed due to a dependency on a privately hosted Docker container. Once the container was recreated locally, the CWL generation succeeded.

It is important to note that the selected workflows were already structured in a way conducive to CWL generation, largely utilizing command-line interfaces. This structure is not guaranteed in typical legacy scientific code. Nevertheless, the results demonstrate that while LLM processes can successfully extract entry points and reproducibility data, relying solely on agentic workflows introduces randomness. To mitigate this, the execution environment must be accessible for validation, and the semantic logic of the workflow must be checked as meticulously as the syntax.

B.3.3. Result 3: Bottom Up Analysis

The last experiment builds on the results from the previous two. Although it is possible to generate the required metadata and even CWLs in some cases using purely agentic approaches, real-life workflows present structural challenges that textual analysis often misses. For instance, a processing step might read a local file hardcoded within a function that is not exposed to the user input/output interface. Another common issue is implicit orchestration: even if all functions are exposed, they often require a specific execution sequence that is not defined in code but described loosely in a README (e.g., "Run Notebook A, then Notebook B"). To account for this, we propose a bottom-up analysis: analyzing the code to separate it into functional chunks that consume inputs and produce outputs, generating a call structure based on the Abstract Syntax Tree (AST), and building the metadata using this structural ground truth.

To test the validity of this approach, we focused on two workflows representing different architectural patterns: the Mangrove detection workflow (a compact, monolithic script) and the CVI workflow (a distributed set of scripts). We generated code graphs for both and confirmed their validity to be consistent with our manual understanding of the code. By separating the code this way, we are able to process it in logically consistent chunks. This allows the LLM process to scale to larger codebases that do not fit into a single context window, while simultaneously analyzing the structure to extract the overall workflow logic.

B.3.3.1. Scenario A: Mangrove Detection (Single-Script Logic)

We first focused on the Mangrove workflow (KindGrove). This workflow is composed of a single, self-contained script. The hybrid analysis easily captured the explicit inputs and outputs. More importantly, the code was further broken down into three separate logical steps based on the clustering of the AST and function embeddings:

```
flowchart TD subgraph "Satellite Mangrove Carbon Analysis"
node_17833407["This workflow step orchestrates the identification,
downloading, and processing satellite imagery to detect mangrove
areas within a user-defined geographic region. It then calculates
mangrove biomass and carbon estimates, finally exporting all derived
data and metadata."] end subgraph "Query, Download & Crop S2 Bands"
node_47564399["This workflow step first queries the AWS STAC catalog
for Sentinel-2 L2A satellite scenes that match specified geographic,
temporal, and cloud cover criteria. Subsequently, it downloads and
crops the red, green, and NIR bands from the selected imagery for
the given study area."] end subgraph "Mangrove Carbon Assessment &
Reporting" node_25548366["This workflow processes satellite imagery to
calculate vegetation indices, then uses these to detect mangrove areas
and quantify their extent. It further estimates above-ground biomass,
carbon stocks, and CO2 equivalents within the detected mangroves,
finally exporting these summary results to CSV files."] end
```

Figure B.6

(The preceding chart is derived from the generated graph)

Using the generated graph and verified inputs/outputs, we were able to consistently create the CWL for the KindGrove workflow. Interestingly, this approach theoretically allows for a detailed breakdown of a workflow into logical steps which can then be modelled as granular substeps within a CWL Workflow structure (rather than a monolithic CommandLineTool). While we have not attempted fully nested generation at this stage, the structural graph significantly enriches the final artifact with deep provenance data.

B.3.3.2. Scenario B: CVI (Multi-File Complexity)

The second workflow we analyzed was the Coastal Vulnerability Index (CVI). This workflow presents a significantly different architectural challenge: it consists of several separate processing steps spread across multiple Python files which are not directly linked by a central orchestrator.

Using the neuro-symbolic approach, we were able to infer the implicit I/O contracts between these disjoint files. The process functioned by first statically crawling the codebase to identify all potential I/O operations, and then using context-aware reasoning to determine the specific contract (e.g., specific file names or variable schemas) and link them to the correct sub-step. This effectively reverse-engineered the execution order. The inferred dataflow between the separate entry points is summarized below:

```
flowchart TD file_45172388["compute_elevation.py\nIn:
transects.geojson, tokens.env, config.json, output_dir,
AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, AWS_DEFAULT_REGION,
AWS_ENDPOINT_URL\nOut: transects_with_elevation.geojson"]
file_96225038["extract_coastline.py\nIn: aoi_csv\nOut:
coastline.gpkg"] file_56722915["setup_env.py\nIn: config.json,
output_dir\nOut: config_validated.json"] file_65367079["compute_cvi.py
\nIn: transects_with_land_cover.geojson, transects_with_slope.geojson,
transects_with_erosion.geojson, transects_with_elevation.geojson,
```

```
config.json, output_dir\nOut: transects_with_cvi_equal.geojson"]
file_26186655["generate_transects.py\nIn: coastline.gpkg, output_dir
\nOut: transects.geojson"] file_96571969["compute_landcover.py
\nIn: transects.geojson, tokens.env, config.json, output_dir,
AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, AWS_DEFAULT_REGION,
AWS_ENDPOINT_URL\nOut: transects_with_land_cover.geojson"]
file_61549461["compute_slope.py\nIn: transects.geojson,
tokens.env, config.json, output_dir, AWS_ACCESS_KEY_ID,
AWS_SECRET_ACCESS_KEY, AWS_DEFAULT_REGION, AWS_ENDPOINT_URL,
Copernicus_DSM_COG_10_DEM.tif\nOut: transects_with_slope.geojson"]
file_75731443["compute_erosion.py\nIn: transects.geojson, config.json,
output_dir\nOut: transects_with_erosion.geojson"] file_45172388 --> |
io| file_65367079 file_45172388 --> |io| file_75731443 file_45172388
--> |io| file_96571969 file_45172388 --> |io| file_61549461
file_75731443 --> |io| file_65367079 file_96571969 --> |io|
file_65367079 file_96571969 --> |io| file_75731443 file_96571969 --> |
io| file_61549461 file_61549461 --> |io| file_65367079 file_61549461
--> |io| file_75731443 file_96225038 --> |io| file_65367079
file_96225038 --> |io| file_45172388 file_96225038 --> |io|
file_75731443 file_96225038 --> |io| file_96571969 file_96225038 --> |
io| file_61549461 file_96225038 --> |io| file_26186655 file_26186655
--> |io| file_65367079 file_26186655 --> |io| file_45172388
file_26186655 --> |io| file_75731443 file_26186655 --> |io|
file_96571969 file_26186655 --> |io| file_61549461 file_56722915 --> |
io| file_65367079 file_56722915 --> |io| file_45172388 file_56722915
--> |io| file_75731443 file_56722915 --> |io| file_96571969
file_56722915 --> |io| file_61549461
```

Figure B.7 — File io

In the generated graph, this architecture is further decomposed across multiple levels of granularity: statements are linked to functions, functions are mapped to logical steps, steps to entry points, and entry points to specific files, with explicit semantic connections defined between each layer.

Using this structural approach, we were able to combine the agentic approach from previous experience by guiding it with the generated graph to create a working CWL for the CVI workflow, successfully resolving the execution order that pure LLM reasoning failed to consistently capture. Additionally, this approach demonstrated significant value in dependency discovery. The static analysis successfully caught latent external dependencies—specifically, calls to the Overpass API and Copernicus data repositories. Although these dependencies are not currently utilized in the primary execution path of the algorithm, detecting them is critical for future provenance modeling and ensures that the generated metadata captures the complete potential footprint of the software.

B.4. Conclusion

This project demonstrates that the automated generation of geospatial metadata becomes feasible when provided with high-quality well described OGC Building Blocks and the correct

validation approach is applied. These results suggest a path forward to make the vast majority of scientific workflows, which are not reproducible, more so. Throughout these experiments, we intentionally utilized lightweight, low-latency models (Gemini 2.5 Flash) to demonstrate that system robustness stems from the validation architecture rather than the raw parameter count of the model.

Based on the validation scenarios, we draw four primary conclusions regarding the nature of metadata generation, the dependencies of execution environments, and Building Block applications, which we describe below.

In addition to these, our work has had this additional impact:

- We automatically generated workflow metadata for 3 of the example workflows, which were reviewed and merged into the Open Science Catalog Staging area;
- We added these as examples for the workflow building blocks; and
- We aim to have the metadata Generation functionality from experiment 1 to be added to upcoming Open Science Catalog library to help users Open Science Catalog users generate data.

B.4.1. Syntactic Compliance vs. Semantic Fidelity

The experiments confirm that automatically generating metadata against a Building Block schema is highly achievable. The iterative, "self-correcting" reflexion loop proved capable of producing syntactically valid JSON outputs for all tested workflows in under one minute. However, a critical distinction must be made between syntactic compliance and semantic utility. Obtaining a minimal "passing" example is straightforward, but extracting a rich, maximum-profile description is difficult and, without structural analysis and actual execution, often results in hallucinatory noise rather than actionable metadata.

We found that incorporating iterative execution attempts and fixing runtime errors, as well as adding Bottom-Up Code Analysis approach — mixing static code analysis with agentic reasoning — improves performance for complex, multi-step workflows. This approach stabilizes the process, allowing the system to uncover hidden dependencies (e.g., the dependency on an external Overpass API in the CVI workflow) that purely semantic analysis misses. Furthermore, by building a deterministic "Code Graph" first, translation to various standards becomes a deterministic transformation rather than a probabilistic guess.

B.4.2. The "Chicken and Egg" Execution Problem

Beyond descriptive metadata, the generation of executable logic (CWL) revealed a recurring dependency on the Execution Environment. Validating the functionality of a generated workflow requires access to the specific container or runtime environment, as well as example input and output data. This is difficult to automate since, defining these often requires understanding the workflow first. Therefore, to fully enable both accurate and automatic mapping or

transformations of workflow metadata the execution environment, input and output examples are needed.

In addition, an indication of the complexity of run is required — a notion of the cost of running a workflow with an input to produce an output — to enable the interactive runtime iterative loop. Running agentic validation loops has a cost that is additional to the cost of running the workflow. If the initial cost of running the workflow is unknown (or unknowable) this makes the generate/static validation/execution validation loop difficult to manage automatically.

B.4.3. Implications for OGC Building Blocks

Our results indicate that LLMs process textual semantics far better than complex graph relationships. Therefore, providing the AI with annotated descriptions and clear examples (Approach 1) yields significantly better results than providing a fully expanded, link-heavy schema (Approach 2). We propose that future iterations that aim to use the OGC Building Blocks will greatly benefit from richer field descriptions and clear field-constraints. With these, a richer approach that combines both examples and dynamic schema creation can be fully leveraged.

The fact that we could generate executable CWLs also suggests that there is significant room to extend the OSPD Building Block schemas. It can evolve toward a structure resembling the Common Workflow Language (CWL) to capture execution logic, not just descriptive metadata. Our experiments confirm that LLM processes are capable of populating these extended fields by analyzing the bottom-up structure of the code to reveal implicit execution sequences. Additionally, we suggest that runtime costs are included in the schema, or at least examples this would enable an automatic generate / validate / runtime — validation loop, which produces significantly better results than a static generate / validate loop.

B.4.4. The Human-in-the-Loop

Ultimately, while the AI model successfully processed data within context windows and identified syntax errors, human oversight remains essential for semantic verification. The experiments revealed instances where the AI generated syntactically valid metadata that was semantically incorrect—for example, confusing a Notebook version of a workflow with its CWL counterpart within the same repository. The value of the AI, therefore, is not to replace the domain expert, but to act as a force multiplier. By reducing the time required to generate initial metadata from hours to seconds, the AI removes the “blank page” barrier, shifting the human role from manual authoring to high-level review.

B.5. Limitations

There are several inherent limitations related to both the selected workflows and the defined LLM processes that must be considered when interpreting these results. First, the validation scenarios were relatively ideal candidates: self-contained, reasonably well-structured, and written exclusively in Python. This homogeneity allowed them to fit easily within standard

LLM context windows, simplifying the analysis. However, less widely used languages, larger monolithic repositories, and fragmented code structures will introduce significant new challenges. The majority of scientific workflows are poorly documented, do not follow best practices and may depend on manually interleaving processing steps that are difficult to capture programmatically.

Second, the methodology relies heavily on the existence of predefined execution environments. To validate a generated workflow, the system requires a functioning runtime environment (e.g., a Docker container). This is a heavy prerequisite, as many scientific repositories lack formal environment definitions (such as `environment.yml` or `Dockerfile`), making the "chicken and egg" problem of validation a persistent barrier to full automation.

Third, the agentic process defined in Experiments 1 and 2 was intentionally kept simple to establish a baseline. While Experiment 3 demonstrated the value of advanced structural analysis, the schema-driven discovery and CWL generation loops relied on relatively straightforward prompt engineering. As the OGC Building Block schemas evolve in complexity—incorporating deeper semantic links and stricter validation rules—these simple agentic workflows will likely need to be upgraded with more robust reasoning capabilities and state management.

B.6. Future work

To validate and extend the findings of this work, we propose building a large catalogue of geospatial scientific code that aims to be representative of the well-known shortcomings of real-world research. Unlike the clean examples used in this pilot, this catalogue should capture the "messy" reality of scientific workflows: underspecified environments, missing comments, and complex data movements across heterogeneous environments (e.g., performing analysis in R, executing a C command-line function, and switching to QGIS for result tuning). Analyzing these fragmented, polyglot pipelines will allow us to test the neuro-symbolic engine against workflows that do not fit into a single LLM context window.

A related suggestion is to curate a list of relevant open geospatial workflows hosted on GitHub and run our Code Graph generation and clustering algorithms at scale. The aim of this empirical analysis is to identify "common steps"—recurring processing patterns across different projects. Performing this bottom-up analysis on this scale will enable us to suggest data-driven additions to the OSPD Building Blocks based on actual usage patterns of the community in a quantitative manner.

Future work should also focus on using Building Blocks that have well described fields, in combination with the OGC Rainbow Graph and build AI workflows to create richly annotated schemas and metadata definitions. Furthermore, we propose exploring more complex agentic architectures that can scale with workflow complexity, such as multimodal and multi-agent systems that can reason around the core data directly (e.g. interpreting and labeling geotiffs).

We suggest replacing the current "static" validation step with a specialized Evaluator Agent. This agent would provide qualitative feedback based on best practices for STAC and OGC API

Records, allowing the system to actively enrich existing metadata with deep provenance and dependency tracking, rather than just checking for syntax errors.

Finally, while this report highlights the success of static and agentic methods, the “gold standard” for analyzing code remains dynamic code analysis. A combination of dynamic analysis (runtime observation), static analysis (structural mapping), and agentic analysis (semantic reasoning) would, in theory, create the most robust method for automated reproducibility. Future iterations of the OSPD should explore this tripartite approach, grounding these advanced analytical methods in the core data profiles of the OGC Building Blocks as a foundational target, validator and constraint.

C

ANNEX C (NORMATIVE) RMIT CONTRIBUTIONS

ANNEX C (NORMATIVE) RMIT CONTRIBUTIONS

C.1. Introduction

One of the main objectives of the OSPD 2025 project was to test the potential of building blocks to support the standardization of metadata for reproducing and reusing scientific workflows. RMIT University contributed to this project by focusing on the reuse of existing standard ontologies that are suitable for capturing workflow provenance. These ontologies can serve as a foundation for establishing an Open Science Ontology and for supporting the development of relevant building blocks that enable standardized metadata and provenance capture for scientific workflows. To carry out the project tasks, we adopted the following approach:

- familiarising ourselves with the concept of building blocks;
- reviewing and identifying existing standard ontologies relevant to the project;
- developing a building block example that describes and captures workflow provenance; and
- evaluating the potential benefits and challenges of using building blocks to support the reproducibility and reuse of scientific workflows.

C.2. The Building Blocks concept

The Building Block is an approach to the containerization of specifications that can be reused across different contexts and applications. In other words, the main idea of building blocks is to link required standards and specifications in an automated manner that supports the FAIR principles for data exchange, quality control, and built-in testing and validation. The primary objective is to enable technology-agnostic dependencies and relationships through a knowledge graph–based data management approach.

C.3. Open Science Ontology

In this project, we proposed the reuse of several already established ontologies for capturing workflow provenance. A review of existing ontologies led to the selection of three core ontologies for the purposes of this project: wf4ever (https://wf4ever.github.io/ro/2016-01-28/wf4ever/), wfDesc (https://wf4ever.github.io/ro/2016-01-28/wfdesc/), and wfProv(https://wf4ever.github.io/ro/2016-01-28/wfprov/). Essentially, all three ontologies have strong links to the PROV-O (https://www.w3.org/TR/prov-o/) ontology, which is the standard ontology for the semantic representation and enrichment of provenance and lineage information.

C.4. Building Blocks for workflow provenance

Using the description of the Kind Grove (https://gitlab.ogc.org/ogc/OSPD-2025/-/wikis/WP-1:-Workflow-discovery-and-evaluation/Mangrove-Detect) scientific workflow example and the identified workflow provenance ontologies, we were able to generate a building block example. We applied all workflow steps and processes to map provenance and workflow descriptions using wf4ever, wfDesc, and wfProv. The mappings of input data, workflow steps, and output data are provided in the Turtle file below.

```
PREFIX prov: <http://www.w3.org/ns/prov#>
PREFIX schema: <https://schema.org/>
PREFIX stac: <https://w3id.org/ogc/stac/core/>
PREFIX wfdesc: <http://purl.org/wf4ever/wfdesc#>
PREFIX wfkg: <https://example.org/kindgrove/>
PREFIX wfprov: <http://purl.org/wf4ever/wfprov#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

wfkg:DefineStudyAreaRun
    a wfprov:ProcessRun ;
    wfprov:describedByProcess wfkg:DefineStudyAreaProcess ;
    wfprov:hasSubProcessRun
        wfkg:AddCenterPointRun ,
        wfkg:AddInfoAnnotationRun ,
        wfkg:AddStudyAreaBoundaryRun ,
        wfkg:CreateBoundingBoxRun ,
        wfkg:CreateLocationSelectorWidgetRun ,
        wfkg:CreateMapRun ,
        wfkg:DisplayStudyAreaRun ,
        wfkg:GetSelectedSiteRun ;
    wfprov:usedInput
        wfkg:DateRange ,
        wfkg:StudyArea ;
    wfprov:wasEnactedBy wfkg:JupyterNotebookEngine ;
    wfprov:wasPartOfWorkflowRun wfkg:KindGroveWorkflowRun ;
.

wfkg:InteractiveVisualisation
    a wfprov:Artifact ;
    wfprov:wasOutputFrom wfkg:DisplaySummaryReportRun ;
```

```
    schema:encodingFormat "text/html" ;
    schema:name "HTML Summary Report" ;
.

wfkg:AWSSTACCatalog
    a
        wfprov:Artifact ,
        stac:Catalog ;
    schema:name "AWS STAC Catalog" ;
    schema:url "https://earth-search.aws.element84.com/v1" ;
.

wfkg:AllometricEquationDesc
    a wfdesc:Process ;
.

wfkg:AllometricEquationRun
    a wfprov:ProcessRun ;
    wfprov:describedByProcess wfkg:AllometricEquationDesc ;
    wfprov:usedInput
        wfkg:MangroveMask ,
        wfkg:NDVI ;
    wfprov:wasEnactedBy wfkg:JupyterNotebookEngine ;
    wfprov:wasPartOfWorkflowRun wfkg:KindGroveWorkflowRun ;
.

wfkg:BiomassEstimate
    a wfprov:Artifact ;
    wfdesc:wasOutputFrom wfkg:AllometricEquationRun ;
.

wfkg:CO2EquivalentDesc
    a wfdesc:Process ;
.

wfkg:CO2EquivalentEstimate
    a wfprov:Artifact ;
    wfdesc:wasOutputFrom wfkg:CO2EquivalentRun ;
.

wfkg:CO2EquivalentRun
    a wfprov:ProcessRun ;
    wfprov:describedByProcess wfkg:CO2EquivalentDesc ;
    wfprov:usedInput wfkg:CarbonStockEstimate ;
    wfprov:wasEnactedBy wfkg:JupyterNotebookEngine ;
    wfprov:wasPartOfWorkflowRun wfkg:KindGroveWorkflowRun ;
.

wfkg:CSVSummary
    a wfprov:Artifact ;
    wfdesc:wasOutputFrom wfkg:ExportResultsRun ;
    schema:encodingFormat "text/csv" ;
.

wfkg:CalculateIndicesProcessDesc
    a wfdesc:Process ;
.

wfkg:CarbonStockDesc
    a wfdesc:Process ;
.

wfkg:CarbonStockEstimate
```

```
    a wfprov:Artifact ;
    wfdesc:wasOutputFrom wfkg:CarbonStockRun ;
.

wfkg:CarbonStockRun
    a wfprov:ProcessRun ;
    wfprov:describedByProcess wfkg:CarbonStockDesc ;
    wfprov:usedInput wfkg:BiomassEstimate ;
    wfprov:wasEnactedBy wfkg:JupyterNotebookEngine ;
    wfprov:wasPartOfWorkflowRun wfkg:KindGroveWorkflowRun ;
.

wfkg:DefineStudyAreaProcess
    a wfdesc:Process ;

.

wfkg:DisplaySummaryReportRun
    a wfdesc:Process ;
.

wfkg:DisplaySummaryReportRun
    a wfprov:ProcessRun ;
    wfprov:describedByProcess wfkg:DisplaySummaryReportRun ;
    wfprov:usedInput
        wfkg:CSVSummary ,
        wfkg:GeoTIFFRaster ;
    wfprov:wasPartOfWorkflowRun wfkg:KindGroveWorkflowRun ;
    prov:endedAtTime "2024-06-01T10:46:00+00:00"^^xsd:dateTime ;
.

wfkg:DownloadBandsProcessDesc
    a wfdesc:Process ;
.

wfkg:ExportResultsDesc
    a wfdesc:Process ;
.

wfkg:GeoTIFFRaster
    a wfprov:Artifact ;
    wfdesc:wasOutputFrom wfkg:ExportResultsRun ;
    schema:encodingFormat "image/tiff" ;
.

wfkg:GreenBand
    a wfprov:Artifact ;
    wfdesc:wasOutputFrom wfkg:DownloadBandsRun ;
.

wfkg:KindGroveWorkflow
    a
        wfdesc:Workflow ,
        schema:ComputationalWorkflow ;
    schema:name "KindGrove Mangrove Carbon Assessment Workflow" ;
    schema:version "1.0" ;
.

wfkg:Myanmar
    a schema:Country ;
    schema:name "Myanmar" ;
.
```

```
wfkg:NDWI
    a wfprov:Artifact ;
    wfdesc:wasOutputFrom wfkg:CalculateIndicesRun ;
.

wfkg:NIRBand
    a wfprov:Artifact ;
    wfdesc:wasOutputFrom wfkg:DownloadBandsRun ;
.

wfkg:RedBand
    a wfprov:Artifact ;
    wfdesc:wasOutputFrom wfkg:DownloadBandsRun ;
.

wfkg:SAVI
    a wfprov:Artifact ;
    wfdesc:wasOutputFrom wfkg:CalculateIndicesRun ;
.

wfkg:SelectScenesProcessDesc
    a wfdesc:Process ;
.

wfkg:SelectScenesRun
    a wfprov:ProcessRun ;
    wfprov:describedByProcess wfkg:SelectScenesProcessDesc ;
    wfprov:usedInput
        wfkg:DateRange ,
        wfkg:Sentinel2L2A ,
        wfkg:StudyArea ;
    wfprov:wasEnactedBy wfkg:JupyterNotebookEngine ;
    wfprov:wasPartOfWorkflowRun wfkg:KindGroveWorkflowRun ;
.

wfkg:SelectedScene
    a wfprov:Artifact ;
    wfdesc:wasOutputFrom wfkg:SelectScenesRun ;
    schema:description "Lowest cloud cover Sentinel-2 scene" ;
.

wfkg:Sentinel2L2A
    a
        wfprov:Artifact ,
        stac:Collection ;
    schema:distribution wfkg:Sentinel2L2A_AWSS3Distribution ;
    schema:isPartOf wfkg:AWSSTACCatalog ;
    schema:name "Sentinel-2 L2A Multispectral Optical Imagery" ;
.

wfkg:Sentinel2L2A_AWSS3Distribution
    a schema:DataDownload ;
    schema:contentUrl "s3://sentinel-inventory/sentinel-s2-l2a/" ;
    schema:isAccessibleForFree true ;
    schema:name "Sentinel-2 L2A AWS Open Data Registry (S3)" ;
    schema:provider "AWS Open Data Registry" ;
.

wfkg:StarlingFoundries
    a prov:Organization ;
    schema:name "Starling Foundries" ;
.
```

```
wfkg:StudyAreaGeometry
    a schema:GeoShape ;
    schema:box "95.15 15.9 95.35 16.1" ;
    schema:center "95.25 16.0" ;
    schema:polygon "95.15 15.9 95.35 15.9 95.35 16.1 95.15 16.1 95.15 15.9" ;
.

wfkg:ThresholdClassificationDesc
    a wfdesc:Process ;
.

wfkg:ThresholdClassificationRun
    a wfprov:ProcessRun ;
    wfprov:describedByProcess wfkg:ThresholdClassificationDesc ;
    wfprov:usedInput
        wfkg:NDVI ,
        wfkg:NDWI ,
        wfkg:SAVI ;
    wfprov:wasEnactedBy wfkg:JupyterNotebookEngine ;
    wfprov:wasPartOfWorkflowRun wfkg:KindGroveWorkflowRun ;
.

wfkg:CameronSajedi
    a prov:Person ;
    schema:affiliation wfkg:StarlingFoundries ;
    schema:name "Cameron Sajedi" ;
.

wfkg:DateRange
    a wfprov:Artifact ;
    schema:description "Temporal range for satellite query" ;
.

wfkg:ExportResultsRun
    a wfprov:ProcessRun ;
    wfprov:describedByProcess wfkg:ExportResultsDesc ;
    wfprov:usedInput
        wfkg:CO2EquivalentEstimate ,
        wfkg:MangroveMask ;
    wfprov:wasEnactedBy wfkg:JupyterNotebookEngine ;
    wfprov:wasPartOfWorkflowRun wfkg:KindGroveWorkflowRun ;
.

wfkg:MangroveMask
    a wfprov:Artifact ;
    wfdesc:wasOutputFrom wfkg:ThresholdClassificationRun ;
.

wfkg:NDVI
    a wfprov:Artifact ;
    wfdesc:wasOutputFrom wfkg:CalculateIndicesRun ;
.

wfkg:StudyArea
    a wfprov:Artifact ;
    schema:containedInPlace wfkg:Myanmar ;
    schema:description "1,800 acres of mangrove restoration in Ayeyarwady Delta"
;
    schema:geo wfkg:StudyAreaGeometry ;
    schema:name "Thor Heyerdahl Climate Park" ;
.

wfkg:CalculateIndicesRun
```

```
    a wfprov:ProcessRun ;
    wfprov:describedByProcess wfkg:CalculateIndicesProcessDesc ;
    wfprov:usedInput
        wfkg:GreenBand ,
        wfkg:NIRBand ,
        wfkg:RedBand ;
    wfprov:wasEnactedBy wfkg:JupyterNotebookEngine ;
    wfprov:wasPartOfWorkflowRun wfkg:KindGroveWorkflowRun ;
.

wfkg:DownloadBandsRun
    a wfprov:ProcessRun ;
    wfprov:describedByProcess wfkg:DownloadBandsProcessDesc ;
    wfprov:usedInput wfkg:SelectedScene ;
    wfprov:wasEnactedBy wfkg:JupyterNotebookEngine ;
    wfprov:wasPartOfWorkflowRun wfkg:KindGroveWorkflowRun ;
.

wfkg:JupyterNotebookEngine
    a wfprov:WorkflowEngine ;
    prov:actedOnBehalfOf wfkg:CameronSajedi ;
    schema:name "Jupyter Notebook" ;
.

wfkg:KindGroveWorkflowRun
    a wfprov:WorkflowRun ;
    wfprov:describedByWorkflow wfkg:KindGroveWorkflow ;
    wfprov:wasInitiatedBy wfkg:CameronSajedi ;
    prov:endedAtTime "2024-06-01T10:45:00+00:00"^^xsd:dateTime ;
    prov:startedAtTime "2024-06-01T10:00:00+00:00"^^xsd:dateTime ;
.
```

Listing C.1

C.4.1. Kind Grove workflow provenance building block example

An example of the building block is available via the GitHub repository (https://nenadradosevic.github.io/bblocks-openscience/bblock/ogc.osc.geodcat-stac-earthcode.experiments/examples). The building block includes Turtle and JSON-LD files that capture the Kind Grove workflow provenance.

C.4.2. Validation

We successfully integrated SHACL rules into the building block example to validate our workflow provenance mappings. The SHACL rules implemented in the building block are provided below. First rule checks that class wfprov:procesRun requires the object property wfprov:describedByProcess and the class wfdesc:Process. The second SHACL rule constrains the target class wfprov:procesRun wfprov:wasPartOfWorkflowRun class wfprov:procesRun requires the object property wfprov:wasPartOfWorkflowRun with the minimun reqired number of 1. Finally, the last SHACL rule validates that each derived worklfow output as wfprov:Artefact must link to another target class wfprov:ProcessRun.

```
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix sh:      <http://www.w3.org/ns/shacl#> .
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .
```

```
@prefix wfprov:  <http://purl.org/wf4ever/wfprov#> .
@prefix wfdesc:  <http://purl.org/wf4ever/wfdesc#> .
@base <https://www.ogc.org/rules/kindgrove/> .
############################################################
# ProcessRun must be described by a wfdesc:Process
############################################################
<#ProcessRunShape>
    a sh:NodeShape ;
    sh:targetClass wfprov:ProcessRun ;
    sh:message "Each ProcessRun must be described by a wfdesc:Process." ;
    sh:property [
        sh:path wfprov:describedByProcess ;
        sh:class wfdesc:Process ;
        sh:minCount 1
    ] .
############################################################
# ProcessRun must belong to a WorkflowRun
############################################################
<#ProcessRunWorkflowShape>
    a sh:NodeShape ;
    sh:targetClass wfprov:ProcessRun ;
    sh:message "Each ProcessRun must be associated with a WorkflowRun." ;
    sh:property [
        sh:path wfprov:wasPartOfWorkflowRun ;
        sh:class wfprov:WorkflowRun ;
        sh:minCount 1
    ] .
############################################################
# Derived artifacts must point to a ProcessRun
############################################################
<#ArtifactProvenanceShape>
    a sh:NodeShape ;
    sh:targetSubjectsOf wfprov:wasOutputFrom ;
    sh:targetSubjectsOf wfdesc:wasOutputFrom ;
    sh:message "Derived artifacts must be linked to a ProcessRun that produced
them." ;
    sh:property [
        sh:path [ sh:alternativePath ( wfprov:wasOutputFrom wfdesc:wasOutputFrom
) ] ;
        sh:class wfprov:ProcessRun ;
        sh:minCount 1 ;
    ] .
```

Listing C.2

C.5. Discussion

Building blocks provide seamless integration of different standards and specifications. Our example demonstrates how building blocks can integrate, describe, and validate scientific workflow provenance, supporting their reuse and interoperability. The capabilities of building blocks extend further by enabling automated integration of all dependencies and relationships between different standards, ontologies, SHACL rules, and metadata statements.

Nevertheless, there are some challenges and limitations regarding building blocks, such as the inherent complexity of integrating multiple specifications. This complexity may pose a challenge

for their adoption among the wider industry and scientific community. We recommend breaking building blocks into simpler tasks that can be easily understood by a broad audience, not just software engineers and developers.

C.6. Future work

Future work may consider further development of an Open Science ontology that incorporates additional existing standard ontologies, enabling better support and faster uptake of the building blocks concept for open and reproducible scientific projects. In addition, future work may include the development of building blocks that capture finer levels of granularity in workflow description, provenance, execution, and other important aspects, thereby improving the reproducibility and transparency of scientific workflows.

D

ANNEX D (INFORMATIVE) IMPROVEMENTS BY KURRAWONG AI TO OGC RAINBOW TO FACILITATE OSPD NEEDS

ANNEX D (INFORMATIVE) IMPROVEMENTS BY KURRAWONG AI TO OGC RAINBOW TO FACILITATE OSPD NEEDS

For KurrawongAI, the main contribution goal was to add capabilities to Prez and prez-ui to support the OSPD ontologies and building blocks, to prepare the OGC Rainbow for hosting them.

Originally, the following capabilities were set to add to Prez in stages:

1. Display of background objects
    a) Ontologies in general
    b) Some specific ontologies
    c) Building Blocks in general
2. Display of simple OSPD-specific objects
    a) OSPD Building Blocks
    b) Workflow runs
3. Display of complex OSPD-specific objects
    a) Building Blocks linked to other things in unusual ways

During the course of the pilot, the following work was done, with some ongoing work that will be added to Prez when it is reviewed for its next version.

First and foremost, to support these additions, OGC's prez-ui application needs to be updated to the latest version of prez-ui, which was done in this PR: https://github.com/ogcincubator/ogc-prez-ui/pull/10

The capabilities added to Prez (and as such, soon the OGC Rainbow) are:

1. Added support to prez-ui and prez-lib for ontologies in general (1.a — https://github.com/RDFLib/prez-ui/pull/240) and tested the following ontologies specifically (1.b — https://github.com/Kurrawong/ogc-test-catalogue):

a) ORE Vocabulary https://www.openarchives.org/ore/1.0/datamodel

b) Workflow Forever Research Object Model https://wf4ever.github.io/ro/

c) Annotation Ontology https://github.com/annotation-ontology/annotation-ontology/

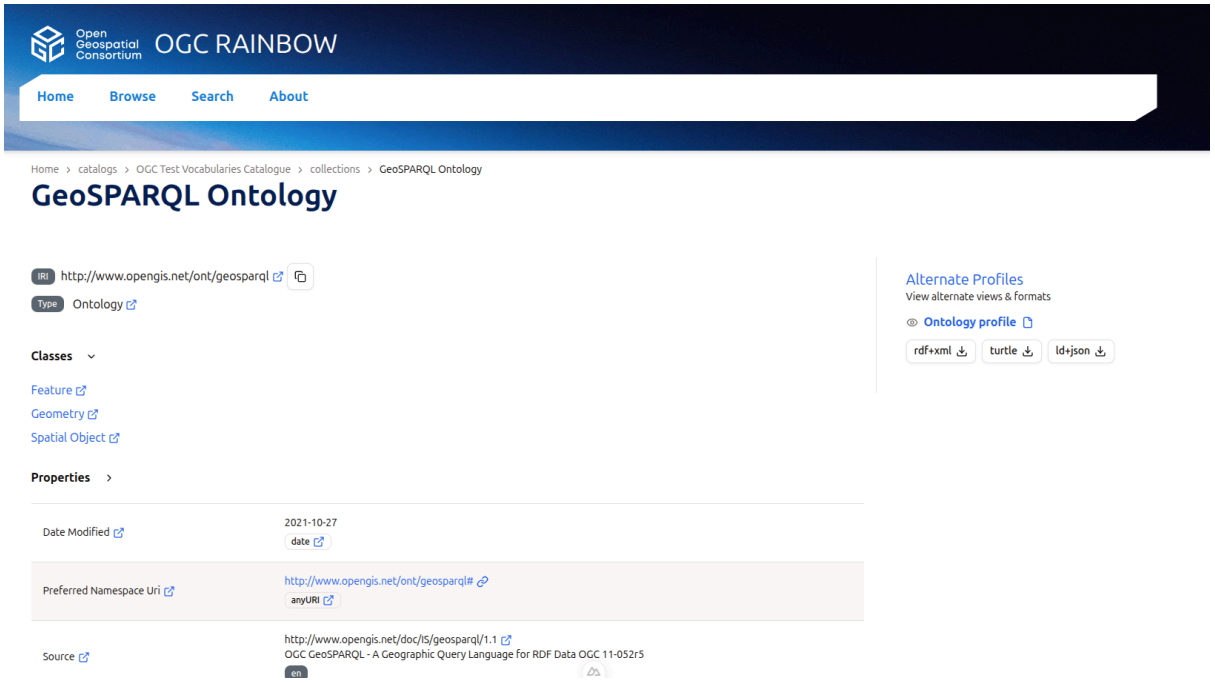

Figure D.1

2. A widget was developed to display OGC Building Blocks (including those made for the OSPD) within the OGC Rainbow (2.a — https://github.com/RDFLib/prez-ui/pull/252), specifically their dependency diagrams, for which a PR exists here: https://github.com/RDFLib/prez-ui/pull/252 . Display of workflow runs (2.b) was not feasible at this time, due to variety of the workflow scenarios, as well as the limited time available.

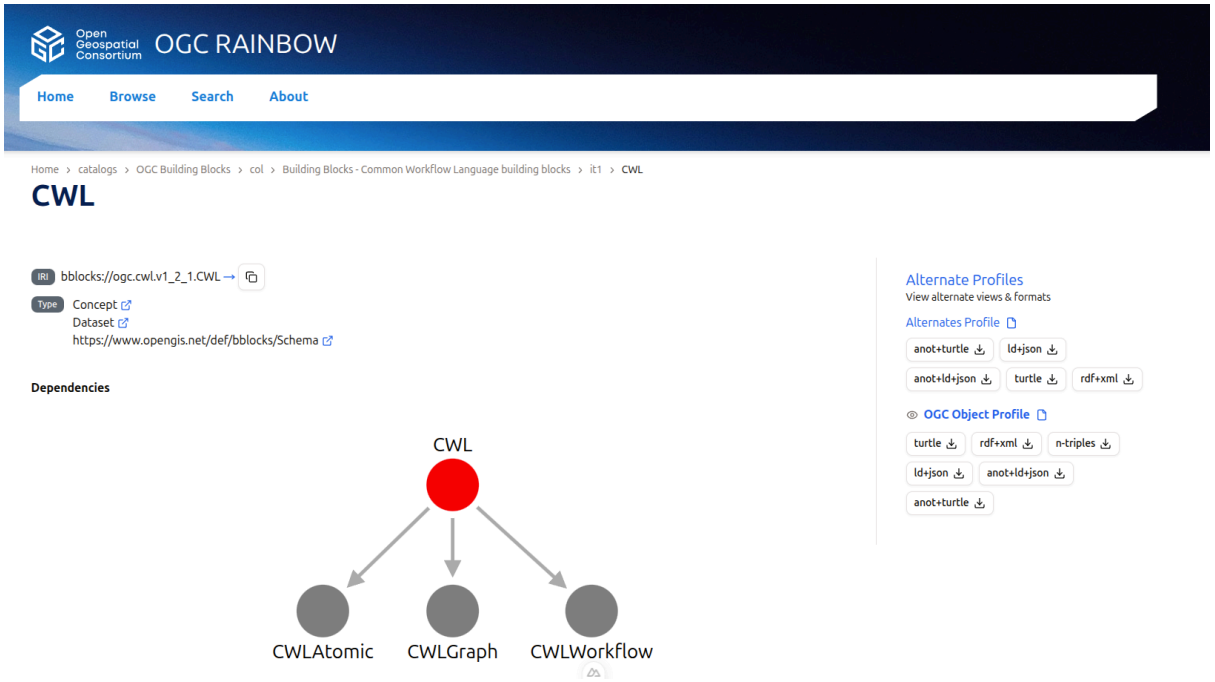

Figure D.2

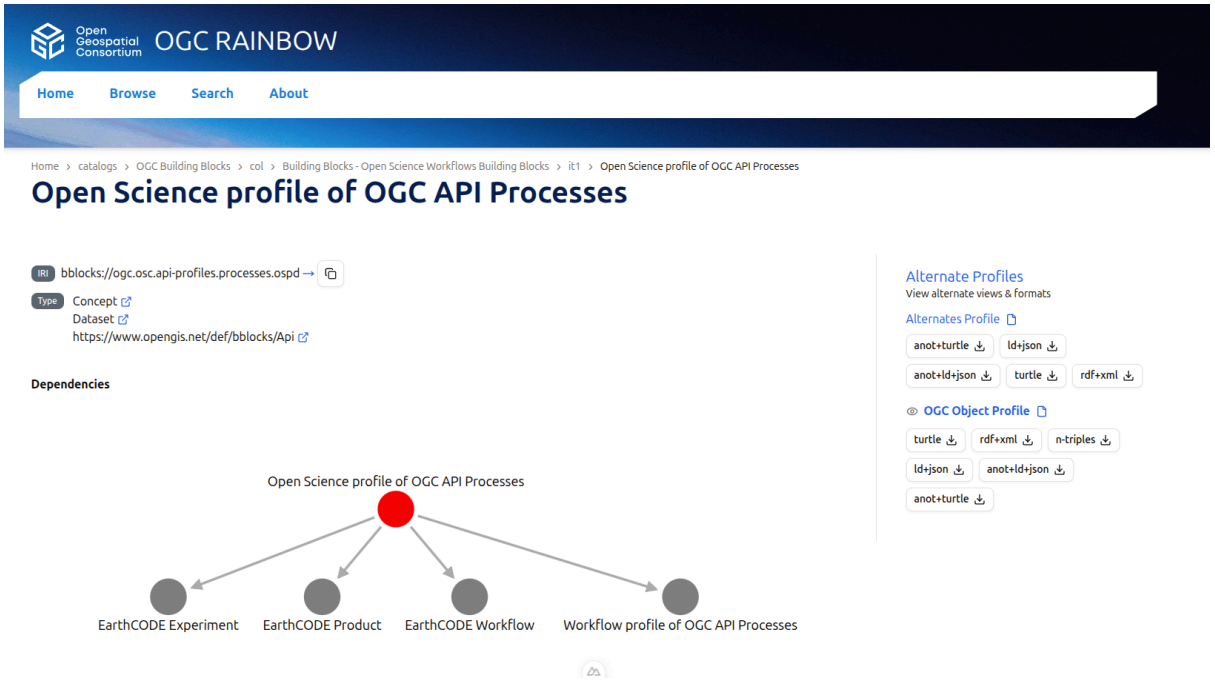

Figure D.3

3. For display of complex OSPD-specific objects (3.a — https://github.com/RDFLib/prez-ui/pull/253), the decision was made to focus on display of workflows and their provenance, as this will be an important part of the data and building blocks referenced by the scientists. When the OSPD workflows indicate inputs and outputs that are described with standardised metadata (e.g., W3C PROV), it will be possible to extend the total provenance "upstream" by reading the provenance

sections of the inputs. Similarly, a profile of these workflows could be flattened down into the metadata records of the Outputs.

In ongoing work, we are exploring on how to best visualize the workflows and their provenance, and what visualization widgets work best. For the OSPD project, we focused on a Sankey-diagram-based widget, which follows the prov:wasDerivedFrom trail for any resource in the OGC Rainbow:

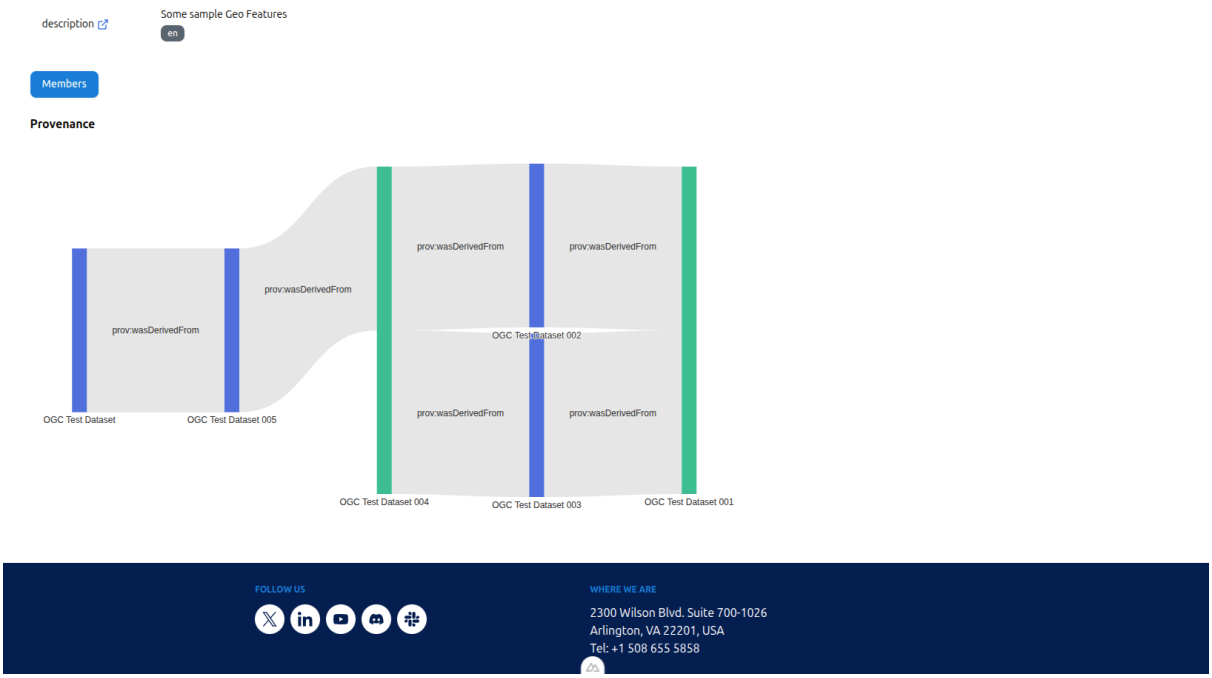

Figure D.4

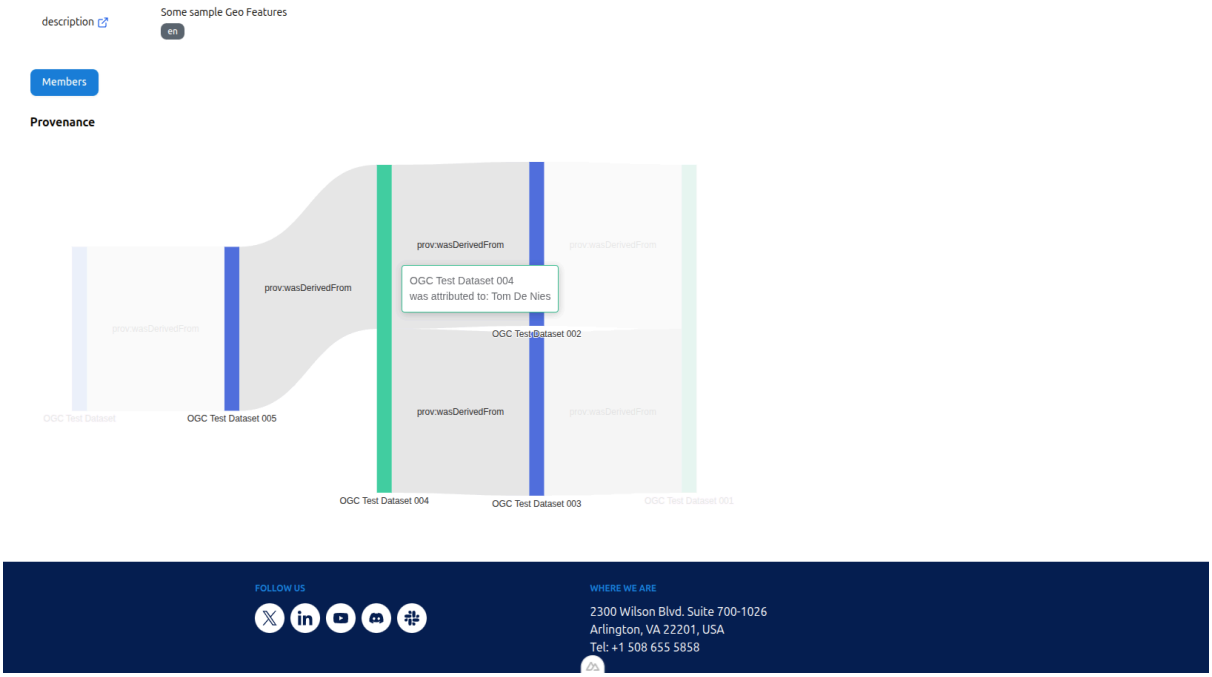

Figure D.5

With this work done and ongoing, the OGC Rainbow will soon be able to display the ontologies in use for these workflows and perhaps also instances of workflows. It may provide links to data directly or indirectly via catalogued resources. As long as Plans are captured, we will enable people to re-run workflows, at least semi-manually, potentially on different systems.

The Rainbow will also be able to provide different *profiles* of workflows and of object's metadata. One profile for a workflow will be the complete WF4Ever content for re-running processes, another should be a provenance "Black Box" which is just the outputs of the workflow linked to the inputs, which may be all someone needs. We can calculate this directly from WF4Ever data as long as the data is valid according to our profile of WF4Ever which we can validate.

We will soon be completing a definition of requirements and a validator for our profile of WF4Ever within our set of OSPD Workflow Provenance resources which is prototyped here: https://github.com/Kurrawong/ospd-wf-prov/blob/main/resources/validators/ospd-wf4ever.ttl.

The PrezManifest tool (https://github.com/Kurrawong/prez-manifest) we use to load data into the Rainbow will soon be capable of inferring new data using queries or rules. When it is, we will be able to create the Black Box view from OSPD Profile of WF4Ever workflow instances using some form of the query we have created for this purpose here: https://github.com/Kurrawong/ospd-wf-prov/blob/main/resources/rules/wf4ever-bb.sparql .

These SPARQL queries may soon be converted to SHACL Rules, as per the developing specification here: https://www.w3.org/TR/shacl12-rules/.

E

ANNEX E (INFORMATIVE) STARLING FOUNDRIES: KINDGROVE BUILDING BLOCKS EXPERIENCE

ANNEX E (INFORMATIVE) STARLING FOUNDRIES: KINDGROVE BUILDING BLOCKS EXPERIENCE

E.1. Context

The KindGrove mangrove biomass workflow served as a practical testbed for evaluating OGC Building Blocks in a real-world Earth observation analysis context. The workflow was used by RMIT University as an example for their wf4ever workflow provenance ontology mapping (see Annex C), and was integrated with CS Group's AntFlow platform for EOAP execution.

This annex documents our experience applying Building Blocks from the perspective of a scientific workflow developer—what worked, what created friction, and what would make the ecosystem more effective.

E.2. Building Blocks Applied

E.2.1. STAC for Data Discovery

The workflow uses STAC APIs (via `pystac-client`) to query the AWS Earth Search catalog for Sentinel-2 L2A imagery. STAC's standardized query interface enabled:

- Spatial filtering using OGC bbox parameters;
- Temporal filtering with ISO 8601 datetime ranges;
- Property filtering (cloud cover percentage); and
- Direct asset access via signed URLs.

E.2.2. Bounding Box Schema

We adopted the OGC Application Package bounding box schema (`ogc.geo.common.data_types.bounding_box`) for Area of Interest definition. This standardized input format ensured compatibility with the CS Group AntFlow platform's dynamic UI generation.

E.2.3. CWL Workflow Packaging

The workflow is packaged as a CWL Application Package, enabling:

- Platform-independent execution;
- Declarative input/output specification; and
- Integration with multiple OSPD execution environments.

E.3. Friction Points and Lessons Learned

E.3.1. Ecosystem Adoption Curve

Early in the pilot, integrating our notebook-based workflow with the emerging Building Blocks infrastructure created significant friction. Documentation was sparse, patterns were still solidifying, and participants were solving similar problems independently.

This friction dissipated noticeably as the OGC Rainbow and Building Blocks concepts gained clearer adoption strategies. By pilot end, the path from working notebook to portable CWL package was well-established.

E.3.2. The Bounding Box Needs a Temporal Complement

The bbox Building Block handles spatial extent well, but nearly every STAC query also requires temporal bounds. In practice, **everyone is doing time bounding**—yet the Building Block is silent on this, leading to repeated miscommunications between parties.

When platform partners attempted to reproduce our workflow results, they requested clarification on the input parameters. The spatial extent was clear—it was standardized via bbox. But establishing the temporal component required multiple rounds of back-and-forth because it was never communicated by the function calls or captured in the results. The data's time bounds existed only in our heads and our notebook comments, not in any formal specification.

A companion `temporal_extent` Building Block (or bbox extension) specifying start/end datetime would eliminate this friction entirely. Some workflows want "all data ever" for a site; others want a specific window. Without a standard way to express this, each cross-party integration required custom negotiation that could have been avoided.

E.3.3. Shapes, Not Boxes: The Cloud Egress Problem

The bbox paradigm requires specifying a rectangle that **contains** the area of interest, not the area itself. For irregularly shaped study sites (coastlines, river deltas, protected area boundaries), this means:

- Requesting significantly more data than needed;
- Mentally tracking the "real" study area separate from the query extent; and
- Either accepting extra data or implementing a post-query trimming step.

We chose to accept the extra data, but the inefficiency compounds at scale. **Cloud egress costs are killing open geospatial.** If workflows could request arbitrary polygons rather than bounding boxes, the bandwidth and cost savings across the ecosystem would be enormous.

This is not a criticism of the bbox Building Block—rectangles are computationally efficient for spatial indexing. But a `geometry` parameter accepting GeoJSON polygons as an alternative query mode would be transformative for cost-sensitive workflows.

E.3.4. Scale-Independent Outputs Are Essential

Variable scene coverage in Earth observation data makes absolute metrics (total area, total carbon) incomparable across time periods. Early in development, we observed an apparent "3× increase in mangrove area"—entirely an artifact of different scene coverage percentages.

We recommend that workflow Building Blocks include guidance on:

- Always reporting coverage percentage alongside absolute values;
- Providing density metrics (per-hectare values) for temporal comparison; and
- Documenting minimum coverage requirements explicitly.

E.3.5. Distributed Execution Was Within Reach

The I-Guide platform could have been used in tandem with CS Group's AntFlow to split workflow execution across environments. Once workflows are CWL-defined, coordinating parallel execution via Dask is straightforward—not a major technical leap from where we ended.

This was initially planned for the demonstration but deprioritized due to time constraints. The Building Blocks infrastructure is ready for this; the limiting factor was pilot duration.

E.4. Recommendations for Future Building Blocks

1. **Temporal extent standardization**: Define a `temporal_extent` Building Block or bbox extension to complement spatial queries. Time bounding is universal in EO workflows but currently unstandardized.
2. **Geometry query support**: Enable polygon-based queries as an alternative to bbox. The efficiency gains for irregular study areas would significantly reduce cloud egress costs.
3. **Coverage metadata**: Standardize output metadata for scene/AOI coverage percentage. This is critical for reproducibility and fair temporal comparison.
4. **Scale-independent defaults**: Encourage workflows to output normalized metrics (density, fraction) alongside absolute values.
5. **Reactive notebook patterns**: Consider Building Block patterns for interactive notebook environments (Marimo, Jupyter) alongside batch CWL workflows. Many scientific workflows begin as exploratory notebooks before formalization.

E.5. References

- KindGrove source: https://github.com/starling-foundries/KindGrove
- RMIT wf4ever provenance mapping: Annex C
- CS Group AntFlow integration: See OGC 25-043 CS Group annex