OGC Testbed-16

Machine Learning Engineering Report

Publication Date: 2021-02-15

Approval Date: 2021-02-10

Submission Date: 2020-11-20

Reference number of this document: OGC 20-015r2

Reference URL for this document: http://www.opengis.net/doc/PER/t16-D015

Category: OGC Public Engineering Report

Editor: Panagiotis (Peter) A. Vretanos

Title: OGC Testbed-16: Machine Learning Engineering Report

OGC Public Engineering Report

LICENSE AGREEMENT

Permission is hereby granted by the Open Geospatial Consortium, ("Licensor"), free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD. THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER'S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR's sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications.

This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the

Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

Table of Contents

Chapter 1. Subject

This engineering report describes the work performed in the Machine Learning Thread of OGC's Testbed-16 initiative.

Previous OGC testbed tasks concerned with Machine Learning (ML) concentrated on the methods and apparatus of training models to produce high quality results. The work reported in this ER, however, focuses less on the accuracy of machine models and more on how the entire machine learning processing chain from discovering training data to visualizing the results of a ML model run can be integrated into a standards-based data infrastructure specifically based on OGC interface standards.

The work performed in this thread consisted of:

1. Training ML models;
2. Deploying trained ML models;
3. Making deployed ML models discoverable;
4. Executing an ML model;
5. Publishing the results from executing a ML model;
6. Visualizing the results from running a ML model.

At each step, the following OGC and related standards were integrated into the workflow to provide an infrastructure upon which the above activities were performed:

1. OGC API - Features: Approved OGC Standard that provides API building blocks to create, retrieve, modify and query features on the Web.
2. OGC API - Coverages: Draft OGC Standard that provides API building blocks to create, retrieve, modify and query coverages on the Web.
3. OGC API - Records: Draft OGC Standard that provides API building block to create, modify and query catalogues on the Web.
4. Application Deployment and Execution Service: Draft OGC Standard that provides API building blocks to deploy, execute and retrieve results of processes on the Web.
    - MapML is a specification that was published by the Maps For HTML Community Group [https://www.w3.org/community/maps4html/]. It extends the base HTML map element to handle the display and editing of interactive geographic maps and map data without the need of special plugins or JavaScript libraries. The Design of MapML [https://www.w3.org/community/maps4html/2019/12/09/the-design-of-mapml/] resolves a Web Platform gap by combining map and map data semantics into a hypermedia format that is syntactically and architecturally compatible with and derived from HTML. It provides a standardized way for declarative HTML content to communicate with custom spatial server software (which currently use HTTP APIs based on multiple queries and responses). It allows map and map data semantics to be either included in HTML directly, or referred to at arbitrary URLs that describe stand-alone layers of map content, including hyper-linked annotations to further content.

Particular emphasis was placed on using services based on the emerging OGC API Framework suite

of API building blocks.

NOTE	This ER does not cover the specific details concerning the discovery and reusability of training data sets. A complete description of this topic can be found in the D016 Machine Learning Training Data Engineering Report [https://docs.ogc.org/per/20-018.html].

Chapter 2. Executive Summary

2.1. Business statement

The integration of Machine Learning (ML) tools into a framework composed of catalogues, data access services and data processing services that comply with OGC standards can result in a ML processing chain that can extract knowledge and insight from the vast amount of geospatial data being collected and deployed in cloud platforms.

2.2. Goals

The OGC Testbed-16 goals for the Machine Learning Thread are:

1. Discovery and reusability of data used to train predictive ML models.
2. The integration of predictive ML models into a standards-based data infrastructure.
3. Cost-effective visualization and data exploration technologies based on the use of the Map Markup Language (MapML).

2.3. Scenario / Use-cases

These goals are explored and addressed using the backdrop of a wildland fires scenario.

Wildland fires are those that occur in forests, shrublands and grasslands. While representing a natural component of forest ecosystems, wildland fires can present risks to human lives and infrastructure. Being able to properly plan for and respond to wildland fire events is thus a critical component of forestry management and emergency response.

Appropriate responses to wildland fire events benefit from planning activities undertaken before events occur. ML presents a new opportunity to advance wildland fire planning using diverse sets of geospatial information such as satellite imagery, Light Detection and Ranging (LiDAR) data, land cover information and building footprints. As much of the required geospatial information is available using OGC standard interfaces and encodings, a requirement exists to understand how well these standards can support ML in the context of wildland fire planning. Testbed-16 explored how to leverage ML, cloud deployment and execution, and geospatial information, provided through OGC standards, to improve planning approaches for wildland fire events. Findings inform future improvement and/or development activities for OGC Standards, leading to improved potential for the use of OGC standards within an infrastructure that includes ML applications.

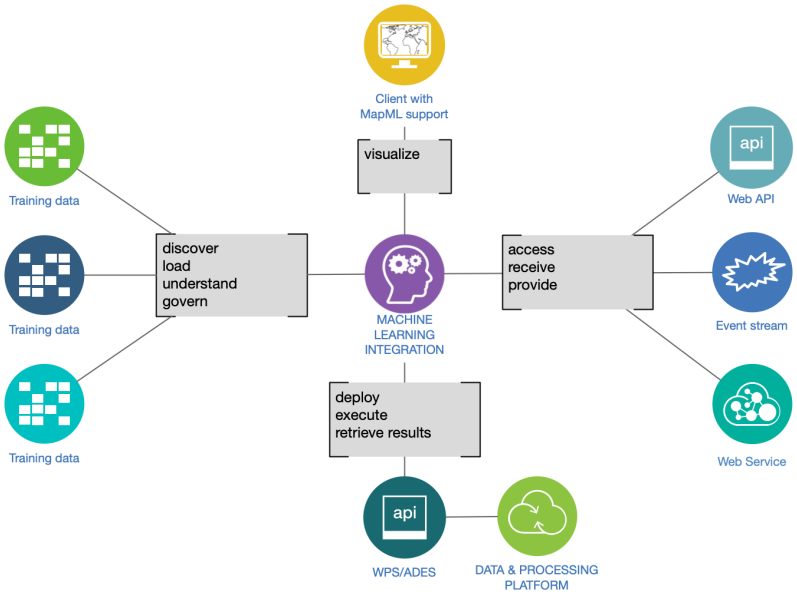


Figure 1. ML and EO integration challenges

Advanced planning for wildland fire events can greatly improve the ability of first responders to address a situation. However, accounting for the many variables (e.g. wind, dryness, fuel loads) and their combinations that will be present at the exact time of an event is very difficult. As such, there is an opportunity to evaluate how ML approaches, combined with geospatial information delivered using OGC Standards, can improve response planning throughout the duration and aftermath of wildland fire occurrences.

Thus, in addition to planning related work, Testbed-16 explored how to leverage ML technologies for dynamic wildland fire response. The planned work provided insight into how OGC Standards can support wildland fire response activities in a dynamic context. Any identified limitations of existing OGC Standards were documented and will be used to plan improvements to these frameworks. The Testbed was also an opportunity to explore how OGC Standards may be able to support the upcoming Canadian WildFireSat mission.

IMPORTANT	Though this task uses a wildland fire scenario, the emphasis is not on the quality of the modelled results, but on the integration of externally provided source and training data, the deployment of the ML model on remote clouds through a standardized interface, and the visualization of model output.

2.4. Research questions

This ER addresses the following research questions:

- Does ML require "data interoperability"? Or can ML enable "data interoperability"?
- How do existing and emerging OGC Standards contribute to a data architecture flow towards "data interoperability"?
- Is it necessary to have analysis ready data (ARD) for ML? Can ML help ARD development?
- What is the value of datacubes for ML?
- How do we address interoperability of distributed datacubes maintained by different organizations?

- What is the potential of MapML in the context of ML? Where does it need to be enhanced?
- How to discover and run an existing ML model?

2.5. Primary findings

The answers to the research questions can be found in the Research questions section. The following additional findings of the ML thread in OGC Testbed-16 are noted:

- The use of catalogues, data access services and data processing services that comply with OGC standards facilitates the modular deployment and expandability of ML processing chains.
- While the older OGC W*S services (e.g. WMS [https://www.ogc.org/standards/wms], WMTS [https://www.ogc.org/standards/wmts], WFS [https://www.ogc.org/standards/wfs], CSW [https://www.ogc.org/standards/cat], etc.) can be suitably integrated into the processing chain, the newer OGC API [https://ogcapi.ogc.org/] interfaces are easier to use and easier to integrate into a ML processing chains.
- The OGC API - Tiles [https://github.com/opengeospatial/OGC-API-Tiles] interface is a good candidate for model training as both value datasets and label datasets can be retrieved.
- The older WMS [https://www.ogc.org/standards/wms] interface can also be used for model training but relies on the existence of optional capabilities (i.e. LegendURL and GetStyles) in the service instances being used.
- The OGC API - Records [https://htmlpreview.github.io/?https://github.com/opengeospatial/ogcapi-records/blob/master/20-004.html] interface offered a high level of flexibility, allowing both the discovery of training datasets and providing the binding information necessary for data extraction by a ML algorithm. The catalogue is capable of harvesting repositories of different typologies and to list the relevant information for ML applications. This is an emerging OGC standard that can potentially contribute to a data architecture flow towards "data interoperability".
- Using Docker [https://www.docker.com/] to encapsulate both trained ML models as well as the entire processing chain facilitated testing the processing chain, as well as scaling it in production.
- The use of Application Deployment and Execution Service (ADES) and Execution Management Service (EMS) allows the dynamic deployment of trained models encapsulated in Docker [https://www.docker.com/] containers and provides a consistent interface for executing ML models and retrieving the results of processing. Those results can be persistently stored for use by downstream actors.
- There are plenty of development libraries available that implement support for OGC standards. Conveniently for the ML domain, numerous Python libraries with OGC standards support exist.

2.6. Future work

The following future-work items where identified:

- **Data Authenticity**: This aspect needs to be investigated in order to be sure that the model is trained and inferred with authentic data (the issue of data tampering in satellite imagery was also noted).

- **Analysis Ready Data (ARD)**: Another important aspect to take into consideration when the framework deals with different data sources like datacubes where some data could be already in ARD format and some other not.
- **ONNX check points**: for the time being the actual model stores its checkpoints in native format, but it could be useful to take into consideration the Open Neural Network eXchange (ONNX) [https://github.com/onnx] format.
- **Training dismissal**: another important aspect to be covered is the expected behavior in case it is required to interrupt the training. For instance, all the intermediate training has to be maintained or not?

Chapter 3. Standard and/or Domain Working Group review

3.1. Overview

The Machine Leaning (ML) thread participants and sponsors believed that the work of the ML task is relevant to work being done in the OGC Standards and Domain Working Groups (SWGs, DWGs) listed below. A request for review of this ER by the SWG/DWG members was forwarded to the working groups by the editor.

3.2. Artificial Intelligence in Geoinformatics (GeoAI) DWG

The Artificial Intelligence in Geoinformatics (GeoAI) DWG is chartered to identify use cases and applications related to Artificial Intelligence (AI) in geospatial domains and focused on the Internet-of-Things (e.g., healthcare, smart energy), robots (e.g., manufacturing, self-driving vehicles), or 'digital twins' (e.g., smart buildings and cities). This DWG provides an open forum for broad discussion and presentation of use cases with the purpose of bringing geoscientists, computer scientists, engineers, entrepreneurs, and decision makers from academia, industry, and government together to develop, share, and research the latest trends, successes, challenges, and opportunities in the field of AI with geospatial data. The working group aims to investigate the feasibility and interoperability of OGC standards in incorporating geospatial information with AI and describe gaps and issues which can lead to new geospatial standardization to advance trustworthiness and accountability for this domain community. Furthermore, existing OGC Web Services need to be carefully examined for changes that may need to be made in the context of AI-empowered applications. As some AI methods are already included in OGC standards, it is expected that AI methods will also impact many OGC standards in the future. For example, routing services have not yet been built according to human-centered AI, despite some suggestions to extend the Open Location Services (OpenLS) standard.

The goal of the ML task in Testbed-16 is to explore how to leverage ML through OGC standards to improve planning approaches for wildland fire events and this seems to align with the goals of a GeoAI DWG especially as it relates to incorporating and integrating geospatial information with AI.

3.3. Document contributor contact points

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

Contacts

Name	Organization	Role
Panagiotis (Peter) A. Vretanos	CubeWerx Inc.	Editor
Samuel Foucher	CRIM	Contributor

Name	Organization	Role
Francis Charette-Migneault	CRIM	Contributor
Matthes Rieke	52°North	Contributor
Andrea Cavallini	RHEA Group	Contributor
Nicola Lorusso	RHEA Group	Contributor
Valerio Fontana	RHEA Group	Contributor

3.4. Foreword

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.

Chapter 4. References

The following normative documents are referenced in this document.

- OGC 19-072, OGC® API - Common - Part 1: Core [http://docs.opengeospatial.org/DRAFTS/19-072.html]
- OGC 19-072, OGC® API - Common - Part 2: Geospatial Data [http://docs.opengeospatial.org/DRAFTS/20-042.html]
- OGC 18-058, OGC® API - Features - Part 1: Core [http://docs.opengeospatial.org/is/17-069r3/17-069r3.html]
- OGC 18-058, OGC® API - Features - Part 2: Coordinate Reference Systems by Reference [http://docs.opengeospatial.org/DRAFTS/18-058.html]
- OGC 19-079, OGC® API - Features - Part 3: Filtering and the Common Query Language [http://docs.opengeospatial.org/DRAFTS/19-079.html]
- OGC 20-004, OGC® API - Records - Part 1: Core [https://htmlpreview.github.io/?https://github.com/opengeospatial/ogcapi-records/blob/master/20-004.html]
- OGC 18-062, OGC® API - Processes - Part 1: Core [http://docs.opengeospatial.org/DRAFTS/18-062.html]
- OGC 20-057, OGC® API - Tiles - Part 1: Core [https://github.com/opengeospatial/OGC-API-Tiles]
- OGC 20-044, OGC® API - Processes - Part 2: Transactions [https://github.com/opengeospatial/wps-rest-binding/tree/master/extensions/transactions/standard]
- OGC Testbed-14: ADES & EMS Results and Best Practices Engineering Report [https://docs.opengeospatial.org/per/18-050r1.html]
- Map Markup Language [https://maps4html.org/MapML/spec/]

Chapter 5. Terms and definitions

For the purposes of this report, the definitions specified in Clause 4 of the OWS Common Implementation Standard OGC 06-121r9 [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2] shall apply. In addition, the following terms and definitions apply.

- **container**

  a software package that contains everything needed to run a program or application; this includes the executable program as well as system tools, libraries, and settings

- **convolutional neural network (CNN)**

  a class of deep neural networks commonly applied to analyzing visual imagery

- **deployment**

  a trained model available for execution behind a standardized API; in OGC this standardized API is defined by the OGC API - Processes specification [http://docs.opengeospatial.org/DRAFTS/18-062.html] with extensions [https://github.com/opengeospatial/wps-rest-binding/tree/master/extensions/transactions/standard].

- **Docker**

  a software platform for building applications based on containers

- **model inference**

  refers to the process of taking a machine learning model that has already been trained and using that trained model to make useful predictions based on new data

- **trained model**

  providing a machine learning algorithm with known data from which it can learn; the model artifact created by the training process is called a trained model

Chapter 6. Abbreviated terms

- ADES - Application Deployment and Execution System
- AI - Artificial Intelligence
- API - Application Programming Interface
- ARD - Analysis Ready Data
- CNN - Convolutional Neural Networks
- CRIM - Computer Research Institute of Montréal
- CRS - Coordinate Reference System
- CSW - Catalogue Service for the Web
- CWL - Common Workflow Language
- DL - Deep Learning
- ER - Engineering Report
- EMS - Execution Management System
- HTTP - Hypertext Transfer Protocol
- JSON - JavaScript Object Notation
- LiDAR - Light Detection and Ranging
- MapML - Map Markup Language
- ML - Machine Learning
- OGC - Open Geospatial Consortium
- ONNX - Open Neural Network Exchange Format
- OWS - OGC Web Services
- Pub/Sub - Publication/Subscription
- REST - Representational State Transfer
- RNN - Recurrent Neural Network
- SAR - Synthetic Aperture Radar
- TIE - Technology Integration Experiments
- URL - Uniform Resource Locator
- VCS - Version Control Systems
- WCS - Web Coverage Service
- WES - Web Enterprise Suite
- WFS - Web Feature Service
- WMS - Web Map Service
- WPS - Web Processing Service
- WPS-T - Transactional Web Processing Service

Chapter 7. Overview

The "Scenario" section provides a detailed description of the scenario and use cases used to drive the participants' implementations.

The "Infrastructure overview" section provides background information about ML and Deep Learning (DL) as well as some of the underlying OGC technologies such as the Application Deployment and Execution Service (ADES) which is an emerging technology developed in OGC Testbeds 13, 14 and 15.

The "Training, deployment and execution of machine learning models" section describes how machine learning models were deployed via existing and emerging OGC APIs in the ML task. Specifically the clause describes how ML models are packaged into containers and deployed via an ADES.

The "Visualization of ML Results" section describes how MapML was used to visualize and interact with geospatial information within a web browser.

The "Research questions" section attempt to answer research-questions [https://portal.ogc.org/files/?artifact_id=91644#_research_questions] originally expressed in the OGC Testbed-16: Call for Participation (CFP) [https://portal.ogc.org/files/?artifact_id=91644] based on the experiences of the thread participants.

The "Issues" section provides a summary of the issues discussed during the Testbed.

Chapter 8. Scenario

8.1. Overview

The Machine Learning task scenario addresses two phases of wildland fire management,

- Wildland Fire Planning
- Wildland Fire Response.

For both scenarios, various steps of training and analysis data integration, processing and visualization were performed as outlined below. The scenarios serve the purpose of guiding the activity through the various steps in the two phases of wildland fire planning and response. The scenarios help to ground all work in a real-world situation.

The Wildland fire planning scenario includes the following major steps:

1. Investigate the application of different ML frameworks (e.g. Mapbox RoboSat, Azavea's Raster Vision, GeoDeepLearning) to multiple types of remotely sensed information such as synthetic aperture radar, optical satellite imagery, and LiDAR. Access to these data sources was provided through OGC standards. The focus was to identify fuel availability within targeted forest regions.
2. Explore interoperability challenges related to ML training data. Develop solutions that allow the wildland fire training data, test data, and validation data be structured, described, generated, discovered, accessed, and curated within data infrastructures.
3. Explore the interoperability and reusability of trained ML models to determine potential for applications using different types of geospatial information. Interoperability, reusability and discoverability are essential elements for cost-efficient ML. The structure and content of the trained ML models have to provide information about its purpose. Questions such as: "What is the ML model trained to do?" or "What data was the model trained on?" or "Where is the model applicable?" need to be answered sufficiently in order to provide guidance on the appropriate use of a model. For example, models trained with data from a specific area that contains a specific features profile (e.g. forested land) may not be appropriate for use in another area with a different features profile (e.g. grassland). Interoperability of training data should be addressed equivalently.
4. Deep Learning (DL) architectures can use LiDAR to classify field objects (e.g. buildings, low vegetation, etc.). These architectures mainly use the TIFF and ASCII image formats. Other DL architectures use 3D data stored in a raster or voxel form. However, 3D voxels or raster forms may have many approximations that make classification and segmentation vulnerable to errors. Therefore, Testbed-16 participants should apply advanced DL architectures directly to the raw point cloud to classify points and segments of individual items (e.g. trees, etc.). The PointNET architecture for this or propose different approaches should be investigated. If different DL architectures are proposed, an alternative to PointNET could be considered.
5. Leverage outcomes from the previous steps to predict wildland fire behavior within a given area through ML. Incorporate training of ML using historical fire information and the Canadian Forest Fire Danger Rating system (fire weather index, fire behavior prediction) leveraging

weather, elevation models, fuels.

6. Using ML to discover and map suitably sized and shaped water bodies for water bombers and helicopters.
7. Investigate the use of ML to develop smoke forecasts based on weather conditions, elevation models, vegetation/fuel and active fires (size) based on distributed data sources and datacubes using OGC standards.

The Wildland fire response scenario includes the following major steps:

1. Explore ML methods for identifying active wildland fire locations through analysis of fire information data feeds (e.g. the Canadian Wildland Fire Information System, the United States Geological Survey LANDFIRE system) and aggregation methods. Explore the potential of MapML as an input to the ML process and the usefulness of a structured Web of geospatial data in this context.
2. Implement ML to identify potential risks to buildings and other infrastructure given identified fire locations. Consider the potential for estimating damage costs.
3. Investigate how existing standards related to water resources (e.g. WaterML, Common Hydrology Features (CHyF), in conjunction with ML, can be used to locate potential water sources for wildland fire event response.
4. Develop evacuation and first responder routes based on ML predictions of active fire behavior and real-time conditions (e.g. weather, environmental conditions).
5. Based on smoke forecasts and suitable water bodies, determine if suitable water bodies are accessible to water bombers and helicopters.
6. Explore the communication of evacuation and first responder routes, as well as other wildland fire information, through Publication/Subscription (Pub/Sub) messaging.
7. Examine how ML can be used to identify watersheds/water sources that will be more susceptible to degradation (e.g. flooding, erosion, poor water quality) after a fire has occurred.
8. Identify how OGC standards and ML may be able to support the goals of the upcoming Canadian WildFireSat mission.

8.2. Components

The following diagram provides an overview of the main work items for this task. The diagram is structured to show the training data at the bottom, existing platforms and corresponding APIs to the left, and Machine Learning models and visualization efforts to the right.

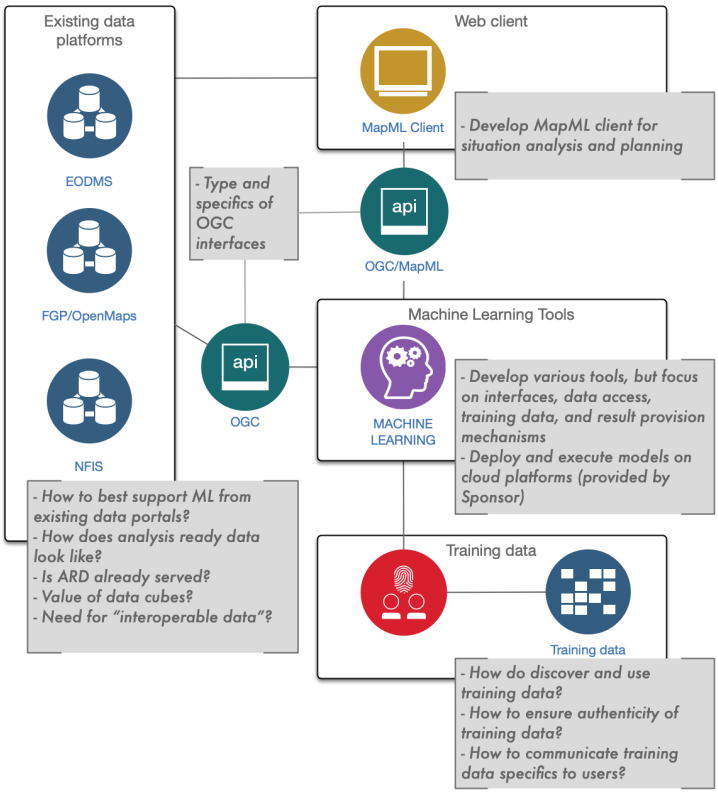


Figure 2. Major components and research aspects

The following overarching research questions further helped to guide the work in this task:

- Does ML require "data interoperability"?
  - Or can ML enable "data interoperability"?
  - How do existing and emerging OGC standards contribute to a data architecture flow towards "data interoperability"?
- Where do trained datasets go and how can they be re-used?
  - See the D016 Machine Learning Training Data ER [https://docs.ogc.org/per/20-018.html].
- How can we ensure the authenticity of trained datasets?
  - See the D016 Machine Learning Training Data ER [https://docs.ogc.org/per/20-018.html].
- Is it necessary to have analysis ready data (ARD) for ML? Can ML help ARD development?
  - For the purposes of serving the data from OGC API sources such as coverage server the data needs to be orthorectified
  - This is probably true for ML models as well
- What is the value of datacubes for ML?
- How do we address interoperability of distributed datacubes maintained by different organizations?
- What is the potential of MapML in the context of ML?
  - Where does it need to be enhanced?
- How to discover and run an existing ML model?

Chapter 9. Infrastructure overview

9.1. Introduction

A primary task of this thread was to explore the use of existing and emerging OGC APIs to enable a processing chain that starts with discovering training data for a specific purpose and ends with a deployed ML model that can be executed and its results visualized using a browser.

The following components diagram shows the interactions of the various ML and OGC components used in the thread:

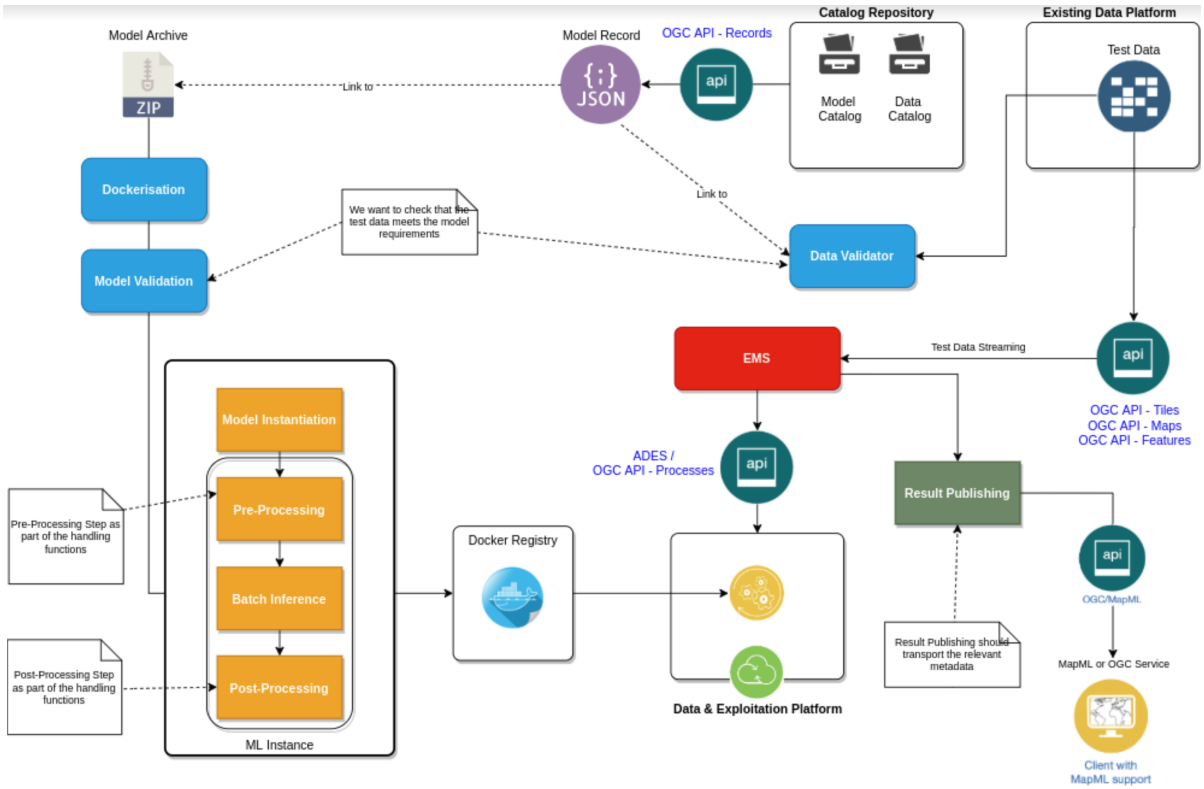


Figure 3. Detailed components diagram

This diagram covers all the components except those related to training data which are discussed in detail in the D016 Machine Learning Training Data ER [https://docs.ogc.org/per/20-018.html].

This section provides a brief overview of each of the OGC components that were integrated to create an infrastructure upon with the activities of the testbed where performed.

9.2. Functional description

A summary of Application Deployment and Execution Service (ADES) and Execution Management Service (EMS) can be found in the OGC Testbed-14: ADES & EMS Results and Best Practices Engineering Report (OGC 18-050r1) [https://docs.opengeospatial.org/per/18-050r1.html].

9.2.1. Dockerization

The left side of the diagram illustrates the bundling of an ML instance into a Docker container. This

Docker container is the execution unit for the ML instance once it is deployed to the ADES.

In order to make it discoverable, a catalogue record describing the details of the ML instance is created.

9.2.2. Deployment

The bottom center of the diagram illustrates an exploitation platform to which the Dockerized ML model is deployed. Once deployed, the model can be executed using the Application Deployment and Execution Service (ADES) and the Execution Management Service (EMS) APIs.

9.2.3. Discovery and execution

The center of the diagram illustrates an Execution Management Service,(EMS).

The EMS interacts with a catalogue, illustrated in the top right of the diagram, to provide discovery capabilities for ML models and data.

Once a suitable combination of the ML model and input data has been identified, the EMS interacts with the ADES to coordinate execution of the model.

9.2.4. Visualization

Once a ML model has been executed the EMS mediates handling of the results. This can involve passing the results (e.g. a GeoTIFF) directly to the client or publishing the results via OGC-based services (e.g. OGC API - Maps [https://github.com/opengeospatial/OGC-API-Maps] or OGC API - Tiles [https://github.com/opengeospatial/OGC-API-Tiles] servers) for later retrieval. In the latter case, technologies such as MapML can be leveraged to view the results and federate with other authoritative sources.

9.3. OGC Interfaces

9.3.1. Overview

For the Testbed ML activity, the preferred OGC interfaces are the those being defined for the OGC API framework. This section provides an overview of the specific APIs referenced in the Figure 3.

Two of the planned OGC APIs, such as OGC API - Features - Part 1:Core [http://docs.opengeospatial.org/is/17-069r3/17-069r3.html] and OGC API - Features - Part 2:Coordinate Reference Systems by Reference [http://docs.opengeospatial.org/is/18-058/18-058.html], are official OGC standards. Others, such as OGC API - Processes [https://github.com/opengeospatial/wps-rest-binding] or OGC API - Coverages [https://github.com/opengeospatial/ogc_api_coverages], will become OGC standards sometime in 2021. Finally, several of the APIs (e.g. OGC API - Common [https://github.com/opengeospatial/oapi_common], OGC API - Maps [https://github.com/opengeospatial/OGC-API-Maps], OGC API - Tiles [https://github.com/opengeospatial/OGC-API-Tiles]) are still under development within the OGC standards process.

9.3.2. OGC API Common

9.3.2.1. Overview

The OGC API framework is organized by resource type (e.g. features, coverages, maps, tiles). Each resource has an associated API standard. These resource-specific API standards are built using shared API modules. The OGC API-Common [https://github.com/opengeospatial/oapi_common] suite of standards stages the requirements and conformance that define these shared modules.

OGC API - Common - Part 1: Core [http://docs.opengeospatial.org/DRAFTS/19-072.html] defines the resources and operations which are be common to all OGC API standards. This draft Standard defines the minimal requirements for an API to be discovered and used by any client.

OGC API - Common - Part 2: Collections [http://docs.opengeospatial.org/DRAFTS/20-024.html] provides a common connection between the API landing page and resource-specific details for collections. That connection includes metadata which describes the collections of hosted resources, common parameters for selecting subsets of those collections, and URI templates for identifying the above.

9.3.2.2. API Summary

Table 1. OGC API - Common resources

Resource	URI	HTTP Method	Description
Landing Page	/	GET	The purpose of the landing page is to provide clients with a starting point for using the API. Any resource exposed through an API can be accessed by following paths or links starting from the landing page.
API definition	/api	GET	Every API should provide an API Definition resource which describes capabilities provided by that API. This resource can be used by developers to understand the API, by software clients to connect to the server, and by development tools to support the implementation of servers and clients.
Conformance	/conformance	GET	Provides a list of conformance classes implemented by an API.
Collections	/collections	GET	Information which describes the set of supported Collections.
Collection	/collections	GET	Information about a specific collection.

Table 2. OGC API - Common parameters

Parameter Name	Target	Description
Bounding Box	Extent	Selects resources which have an Extent element that intersects the bounding box
Date-Time	Extent	Selects resources which have an Extent element that intersects the specified time period
Limit	Result set	Limits the number of resources returned in a single response

9.3.3. OGC API - Features

9.3.3.1. Overview

The OGC API - Features [https://www.ogc.org/standards/ogcapi-features] Standard defines API building blocks to create, modify and query features on the Web. OGC API - Features [https://www.ogc.org/standards/ogcapi-features] is comprised of multiple parts, each of which is a separate standard.

OGC API - Features - Part 1: Core [http://docs.opengeospatial.org/DRAFTS/17-069r4.html] specifies the core capabilities and is restricted to fetching features where geometries are represented in the World Geodetic System 1984 coordinate reference system (WGS 84) with axis order longitude/latitude. The API further specifies discovery and query operations that are implemented using the HTTP GET method.

By default, every API implementing Part 1 will provide access to a single dataset. Rather than sharing the data as a complete dataset, OGC API Features offers direct, fine-grained access to the data at the feature (object) level.

Discovery operations defined in Part 1 enable clients to interrogate the API to determine its capabilities and retrieve information about this distribution of the dataset, including the API definition and metadata about the feature collections provided by the API.

Query operations defined in Part 1 enable clients to retrieve features from the underlying data store based upon simple selection criteria, defined by the client.

OGC API - Features - Part 2: Coordinate Reference Systems by Reference [http://docs.opengeospatial.org/DRAFTS/18-058.html] extends Part 1 to support additional coordinate reference systems in addition to WGS84 CRS specified in the core. This is an approved OGC standard.

OGC API - Features - Part 3: Filtering and the Common Query Language [http://docs.opengeospatial.org/DRAFTS/19-079.html] (CQL) extends Part 1 to support richer filtering capabilities beyond the simple selection capabilities defined in Part 1. This is a draft standard.

The API building blocks specified in the OGC API - Features suite of standards are consistent with the architecture of the Web. In particular, the API design is guided by the Internet Engineering Task Force (IETF) HTTP [https://tools.ietf.org/html/rfc7231]/https://tools.ietf.org/html/rfc2818[HTTPS] RFCs, the W3C Data on the Web Best Practices [https://www.w3.org/TR/dwbp/], the W3C/OGC Spatial Data on the Web Best Practices [https://www.w3.org/TR/sdw-bp/] and the emerging OGC Web API Guidelines [https://github.com/opengeospatial/OGC-Web-API-Guidelines]. A particular example is the use of the concepts

of datasets and dataset distributions as defined in the W3C Data Catalog Vocabulary (DCAT) [https://www.w3.org/TR/vocab-dcat-2/] and used in schema.org [https://schema.org].

In this thread, a features server was deployed to offer vector data for the training of ML models and as input data when executing trained models. Specifically, an OGC API Features server supporting both parts 2 and 3 was deployed to host the Canvec Series of Topographic Data of Canada [https://open.canada.ca/data/en/dataset/8ba2aa2a-7bb9-4448-b4d7-f164409fe056].

9.3.3.2. API Summary

Table 3. OGC API - Features resources

Resource	URI	HTTP Method	Description
Features	/collections/{collectionId}/items	GET	The collection of features
Feature	/collections/{collectionsId}/items/{featureId}	GET	A specific feature

9.3.4. OGC API - Coverages (Draft)

9.3.4.1. Overview

Coverages - as per OGC, and ISO standardization - constitute a unifying paradigm for the digital representations of space/time varying phenomena. Specifically, these are data defined as spatio-temporal regular and irregular grids, point clouds, and general meshes. In particular, multi-dimensional datacubes fall under this category, such as 1-D sensor time series, 2-D satellite imagery, 3-D x/y/t image time series and x/y/z geoscientific models, 4-D x/y/z/t climate and ocean data sets, and more.

The draft OGC API - Coverages [https://github.com/opengeospatial/ogc_api_coverages] standard establishes how to access coverages. Like OGC API Features [https://www.ogc.org/standards/ogcapi-features], OGC API - Coverages [https://github.com/opengeospatial/ogc_api_coverages] is comprised of multiple parts, each of them as a separate standard.

The draft OGC API - Coverages - Part 1: Core [http://docs.opengeospatial.org/DRAFTS/19-072.html], established how to access coverages as defined by the Coverage Implementation Schema (CIS) 1.1 [http://docs.opengeospatial.org/is/09-146r6/09-146r6.html].

Part 1 defines coverage extraction and includes subsetting capabilities, including band subsetting, scaling, CRS conversion and data format encoding. Gridded coverages or 'point-cloud' multi-point coverages are the primary data types considered but other kinds of datasets are not excluded.

In this Testbed thread, a coverage server was deployed serving Sentinel S1A and S1B satellite imagery from Canada. This data was used for training ML models and as input when executing trained models.

9.3.4.2. API Summary

Table 4. OGC API - Features resources

Resource	URI	HTTP Method	Description
Coverage	/collections/{coverageId}/coverage	GET	Returns a coverage and all its components
Rangeset	/collections/{coverageId}/coverage/rangeset	GET	Returns the rangeset of the coverage
Rangetype	/collections/{coverageId}/coverage/rangetype	GET	Returns the rangetype of the coverage
Features	/collections/{coverageId}/coverage/domainset	GET	Returns the domainset of the coverage
Metadata	/collections/{coverageId}/coverage/metadata	GET	Returns associated metadata defined by the CIS [http://docs.opengeospatial.org/is/09-146r6/09-146r6.html] model

9.3.5. OGC API - Maps and OGC API - Tiles (Drafts)

OGC API - Tiles - Part 1: Core [https://github.com/opengeospatial/ogcapi-tiles] specifies the behavior of Web APIs that provide access to tiles of one geospatial data resource or more than one geospatial data resource. This standard defines how to discover which resources offered by the API can be retrieved as tiles, what are the tile matrix sets supported by the geospatial data resources, which are the limits of the tiled space and how to request one tile at a time.

OGC API - Maps - Part 1: Core [https://github.com/opengeospatial/ogcapi-maps] describes an API that presents maps portraying data that has been rendered according to a style. The maps served by implementations of the draft OGC API - Maps Standard are retrieved as images of any size, generated on-the-fly, representing a geospatial extent determined by the client application, and with the styling also determined by the client application. When used together with OGC API - Tiles, the two APIs enable support for map tiles.

In this thread, a server was deployed that offered map and tile access OGC API interfaces to Sentinel S1A and S1B satellite imagery from Canada and to the Canvec Series of Topographic Data of Canada [https://open.canada.ca/data/en/dataset/8ba2aa2a-7bb9-4448-b4d7-f164409fe056]. Both the Sentinel and Canvec data were used to train ML model and as input data when executing trained models.

9.3.5.1. API Summary

Table 5. OGC API - Tiles resources

Resource name	Common path
Tiling Schemas	`/tileMatrixSets`
Tiling Schema	`/tileMatrixSets/{tileMatrixSetId}`
Tiles	

Resource name	Common path
Vector Tiles description	`/collections/{collectionId}/tiles`
Vector Tiles description in one tile matrix set	`/collections/{collectionId}/tiles/{tileMatrixSetId}`
Vector Tile	`/collections/{collectionId}/tiles/{tileMatrixSetId}/{tileMatrix}/{tileRow}/{tileCol}`
Vector Tiles description (geospatial resources[1])	`/tiles`
Vector Tile	`/collections/{collectionId}/tiles/{tileMatrixSetId}/{tileMatrix}/{tileRow}/{tileCol}`
Vector tile (geospatial resources[1])	`/tiles/{tileMatrixSetId}/{tileMatrix}/{tileRow}/{tileCol}`
Maps	
Maps description[2]	`/collections/{collectionId}/map`
Maps description (geospatial resources[1])[2]	`/map`
Map tiles	
Map tiles description	`/collections/{collectionId}/map/tiles`
Map tiles description in one tile matrix set	`/collections/{collectionId}/map/tiles/{tileMatrixSetId}`
Map tiles description (geospatial resources[1])	`/map/tiles`
Map tiles description (geospatial resources[1]) in one tile matrix set	`/map/tiles/{tileMatrixSetId}`
Map tile	`/collections/{collectionId}/map/{styleId}/tiles/{tileMatrixSetId}/{tileMatrix}/{tileRow}/{tileCol}`
Map tile (geospatial resources[1])	`/map/tiles/{tileMatrixSetId}/{tileMatrix}/{tileRow}/{tileCol}`

[1]: The expression "geospatial resources" means "from more than one geospatial resource or collection" [2]: Specified in the draft OGC API - Maps – Part 1: Core specification.

9.3.6. OGC API - Processes / Application Deployment and Execution Service (Draft)

9.3.6.1. Overview

OGC API - Processes - Part 1: Core [https://htmlpreview.github.io/?https://github.com/opengeospatial/wps-rest-binding/blob/master/docs/18-062.html], specifies building blocks that enable the execution of geospatial computing processes on the Web. The draft standard defines requirements and conformance for providing process inputs and for retrieving the results of processing. Processes can be executed synchronously or asynchronously. The API also defines discovery resources for the retrieval of metadata describing the purpose and functionality of each process. Typically, these processes

combine raster, vector, coverage and/or point cloud data with well-defined algorithms to produce new raster, vector, coverage and/or point could information.

Part 1 of OGC API - Processes [https://github.com/opengeospatial/wps-rest-binding] only offers a static set of processes. Previous OGC Testbed work in OGC Testbeds 13 [http://docs.opengeospatial.org/per/17-024.html] and 14 [https://docs.opengeospatial.org/per/18-050r1.html] extended the processes API to allow for the dynamic deployment of processes. This extension is currently referred to as the Application Deployment and Execution Service.

The draft OGC API - Processes - Part 2: Transactions [https://github.com/opengeospatial/wps-rest-binding/tree/master/extensions/transactions] specification formalizes the work done in testbeds 13, 14 and 15 to extend the core to add transactional capability; that is the ability to dynamically deploy and "undeploy" processes.

9.3.6.2. Application Deployment and Execution Service (ADES)

A service that implements these two parts of the OGC API - Processes [https://github.com/opengeospatial/wps-rest-binding] suite of standards is referred to as an "Application Deployment and Execution Service".

The ADES allows clients to deploy processes bundled in Docker containers and manages the environment required to execute those containers. Typically, this requires that an ADES understand:

1. The data that is available for processing and the interfaces required to access that data.
2. Initialize the execution environment for the Docker container.

The first item requires that the ADES understands how to access data and stage it for use by an application in a Docker container. This may require that the ADES know how to retrieve data from an OGC API such as OGC API - Features [https://www.ogc.org/standards/ogcapi-features] or it may require that the ADES know how to read data from other APIs such as Amazon's S3 [https://aws.amazon.com/s3/] object store.

The second item requires that the ADES understand how to interact with the specific platform upon which it is deployed in order to stage data for the Dockerized application and execute the Docker container. For example, an ADES deployed to an Amazon Cloud needs to understand how to interact with AWS's Elastic Container Service (ECS) [https://aws.amazon.com/ecs] or Elastic Kubernetes Service (EKS) [https://aws.amazon.com/eks] in order to execute Docker containers at scale. Similar statements are applicable for ADES' deployed on the Google Cloud Platform [https://cloud.google.com/gcp] or on OpenStack [https://www.openstack.org/]-based cloud such as the Boreal Cloud of Natural Resources Canada (NRCan).

9.3.6.3. Execution Management Service (EMS)

The Execution Management Service (EMS) is basically an ADES that does not actually "execute" processes but rather acts as a coordinating node in a federation of ADESs. Typically, an EMS knows how to:

1. Coordinate the deployment of new processes across the federation.

2. Interact with catalogues to facilitate the discovery of data and processes required to perform a desired analysis.
3. Dispatch the execution of a process to an appropriate ADES node in the federation that offers the discovered data and processing capabilities.
4. Coordinate the execution of a workflow composed of a number of processing steps across the federation.
5. Combine or aggregate the results of each processing step to present the final result of executing a workflow to the client.

So, even though the API presented by the EMS is the same as that presented by the ADES the role of each component is very different. It should be noted, however, that many implementations can be deployed for fulfill either or both roles (i.e. EMS or ADES).

9.3.6.4. API Summary

The following API summary also includes elements of the OGC API - Processes transactional extension [https://github.com/opengeospatial/wps-rest-binding/tree/master/extensions/transactions] which allows processes to be dynamically deployed and undeployed thus extending the API.

Table 6. OGC API - Features resources

Resource	URI	HTTP Method	Description
Processes	/processes	GET	List of available processes
		POST	Deploy a new process
Process	/processes/{processId}	GET	Get a description of a specific process
		PUT	Update a deployed job
		DELETE	Undeploy a job
List of jobs	/processes/{processId}/jobs	GET	Get a list of jobs for the specified process
Job status	/processes/{processId}/jobs/{jobId}	GET	Get the status of the specified job
Job results	/processes/{processId}/jobs/{jobId}/results	GET	Get the results for the specified job

NOTE Although this clause focuses primarily on describing the upcoming OGC API interface for the ADES/EMS, the previous version of these servers — based on the current Web Processing Service [https://www.ogc.org/standards/wps] standard — was also used in the Testbed-16 ML Thread.

9.3.7. OGC API - Records (Draft)

9.3.7.1. Overview

OGC API - Records - Part 1: Core [https://htmlpreview.github.io/?https://github.com/opengeospatial/ogcapi-records/blob/master/20-004.html] defines an API that supports the ability to search collections of descriptive information, called records, about resources such as data collections, services, processes, styles, code lists and other related resources. Records represent resource characteristics that can be queried and presented for evaluation and further processing by both humans and software.

Part 1 defines a set of core queryables that represent a list of mandatory and optional properties that a catalogue records should contain to provide a minimum, useful amount of information about the resource the record is describing.

The draft API for Part 1 is very similar to that for OGC API - Features - Part 1: Core [http://docs.opengeospatial.org/is/17-069r3/17-069r3.html] but also includes additional query parameters that extend the query capabilities of the API.

Part 1 also makes recommendations about the encoding of a record. GeoJSON [https://tools.ietf.org/html/rfc7946] is specified as the JSON encoding of a record. There are recommendations for XML and HTML encoding but these where not used in the ML Testbed activity.

9.3.7.2. API Summary

The following table lists mandatory (indicated by "M") and recommended optional ("O") properties that every catalogue record should include. This list of properties is referred to as the "core queryables".

Table 7. OGC API - Records queryables

Queryable	O/M	Description
recordId	M	A unique record identifier assigned by the catalogue.
recordcreated	M	The date the records was created in the catalogue.
recordmodified	M	The most recent date on which the record was changed.
title	M	A human-readable name given to the resource.
description	M	A free-text description of the resource.
keywords	M	A list of keywords or tags describing the resource.
type	M	The nature or genre of the resource.
language	O	This refers to the natural language used for textual values (i.e. titles, descriptions, etc) of a resource.
externalId	O	An identifier for the resource assigned by an external entity (i.e. not the catalogue).
modified	O	Most recent date on which the resource was changed.

Queryable	O/M	Description
publisher	O	The entity for making the resource available.
themes	O	A knowledge organization system used to classify the record.
formats	O	A list of available distribution formats for the resource.
contactPoint	O	An entity to contact about the resource.
license	O	A legal document under which the resource is made available.
rights	O	A statement that concerns all rights not addressed by the license, such as copyright statements.
extent	O	The spatio-temporal coverage of the resource.
links	O	A list of links for navigating the catalogue API.
associations	O	A list of links to resources associated with this resource.

The following table lists the set of query parameters that every implementation of the OGC API - Records draft standard must provide. The `bbox`, `datetime` and `limit` parameters are inherited from OGC API - Features - Part 1: Core [http://docs.opengeospatial.org/is/17-069r3/17-069r3.html] standard.

Table 8. OGC API - Records query parameters

Parameter name	Description
bbox	A bounding box. If the spatial extent of the record intersects the specified bounding box then the record shall be presented in the response document.
datetime	A time instance or time period. If the temporal extent of the record intersects the specified data/time value then the record shall be presented in the response document.
limit	The number of records to be presented in a response document.
type	A resource type. Only records of the specified type shall be presented in the response document.
q	A space-separated list of search terms. If any server-chosen text field in the record contains 1 or more of the terms listed then this records hall appear in the response set.
externalIds	A comma-separated list of external identifiers. Only records with the specified names shall appear in the response document.

Chapter 10. Training, deployment and execution of machine learning models

10.1. Overview

The topic of training ML models is covered in the D016 Machine Learning Training Data ER [https://docs.ogc.org/per/20-018.html].

Once the model is trained the next step in the machine learning life-cycle is the deployment of the trained model. In the context of the Testbed-16 ML activity, deploying a trained model means that:

- The model is deployed behind a standards-based interface.
- A description of the model is published to a catalogue.

The first point hides the specific details of the model (algorithm, ML platform, programming language, etc.) behind a standards-based API. This allows the model to be consistently invoked with new input data and the results of the run to be retrieved or published in a manner that allows for later retrieval, ideally also through a standards-based API. In the Testbed activity the ADES was used as the API for deploying and invoking ML models.

The second point enables a user to search a catalogue for data and models that meet their needs and invoke the discovered model with the discovered input data. The goal was to use the draft OGC API - Records [https://htmlpreview.github.io/?https://github.com/opengeospatial/ogcapi-records/blob/master/20-004.html] interface as the discovery API.

This section describes the standards-based machine learning environments implemented by the Testbed-16 ML participants.

10.2. Machine Learning Environment 1 (D132 - 52°North)

52°North targeted the implementation of an ML environment that focused on the detection of water bodies using satellite radar data. The overall goal was to create a trained model that was ready for execution within a Docker container. The following figure illustrates the steps involved to prepare the model.

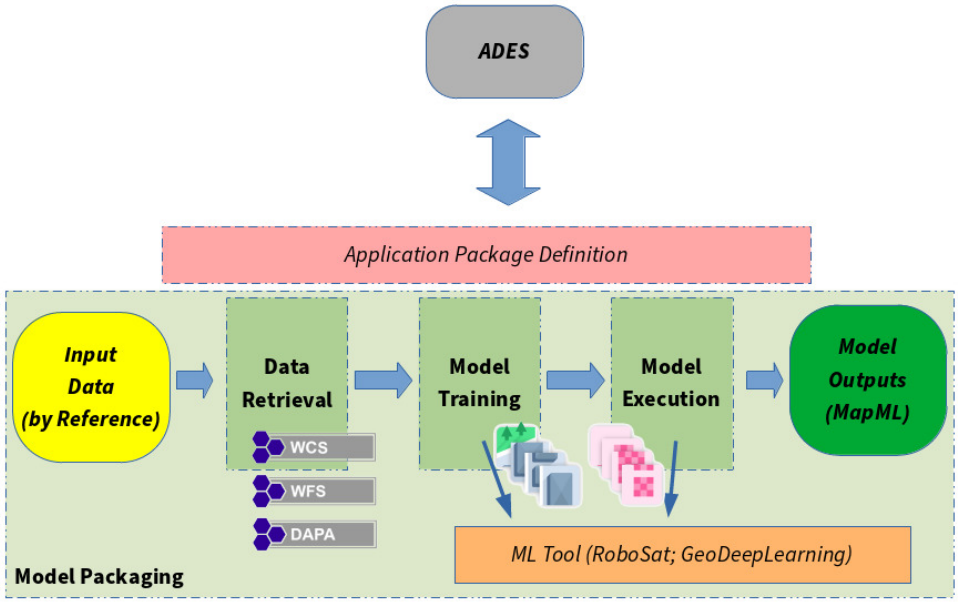


Figure 4. 52North Model Preparation

10.2.1. Use Cases

Since forest fires regularly occur in Canada, obtaining knowledge on water bodies that are candidates for serving water-bomber planes or helicopters is of high value. Many of the natural water bodies in the Canada backcountry vary in extent and water level. A ML model can provide insightful information using up-to-date radar measurements such as Synthetic Aperture Radar (SAR) data. Therefore, the goal was to train a model with historic SAR data and corresponding labels in order to apply the model to recent SAR data.

10.2.2. Data and Training Data Considerations

10.2.2.1. Radarsat-1

Early in the preparation phase of the model development, different data sources were taken into consideration. In particular, the Radarsat-1 data provided by NRCan was assessed with regards to its feasibility within the ML application. For pre-processing the Sentinel Application Platform [http://step.esa.int/main/toolboxes/snap/] (SNAP) built-in Radarsat importer was used. A set of issues arose in the course of this workflow:

- **File structure conventions**: Two different variants of Radarsat-1 data structures were identified. One variant was not supported by SNAP. The `*01f.sard` format (used from 2010 onward) could not be imported while the older format (`dat_01.001`, ...) could be loaded.
- **Geographic Inaccuracy**: after applying a default set of pre-processing steps (i.e. Speckle Filtering with "lee filter" and Terrain Correction using Ellipsoid Correction to Geogrid Location from the SNAP toolkit) the resulting raster data always featured an offset, varying in extremity. The below Figure illustrates the issue.

Figure 5. Radarsat-1 Offset after Pre-processing

After consideration with SAR experts at NRCan participants decided to opt for a Sentinel-1 based model approach as the data was very well supported in terms of pre-processing functionality and accuracy.

10.2.2.2. Training Data

Two training datasets where used: pre-processed Sentinel-1 scenes and water body data (the CanVec dataset "Lakes, Rivers and Glaciers in Canada - CanVec Series - Hydrographic Features"; see https://open.canada.ca/data/en/dataset/9d96e8c9-22fe-4ad2-b5e8-94a6991b744b). As described in section OGC API - Maps and OGC API - Tiles (Drafts), both the Sentinel-1 and the label data were provided by a prototype OGC API Tiles implementation. The below Figure illustrates the tile-based approach.

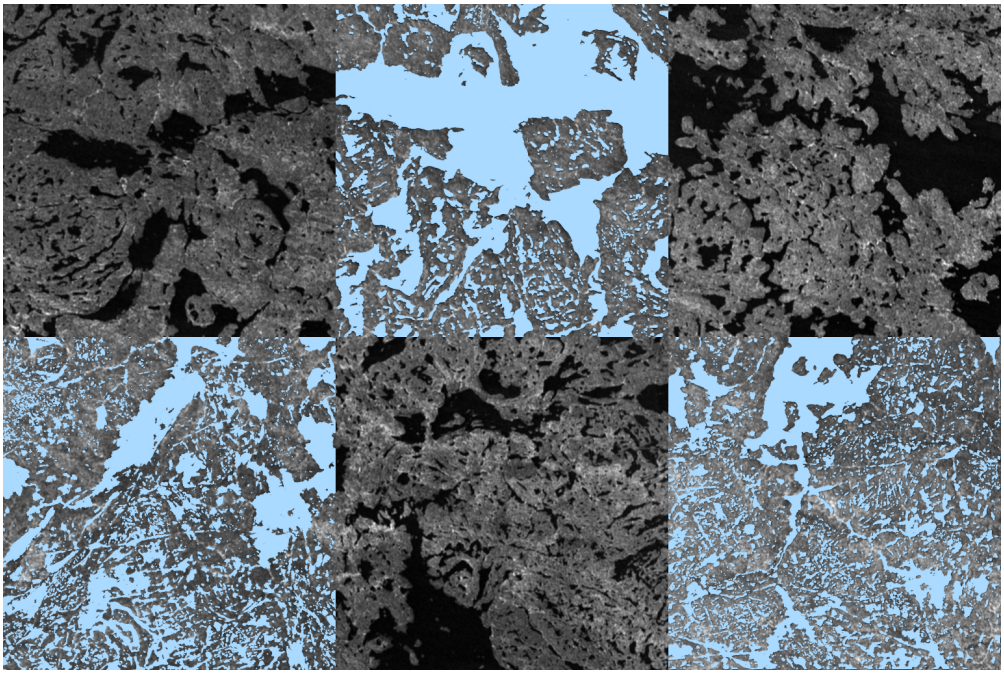


Figure 6. Tile-Based Approach

10.2.3. Model Training

A specialized convolutional neural network (CNN) for training the model was used. This was the U-Net CNN developed for biomedical image segmentation. U-Net provides very good performance using modern GPUs, applying a segmentation of 512x512 images (the size of a tile) in approximately one second.

A dedicated data retrieval process was used to download the tiles for a specific area of interest from the prototype OGC API - Tiles instance. The retrieval process was implemented using a Jupyter Notebook and is available at https://nbviewer.jupyter.org/github/52North/testbed16-jupyter-notebooks/blob/master/ml/tiles/tile-resolution.ipynb. The prototype OGC API - Tiles integration made the usage of training data seamless and straightforward. The amount of training data can be very easily be scaled by applying a larger area of interest. The only mandatory manual step was the identification of the optimal zoom level. This was due to Sentinel-1 data only providing limited spatial resolution.

The model was then executed using the set of downloaded tiles. The matching tiles of the Sentinel-1 and the water body label data share the same file name in different folders which allowed the application of the U-Net segmentation in an efficient manner.

10.2.4. Model Inference Results

The model trained with the tiles can then be applied to other Sentinel-1 scenes. The only prerequisite is that the scenes are pre-processed in the same way as the retrieved tiles were. Using GeoTIFF as the input format, an output raster (also GeoTIFF) with Boolean pixel values (1 = water body) can be created. An example is illustrated in the below Figure (base layer © OpenStreetMap contributors), where the red areas are the overlaid inference results.

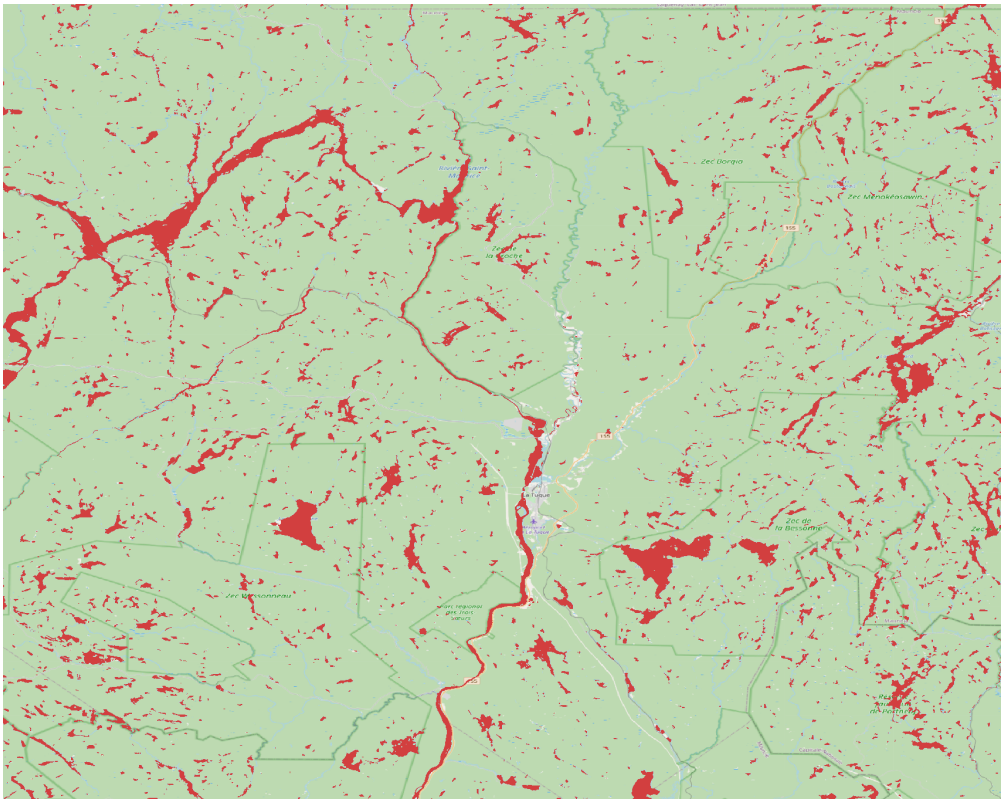


Figure 7. Example Model Inference Results

10.2.5. Execution and ADES Integration

The integration of the model into ADES follows the approach established during the former OGC Testbeds. In summary, a Common Workflow Language (CWL) definition is used as the interface between the model execution within a Docker container and the ADES. As the model has been manifested with the test data in the previous development steps, it has been integrated into a Docker image that allows a platform-independent execution.

The deployment of the model into the ADES followed the approach established in previous OGC Testbeds (see: OGC Testbed-15: Machine Learning Engineering Report [http://docs.opengeospatial.org/per/19-027r2.html] and OGC Testbed-14: Machine Learning Engineering Report [http://docs.opengeospatial.org/per/18-038r2.html]). Specifically,

1. The trained model was bundled into a Docker image that allows platform-independent execution.
2. A process description was created to define the interface of the model when it is invoked via the ADES.
3. A Common Workflow Language [https://www.commonwl.org/] (CWL) definition was created to specify, for the ADES, how to execute the model in the Docker container. It should be noted that the process description for executable, trained model via the ADES can be auto-generated from the CWL definition.

The CWL workflow definition used in this thread is provided below. An ADES can invoke the workflow using any number of CWL runner applications such as `cwltool` or `cwl-runner`.

```
#!/usr/bin/env cwl-runner
cwlVersion: v1.0
class: Workflow
inputs:
  inputScene: File
  model: string
outputs:
  water_mask_output:
    type: File
    streamable: false
    outputSource: execute/water_mask_output
steps:
  execute:
    in:
      inputScene: inputScene
      model: model
    out: [water_mask_output]
    run:
      class: CommandLineTool
      baseCommand: python3
      hints:
        DockerRequirement:
          dockerPull: 52north/tb16-s1ml-with-model:latest

      inputs:
        inputScene:
          type: File
        model:
          type: string
      arguments: ["/s1ml/cli/predict.py", "--input-path", $(inputs.inputScene),
          "--output-path", "water_mask_output.tif", "--model-path",
"/$(inputs.model).h5"]
      outputs:
        water_mask_output:
          type: File
          outputBinding:
            glob: water_mask_output.tif
```

The following listing illustrates a CWL job that can be used to execute the model:

```
inputScene:
  class: File
  path: s1_test2.tif
model: "api_tiles_model"
```

The Docker image used in the ML Testbed activity for the D132 component holds two independently trained models:

- api_tiles_model: A model trained with Sentinel 1 scenes provided via the prototype OGC API - Tiles (with one "backscatter" band) service.
- unet_dice_vvvh: A model trained with Sentinel 1 scenes pre-processed manually (with VV and VH bands).

The `model` parameter allows the selection of one of these two model.

The `inputScene` parameter references a raster scene to process and should match the pre-processing and data format of the referenced model (GeoTIFF in this case).

10.3. Machine Learning Environment 2 (D133 - RHEA)

10.3.1. Introduction

Since 1990, wildland fires across Canada have consumed an average of 2.5 million hectares a year. Even if these events represent a natural component of forest ecosystems maintaining forest health and diversity, this is a challenge to forest management entities. This is because the events at once can be potentially harmful (risky for human being, a threat to communities and infrastructures, can destroy vast amounts of timber resources resulting in costly losses) and beneficial (renewing and maintaining healthy ecosystems and enhancing ecological conditions and eliminating excessive fuel build-up). Beside natural fires and prescribed burns set by authorized forest managers, some uncontrolled wildfires are started by lightning or human carelessness. Being able to properly plan, control and respond to this very complex phenomenon is thus both a critical and vital component of forestry and emergency management in Canada. Artificial Intelligence (both Machine Learning and Deep Learning) presents a valuable opportunity thanks to multiple sources of geospatial information like satellite imagery and data coming from the Canadian Geospatial Data Infrastructure (CGDI).

Within the frame of the activities carried on in D133 Machine Learning Environment 2 deliverable, participants explored how to leverage ML technologies in dynamic context like wildland fire planning and response, exploiting the use of OGC Standards.

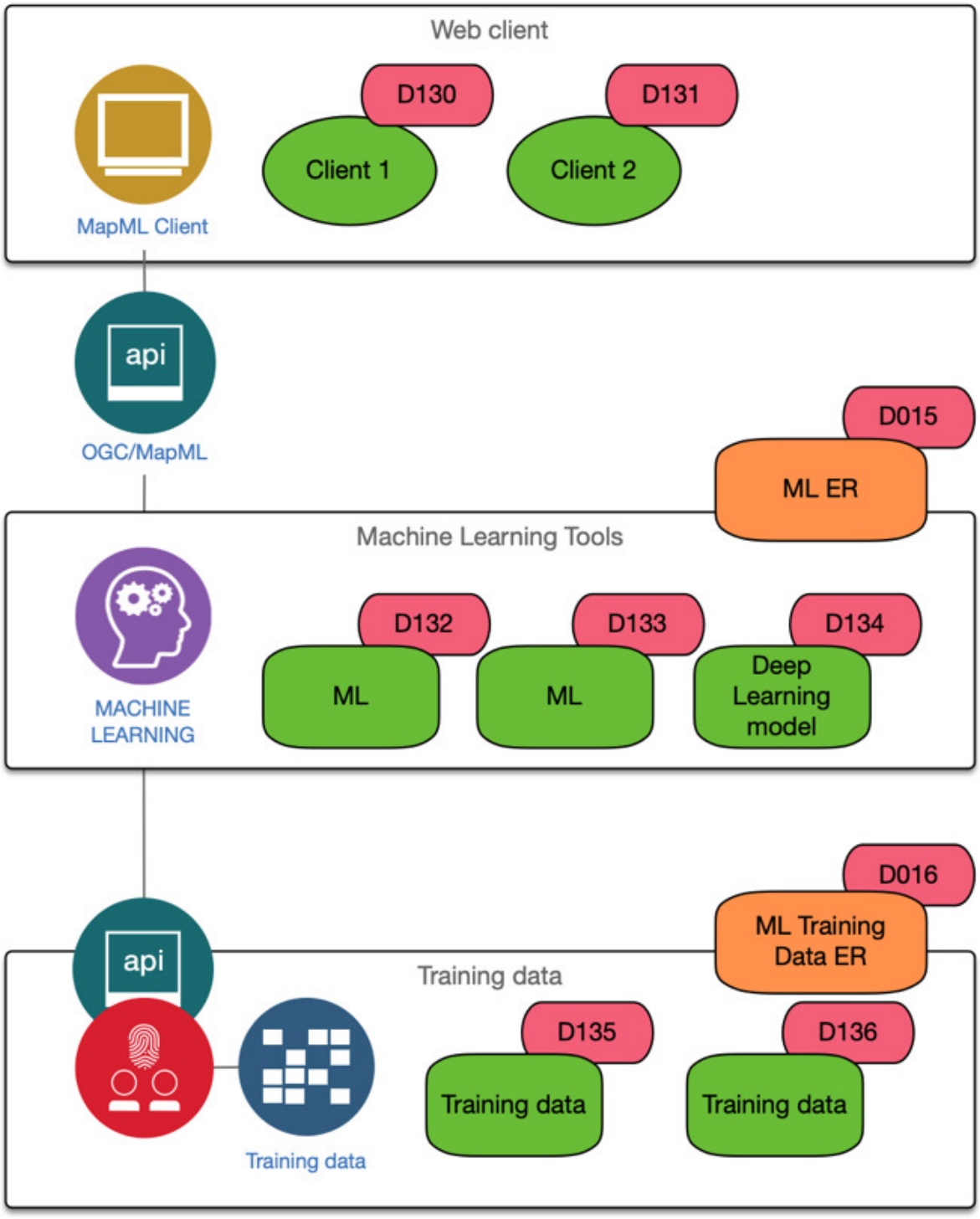


Figure 8. OGC Testbed-16 Context, deliverables and interfaces

The aim of the D133 deliverable was to provide a common framework for supporting Machine Learning (ML) applications by:

- Defining a standardized way to discover, deploy and run ML models on cloud platforms by means of ADES interface
- Defining a standardized way (both data and infrastructures) to discover, expose and use/re-use ML training datasets aimed to interoperability exploiting OGC Standards (such as the OGC API – Records for instance)
- Providing model results in a standardized way exploiting the power of Map Markup Language (MapML)

While the focus of the activity focused on the design and development of the framework, an ML model to estimate fuel loads from Sentinel-1 data was developed and deployed in the framework as a test case to show overall framework functionality and to provide real a sample use case.

10.3.2. Architecture

10.3.2.1. Overview

Figure 9 is shown the high-level architecture of the main components of the deliverable.

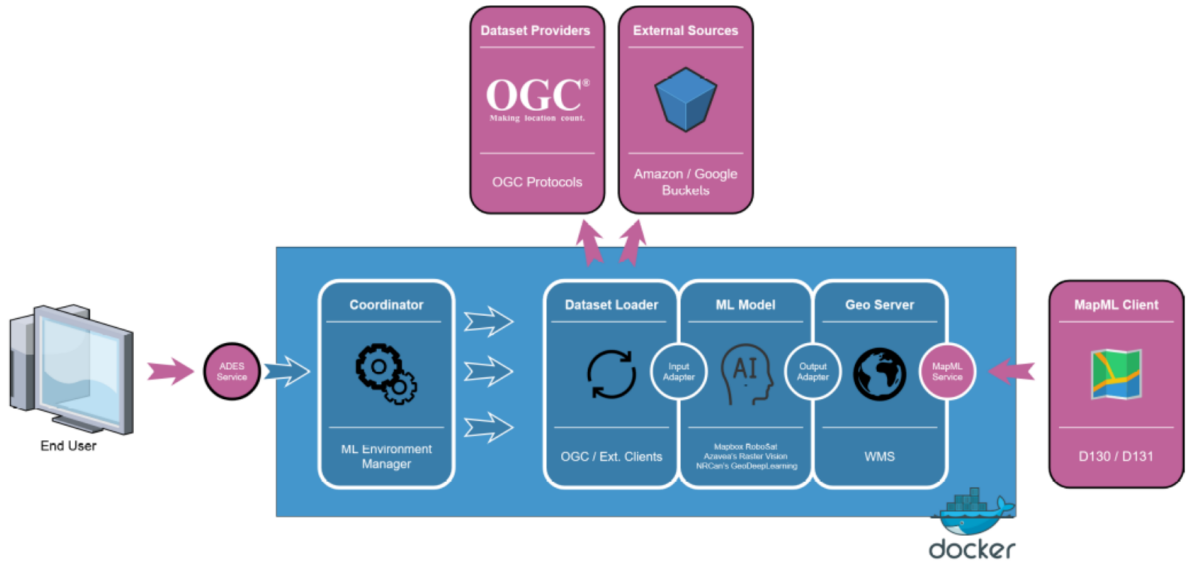


Figure 9. D133 High Level Architecture

The ML environment framework is composed of the following main components:

- **ADES Service**: The entry point of the framework exposing an OGC Standard interface aimed to browse, deploy and run services.
- **Coordinator**: The main component in charge of orchestrating the workflow within the framework calling all the other components and dealing with two distinct scenarios:
  - Training of ML model
  - Run inference on pre-trained ML model
- **Dataset Loader**: Responsible for discovery and collection of data using OGC Standards and other third-party protocols.
- **Input Adapter**: In charge of manipulating data retrieved by Dataset Loader converting it for specific ML Model contained in the environment
- **ML Model**: The core element of the framework
- **Output Adapter**: Symmetrically to the Input Adapter, it is in charge of adapting ML Model component result in order to be correctly processed and published by GeoServer [http://geoserver.org/]
- **GeoServer**: In charge of exposing model results to relevant MapML clients using an OGC Web Map Service [https://www.ogc.org/standards/wms] (WMS) Standard implementation.

The current ML framework is equipped with just one registered model aimed to predict fuel loads

from Sentinel-1 data and is encapsulated in a Docker container for quick and easy deployment mainly, but not only, into Cloud environments.

10.3.2.2. ADES Service

The ML environment framework exposes a dedicated Application Deployment and Execution Service (ADES) interface aimed at triggering, deploying and running a specific ML model from the models available. By means of ADES, it is possible to deliver the framework as an Infrastructure as a Service (IaaS) [https://en.wikipedia.org/wiki/Infrastructure_as_a_service] platform, managing its execution via a WPS interface. The ADES can perform the deployment of applications in the form of Docker containers and control their execution using the application definition provided by a Common Workflow Language [https://www.commonwl.org/] (CWL) configuration file. For the ML Thread activity two distinct processes were deployed to the ADES:

- Training of a ML model over a specific dataset
- Inferencing data using a pre-trained ML model

The workflow execution is handled by the Coordinator component.

10.3.2.3. Coordinator

The Coordinator component is in charge of orchestrating all the activities of the software delivered within the container. The coordinate calls in sequence:

- The **Data Loader** for the retrieval of both input data and ground truth in case of training or only data in case of inference
- The **Input Adapter** for model data preparation
- The **Model** either for training or inferences
- The **Output Adapter** for output data manipulation and publication (in case of inferencing) on the GeoServer [http://geoserver.org/]

10.3.2.4. Dataset Loader

The role of Dataset Loader is to discover and download data, as per requests coming from input parameters provided by the WPS input parameters. These will be used later for either training or inferencing the ML model. The scope of this component is to generalize the data access service and, potentially, cope with several data sources exposing different interfaces such as OGC CSW [https://www.ogc.org/standards/cat], WCS [https://www.ogc.org/standards/wcs], WMS [https://www.ogc.org/standards/wms] or Amazon S3 [https://aws.amazon.com/s3/] / Google Cloud Storage [https://cloud.google.com/storage]. Simply extending this component it is possible to integrate new protocols and data sources without impacts on the rest of the Framework. As the main interface for data discovery and access, even if the standards work is still in progress, the draft OGC API - Records [https://htmlpreview.github.io/?https://github.com/opengeospatial/ogcapi-records/blob/master/20-004.html] specification was selected by the Testbed-16 participants and used for the testing purposes.

10.3.2.5. Model

The whole ML framework is built to virtually handle and to encapsulate any potential AI model

independent from its scientific purpose. Of course, every model comes with a specific way to deal with input data and results. Indeed, even if from one side the model defines the scope of the network, on the other side it constrains the way input data is expected and the way the results are going to be provided. In this context, the focus is not on a particular model. This is so even if the model is the core element of the environment. The question is how to generalize the model integration within the framework. This modularity is established and guaranteed by specific implementations of Input Adapter and Output Adapter components (refer to 1.3.5).

10.3.2.6. Input/Output Adapters

As stated before, one of the more critical aspects of the framework design concerns how to cope with different and extremely specific ways that each ML model handles input and output data. So, the role of both the Input Adapter and the Output Adapter is to provide the bridge between framework internal flows and the model itself thus providing system modularity. The Input Adapter deals with data retrieved by the Dataset Loader that needs to be converted and formatted as specified by the model. Moreover, the data downloaded by the Dataset Loader might be not usable as retrieved from the data source and some pre-processing activities would be needed (e.g. data calibration, orthorectification and so on). The Output Adapter, instead, acts in a symmetrical way taking the results coming from the model, converting the data, and pushin it to the GeoServer [http://geoserver.org/] via a dedicated REST interface.

10.3.2.7. GeoServer MapML

To deliver the results of the model to the external clients a standard GeoServer [http://geoserver.org/] implementation was used along with a dedicated module (developed by the GeoServer [http://geoserver.org/] community) that exposes a MapML interface based on the OGC WMS Standard [https://www.ogc.org/standards/wms]. The Output Adapter (as described in paragraph 1.3.5) is in charge of interacting with the GeoServer [http://geoserver.org/] instance using the REST API in order to publish and make available the output of the model results.

10.3.3. Dataset specification

In the specific context of Testbed-16, the input data is discoverable at the following URL:

https://eratosthenes.pvretano.com/cubewerx/cubeserv/default/ogcapi/catalogues/collections/tb16cat

The catalogue (compliant with the draft OGC API – Records specifications) contains several Sentinel-1 acquisitions to be used as input to the ML model for both training and inferencing as well as calculating total biomass ground-truth.

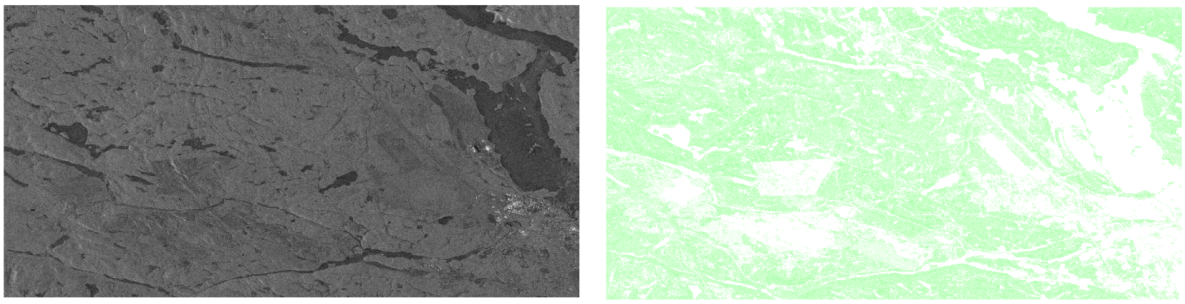


Figure 10. Total biomass ground-truth

Sentinel-1 acquisition can be downloaded via a standard HTTP request using the URL link found in the metadata returned by the catalogue. The ground-truth, instead, is served by exposing a standard OGC WMS service [https://www.ogc.org/standards/wms] exposing the total biomass estimations (layer tot_bio_r).

10.3.4. Components Details Design

10.3.4.1. Scope

This section provides the detailed descriptions of each components composing the ML environment framework.

10.3.4.2. ADES

10.3.4.2.1. Overview

All the requests for the discovery, deployment and running of a Machine Learning model are made through a dedicated ADES service exposed. The ADES server was based on a Weaver implementation starting from original Docker image. Weaver is primarily an EMS that allows the execution of workflows chaining various applications and Web Processing Services (WPS) inputs and outputs. Remote execution of each process in a workflow chain is dispatched by the EMS to one or many registered ADESs by ensuring the transfer of files accordingly between instances when located across multiple remote locations. In the current implementation, instead, the service has been configured as a simple ADES server by means of specific parameter in **weaver.ini** configuration file changing **weaver.configuration** = **ems** to **weaver.configuration** = **ades**.

For this deliverable, only two different types of services are registered in the ADES: model training and model inferencing. Beside these, the ADES service exposes standard calls such as the GetCapabilities and DescribeProcess to discover and browse available services. Additionally to ADES service, it is has to be installed **MongoDB** on host machine. This is needed to register and discover all the configured WPS services to be retrieved later, for instance, via **GetCapabilities**.

In order to start the CRIM ADES Docker container:

```
docker run -it --name ades_weaver -v /var/run/docker.sock:/var/run/docker.sock -v
/usr/local/bin/docker:/usr/bin/docker -v ${HOST_WPS_WORKDIR}:${WEAVER_WPS_WORKDIR} -p
4001:4001 pavics/weaver
```

where:

- **${HOST_WPS_WORKDIR}**: Path on the host machine where temporary result files from Weaver WPS process are stored
- **${WEAVER_WPS_WORKDIR}**: Path on Weaver docker where temporary result files from Weaver WPS process are stored

The **${HOST_WPS_WORKDIR}** and **${WEAVER_WPS_WORKDIR}** folders must point to the SAME valid path on the host machine.

WARNING	Just note that in case the CRIM Weaver ADES server is running on Windows machine, if needed to bind host folders with Docker folders, the path shall be converted in UNIX notation. E.g. the original path **C:\Users\ades_user\Documents\RHEA\Projects\ADES_Weaver\wpsworkdir** for **${HOST_WPS_WORKDIR}** shall be written inside **weaver.ini** configuration file as **/c/Users/ades_user/Documents/RHEA/Projects/ADES_Weaver/wpsworkdir**.

10.3.4.2.2. GetCapabilities

The GetCapabilities request is aimed at retrieval of the list of services registered on the ADES, including service metadata and metadata describing the available processes. The response is an XML document called the capabilities document, which contains a list of all available services. An example of a GetCapabilities request is:

${WEAVER_URL}/ows/wps?service=wps&request=getcapabilities

The response is a standard WPS GetCapabilities XML response. The following is a snippet of the services offered by the D133 ADES component:

```
<!-- PyWPS 4.2.8 -->
<wps:Capabilities service="WPS" version="1.0.0" xml:lang="en-US" xmlns:xlink=
"http://www.w3.org/1999/xlink" xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows
="http://www.opengis.net/ows/1.1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0
../wpsGetCapabilities_response.xsd" updateSequence="1">
...
...
  <wps:ProcessOffering>
    <wps:Process wps:processVersion="1.0.0">
      <ows:Identifier>train_biomass_model</ows:Identifier>
      <ows:Title>BIOMASS Train Model</ows:Title>
      <ows:Abstract>Trigger a process to train the ML model</ows:Abstract>
    </wps:Process>
    <wps:Process wps:processVersion="1.0.0">
      <ows:Identifier>inference_biomass_model</ows:Identifier>
      <ows:Title>BIOMASS Inference Model</ows:Title>
      <ows:Abstract>Trigger a process to perform ML model inference on input
data</ows:Abstract>
    </wps:Process>
  </wps:ProcessOfferings>
  ...
  ...
</wps:Capabilities>
```

10.3.4.2.3. DescribeProcess

The DescribeProcess operation requests details of any services offered by the D133 component. An example of a DescribeProcess request for a train_biomass_model service is:

${WEAVER_URL}/ows/wps?service=wps&request=describeprocess&identifier=train_biomass_model&version=1.0.0

All the available parameters, their nature and possible values if constrained are provided in a standard response package. In the following sections are described all the available services with relevant parameters.

10.3.4.2.4. Training

In order to trigger the model training, a specific WPS execute service is exposed with the following parameters:

Keyword	Description	Sample Values
input_data_server_endpoint	Endpoint of the server hosting input data	https://eratosthenes.pvretano.com/cubewerx/cubeserv/default/ogcapi/catalogues
input_data_server_catalogue	Catalogue name where input data can be found	tb16cat
input_data_server_protocol	Type of protocol for accessing data supported by the server where input data are. Supported protocols so far are OGC API – Records only	ogc_api_records
input_data_identifier	Any form of filters compositions available by the protocol supported by the server to identify input data	type=urn:cw:def:ebRIM-ObjectType:CubeWerx:DataProduct
gt_data_server_endpoint	Endpoint of the server hosting ground truth data	https://eratosthenes.pvretano.com/cubewerx/cubeserv/default/ogcapi/catalogues
gt_data_server_catalogue	Catalogue name where ground truth data can be found	tb16cat
gt_data_server_protocol	Type of protocol for accessing data supported by the server where ground truth data are. Supported protocols so far are OGC API – Records only	ogc_api_records
gt_data_identifier	Any form of filters compositions available by the protocol supported by the server to ground truth input data	q=tot_bio
epochs	Number of epochs the training of ML model will run through	50

Keyword	Description	Sample Values
checkpoint	Number of epoch identifying the checkpoint file from which the training is resume. If left empty, ML model will be trained from scratch. If checkpoint file is not found, process execution will fail	100
checkpoint_save_freq	Period expressed in epoch number between checkpoint saving	50

An example of a service call is given below with the proper JSON payload to be submitted:

POST ${WEAVER_URL}/processes/train_biomass_model/jobs

```
{
  "mode": "async",
  "response": "document",
  "inputs": [
    {
      "id": "input_data_server_endpoint",
      "data":
"https://eratosthenes.pvretano.com/cubewerx/cubeserv/default/ogcapi/catalogues "
    },
    {
      "id": "input_data_server_catalogue",
      "data": "tb16cat"
    },
    {
      "id": "input_data_server_protocol",
      "data": "ogc_api_records"
    },
    {
      "id": "input_data_identifier",
      "data": "type=urn:cw:def:ebRIM-ObjectType:CubeWerx:DataProduct"
    },
    {
      "id": "gt_data_server_endpoint",
      "data":
"https://eratosthenes.pvretano.com/cubewerx/cubeserv/default/ogcapi/catalogues"
    },
    {
      "id": "gt_data_server_catalogue",
      "data": "tb16cat"
    },
    {
      "id": "gt_data_server_protocol",
      "data": "ogc_api_records"
    },
    {
      "id": "gt_data_identifier",
```

```
      "data": "q=tot_bio"
    },
    {
      "id": "epochs",
      "data": 50
    },
    {
      "id": "checkpoint",
      "data": 100
    },
        {
      "id": "checkpoint_save_freq",
      "data": 50
    }
  ],
  "outputs": []
}
```

If the request is accepted and queued correctly, the client is provided with the Job ID (e.g. cb0a65a6-01b6-4672-becd-2bd1c6b1ce68) uniquely identifying the request. This Job Id is needed to perform a status query as provided in the **location** field in the JSON response to the call.

```
{
    "jobID": "cb0a65a6-01b6-4672-becd-2bd1c6b1ce68",
    "status": "accepted",
    "location": "${WEAVER_URL}/processes/inference_biomass_model/jobs/cb0a65a6-01b6-
4672-becd-2bd1c6b1ce68"
}
```

10.3.4.2.5. Inferences

In order to trigger the model inferences, a specific WPS execute service is exposed with the following parameters:

Keyword	Description	Sample Values
input_data_server_endpoint	Endpoint of the server hosting input data	https://eratosthenes.pvretano.com/cubewerx/cubeserv/default/ogcapi/catalogues
input_data_server_catalogue	Catalogue name where input data can be found	tb16cat
input_data_server_protocol	Type of protocol for accessing data supported by the server where input data are. Supported protocols so far are OGC API – Records only	ogc_api_records

Keyword	Description	Sample Values
input_data_identifier	Any form of filters compositions available by the protocol supported by the server to identify input data	type=urn:cw:def:ebRIM-ObjectType:CubeWerx:DataProduct
checkpoint	Number of epoch identifying the checkpoint file from which ML model is restored	100

An example of a service call is given below with the proper JSON payload to be submitted:

POST ${WEAVER_URL}/processes/train_biomass_model/jobs

```
{
  "mode": "async",
  "response": "document",
  "inputs": [
    {
      "id": "input_data_server_endpoint",
      "data":
"https://eratosthenes.pvretano.com/cubewerx/cubeserv/default/ogcapi/catalogues "
    },
    {
      "id": "input_data_server_catalogue",
      "data": "tb16cat"
    },
    {
      "id": "input_data_server_protocol",
      "data": "ogc_api_records"
    },
    {
      "id": "input_data_identifier",
      "data": "type=urn:cw:def:ebRIM-ObjectType:CubeWerx:DataProduct"
    },
    {
      "id": "checkpoint",
      "data": 100
    }
  ],
  "outputs": []
}
```

If the request is accepted and queued correctly, the client is provided with the Job ID (e.g. cb0a65a6-01b6-4672-becd-2bd1c6b1ce68) uniquely identifying the request. This Job Id is needed to perform a status query as provided in the **location** field in the JSON response to the call.

```
{
    "jobID": "cb0a65a6-01b6-4672-becd-2bd1c6b1ce68",
    "status": "accepted",
    "location": "${WEAVER_URL}/processes/inference_biomass_model/jobs/cb0a65a6-01b6-
4672-becd-2bd1c6b1ce68"
}
```

When the status query indicates processing has completed for the required Job ID, the results of the inference of the ML model on input data are published through Output Adapter on the GeoServer [http://geoserver.org/].

10.3.4.2.6. GetStatus

ADES features built-in processes execution monitoring service. A process status can be requested using a GetStatus request with the following input parameters:

Keyword	Description	Sample Values
process_id	Identifier of the process	train_biomass_model
job_id	The Job ID returned in the XML response when a process is submitted	a9d14bf4-84e0-449a-bac8-16e598efe807

An example of **GetStatus** query is given below:

GET {WEAVER_URL}/processes/{process_id}/jobs/{job_id}

The status of a process can be one of the following values:

- **accepted**: This is the status of a process when submitted. This effectively means that the process is pending execution, but is not yet executing.
- **started**: This is the status assumed when a worker retrieves the process it for execution and is making preparation steps.
- **succeeded**: This is the status when all its operations are completed successfully.
- **failed**: This is the status when errors occurred during processing thus preventing the process from completing its execution.

10.3.4.3. Coordinator

The Coordinator acts as a process dispatcher and orchestrator between the components of the ML environment framework in order to execute two typical services of the architecture:

- Train a model
- Trigger model inferences

This component is called by the ADES interface and triggered by a client via a WPS call. The parameters coming from the WPS call are gathered by the Coordinator using CWL and used to

control the other components execution for the correct exploitation of the service. When triggered, the Coordinator will run two different workflows according to the service required and sharing some common steps.

As a first step, the Coordinator invokes the Dataset Loader to retrieve the data using the parameters in the request (see Dataset Loader) that is either to retrieve pairs of input and ground-truth data or just test data to run ML model inference on. When data is available on local storage, the Coordinator invokes the Input Adapter to manipulate, if necessary, data to cope with ML model input data requirements. At this stage, data is ready to be used by the ML model.

If the training service has been requested, the Coordinator calls the ML model to train on input and ground truth data, either resuming from an existing checkpoint or starting from scratch (see Training for the full list of training parameters). Instead, if it is triggered by the inference service, the Coordinator issues the loading of the ML model checkpoint file and runs inference on input data (see Inferences).

When inference is completed, the Output Adapter is called by the Coordinator to handle the results from the ML model in order to be converted and published on the GeoServer to allow MapML clients to access the generated output.

10.3.4.4. Data Loader External Data Sources

10.3.4.4.1. Overview

The Dataset Loader is the main component in charge of discovering and downloading data from external data providers. The Dataset Loader was designed to generalize access to data and to enable handling data coming from providers exposing different protocols and standards, such as OGC CSW, WCS, WMS, Google / Amazon Buckets and so on. The Data Loader component performs a set of actions that can be grouped into two different stages:

- A metadata discovery stage to retrieve the metadata associated to the queried resources.
- A resources download stage, where the resource is actually downloaded used different protocol (e.g. CSW,WMS,WCS,...) based on the information contained in the metadata.

In order to discover and access the data, the following input parameters coming from CWL are required:

- Endpoint of the server catalogue.
- Catalogue name.
- A resource name or any form of filtering supported by the relevant catalogue that could be used to identify resources.

The following table summarizes input parameters required by Data Loader component:

Keyword	Description	Sample Values
input_data_server_endpoint	Endpoint of the server hosting input data	https://eratosthenes.pvretano.com/cubewerx/cubeserv/default/ogcapi/catalogues
input_data_server_catalogue	Catalogue name where input data can be found	tb16cat
input_data_server_protocol	Type of protocol for accessing data supported by the server where input data are. Supported protocols so far are OGC API – Records only	ogc_api_records

10.3.4.4.2. Metadata discovery

The discovery of a resource in a catalogue is the Data Loader's first step to access the data. This is accomplished using one of the possible protocol interfaces (e.g. OGC, CSW, OGC API - Records) made available by the catalogue hosting the dataset.

The result of the discovery, if any, is a metadata file containing several information elements about the queried resource together with the endpoint on which the resource can be downloaded (e.g. simple HTTP URL, endpoint of the WMS server hosting the layer and so on).

An example of a HTTP query, in accordance with the draft OGC API – Records specification, to search all Sentinel-1 products on the catalogue is shown below:

https://eratosthenes.pvretano.com/cubewerx/cubeserv/default/ogcapi/catalogues/collections/tb16cat/items?type=urn:cw:def:ebRIM-ObjectType:CubeWerx:DataProduct&f=JSON

The response of the server catalogue is a metadata file in JSON format containing the information about how many items have been found with relevant metadata information. Following is a snippet of the catalogue response.


```
{
  "timeStamp": "2020-09-28T10:42:06-04:00",
  "numberMatched": 11,
  "numberReturned": 10,
  "links": [
    {
      "href":
"https://eratosthenes.pvretano.com/cubewerx/cubeserv/default/ogcapi/catalogues/collect
ions/tb16cat/items?type=urn:cw:def:ebRIM-ObjectType:CubeWerx:DataProduct&f=JSON",
      "rel": "self"
    },
    {
      "href":
"https://eratosthenes.pvretano.com/cubewerx/cubeserv/default/ogcapi/catalogues/collect
ions/tb16cat/items?type=urn:cw:def:ebRIM-
```

```
ObjectType:CubeWerx:DataProduct&f=JSON&limit=10&offset=11",
      "rel": "next"
    }
  ],
  "records": [
    {
      "id": "urn:uuid:b6713356-f4a9-11ea-889c-933c4b707059",
      "type": "feature",
      "@type": "urn:cw:def:ebRIM-ObjectType:CubeWerx:DataProduct",
      "geometry": {
        "type": "Polygon",
        "coordinates": [
          [
            [
              -78.98628879,
              44.84279684
            ],
            [
              -78.98628879,
              46.88806826
            ],
            [
              -75.00402221,
              46.88806826
            ],
            [
              -75.00402221,
              44.84279684
            ],
            [
              -78.98628879,
              44.84279684
            ]
          ]
        ]
      },
      "properties": {
        "title": "SENTINEL-1 Product
(S1A_IW_GRDH_1SDV_20150317T230010_20150317T230035_005078_00661A_BB3C)",
        "date": "2015-03-17T23:00:10",
        "urn:cw:def:ebRIM-SlotName:crs": "http://www.opengis.net/def/crs/EPSG/0/4326",
        "urn:cw:def:ebRIM-SlotName:hasRepositoryItem": true,
        "urn:cw:def:ebRIM-SlotName:qdate": "2015-03-17T23:00:10Z",
        "urn:cw:def:ebRIM-SlotName:sentinel1:absoluteOrbitNumber": 5078,
        "urn:cw:def:ebRIM-SlotName:sentinel1:enclosure":
"https://www.pvretano.com/Projects/tb16/data/sentinel-
s1/S1A_IW_GRDH_1SDV_20150317T230010_20150317T230035_005078_00661A_BB3C.SAFE.zip",
        "urn:cw:def:ebRIM-SlotName:sentinel1:mapOverlay":
"https://www.pvretano.com/Projects/tb16/data/sentinel-
s1/S1A_IW_GRDH_1SDV_20150317T230010_20150317T230035_005078_00661A_BB3C.SAFE/preview/ma
p-overlay.kml",
```

```
        "urn:cw:def:ebRIM-SlotName:sentinel1:measurement": [
          "https://www.pvretano.com/Projects/tb16/data/sentinel-
s1/S1A_IW_GRDH_1SDV_20150317T230010_20150317T230035_005078_00661A_BB3C.SAFE/measuremen
t/s1a-iw-grd-vh-20150317t230010-20150317t230035-005078-00661a-002.tiff",
          "https://www.pvretano.com/Projects/tb16/data/sentinel-
s1/S1A_IW_GRDH_1SDV_20150317T230010_20150317T230035_005078_00661A_BB3C.SAFE/measuremen
t/s1a-iw-grd-vv-20150317t230010-20150317t230035-005078-00661a-001.tiff"
        ],
        "urn:cw:def:ebRIM-SlotName:sentinel1:missionDataTakeId": "00661A",
        "urn:cw:def:ebRIM-SlotName:sentinel1:missionId": "S1A",
        "urn:cw:def:ebRIM-SlotName:sentinel1:mode": "IW",
        "urn:cw:def:ebRIM-SlotName:sentinel1:path":
"GRD/2015/03/17/IW/DV/S1A_IW_GRDH_1SDV_20150317T230010_20150317T230035_005078_00661A_B
B3",
        "urn:cw:def:ebRIM-SlotName:sentinel1:polarization": "DV",
        "urn:cw:def:ebRIM-SlotName:sentinel1:processingLevel": 1,
        "urn:cw:def:ebRIM-SlotName:sentinel1:productClass": "standard",
        "urn:cw:def:ebRIM-SlotName:sentinel1:productId":
"S1A_IW_GRDH_1SDV_20150317T230010_20150317T230035_005078_00661A_BB3C",
        "urn:cw:def:ebRIM-SlotName:sentinel1:productPreview":
"https://www.pvretano.com/Projects/tb16/data/sentinel-
s1/S1A_IW_GRDH_1SDV_20150317T230010_20150317T230035_005078_00661A_BB3C.SAFE/preview/pr
oduct-preview.html",
        "urn:cw:def:ebRIM-SlotName:sentinel1:productType": "S1A",
        "urn:cw:def:ebRIM-SlotName:sentinel1:productUniqueIdentifier": "BB3C",
        "urn:cw:def:ebRIM-SlotName:sentinel1:quickLook":
"https://www.pvretano.com/Projects/tb16/data/sentinel-
s1/S1A_IW_GRDH_1SDV_20150317T230010_20150317T230035_005078_00661A_BB3C.SAFE/preview/qu
ick-look.png",
        "urn:cw:def:ebRIM-SlotName:sentinel1:resolutionClass": "high",
        "urn:cw:def:ebRIM-SlotName:sentinel1:startTime": "2015-03-17T23:00:10Z",
        "urn:cw:def:ebRIM-SlotName:sentinel1:stopTime": "2015-03-17T23:00:35Z",
        "urn:cw:def:ebRIM-SlotName:sentinel1:thumbNail":
"https://eratosthenes.pvretano.com/cubewerx/cubeserv.cgi?service=WRS&version=3.0&catal
ogueId=tb16cat&request=GetRepositoryItem&id=urn%3Auuid%3Ab6713356-f4a9-11ea-889c-
933c4b707059&repoId=21"
      }
    },
    ...
  ]
}
```


The `records` tag in the response is an object array containing metadata information about each Sentinel-1 product available on the catalogue. The tag `urn:cw:def:ebRIM-SlotName:sentinel1:measurement` holds the URL to download Sentinel-1 data as GeoTIFF image files using simple HTTP protocol.

Similarly, an example of a HTTP query, in accordance with the draft OGC API – Records specification, to search for `tot_bio_r` layer in the catalogue is shown below:

https://eratosthenes.pvretano.com/cubewerx/cubeserv/default/ogcapi/catalogues/collections/tb16cat/items?q=tot_bio&bbox=45.7912,-77.4114,46.0176,-77.1903,4326&f=JSON

The response of the server catalogue is given below:


```
{
  "timeStamp": "2020-09-29T11:43:55-04:00",
  "numberMatched": 1,
  "numberReturned": 1,
  "links": [
    {
      "href":
"https://eratosthenes.pvretano.com/cubewerx/cubeserv/default/ogcapi/catalogues/collect
ions/tb16cat/items?q=tot_bio&bbox=45.7912,-77.4114,46.0176,-77.1903,4326&f=JSON",
      "rel": "self"
    }
  ],
  "records": [
    {
      "id": "urn:uuid:c170ea74-c067-11ea-ad05-97071d6a09dd",
      "resourceId": "tot_bio",
      "type": "feature",
      "@type": "urn:cw:def:ebRIM-ObjectType:CubeWerx:WMS:Layer",
      "geometry": {
        "type": "Polygon",
        "coordinates": [
          [
            [
              -176.412,
              34.3112
            ],
            [
              -176.412,
              83.977
            ],
            [
              -10.8073,
              83.977
            ],
            [
              -10.8073,
              34.3112
            ],
            [
              -176.412,
              34.3112
            ]
          ]
        ]
      },
```

```
      "properties": {
        "title": "tot_bio_r",
        "description": "Total aboveground biomass. Individual tree total aboveground
biomass is calculated using species-specific equations. In the measured ground plots,
aboveground biomass per hectare is calculated by summing the values of all trees
within a plot and dividing by the area of the plot. Aboveground biomass may be
separated into various biomass components (e.g. stem, bark, branches, foliage) (units
= t/ha). Products relating the structure of Canada's forested ecosystems have been
generated and made openly accessible. The shared products are based upon peer-reviewed
science and relate aspects of forest structure including: (i) metrics calculated
directly from the lidar point cloud with heights normalized to heights above the
ground surface (e.g., canopy cover, height), and (ii) modelled inventory attributes,
derived using an area-based approach generated by using co-located ground plot and ALS
data (e.g., volume, biomass). Forest structure estimates were generated by combining
information from 'lidar plots' (Wulder et ",
        "time": "2015-01-01T00:00:00Z",
        "urn:cw:def:ebRIM-SlotName:accessURLTemplate":
"https://opendata.nfis.org/mapserver/cgi-
bin/wms_change.cgi?version=1.3.0&request=GetMap&layers=tot_bio&styles=%7Bstyles%7D&crs
=%7Bcrs%7D&bbox=%7Bbbox%7D&width=%7Bwidth%7D&height=%7Bheight%7D&format=%7Bformat%7D",
        "urn:cw:def:ebRIM-SlotName:crs": [
          "EPSG:3979",
          "EPSG:42101"
        ],
        "urn:cw:def:ebRIM-SlotName:keyword": "forest biomass",
        "urn:cw:def:ebRIM-SlotName:outputFormat": [
          "image/png; mode=8bit",
          "image/tiff"
        ],
        "urn:cw:def:ebRIM-SlotName:queryable": 1
      },
      "links": [
        {
          "href":
"https://eratosthenes.pvretano.com/cubewerx/cubeserv/default/ogcapi/catalogues/collect
ions/tb16cat/items/urn:uuid:6ab874b2-c063-11ea-a3c7-b3754f7b7808",
          "rel": "OperatesOn",
          "type": "urn:oasis:names:tc:ebxml-regrep:ObjectType:RegistryObject:Service",
          "title": "Operated upon Service \"High Resolution Satellite Forest
Information for Canada \""
        },
        {
          "href":
"https://eratosthenes.pvretano.com/cubewerx/cubeserv/default/ogcapi/catalogues/collect
ions/tb16cat/items/urn:uuid:6ad2b020-c063-11ea-8608-673aff7e781a",
          "rel": "ParentOf",
          "type": "urn:cw:def:ebRIM-ObjectType:CubeWerx:WMS:Layer",
          "title": "Child Of WMS Layer \"High Resolution Satellite Forest Information
for Canada \""
        }
      ]
```

```
    }
  ]
}
```

In order to download the data from the layer, a WMS request to the WMS server hosting the layer is made. The address of the WMS server is retrieved using the field `urn:cw:def:ebRIM-SlotName:accessURLTemplate` from the metadata.

10.3.4.4.3. Products Download

After the discovery of the resource has been performed, the resource can be downloaded using a specific protocol based on resource type information gathered by the metadata file (e.g. OGC WMS, WCS). Below is an example of a WMS request used to download data from total_bio_r layer

WARNING	Just note that in some cases the request must be refined due to some limits on queried service. Following a sample response of an **OGC WMS** endpoint with a request exceeding the image sizes (set to maximum 4096 x 4096 pixels). In this case, before calling the Input Adapter and then the model, the data must be downloaded in slices and merged together once all the slices are available.

```
<?xml version="1.0" encoding="windows-1252" standalone="no"?>
<ServiceExceptionReport version="1.3.0" xmlns="http://www.opengis.net/ogc" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"http://www.opengis.net/ogc http://schemas.opengis.net/wms/1.3.0/exceptions_1_3_0.xsd
">
    <ServiceException>msWMSLoadGetMapParams(): WMS server error. Image size out of
range, WIDTH and HEIGHT must be between 1 and 4096 pixels.</ServiceException>
</ServiceExceptionReport>
```

The downloaded data is stored on a local folder to be later accessed by the Input Adapter.

10.3.4.5. Input Adapter

In the context of total biomass estimation from Sentinel-1 data, a custom Input Adapter was developed to convert data from the catalogue to the format requirements posed by the ML model. These adaptations involved both Sentinel-1 data as well as data for the ground truth. At the same time, Sentinel-1 data coming from the catalogue is not orthorectified and in linear scale. To orthorectify Sentinel-1 data and convert the values in a logarithm scale, a dedicated SNAP (Sentinel Application Platform) workflow was implemented and integrated within the Input Adapter.

Figure 11. SNAP workflow executed by Input Adapter

Moreover, since Sentinel-1 spatial resolution is 10m while ground truth data is 30m, Sentinel-1 data is resampled by the Input Adapter to match the spatial resolution of 30m.

The ground truth is encoded as an 8-bit unsigned integer RGBA image where total biomass is displayed in green shades where darker green means high amount of biomass while lighter green means low concentration. Since only the green channel contains valid information for the training of the ML model (Red and Blue are fixed to 255 and no transparencies in Alpha), the Input Adapter retain only this one.

10.3.4.6. AI Model

The default AI model delivered within the environment is a deep neural network based on U-Net architecture. The network takes as input a GeoTIFF image containing the VV polarization channel of Sentinel-1 and predicts, as single channel GeoTIFF image in the range [0,255], the total biomass where higher concentration of total biomass is associated to higher pixel value in the image.

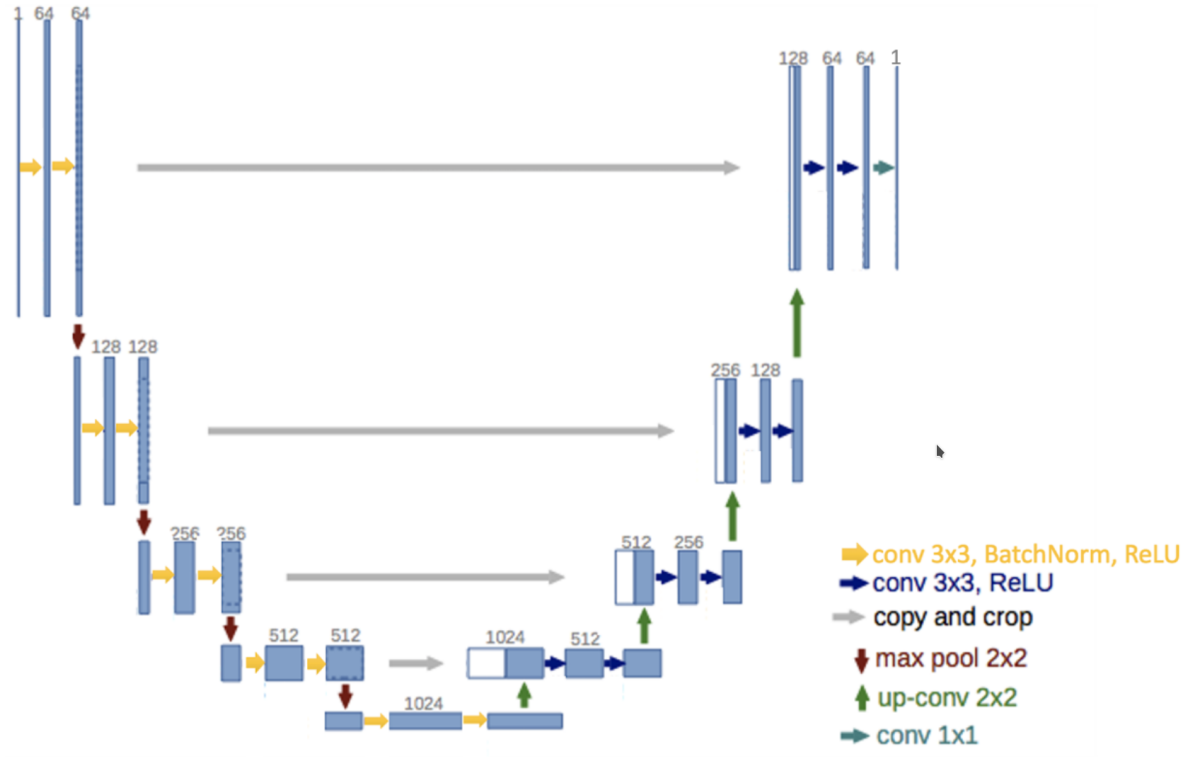


Figure 12. Model U-Net architecture

The predicted total biomass image coming from the model is then converted to a green shade image by taking the single channel in the image as green channel and adding red and blue channels fixed to the 8-bit value of 255 (see paragraph 1.5.6).

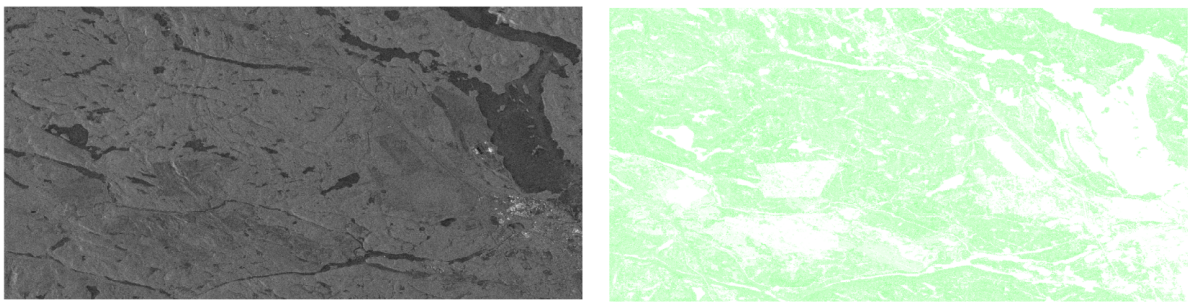


Figure 13. Sentinel-1 VV polarization and total biomass ground truth

ML model checkpoints generated during training are saved in a pre-configured folder. This folder is associated with the ML model. This is so the information is accessible either to resume training

from a specific checkpoint file or to load a checkpoint in order to perform inference with ML model.

NOTE	None of the ML frameworks suggested in the CFP (Call For Proposal) was used since all of those frameworks just deal with object detection, classification ad segmentation tasks. The problem addressed here, as required in TB-16 meetings, is regression. Thus designing a custom framework from scratch was preferred.

10.3.4.7. Output adapter

The output of the ML model, as with the input, comes in a format specific to the target model. The result image/data needs to be converted to one of the formats supported by GeoServer [http://geoserver.org/] for publication. Being able to support images (both gray scale and color) with one or many bands / layers, GeoTIFF was chosen as the output image format. Additionally, the GeoTIFF format is able to store any kind of geographical information in its metadata. Once the result image/data coming from the ML model is converted into a GeoTIFF image, this file is sent to GeoServer [http://geoserver.org/] by means of a custom library wrapping the GeoServer [http://geoserver.org/] REST API in order to create a new SourceDataset containing the image and to create a new Layer to expose the output to the users. Details of the GeoServer [http://geoserver.org/] REST API can be found in the next section.

10.3.4.8. GeoServer

Using GeoServer [http://geoserver.org/] is the last step of the ML framework chain. GeoServer's role is to make publicly available the results of the ML model inference in both the MapML format and via a WMS interface. In order to activate support for MapML, a dedicated module has been installed into the bare-bone GeoServer [http://geoserver.org/] delivery. The GeoServer [http://geoserver.org/] MapML support is still in a draft status but already exists as a community extension (https://docs.geoserver.org/latest/en/user/community/mapml/index.html). The use of MapML has already been exploited in previous OGC Testbed Initiatives:

- OCG Testbed-13 https://docs.ogc.org/per/17-019.html
- OGC Testbed-14 http://docs.opengeospatial.org/per/18-023r1.pdf
- OGC Testbed-15 https://docs.opengeospatial.org/DRAFTS/19-046.html

The MapML modules include support for styles, tiling, querying, shading, and dimensions options for WMS layers, and also provide a MapML outputFormat for WMS GetFeatureInfo.

An example can be found at following link: https://borealweb.nfis.org/tb16d133/geoserver/mapml/TestWorkSpace:tot_bio_petawawa/osmtile/

The embedded MapML client triggers a WMS request to access the image: https://borealweb.nfis.org/tb16d133/geoserver/TestWorkSpace/wms?version=1.3.0&service=WMS&request=GetMap&crs=urn:x-ogc:def:crs:EPSG:3857&layers=TestWorkSpace:tot_bio_petawawa&styles=&bbox=-8659208.490311604,5742535.3559917845,-8580936.973361604,5801238.9937042855&format=image/png&transparent=true&width=1024&height=768

The publication of the image is done by means of the GeoServer [http://geoserver.org/] REST API. Two

steps are necessary in order to expose the image:

- Create a DataStore with a URI link to the image to make the image recognized by the GeoServer [http://geoserver.org/]
- Create a Layer to make the image available to users

Assuming that a default Workspace [https://docs.geoserver.org/stable/en/user/rest/workspaces.html] is made already available by the initial configuration of the GeoServer [http://geoserver.org/], the first step is to define a DataStore with the image reference. The DataStore is created with a HTTP POST call to https://borealweb.nfis.org/tb16d133/geoserver/rest/workspaces/TestWorkSpace/coveragestores/ with parameters sent in the request body and encoded in XML or JSON format. An XML example HTTP POST body is:

```
<?xml version="1.0" encoding="UTF-8"?>
<CoverageStoreInfo>
    <name>BiomasDataStore</name>
    <type>GeoTIFF</type>
    <enabled>true</enabled>
    <workspace>
        <name>TestWorkSpace</name>
    </workspace>
    <url>file:biomas/tot_bio_petawawa.tif</url>
</CoverageStoreInfo>
```

The second step is to create the Layer with a HTTP POST call to https://borealweb.nfis.org/tb16d133/geoserver/rest/workspaces/TestWorkSpace/wmslayer still with parameters sent in the request body and encoded in XML or JSON format. An XML example post body is:

```
<?xml version="1.0" encoding="UTF-8"?>
<wmsLayer>
    <name>Biomass</name>
    <nativeName>Biomass</nativeName>
    <namespace>
        <name>string</name>
        <link>string</link>
    </namespace>
    <title>Biomass</title>
    <abstract>The biomass</abstract>
    <nativeCRS>EPSG:4326</nativeCRS>
    <srs>string</srs>
    <nativeBoundingBox>
        <minx>-77.616780646139</minx>
        <maxx>45.85534888786863</maxx>
        <miny>-77.25408105847575</miny>
        <maxy>46.0388387807033</maxy>
    </nativeBoundingBox>
    <latLonBoundingBox>
        <minx>-77.616780646139</minx>
        <maxx>45.85534888786863</maxx>
        <miny>-77.25408105847575</miny>
        <maxy>46.0388387807033</maxy>
    </latLonBoundingBox>
    <projectionPolicy>FORCE_DECLARED</projectionPolicy>
    <enabled>true</enabled>
    <metadata>
        <@key>regionateStrategy</@key>
        <text>string</text>
    </metadata>
    <store>
        <name>BiomasDataStore</name>
    </store>
</wmsLayer>
```

As an example, the following figure shows the rendering on a MapML internal client of the GeoServer [http://geoserver.org/]

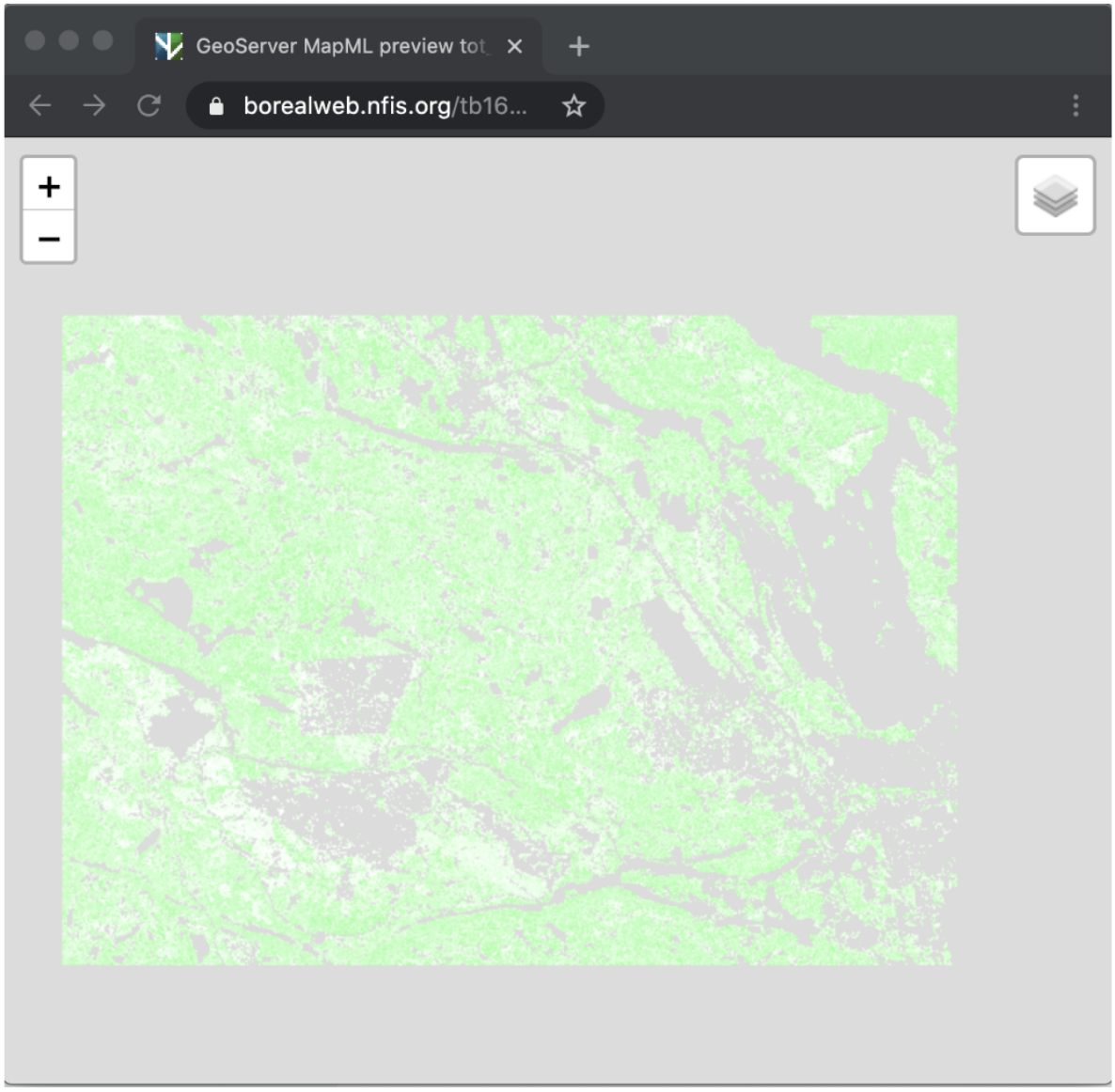


Figure 14. MapML client accessing GeoServer WMS service

10.3.5. Conclusions

Several interesting outcomes come with this activity. A framework has been implemented that can be used as a skeleton for theoretically any Machine Learning model (thanks to Input / Output adapters) and virtually coping with tons of different heterogeneous data sources. The modularity of this framework also permits any extension on any step of the whole chain. For instance, it would be possible to change to final output just by simply swapping the MapML with a different standard and only updating the relevant Output Adapter maintaining the rest of the structure as is with no impact and side effects.

As already highlighted in **Testbed-15**, speaking merely about development, having plenty of Python libraries available that implement OGC standards is really useful, and also having the **ADES WPS** server already available has been a valuable benefit. Further, having the framework encapsulated in a Docker container helped a lot to test the whole application independently for the environment and to scale it in production.

The use of the draft OGC API - Records [https://htmlpreview.github.io/?https://github.com/opengeospatial/ogcapi-records/blob/master/20-004.html] Standard eased the development and the querying mechanism

having to deal with a single standard. Some additional improvements are needed (e.g. some specific identifiers to find exactly the resource and format needed). At the same time, it is really simple to integrate within the framework other catalogues served for instance by OGC CSW Standard [https://www.ogc.org/standards/cat] and data providers exposing OGC WCS services [https://www.ogc.org/standards/wcs] for the data retrieval. In addition, just by upgrading the data loader component it would be possible to integrate additional repository like Amazon S3 [https://aws.amazon.com/s3/] / Google Cloud Storage [https://cloud.google.com/storage].

The following list presents some additional notes suggested to be considered for future developments:

- **Data Authenticity**: This aspect needs to be investigated in order to be sure that the model is trained and inferred with authentic data (the issue of data tampering in satellite imagery was also noted).
- **Analysis Ready Data (ARD)**: Another important aspect to take into consideration when the framework deals with different data sources like datacubes where some data could be already in ARD format and some other not.
- **ONNX check points**: for the time being the actual model stores its checkpoints in native format, but it could be useful to take into consideration **ONNX** format
- **Training dismissal**: another important aspect to be covered is the expected behavior in case it is required to interrupt the training. For instance, all the intermediate training has to be maintained or not? This should be further investigated.

In conclusion, the team demonstrated not only the feasibility of an ML Framework Environment in a generic and complex context but also the potential added benefit of this approach in terms of modularity and expandability.

10.4. Deep Learning Environment (D134 - CRIM)

10.4.1. Deep Learning Models Applied to LiDAR Datasets

10.4.1.1. Training Set

The Dayton Annotated Laser Earth Scan (DALES) [https://udayton.edu/engineering/research/centers/vision_lab/research/was_data_analysis_and_processing/dale.php] was recently proposed as a new benchmark for point cloud classification techniques. DALES generates a very high density sampling with over a half-billion points spanning 10 square kilometers of area. The data was hand labeled by a team of expert LiDAR technicians into eight categories: ground, vegetation, cars, trucks, poles, power lines, fences and buildings. The goal of this data set is to help advance the field of deep learning within aerial LiDAR. The data set is pre-split into 29 training files and 11 testing files, with the following categories: ground(1), vegetation(2), cars(3), trucks(4), power lines(5), fences(6), poles(7) and buildings(8). The original dataset is high density (50ppm), it has been downsampled at 10ppm in order to match NRCan tiles density using LAStools.

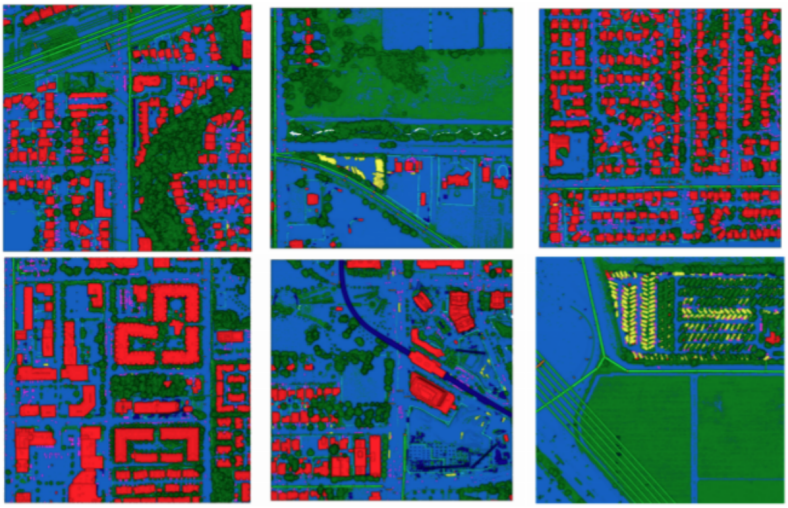


Figure 15. Example of DALES tiles, Semantic classes are labeled by color; ground (blue), vegetation (dark green), power lines (light green), poles (orange), buildings (red), fences (light blue), trucks (yellow), cars (pink), unknown (dark blue).

10.4.1.2. Neural Network Architecture

Several state-of-the art methods, mostly based on deep learning, were also compared using the DALES dataset [https://arxiv.org/abs/2004.11985]. The participants chose the ConvPoint method (https://arxiv.org/pdf/1904.02375.pdf) for its ease of use and performance on large scale LIDAR datasets. The architecture follows a typical UNet approach as shown on Figure 16 below. A few modifications were added to the PyTorch implementation with respect to the data loader in order to be able to load LIDAR tiles in .las format.

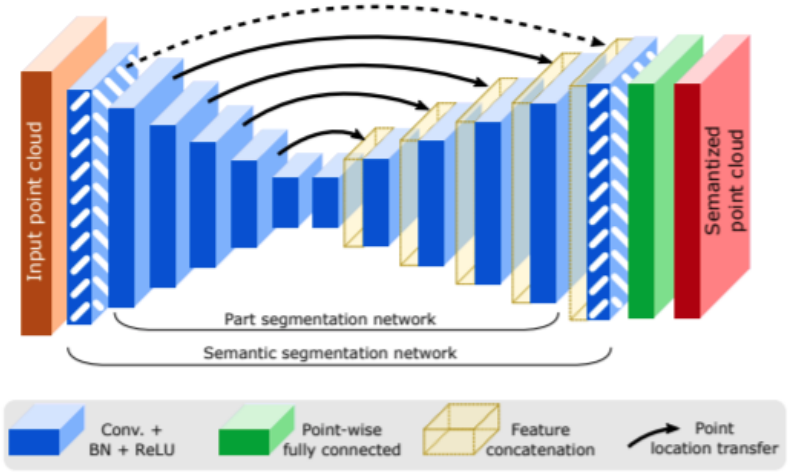


Figure 16. Segmentation networks. The network made of an encoder (progressive reduction of the point cloud size) followed by a decoder (to get back to the original point cloud size). Skip connections (black arrows) allow information to flow directly from encoder to decoder (taken from https://arxiv.org/pdf/1904.02375.pdf).

Once the model is trained on DALES, the inference is performed on a few tiles taken from the Service New Brunswick website (https://geonb.snb.ca/li/).

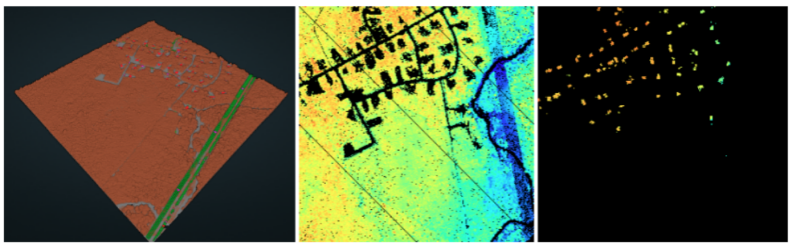


Figure 17. Test on a New Brunswick tiles (from left to right, all classes, vegetation and building classes).

10.4.2. ONNX Packaging

Machine learning frameworks like Pytorch or Tensorflow provide interfaces that help developers to build and run computation graphs representing neural networks. Most of those frameworks provide similar capabilities but have their own format for representing these graphs. The main goal of ONNX [https://onnx.ai/] is to offer a unified system API for switching between machine learning frameworks. ONNX provides a definition of a computation graph model, and definitions of built-in operators and standard data types. Each computation dataflow graph is structured as a list of nodes that form an acrylic graph and each node is a call to an operator. The ONNX graph also maintains metadata to help document its purpose, author, etc. The idea is that a user can train a model with one tool stack and then deploy the model using another for inference and prediction. To ensure this interoperability the user must export this model in the **model.onnx** format which is basically a serialized representation of the model as Protocol Buffers [https://developers.google.com/protocol-buffers] (protobuf). More precisely, the objectives of ONNX are:

1. **Framework interoperability**: Each framework is optimized for specific tasks such as fast training, inferencing on mobile devices or supporting flexible network architectures and so forth. The requirements most important during research and development are usually different from those for shipping and production. This may lead to inefficiencies from not using the right framework or significant delays due to conversion of models between frameworks. Frameworks that use the ONNX representation simplify this and enable developer's efficiency.
2. **Shared optimization**: Usually optimizations need to be integrated separately into each framework which can be a time-consuming process. Hardware vendors and developers with optimizations for improving the performance of their neural networks can impact multiple frameworks at once by targeting the ONNX representation.

10.4.2.1. Limits of ONNX

The use of ONNX is straightforward as long as the project meets these two conditions:

1. Developers are using supported data types and operations of the ONNX specification.
2. Developers do not do any custom development in terms of specific custom layers/operations.

If these conditions are not met, the functionality has to be implemented in an ONNX backend. An ONNX backend is a library that can run ONNX models. The custom implementation can turn out to be very time-consuming and laborious despite ONNX providing an API. A very important point is that the developers must first double-check that the used operations and functions are implemented in the backend for the export and import. If a project is carried out within this framework, the use of ONNX is effective. A second point to check is that not all frameworks do the export to ONNX and import from ONNX. For instance, the Pytorch framework still does not provide

any imports from ONNX. The developer needs to understand that ONNX is packaging only the neural network implementation (the graph) and its parameters. There is no information related to pre-processing, post-processing, metadata or semantics. The ONNX project is developing at a rapid pace and is continually releasing new versions that enhance the compatibility between the frameworks.

10.4.3. OGC API - Records for LiDAR Datasets

The main goal of this task was to explore possible metadata that could be extracted from LIDAR tiles that could be useful to build training and testing sets. About twenty tiles were downloaded from the New Brunswick website [https://geonb.snb.ca/li/].

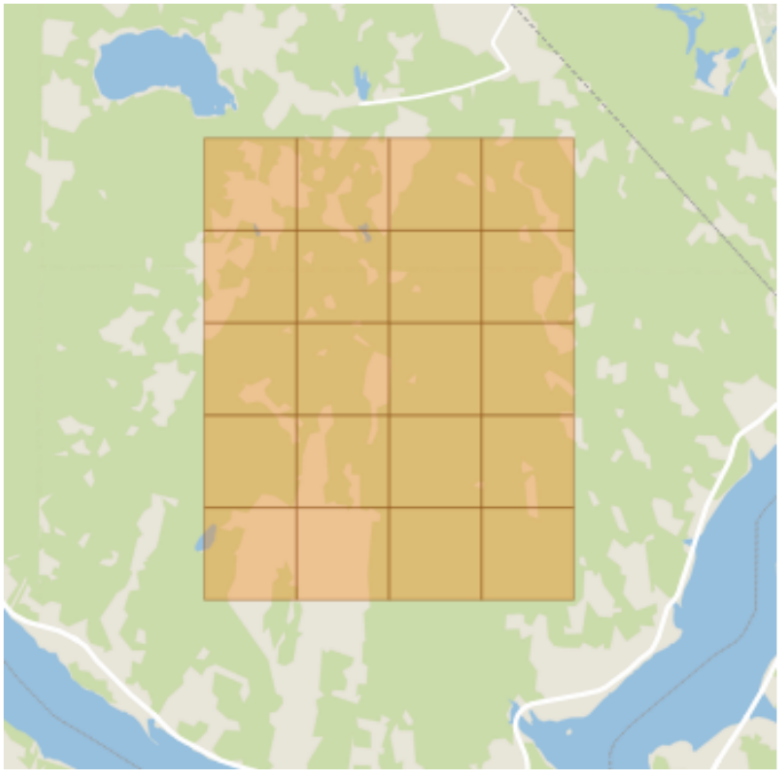


Figure 18. Footprint of the NRCan tiles taken from https://geonb.snb.ca/li/

One issue in building a training task (or even testing) is to make sure that the training set is balanced with nearly the same number of samples per relevant classes. Therefore, the relevant information could be the class names with their frequency or occurrences. For instance, the following statistics can be extracted using lastools [https://rapidlasso.com/lastools/lasinfo/]:

```
histogram of classification of points:
          490  never classified (0)
      4423717  unclassified (1)
     17817147  ground (2)
         4120  low vegetation (3)
         1899  medium vegetation (4)
        10282  high vegetation (5)
          596  building (6)
          703  noise (7)
       145406  keypoint (8)
```

Lastools provides the number of points per class which can be useful to build a training/testing dataset. Also, the general metadata contains class definitions: Using a Python script and the LASTool library [https://rapidlasso.com/lastools/], metadata are extracted from the .laz tiles to form Feature records:


```
   {
     "type": "Feature",
     "id": "nb_2018_2465000_7438000",
     "properties": {
       "name": "nb_2018_2465000_7438000",
       "crs": {
         "type": "name",
         "properties": {
           "name": "urn:ogc:def:crs:EPSG::2953"
         }
       },
       "featureclass": "LIDAR",
       "LAZ Metadata":
"https://geonb.snb.ca/downloads2/lidar/2018/snb/aoi1/meta/meta_2018_aoi1.xml",
       "onlink": "https://geonb.snb.ca/li/",
       "class_histograms": [
         {
           "name": "unclassified",
           "label": 1,
           "count": 24
         },
         {
           "name": "ground",
           "label": 2,
           "count": 1879673
         },
         {
           "name": "low vegetation",
           "label": 3,
           "count": 1551931
         },
         {
           "name": "medium vegetation",
           "label": 4,
           "count": 1453260
         },
         {
           "name": "high vegetation",
           "label": 5,
           "count": 18614547
         },
         {
           "name": "noise",
           "label": 7,
           "count": 3906
```

```
      },
      {
        "name": "keypoint",
        "label": 8,
        "count": 10709
      },
      {
        "name": "water",
        "label": 9,
        "count": 68
      }
    ],
    "class_names": [
      "unclassified",
      "ground",
      "low vegetation",
      "medium vegetation",
      "high vegetation",
      "noise",
      "keypoint",
      "water"
    ],
    "class_labels": [
      1,
      2,
      3,
      4,
      5,
      7,
      8,
      9
    ],
    "class_count": [
      24,
      1879673,
      1551931,
      1453260,
      18614547,
      3906,
      10709,
      68
    ],
    "class_frequency": [
      0.0001020663415910391,
      7.993806104060548,
      6.599996648821785,
      6.180372149191392,
      79.16327969435213,
      0.01661129709394161,
      0.04554285217076821,
      0.0002891879678412773
```

```
      ],
      "bbox": [
        2465000,
        7438000,
        2465999,
        7438999
      ]
    },
    "geometry": {
      "type": "Polygon",
      "coordinates": [
        [
          [
            2465000,
            7438000
          ],
          [
            2465999,
            7438000
          ],
          [
            2465999,
            7438999
          ],
          [
            2465000,
            7438999
          ]
        ]
      ]
    }
  }
```

A GeoJSON encoding containing the Feature Collection is then pushed into a pygeoapi server (https://pygeoapi.io/) with the following configuration file:

```
nb_lidar:
    type: collection
    title: NB Lidar metadata record
    description: NB Lidar Data
    keywords:
        - LIDAR
    links:
        - type: text/html
          rel: canonical
          title: information
          href: https://geonb.snb.ca/li/
          hreflang: en-US
    extents:
        spatial:
            bbox: [-69.05, 44.56, -63.7, 48.07]
            crs: http://www.opengis.net/def/crs/EPSG/9.8.15/2953
        temporal:
            begin: 2011-11-11
            end: null  # or empty (either means open ended)
    providers:
        - type: feature
          name: GeoJSON
          data: tests/data/nb_lidar.json
          id_field: id
```

Records are showing up on the server (http://localhost:5000/collections/nb_lidar/items?f=html)

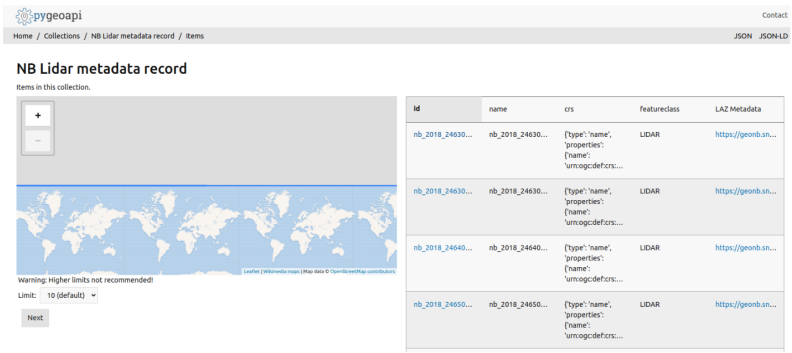


Figure 19. LIDAR Tile Record shown in the pygeoapi server.

Recommendations for the LIDAR records:

1. Add information about annotations and class frequency.
2. A quickview of the tiles as a PNG showing classes could also be interesting.

10.4.4. OGC API - Records Queries for building training set

The objective here is to simulate queries of records that could be useful for ML. Unfortunately queries using CQL are too limited for the LIDAR use case as they cannot deal with vectors or arrays of values.

NOTE	A change request (see: https://github.com/opengeospatial/ogcapi-features/issues/384) has been submitted against the draft OGC API - Features - Part 3: Filtering and the Common Query Language (CQL) [http://docs.opengeospatial.org/DRAFTS/19-079.html] to add support for vectors or arrays of values.

One possibility is to use ElasticSearch [https://www.elastic.co/elasticsearch/] as a provider as it has a good query engine. Instead, the participants chose a Python query using QGIS API [https://qgis.org/api/]. In the code fragment below, the participants directly query the LIDAR collections to identify tiles that contain our class of interest (for instance buildings or water).

These tiles can be loaded directly into QGIS as a vector layer (http://localhost:5000/collections/nb_lidar/items?f=json&limit=20). The QGIS Python API can then be used to query the records directly:

```
vlayer = QgsVectorLayer(
'http://localhost:5000/collections/nb_lidar/items?f=json&limit=20', "my wfs layer",
"ogr")
features = vlayer.getFeatures()
import json
from collections import Counter
cnt_all = Counter()
all_classes = [ 'ground', 'low vegetation', 'medium vegetation', 'high vegetation',
'building', 'water']
selection= list()
Class_of_interest = 'building'
for feature in features:
   # retrieve every feature with its geometry and attributes
   print("Feature ID: ", feature.id())
   class_histograms = json.loads(feature["class_histograms"])
   cnt = Counter()
   for cl in class_histograms:
       # print(cl)
       cnt[cl["name"]] += cl["count"]
   most_common = cnt.most_common()[0]
   least_common = cnt.most_common()[-2]
   missing = [k  for k in all_classes if k not in cnt.keys()]
   print(f'Most common: {most_common} Least common: {least_common}')
   print(f'Missing {missing}')
   cnt_all += cnt
   if Class_of_interest in cnt.keys():
       selection.append(feature)
print(cnt_all)
iface.activeLayer().selectByIds([s.id() for s in selection])
```

Below are two examples of queries requesting some tiles with specific classes:

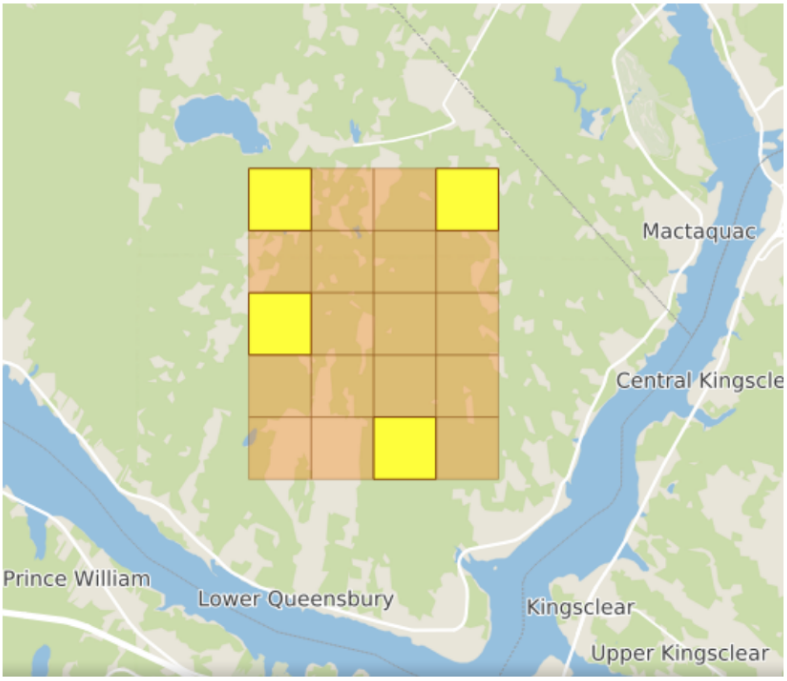


Figure 20. Looking for tiles containing the Building class (Class_of_interest = Building).

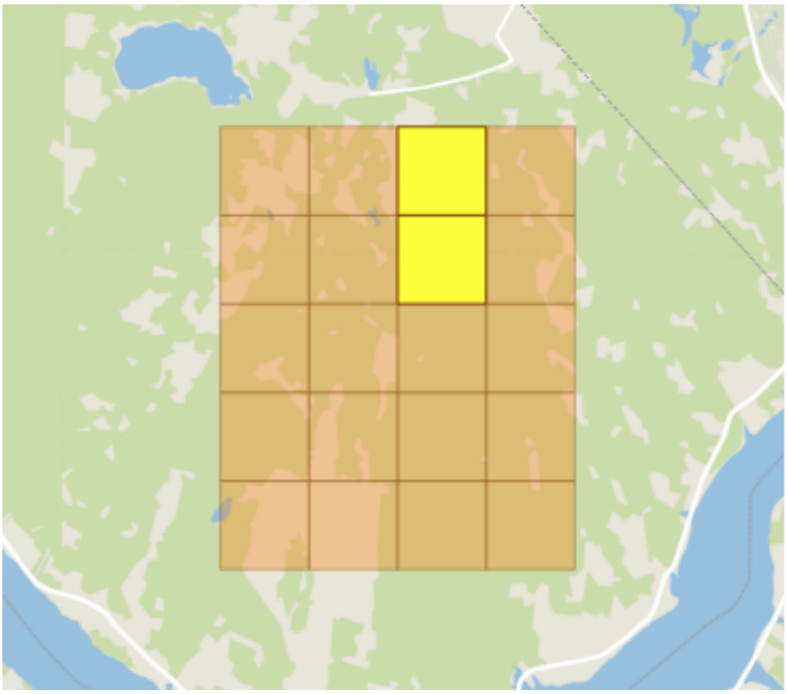


Figure 21. Looking for the Water class (Class_of_interest = Building)..

10.4.5. OGC API - Records queries for ML models

The goal of this task was to enable the querying of a catalogue of trained models. ML models are minimally defined by their architecture and their parameters resulting from the training phase. However, this information is insufficient to form a viable ML pipeline that can be executed on new data. In this case additional information must be provided:

1. Optional pre-processing before calling the inference (format conversion, statistical normalization, etc.)
2. Expected input format (e.g. number of channels, image size, etc.)

3. Semantic information about the model output which maps the model output to the semantic information (e.g. actual class names)
4. Optional post-processing
5. The type of machine learning task such as classification, detection, semantic segmentation, instance detection, etc.

Several ML libraries or frameworks provide ways to package ML pipelines, we can mention MXNet [https://rapidlasso.com/lastools/], TorchServe [https://rapidlasso.com/lastools/], TensorFlowX [https://www.tensorflow.org/tfx], thelper (https://github.com/plstcharles/thelper), MLflow (https://mlflow.org/).

- The ONNX format contains only the model (as a graph) and its parameters. There is no information about pre/post-processing, output description, etc.
- MXNet has a richer model archive:
  - Pre-trained MXNet Model (it can be an ONNX file)
  - A 'signature.json' describing the inputs and outputs of the models.
  - Semantic information as a 'synset.txt' containing the class names and synonyms (a Synset is a special kind of a simple interface that is used to look up words in WordNet (https://wordnet.princeton.edu/). Synset instances are the groupings of synonymous words that express the same concept. Some of the words have only one Synset and some have several).
  - Custom model service files for pre-processing.

MLflow (https://mlflow.org/) provides a fairly complete model development and management framework from model training to model packaging and deployment:

- Mlflow projects (https://mlflow.org/docs/latest/projects.html) provides a way to package models in order to ensure reproducibility.
- MLflow registry (https://mlflow.org/docs/latest/model-registry.html) acts as a model store with search functionalities (https://mlflow.org/docs/latest/model-registry.html#listing-and-searching-mlflow-models).

The testbed participants harvested metadata from a model repo containing MXNet models: https://github.com/awslabs/multi-model-server/blob/master/docs/model_zoo.md

An example is shown below for the Alexnet model:


```
{
       "type": "Feature",
       "id": "alexnet",
       "properties": {
         "name": "alexnet",
         "url": "https://s3.amazonaws.com/model-server/model_archive_1.0/alexnet.mar",
         "content": [
           "/content/models/alexnet/alexnet-symbol.json",
           "/content/models/alexnet/synset.txt",
```

```
        "/content/models/alexnet/model_handler.py",
        "/content/models/alexnet/signature.json",
        "/content/models/alexnet/alexnet-0000.params",
        "/content/models/alexnet/mxnet_model_service.py",
        "/content/models/alexnet/mxnet_vision_service.py",
        "/content/models/alexnet/MAR-INF/MANIFEST.json",
        "/content/models/alexnet/mxnet_utils/__init__.py",
        "/content/models/alexnet/mxnet_utils/nlp.py",
        "/content/models/alexnet/mxnet_utils/ndarray.py",
        "/content/models/alexnet/mxnet_utils/image.py"
      ],
      "signature.json": {
        "inputs": [
          {
            "data_shape": [
              0,
              3,
              224,
              224
            ],
            "data_name": "data_0"
          }
        ],
        "input_type": "image/jpeg",
        "outputs": [
          {
            "data_shape": [
              0,
              1000
            ],
            "data_name": "softmax"
          }
        ],
        "output_type": "application/json"
      },
      "MANIFEST.json": {
        "modelServerVersion": "1.0",
        "specificationVersion": "1.0",
        "model": {
          "handler": "mxnet_vision_service:handle",
          "modelName": "alexnet"
        },
        "runtime": "python",
        "implementationVersion": "1.0"
      },
      "model_handler": [
        "model_handler.py",
        "mxnet_model_service.py",
        "mxnet_vision_service.py",
        "__init__.py",
        "nlp.py",
```

```
            "ndarray.py",
            "image.py"
          ],
          "synset.txt": [
            {
              "class_id": "n01440764",
              "synonims": [
                "tench,",
                "Tinca tinca"
              ]
            }
          ]
        }
}
```

Below are some possible queries that could be useful:

1. Query if the models have the relevant semantic outputs;
2. The machine learning task (classification, semantic segmentation, instance recognition, etc.);
3. The size of the model (number of parameters);
4. Pre-processing or post-processing that must be applied;
5. Performance on some known benchmarks (e.g. ImageNet);

Once the model archive is selected, the content of the .mar archive can be used to initialize a model.

```
# Download pre-trained resnet model - json and params by running following code.
path='http://data.mxnet.io/models/imagenet/'
[mx.test_utils.download(path+'resnet/18-layers/resnet-18-0000.params'),
mx.test_utils.download(path+'resnet/18-layers/resnet-18-symbol.json'),
mx.test_utils.download(path+'synset.txt')]


ctx = mx.cpu()
with open('synset.txt', 'r') as f:
 labels = [l.rstrip() for l in f]
sym, args, aux = mx.model.load_checkpoint('resnet-18', 0)
```

Or if the archive contains an ONNX file it can also be used to initialize a model within MXNet. Note that PyTorch can export in ONNX format but cannot yet import ONNX [https://www.tensorflow.org/tfx].

Some recommendations for the model metadata:

1. The model architecture can also be visualized using mx.viz.plot_network (see Figure 22 below) and could be included in the metadata as a link to a pdf file.
2. Number of parameters (or memory size).
3. A short description similar to this one: https://github.com/dmlc/mxnet-model-gallery/blob/master/imagenet-1k-caffenet.md.

4. Links to the original paper should be added to the model records also.

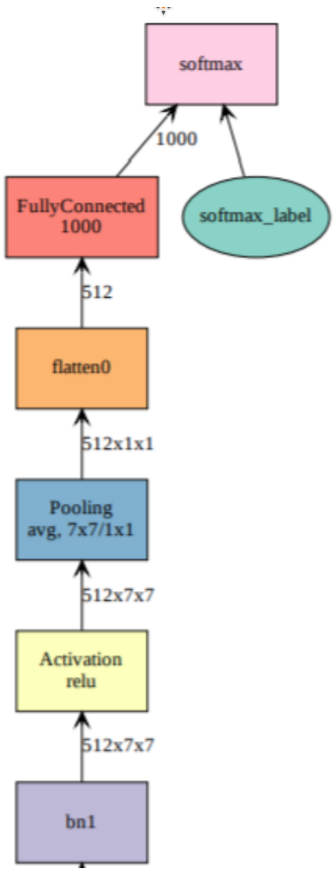


Figure 22. Graph of the top part of a Resnet-18 (partial view)

10.4.6. Inference deployment on ADES/EMS

The LIDAR inference application was packaged using the thelper [https://github.com/plstcharles/thelper] library. The resulting image provides thelper [https://github.com/plstcharles/thelper] (via base image) and all its sub-dependencies (although it is not explicitly required by itself for this application).

Some of the tools from LAStools [https://rapidlasso.com/lastools/] suite are commercial (cannot be used without licence), but laszip specifically is publicly available. The model [https://arxiv.org/abs/2004.11985] targets specific version tracking using remote Git commit within the Dockerfile to ensure traceability in case of future updates.

The model [https://arxiv.org/abs/2004.11985] targets specific version tracking using remote Git commit within the Dockerfile to ensure traceability in case of future updates.

The inference application defined by the Docker container is wrapped by CWL and deployed as process at the following URL location:

https://ogc-ades.crim.ca/ADES/processes/ogc-tb16-lidar

The definition of deployment and example job execution are available within the source code repository.

Once the inference is performed, the new .las is converted in a raster format (.tif) showing the classification values. This application is defined by a docker container wrapped by CWL and

deployed as process at the following URL location:

https://ogc-ades.crim.ca/ADES/processes/las2tif

Example of a successful job execution: https://ogc-ades.crim.ca/ADES/processes/las2tif/jobs/6a22e5e7-adc4-457d-8468-93e02ea6b471

10.5. OGC API - Records server (CubeWerx)

10.5.1. Overview

CubeWerx provided a catalogue for the Testbed-16 ML thread. The purpose of the catalogue was to provide the following services:

- A model catalogue to make trained data models discoverable
- A data catalogue to make training and other data discoverable

The CubeWerx catalogue implements the current draft of the DRAFT OGC API -Records - Part 1: Core [https://htmlpreview.github.io/?https://github.com/opengeospatial/ogcapi-records/blob/master/20-004.html] standard but also includes an implementation of the draft OGC API - Features - Part 3 Filtering and Common Query Language (CQL) [http://docs.opengeospatial.org/DRAFTS/19-079.html] to provide additional, advanced query capability.

The catalogues deployed for the ML Thread can be found at this URL: https://eratosthenes.pvretano.com/cubewerx/cubeserv/default/ogcapi/catalogues.

10.5.2. Summary of OGC API - Records - Part 1 Core

At its core, the OGC API - Records [https://htmlpreview.github.io/?https://github.com/opengeospatial/ogcapi-records/blob/master/20-004.html] specification uses the API defined by the OGC API - Feature - Parts 1: Core [http://docs.opengeospatial.org/is/17-069r3/17-069r3.html] specification (with enhancements) and a datamodel composed of a set of `core` queryables that a catalogue implementation should use to describe resources.

The OGC API - Records - Part 1: Core [https://htmlpreview.github.io/?https://github.com/opengeospatial/ogcapi-records/blob/master/20-004.html] specification defines the following conformance classes:

1. Core: The minimum set of catalogue functionality.
2. Sorting: A parameter to specify sorting.
3. OpenSearch: Additional query parameters that are OpenSearch compatible.
4. JSON: A GeoJSON encoding for a catalogue record.
5. ATOM: An XML-encoding for a catalogue record.
6. HTML: A HTML-encoding for a catalogue record.

10.5.3. Core conformance class

The core conformance class includes a set of core properties also referred to as `core queryables` and

a set of core query parameters.

The list of core queyrables is provided in the following table:

Table 9. Table of Core Queryables

Queryables	Requirement	Description
recordid	M	A unique record identifier assigned by the server.
recordcreated	M	The date this record was created in the server.
recordmodified	M	The most recent date on which the record was changed.
title	M	A human-readable name given to the resource.
description	M	A free-text description of the resource.
keywords	M	A list of keywords or tag associated with the resource.
type	M	The nature or genre of the resource.
language	M	This refers to the natural language used for textual values (i.e. titles, descriptions, etc) of a resource.
externalid	O	An identifier for the resource assigned by an external entity.
modified	O	The more recent date on which the resource was changed.
publisher	O	The entity making the resource available.
themes	O	A knowledge organization system used to classify the resource.
formats	O	A list of available distributions for the resource.
contactpoint	O	An entity to contact about the resource.
license	O	A legal document under which the resource is made available.
rights	O	A statement that concerns all rights not addressed by the license such as a copyright statement.
extent	O	The spatio-temporal coverage of the resource.
links	O	A list of links for navigating the API.

Queryables	Requirement	Description
associations	O	A list of links for accessing the resource or links to other resources associated with this resource.

The list of core query parameters is provided in the following table:

Table 10. Table of Query Parameters

Query Parameter	Description
bbox	A bounding box. If the spatial extent of the record intersects the specified bounding box then the record shall be presented in the response document.
datetime	A time instance or time period. If the temporal extent of the record intersects the specified data/time value then the record shall be presented in the response document.
limit	The number of records to be presented in a response document.
type	A resource type. Only records of the specified type shall be presented in the response document.
q	A space-separated list of search terms. If any server-chosen text field in the record contains 1 or more of the terms listed then this records hall appear in the response set.
externalIds	A comma-separated list of external identifiers. Only records with the specified names shall appear in the response document.
sortby	A comma-separated list of queryables specifying how the records in the response shall be ordered for presentation.

The `bbox` parameter is composed of either 4 or 6 number values that define a bounding box. The default CRS is http://www.opengis.net/def/crs/OGC/1.3/CRS84. Any record whose spatial component intersects the bounding box shall be included in the result set of a query.

Example: ...&bbox=43.5805,-79.6390,43.8782,-79.1569&...

The `datetime` parameter is a string. The format of the string is an ISO8601 date string or period. The string '..' may be used to denote "since the beginning of time" or "to the end of time".

Example: ...&datetime=2019-10-01T13:45:17/2019-10-31T22:10:14&...

The `limit` parameter is used to specify the number of records that shall appear in single response document. If there are additional records, `next` and `prev` links shall be included to allow navigation to the next or previous set of results.

Example: ...&limit=27&...

The `type` parameter is a string. Its value is a type identifier and only records of that type shall appear in the response document.

Example: …&type=http://www.opengis.net/def/object-type/ogc-csw-ebrim/0/dataset&…

The `externalIds` parameter is a comma-separated list of external identifiers. Only records with the specified external identifiers shall appear in a response document.

Example: …&externalIds=id1,id2,id3&…

10.5.4. Sorting conformance class

The sorting conformance class defines the `sortby` parameter. The `sortby` parameter is a comma-separated list of core queryable names. The results in a response document shall be sorted by the specified queryables in the order in which they are specified. Each specified queryable can be additionally qualified to indicate a sort direction.

Example: …&sortby=modified:desc&…

10.5.5. OpenSearch conformance class

The core conformance class provides a minimum query capability via the `bbox`, `datetime`, `type`, `q` and `externalIds` parameters. The OpenSearch conformance class provide a additional level of query capability that is also compatible with the OpenSearch [https://github.com/dewitt/opensearch/blob/master/opensearch-1-1-draft-6.mdhttps://github.com/dewitt/opensearch/blob/master/opensearch-1-1-draft-6.md] specification and the OGC® OpenSearch Geo and Time Extensions [https://portal.opengeospatial.org/files/?artifact_id=56866].

The OGC® OpenSearch Geo and Time Extensions extend OpenSearch to provide the following query capabilities:

- Arbitrary geometry search;
- Point and radius search;
- Support for spatial operators;
- Support for temporal operators;
- Search by place name;
- Temporal search;
- Support for temporal operators;

The OpenSearch conformance class is presented here for completeness but was not used by ML Thread participants.

10.5.6. JSON conformance class

The JSON conformance class defines a JSON encoding for a catalogue record by mapping the core queryables into a GeoJSON [https://tools.ietf.org/html/rfc7946] object. This was the primary output format used by the Thread participants. The following is an example of a JSON-encoded catalogue record:

```
{
```

```
    "id": "urn:uuid:452b6afc-c062-11ea-aa84-53b2ade463d0",
    "resourceId": "NFIS_High_Resolution_Forest_Data",
    "type": "feature",
    "@type": "urn:cw:def:ebRIM-ObjectType:CubeWerx:WMS:Layer",
    "geometry": {
      "type": "Polygon",
      "coordinates": [[[-170,40],[-170,60],[0,60],[0,40],[-170,40]]]
    },
    "properties": {
      "title": "High Resolution Satellite Forest Information for Canada",
      "description": "NFIS Project Office.  This Web services are for forest change
products that represents the first wall-to-wall characterization of wildfire and
harvest in Canada at a spatial resolution commensurate with human impacts.  The
information outcomes represents 25 years of stand replacing change in Canada's forests
derived from a single consistent spatially-explicit data source and derived in a fully
automated manner.",
      "accessURLTemplate": "https://opendata.nfis.org/mapserver/cgi-
bin/wms_change.cgi?version=1.3.0&request=GetMap&layers=NFIS_High_Resolution_Forest_Dat
a&styles=%7Bstyles%7D&crs=%7Bcrs%7D&bbox=%7Bbbox%7D&width=%7Bwidth%7D&height=%7Bheight
%7D&format=%7Bformat%7D",
      "urn:cw:def:ebRIM-SlotName:crs": ["EPSG:4617","EPSG:3979","EPSG:3978",
"EPSG:3857","EPSG:42304","EPSG:4326","EPSG:4269","EPSG:42101"],
      "urn:cw:def:ebRIM-SlotName:keyword": "Change",
      "urn:cw:def:ebRIM-SlotName:outputFormat": ["image/tiff","image/png; mode=8bit"
,"image/jpeg","image/png"],
      "urn:cw:def:ebRIM-SlotName:queryable": 0
    },
    "associations": [
      {
        "href":
"https://eratosthenes.pvretano.com/cubewerx/cubeserv/default/ogcapi/catalogues/collect
ions/tb16cat/items/urn:uuid:45334ccc-c062-11ea-bdc3-ebf0bb16ce92",
        "rel": "ParentOf",
        "type": "urn:cw:def:ebRIM-ObjectType:CubeWerx:WMS:Layer",
        "title": "Parent Of WMS Layer \"ca_prov_r \""
      },
      {
        "href":
"https://eratosthenes.pvretano.com/cubewerx/cubeserv/default/ogcapi/catalogues/collect
ions/tb16cat/items/urn:uuid:4535ce98-c062-11ea-bff2-87ce08cbef28",
        "rel": "ParentOf",
        "type": "urn:cw:def:ebRIM-ObjectType:CubeWerx:WMS:Layer",
        "title": "Parent Of WMS Layer \"ca_change_nochange_r1984 \""
      },
      {
        "href":
"https://eratosthenes.pvretano.com/cubewerx/cubeserv/default/ogcapi/catalogues/collect
ions/tb16cat/items/urn:uuid:4538463c-c062-11ea-8f4c-8f86afc502a1",
        "rel": "ParentOf",
        "type": "urn:cw:def:ebRIM-ObjectType:CubeWerx:WMS:Layer",
        "title": "Parent Of WMS Layer \"ca_change_year_r1984 \""
```

```
      },
      {
        "href":
"https://eratosthenes.pvretano.com/cubewerx/cubeserv/default/ogcapi/catalogues/collect
ions/tb16cat/items/urn:uuid:453a8fc8-c062-11ea-a121-337a0f5ce5ea",
        "rel": "ParentOf",
        "type": "urn:cw:def:ebRIM-ObjectType:CubeWerx:WMS:Layer",
        "title": "Parent Of WMS Layer \"ca_change_type_r \""
      },
      {
        "href":
"https://eratosthenes.pvretano.com/cubewerx/cubeserv/default/ogcapi/catalogues/collect
ions/tb16cat/items/urn:uuid:453c9570-c062-11ea-bd9d-c74b345a6322",
        "rel": "ParentOf",
        "type": "urn:cw:def:ebRIM-ObjectType:CubeWerx:WMS:Layer",
        "title": "Parent Of WMS Layer \"ca_change_nochange_r2012 \""
      },
      {
        "href":
"https://eratosthenes.pvretano.com/cubewerx/cubeserv/default/ogcapi/catalogues/collect
ions/tb16cat/items/urn:uuid:453e6788-c062-11ea-9ad0-7b8d695e5ebe",
        "rel": "ParentOf",
        "type": "urn:cw:def:ebRIM-ObjectType:CubeWerx:WMS:Layer",
        "title": "Parent Of WMS Layer \"ca_change_year_r2012 \""
      },
      {
        "href":
"https://eratosthenes.pvretano.com/cubewerx/cubeserv/default/ogcapi/catalogues/collect
ions/tb16cat/items/urn:uuid:454046fc-c062-11ea-bbb9-3fd8b87ab6fe",
        "rel": "ParentOf",
        "type": "urn:cw:def:ebRIM-ObjectType:CubeWerx:WMS:Layer",
        "title": "Parent Of WMS Layer \"ca_change_type_r2012 \""
      },
      {
        "href":
"https://eratosthenes.pvretano.com/cubewerx/cubeserv/default/ogcapi/catalogues/collect
ions/tb16cat/items/urn:uuid:45420b54-c062-11ea-ae63-e741378866ae",
        "rel": "ParentOf",
        "type": "urn:cw:def:ebRIM-ObjectType:CubeWerx:WMS:Layer",
        "title": "Parent Of WMS Layer \"ca_rgb_2015_wkg_r \""
      },
      {
        "href":
"https://eratosthenes.pvretano.com/cubewerx/cubeserv/default/ogcapi/catalogues/collect
ions/tb16cat/items/urn:uuid:44f58374-c062-11ea-86c1-cf8ce91dab91",
        "rel": "OperatesOn",
        "type": "urn:oasis:names:tc:ebxml-regrep:ObjectType:RegistryObject:Service",
        "title": "Operated upon Service \"High Resolution Satellite Forest
Information for Canada \""
      }
    ]
```

```
    }
```

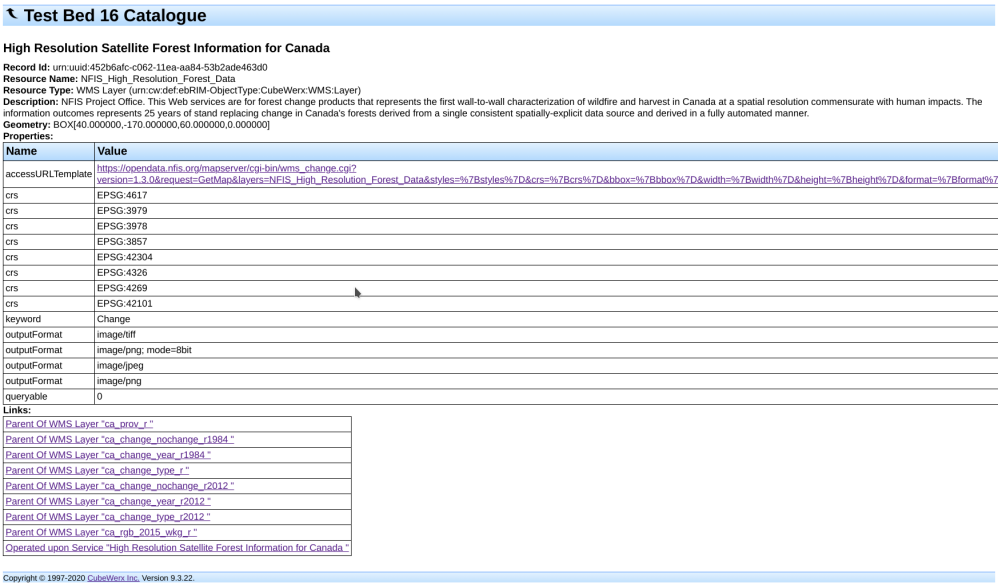
10.5.7. ATOM conformance class

The ATOM conformance class defines requirements for an XML output format of catalogue records that conform to The Atom Syndication Format [https://tools.ietf.org/html/rfc4287].

The ATOM conformance class is presented here for completeness but was not used by Testbed-16 ML Thread participants.

10.5.8. HTML conformance class

The HTML conformance class defines requirements for an HTML output format of catalogue records. The requirements basically recommends that catalogues implementing the OGC API - Records - Part 1: Core [https://htmlpreview.github.io/?https://github.com/opengeospatial/ogcapi-records/blob/master/20-004.html] specification provide HTML as a supported output format. The following example illustrates a catalogue record encoded as HTML:



Test Bed 16 Catalogue

High Resolution Satellite Forest Information for Canada

Record Id: urn:uuid:452b6afc-c062-11ea-aa84-53b2ade463d0
Resource Name: NFIS_High_Resolution_Forest_Data
Resource Type: WMS Layer (urn:cw:def:ebRIM-ObjectType:CubeWerx:WMS:Layer)
Description: NFIS Project Office. This Web services are for forest change products that represents the first wall-to-wall characterization of wildfire and harvest in Canada at a spatial resolution commensurate with human impacts. The information outcomes represents 25 years of stand replacing change in Canada's forests derived from a single consistent spatially-explicit data source and derived in a fully automated manner.
Geometry: BOX[40.000000,-170.000000,60.000000,0.000000]
Properties:

Name	Value
accessURLTemplate	https://opendata.nfis.org/mapserver/cgi-bin/wms_change.cgi?version=1.3.0&request=GetMap&layers=NFIS_High_Resolution_Forest_Data&styles=%7Bstyles%7D&crs=%7Bcrs%7D&bbox=%7Bbbox%7D&width=%7Bwidth%7D&height=%7Bheight%7D&format=%7Bformat%7
crs	EPSG:4617
crs	EPSG:3979
crs	EPSG:3978
crs	EPSG:3857
crs	EPSG:42304
crs	EPSG:4326
crs	EPSG:4269
crs	EPSG:42101
keyword	Change
outputFormat	image/tiff
outputFormat	image/png; mode=8bit
outputFormat	image/jpeg
outputFormat	image/png
queryable	0

Links:

Parent Of WMS Layer "ca_prov_r."
Parent Of WMS Layer "ca_change_nochange_r1984."
Parent Of WMS Layer "ca_change_year_r1984."
Parent Of WMS Layer "ca_change_type_r."
Parent Of WMS Layer "ca_change_nochange_r2012."
Parent Of WMS Layer "ca_change_year_r2012."
Parent Of WMS Layer "ca_change_type_r2012."
Parent Of WMS Layer "ca_rgb_2015_wkg_r."
Operated upon Service "High Resolution Satellite Forest Information for Canada."

Copyright © 1997-2020 CubeWerx Inc. Version 9.3.22.

Figure 23. Sample HTML-encoded catalogue record

10.5.9. Extensions for the ML Thread

10.5.9.1. Overview

The section discusses extensions made by CubeWerx to their catalogue in order to satisfy participant requirements. In most cases, the extensions were added to make binding to a discovered resource (e.g. a WMS layer) more convenient. In some cases, the extensions were added to overcome some complicated access mechanism that was orthogonally related to the goals of the ML Thread (e.g. fetching data from Amazon's S3).

10.5.9.2. Retrieving a map (accessURLTemplate)

The map servers provided by NRCan for the Thread did not provide an OGC API interface. Rather they implemented the OpenGIS Web Map Service (WMS) Implementation Specification

[http://portal.opengeospatial.org/files/?artifact_id=14416]. In order to alleviate clients from having to possess in-depth knowledge of this specification, a URL template was added to each WMS layer record in the catalogues via the `accessURLTemplate` queryable. The function of this queryable is to provide a URL template of a GetMap request for the corresponding layer that allows the client to construct a valid GetMap request to fetch layer data. Here is an example of such a URL template:

```
https://opendata.nfis.org/geoserver/OGC_TB15_ML_D104/ows?SERVICE=WMS&version=1.3.0&req
uest=GetMap&layers=OGCTestbed15%20D104%20Quebec%20river-
lake%20vectorization%20ML&styles={styles}&crs={crs}&bbox={bbox}&width={width}&height={
height}&format={format}
```

In order to construct a GetMap request for the layer, the client needs to only provide values for the substitution variables. Lists of values are provided in the corresponding catalogue record for the styles, crs and format substitution variables so the client does not need to guess values for those variables. The remaining variables, width, height and bbox, are filled in with values suited to the client's purpose (i.e. the width and height of map they desire in the area of interest). In this way, with very little a-priori knowledge, a client can construct a WMS GetMap request and fetch the desired layer data.

10.5.9.3. Retrieving a map legend (legendURL)

The value of the `legendURL` queryable is a link that is a GetLegendGraphic request for a discovered layer. This URL allows a client to retrieve a legend graphic for a layer discovered in a catalogue search. In the Thread these legend graphics were used to provide classification categories for training ML models from data discovered using the catalogue; specifically Sentinel-1 data.

10.5.9.4. Retrieving a style (styleURL)

The value of the `styleURL` query parameter is a link that is a GetStyle request for a discovered layer. This URL allows a client to retrieve an SLD definition associated with a layer discovered in a catalogue search. The intent was to use this Styled Layer Descriptor (SLD) [https://www.ogc.org/standards/sld] information to derive the classifications or categories associated with various colors in a raster image for the purpose of training an ML model. In the end however, this approach did not work very well because the GetStyle operation is not mandatory in the WMS specification and none of the WMS's used in the Thread implemented it. Instead, the RGB values of the color scale illustrated in a legend graphic, obtained via the `legendURL`, were used for this purpose.

10.5.9.5. Retrieving Sentinel-1 from Amazon's S3 (enclosure)

Each catalogue record is accompanied by a list of links to resources related to that record. For example, a download link might be included to allow the resource being described by the catalogue record to be retrieved from its source. Such a link is identified using the link relation `enclosure`.

The catalogue of Sentinel-1 products deployed by CubeWerx for the ML thread of Testbed-16, catalogued product metadata harvested from the Sentinel-1 Registry of Open Data on AWS [https://registry.opendata.aws/sentinel-1/]. As such all the download links for the data are references to Amazon's Simple Storage Service [https://aws.amazon.com/s3/?nc2=h_ql_prod_st_s3] (S3) locations. The mechanics of resolving an S3 reference are not trivial requiring Amazon credentials and either

special libraries provided by Amazon, the use of Amazon's free command line tool (i.e. AWS) or an in-depth knowledge of Amazon's download URL scheme and signing protocol [https://docs.aws.amazon.com/general/latest/gr/sigv4_signing.html] in order construct a valid download URL. To alleviate ML thread participants of the burden of being familiar with at least one of these access mechanisms to retrieve a discovered Sentinel-1 product, an `enclosure` link was included with each record that retrieved the Sentinel product from a local copy stored on a CubeWerx server that was open to ML Thread participants.

Chapter 11. Visualization of ML Results

11.1. Overview

With planning and response activities for wildland fire events, it is critical that stakeholders (e.g. planners, first responders, residents, policy makers) are able to visualize related geospatial information quickly and accurately. Typically, such visualization requires users to have access to specialized software and skills.

This thread task evaluated the utility of MapML for providing a geospatial information visualization and interaction interface in the context of wildland fire planning and response.

MapML enables viewing and interaction with geospatial information within web browsers which are ubiquitous across multiple devices.

All geospatial information results from the ML models used in Testbed-16 were published to servers that implement the OGC Web Service or OGC API interfaces and are additionally augmented to support MapML.

11.2. MapML Client 1 (D130 - ASU)

11.2.1. Introduction of MapML Client

This client is developed to visualize the geographical data which is published via the MapML specification, generated by machine learning and deep learning modules. MapML is short for Map Markup Language. It encodes map information using text format for the World Wide Web [1]. The author of a HTML document should be able to use the <map> element and the link of a MapML document to declare a sub-area of HTML page to create a two-dimensional map with general map operations, such as panning and zooming. The MapML specification should ultimately provide a convenient and declarative manner for HTML authors to include maps in a web page. However, finding a browser that supports the MapML application is currently difficult. Therefore, a common solution is to implement the MapML specification via JavaScript. This client implementation was developed based on a JavaScript library called Web-Map-Custom-Element [https://github.com/Maps4HTML/Web-Map-Custom-Element] (WMCE).

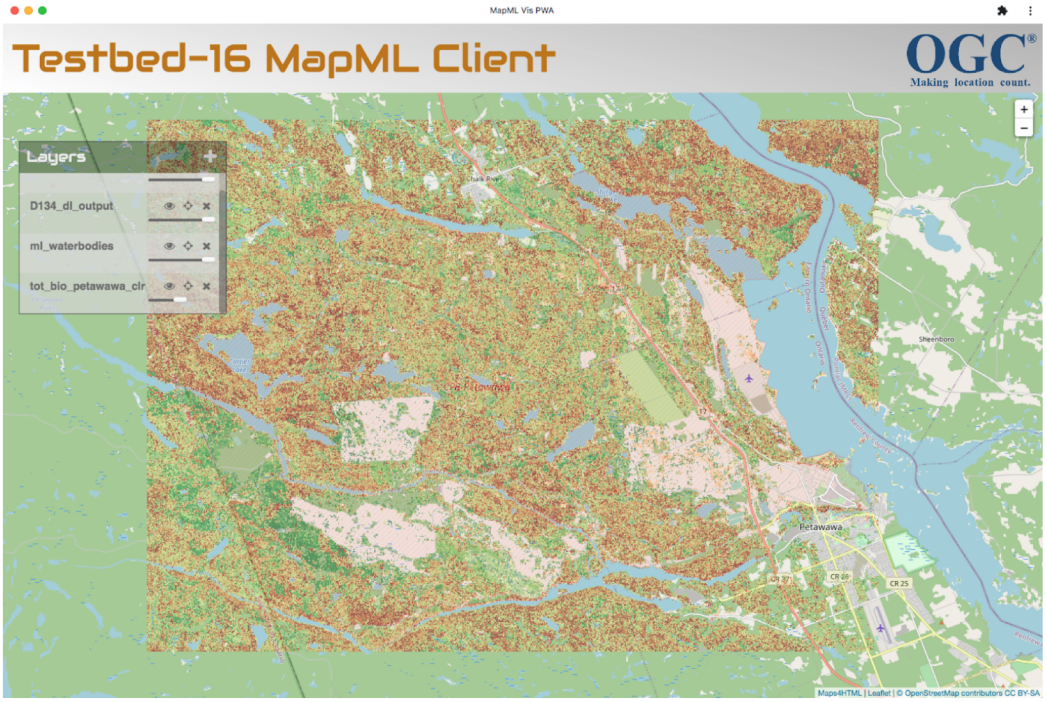


Figure 24. A screenshot of MapML Client 1

Except for the supported general map operations, this client realized the multi-layer management. With the client, users can view all the maps/layers they are interested in simultaneously displayed over any arbitrary basemap. Many layer operations are implemented, such as adding, removing, toggling visibility, adjusting transparency, zooming to the selected layer, and etc., to meet the requirements of different application scenarios. Finally, this client is wrapped and published as a Progressive Web Application [https://en.wikipedia.org/wiki/Progressive_web_application] (PWA), so that the client acquires following advantages:

1. Working like a native application on users' OS.
2. Loading much faster than a traditional web application.
3. The capability of working offline.
4. The capability of deploying to mobile environments with less effort.

11.2.2. Including Maps with MapML

With the WMCE library, maps can be included in the HTML document just following MapML specification. An example using the `<map>` element is demonstrated in the below code snippet.

```
    <map is="web-map"
         projection="OSMTILE"
         zoom="4" lat="42.13082130188811"
         lon="-102.39257812500001"
         controls=""
         controlslist="nofullscreen"
         style="display: block;">
         <layer- label="OpenStreetMap"
                 src="https://geogratis.gc.ca/mapml/en/osmtile/osm/" checked>
         </layer->
         <layer- label="tot_bio_petawawa_clr"
                 src=
"http://cici.lab.asu.edu/geoserver217/mapml/sf:tot_bio_petawawa_clr/OSMTILE?style="
checked>
         </layer->
    </map>
```

The <map> element declares a sub-area in the web page as a map viewer for displaying maps. The attributes of the element indicate the projection information and how the viewer is initialized, such as zoom level and center location.

By adding child <layer> elements to the <map> element, a certain map/layer can be included in the map viewer. This step can be accomplished declaratively while drafting the HTML document. It also allows developers to inject the <layer> element programmatically and dynamically, and this feature was used in the ASU client to add functions for adding and removing maps.

When declaring a <layer> element, only the "src" attribute is required, which provides the link directly to the MapML document. The label attribute is optional. This information can be automatically extracted from MapML metadata. Similar to other HTML elements, such as <img>, the source link of MapML can be hosted in any servers, from any origin. For instance, the two layers included in the above code snippet are hosted in different servers, from the US and Canada respectively.

11.2.3. MapML Metadata Acquisition

The WMCE library extracts most metadata from the MapML documents and is accessible via <layer> element object. However, while developing the client based on WMCE library, the participants encountered different fashions of exposing the MapML metadata. Some simple information such as title and legend information can be acquired by a straightforward format, a short plain text or a URL. However, the complex information such as layer extent is wrapped in a MicroXML. Below is an example of a MicroXML of layer extent:

```
   <extent units="OSMTILE">
      <input name="z" type="zoom" value="18" min="0" max="18"/>
      <input name="xmin" type="location" rel="map" position="top-left" axis="easting"
units="pcrs" min="-8240672.435241637" max="-8227290.1626559235"/>
      <input name="ymin" type="location" rel="map" position="bottom-left" axis=
"northing" units="pcrs" min="4965875.319295468" max="4994389.518266143"/>
      <input name="xmax" type="location" rel="map" position="top-right" axis="easting"
units="pcrs" min="-8240672.435241637" max="-8227290.1626559235"/>
      <input name="ymax" type="location" rel="map" position="top-left" axis="northing"
units="pcrs" min="4965875.319295468" max="4994389.518266143"/>
      <input name="w" type="width" min="1" max="10000"/>
      <input name="h" type="height" min="1" max="10000"/>
      <link tref="http://cici.lab.asu.edu/geoserver217/tiger/wms?version=1.3.0
&amp;service=WMS&amp;request=GetMap&amp;crs=urn:x-ogc:def:crs:EPSG:3857
&amp;layers=tiger_roads&amp;styles=line&amp;bbox={xmin},{ymin},{xmax},{ymax}&amp;forma
t=image/png&amp;transparent=true&amp;width={w}&amp;height={h}" rel="image"/>
      <input name="i" type="location" axis="i" units="map"/>
      <input name="j" type="location" axis="j" units="map"/>
      <link tref="http://cici.lab.asu.edu/geoserver217/tiger/wms?version=1.3.0
&amp;service=WMS&amp;request=GetFeatureInfo&amp;FEATURE_COUNT=50&amp;crs=urn:x-
ogc:def:crs:EPSG:3857&amp;layers=tiger_roads&amp;query_layers=tiger_roads&amp;styles=l
ine&amp;bbox={xmin},{ymin},{xmax},{ymax}&amp;width={w}&amp;height={h}&amp;info_format=
text/mapml&amp;transparent=true&amp;x={i}&amp;y={j}" rel="query"/>
</extent>
```

This requires applications to implement their own module to extract the extent coordinates, which should be a general function included in the library. Moreover, only the projected coordinates are provided in the MicroXML. Other supported coordinate systems are missing, such as the commonly used geographic coordinates. Communicating with WMCE side, all mentioned issues were fixed. The extent information is now wrapped in an object, shown as follows:

```
extent:
        bottomRight:
            gcrs: {horizontal: -77.25408105847575, vertical: 45.85534888786863}
            pcrs: {horizontal: -8599884.9651318, vertical: 5757199.001661819}
            tcrs: Array(25)
            tilematrix: Array(25)
        topLeft:
            gcrs: {horizontal: -77.616780646139, vertical: 46.0388387807033}
            pcrs: {horizontal: -8640260.498541405, vertical: 5786575.348034253}
            tcrs: Array(25)
            tilematrix: Array(25)
        projection: "OSMTILE"
        zoom: {minZoom: 0, maxZoom: 24, minNativeZoom: 0, maxNativeZoom: 18}
```

Now, extent coordinates are easily accessible, and multiple coordinates systems are supported.

11.2.4. Map Publishing and Customization

To examine the functions of the ASU client and explore the capability of map customization in the context of MapML based implementation, ASU obtained some sample output datasets from machine learning and deep learning modules from the other Testbed-16 ML Thread participants. The output samples are all raster data and formatted as GeoTIFF. They are published via an OGC Web Coverage Service [https://www.ogc.org/standards/wcs] (WCS) endpoint leveraging GeoServer. A GeoServer MapML plug-in [2] wraps the published OGC services into MapML, which can be directly visualized in the client. The code snippet below presents an example of MapML document response from GeoServer:

```
   <mapml>
      <head>
         <title>tot_bio_petawawa_clr</title>
         <base href="http://cici.lab.asu.edu/geoserver217/mapml"/>
         <meta charset="utf-8"/>
         <meta content="text/mapml;projection=OSMTILE" http-equiv="Content-Type"/>
         <link href=
"http://cici.lab.asu.edu/geoserver217/mapml/sf:tot_bio_petawawa_clr/OSMTILE?style="
rel="self style" title="raster"/>
         <link href=
"http://cici.lab.asu.edu/geoserver217/mapml/sf:tot_bio_petawawa_clr/CBMTILE?style="
rel="alternate" projection="CBMTILE"/>
         <link href=
"http://cici.lab.asu.edu/geoserver217/mapml/sf:tot_bio_petawawa_clr/APSTILE?style="
rel="alternate" projection="APSTILE"/>
         <link href=
"http://cici.lab.asu.edu/geoserver217/mapml/sf:tot_bio_petawawa_clr/WGS84?style=" rel
="alternate" projection="WGS84"/>
      </head>
      <body>
         <extent units="OSMTILE">
            <input name="z" type="zoom" value="18" min="0" max="18"/>
            <input name="xmin" type="location" rel="map" position="top-left" axis=
"easting" units="pcrs" min="-8640260.498541405" max="-8599884.9651318"/>
            <input name="ymin" type="location" rel="map" position="bottom-left" axis=
"northing" units="pcrs" min="5757199.001661819" max="5786575.348034253"/>
            <input name="xmax" type="location" rel="map" position="top-right" axis=
"easting" units="pcrs" min="-8640260.498541405" max="-8599884.9651318"/>
            <input name="ymax" type="location" rel="map" position="top-left" axis=
"northing" units="pcrs" min="5757199.001661819" max="5786575.348034253"/>
            <input name="w" type="width" min="1" max="10000"/>
            <input name="h" type="height" min="1" max="10000"/>
            <link tref="http://cici.lab.asu.edu/geoserver217/sf/wms?version=1.3.0
&amp;service=WMS&amp;request=GetMap&amp;crs=urn:x-ogc:def:crs:EPSG:3857
&amp;layers=tot_bio_petawawa_clr&amp;styles=&amp;bbox={xmin},{ymin},{xmax},{ymax}&amp;
format=image/png&amp;transparent=true&amp;width={w}&amp;height={h}" rel="image"/>
            <input name="i" type="location" axis="i" units="map"/>
            <input name="j" type="location" axis="j" units="map"/>
            <link tref="http://cici.lab.asu.edu/geoserver217/sf/wms?version=1.3.0
&amp;service=WMS&amp;request=GetFeatureInfo&amp;FEATURE_COUNT=50&amp;crs=urn:x-
ogc:def:crs:EPSG:3857&amp;layers=tot_bio_petawawa_clr&amp;query_layers=tot_bio_petawaw
a_clr&amp;styles=&amp;bbox={xmin},{ymin},{xmax},{ymax}&amp;width={w}&amp;height={h}&am
p;info_format=text/mapml&amp;transparent=true&amp;x={i}&amp;y={j}" rel="query"/>
         </extent>
      </body>
   </mapml>
```

This MapML document publishes the total biomass of the Petawawa area dataset. This document is mainly composed of title information, available projections and corresponding links, and the extent

information. The following figure demonstrates how this layer is displayed in the ASU client.

From the MapML document it is evident that the tile data required to render the map on the page will be retrieved from a WMS [https://www.ogc.org/standards/wms] endpoint. The WMS link is shown in the `<link>` element which is a child of the `<extent>` element in the MapML document above. The tile images retrieved from the WMS [https://www.ogc.org/standards/wms] are rendered from the original WCS [https://www.ogc.org/standards/wms] endpoint when publishing the data. Therefore customizing the style of the layer that is published via MapML on the client side is challenging because the client only has access to the rendered images rather than the underlying source data. The settled-upon solution to this styling problem was to edit the original GeoTIFF file to achieve certain style customizations and to publish the result as another MapML layer. The following figure presents the MapML visualization of the original total biomass layer and the one rendered with a red-yellow-green color ramp which indicates the value range from low to high.

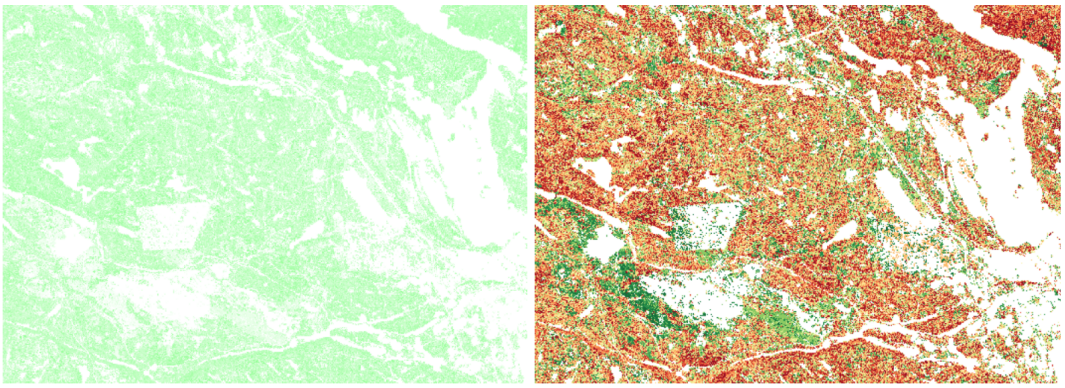


Figure 25. MapML layer customization

Left: original layer; Right: customized layer

11.3. MapML Client 2 (D131 - Bocoup)

11.3.1. Standardizing Web Maps to Increase Adoption among Browser Vendors

The objectives for this deliverable were to:

- Evaluate the MapML proposal and provide feedback,
- Identify low level primitives to standardize, and
- Discuss with stakeholders and document impact for a geospatial community audience.

Potential stakeholders included:

- Browser vendors,
- Web maps library and framework authors,
- Web developers,
- Geospatial community, and
- Accessibility community.

One goal of this work was to begin the process of gaining agreement from two or more browser implementers by focusing on a small primitive subset and getting feedback from framework authors and/or web developers.

A second goal was to follow the Extensible Web Manifesto [https://extensiblewebmanifesto.org/], the WHATWG process [https://whatwg.org/faq] and the W3C's HTML Design Principles [https://www.w3.org/TR/html-design-principles/], and browser engines' processes for shipping new features (e.g., the Chromium intent process [https://www.chromium.org/blink/launching-features]).

The work involved four streams:

1. Review documentation, proposals, and make recommendations,
2. Identify missing primitives in the web platform that would benefit JavaScript libraries and the polyfill capability,
3. Interview web maps library/frameworks authors and web developers, and
4. Summarize feedback and impact for the geospatial community audience.

The following documents were reviewed as part of this work (each item is described in more detail below):

- The MapML (Map Markup Language) proposal [https://github.com/Maps4HTML/MapML-Proposal] (explainer),
- The HTML <map> Element proposal [https://maps4html.org/MapML/spec/#the-map-element-0],
- Map Markup Language [https://maps4html.org/MapML/spec/], and
- Use Cases and Requirements for Standardizing Web Maps [https://maps4html.org/HTML-Map-Element-UseCases-Requirements/].

This review led to reporting several issues including with feedback and recommendations for a process for standardizing web maps to increase the likelihood of adoption among browser vendors. Details are provided below. The review also identified two missing web platform primitives that are fundamental to web maps' interaction model: panning and zooming. An issue was also reported to the W3C Cascading Style Sheets Working Group to initiate a discussion.

11.3.2. MapML Explainer

An "explainer" is a short design document with code examples showing how the solution is intended to be used. The intended audience is the web developer. The W3C Technical Architecture Group provides a document on what the purpose of explainers and what they should contain [https://w3ctag.github.io/explainers].

Based on the evaluation the current MapML (Map Markup Language) proposal [https://github.com/Maps4HTML/MapML-Proposal] is not an effective explainer, and should be rewritten for clarity and brevity. The reasons are:

- Does not clearly explain the problem.
- Makes various claims that are unclear how they follow from one to the other.
- Lacks separation between problem description and the proposal.

- The document is too long.

The full review is available at MapML-Proposal issue #17 [https://github.com/Maps4HTML/MapML-Proposal/issues/17].

11.3.3. The HTML <map> Element proposal

The HTML <map> Element proposal [https://maps4html.org/MapML/spec/#the-map-element-0] redefines the existing HTML <map> and <area> elements, and defines new <layer> element.

The finding was that the model for client-side image maps (the existing semantic of the <map> element) and web maps (the proposed new semantic) are fundamentally different. For the former, an <img> element renders the image, and the <map> only provides the association between the <img> and the <area> elements, while for web maps the <map> element owns the rendering and interaction model. For this reason, the recommendation is not to reuse the `<map>` element for web maps.

As discussed in the full review at HTML-Map-Element issue #97 [https://github.com/Maps4HTML/MapML/issues/97] there are additional problems with reusing `<map>`.

Sixteen specific issues [https://github.com/Maps4HTML/MapML/issues?q=is:issue+author:zcorpan+created:2020-06-29] were also raised, with questions and feedback on web compatibility, API design choices, and technical errors.

11.3.4. Map Markup Language

The Map Markup Language [https://maps4html.org/MapML/spec/] specification defines a new markup language, as a MicroXML [https://dvcs.w3.org/hg/microxml/raw-file/tip/spec/microxml.html] format but inspired by HTML. Being MicroXML means that it is XML but without using any XML namespace.

In this blog post [https://www.w3.org/community/maps4html/2019/12/09/the-design-of-mapml/], there is a statement that MicroXML might not be the right solution, and instead suggests something more HTML-like: “The new document type should be readily parseable with a (minimally) enhanced HTML parser”.

In the `<map>` element proposal, it is defined that the <layer> element can optionally contain inline MapML markup.

For these ideas to work, all MapML elements need to be HTML elements directly. MicroXML cannot be used inline in an HTML document. By changing MapML to be a direct extension of HTML, the HTML parser can be used without changes.

The implications of this recommendation are reconsideration of the MIME type, the doctype, the root element, and audit all MapML elements for either reuse and extend an existing HTML element, or rename to avoid clashing with HTML.

The full review is available at MapML issue #70 [https://github.com/Maps4HTML/MapML/issues/70].

Another aspect of the MapML specification is its statement that it should be possible to style MapML documents CSS. This sounds good but it is unclear how this would work. In particular, CSS may

need new primitives and possibly a new rendering model, such as panning and zooming. Rendering MapML content should then be defined in terms of CSS (see: MapML issue #71 [https://github.com/Maps4HTML/MapML/issues/71]).

11.3.5. Use Cases and Requirements for Standardizing Web Maps

The Use Cases and Requirements for Standardizing Web Maps [https://maps4html.org/HTML-Map-Element-UseCases-Requirements/] document has excellent structure and clarity. It does not leave the reader with questions and concerns as reading the explainer did. This document also provides a process for the work, and a plan for accessibility, privacy, security. There is no feedback for improvements to this document. A summary of this document would make for an excellent part of a new explainer.

11.3.5.1. Process Recommendations

With the goal of making web maps a first-class feature on the web platform, to having the relevant stakeholders at the table throughout the process is imperative. Designing a solution to be implemented in browser engines without having browser engine representatives involved is unlikely to be successful.

Following the informal process outlined in the WHATWG FAQ for adding features to the web platform [https://whatwg.org/faq#adding-new-features] (with one modification) is recommended. This FAQ states that documenting the use cases and requirements comes first, then research existing solutions and propose new solutions, then evaluate how well each solution solves the use cases and meets the requirements.

The suggested modification is to get browser vendors and maps vendors to agree to the use cases and requirements, and take part in the design of the proposed solution.

Considerations for accessibility, privacy, and security should be foundational in the design. Adding them as an afterthought usually results in suboptimal results at best. At worst, some vendors may refuse to implement or even get involved. Work together with subject matter experts early in the design process is recommended.

Following the Extensible Web Manifesto [https://extensiblewebmanifesto.org/] is also recommended. Start with standardizing underlying primitives that can be used for implementing the high-level feature (both in author-level JavaScript and in the browser engine). This approach does not seem to be currently followed by the proposed specifications, as they only describe high-level features.

11.3.5.2. Identify Missing Primitives

The following primitives were identified as missing in the web platform. Note that this is not an exhaustive list of missing primitives that web maps would need- there are undoubtedly others.

- Panning. Scrolling is similar, but not identical. Scrolling areas have edges.
- Zooming. While browsers support “page zoom” and “pinch zoom” [https://www.w3.org/TR/cssom-view-1/%23zooming#zooming], there is no per-element zooming primitive.

To initiate a discussion about these missing primitives, an issue was reported to the CSS Working

Group [https://github.com/w3c/csswg-drafts/issues/5275].

11.3.6. Maps for the Web Workshop

The team participated in the joint W3C/OGC Maps for the Web Workshop [https://www.w3.org/2020/maps/agenda] in September-October 2020, and held a presentation to present the review of the MapML proposal:

- Presentation in YouTube [https://youtu.be/zJkJcvmUA5o?t=2089] (duration six minutes).
- Slide deck in slides.com [https://slides.com/zcorpan/bocoup-review-mapml/].
- Discussion in Discourse [https://discourse.wicg.io/t/the-mapml-proposal-goals-and-details/4848].
- Bocoup's position statement for the workshop [https://www.w3.org/2020/maps/supporting-material-uploads/position-statements/Simon_Pieters-Bocoup.pdf] (in w3.org).

11.3.7. Future Recommendations

Future work in this area should explore ways to collect additional input from various stakeholders. Ideas include developing a research design with questions for each group of stakeholders, and to reach out to them directly. For web developers, questions could be added to the MDN Developer Needs Survey [https://insights.developer.mozilla.org/] to learn about their pain points with web maps.

Chapter 12. Research questions

12.1. Overview

The OGC Testbed-16 call for proposals posed a number of research questions [https://portal.ogc.org/files/?artifact_id=91644#_research_questions] that this section of the ER attempts to answer based on the experiences of the thread participants during the Testbed.

12.2. Does ML require "data interoperability"?

12.2.1. Additional related questions

- Or can ML enable "data interoperability"?
- How do existing and emerging OGC standards contribute to a data architecture flow towards "data interoperability"?

12.2.2. Response

- ML requires data interoperability to make sure the right data, following the model requirements, is used for training and testing
  - Data interoperability will facilitate reproducibility, the ability of a user to duplicate the results of a prior study as defined in this document:
    - https://www.amstat.org/asa/files/pdfs/POL-ReproducibleResearchRecommendations.pdf)
    - McGill checklist for ML reproducibility: https://www.cs.mcgill.ca/~jpineau/ReproducibilityChecklist.pdf
  - The OGC API - Tiles [https://github.com/opengeospatial/OGC-API-Tiles] specification seems to be a good candidate for model training as both value datasets and label datasets can be retrieved as tiles which makes model training straight-forward (as most work on batches of patches [e.g. 512x512]). Bands (e.g. RGB only) are a possible limitation, depending on the model use case.
  - [Skymantics] Open WMS [https://www.ogc.org/standards/wms] has some shortcomings for model training, although they can be overcome and can potentially contribute to data interoperability. From the perspective of ML data interoperability, the legend graphic is particularly useful because it encodes, in image format, scale ranges that can be used in model training. More importantly, the underlying information used to generate the legend graphic is useful because it encodes the scale ranges represented in the legend graphic in a machine readable format (i.e. SLD). The issue with WMS [https://www.ogc.org/standards/wms] is that both the LegendURL (used to retrieve the legend graphic) and the GetStyles operation (used to retrieve the underlying SLD) are optional extensions in the WMS [https://www.ogc.org/standards/wms] specification. So, technically, it is possible to use an OGC WMS [https://www.ogc.org/standards/wms] for model training but only if these optional components are implemented. This consideration is particularly important given the large number of existing WMS [https://www.ogc.org/standards/wms] instances that represent a large pool of available tarining data if only these optional components were mandatory.

- [Skymantics] The experimental prototype of an OGC API - Records [https://htmlpreview.github.io/?https://github.com/opengeospatial/ogcapi-records/blob/master/20-004.html] catalogue used during the testbed offered a high level of flexibility, facilitating both the discovery of training datasets and providing binding information for data extraction by a ML algorithm. The catalogue is capable of harvesting repositories of different typologies and to list the relevant information for ML applications. This is an emerging OGC standard that can potentially contribute to a data architecture flow towards "data interoperability".

12.3. Where do trained datasets (i.e. trained model and training datasets) go and how can they be re-used?

12.3.1. Response

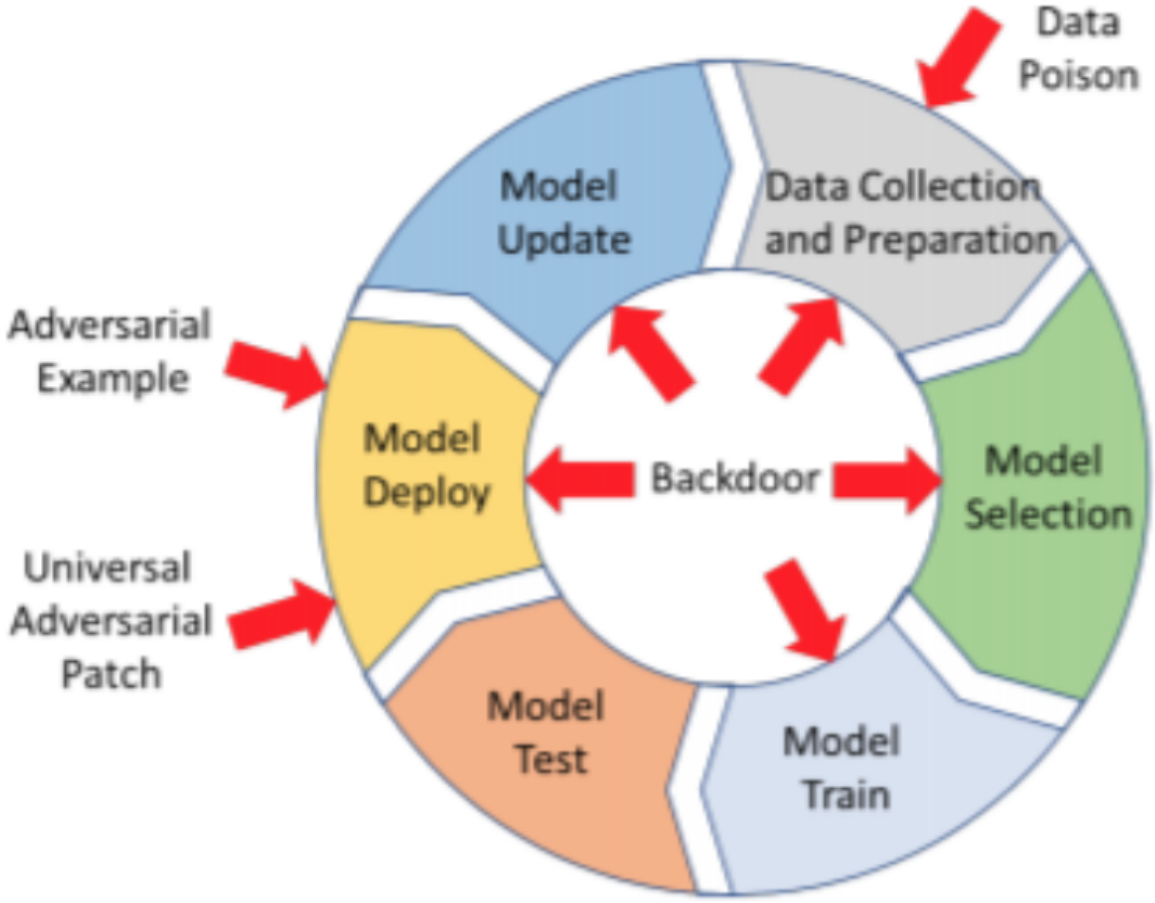
See: D016 Machine Learning Training Data ER [https://docs.ogc.org/per/20-018.html].

Additional comments follow:

- [CRIM] Training set:
  - They can be reused and should be made available for research reproducibility
  - They can be quite large (millions of samples), for instance Imagenet (http://image-net.org/download-API) in vision
- Trained models:
  - Trained model parameters are available in so-called model zoo, they are not easily queryable right now. They are usually used for benchmarking / comparing results against old/new training datasets.
  - Trained models are required for reproducibility as well as for benchmarking / comparing results against old/new training datasets.

12.4. How can we ensure the authenticity of trained datasets?

- [CRIM] It's an important and complex question, there is evidence that models can be attacked by either training dataset manipulation (e.g. label poisoning) or by directly manipulating the model parameters (backdoor attacks):
  - Barni, Mauro, Kassem Kallas, and Benedetta Tondi. "A new backdoor attack in CNNs by training set corruption without label poisoning." 2019 IEEE International Conference on Image Processing (ICIP). IEEE, 2019. https://arxiv.org/abs/1902.11237.
- There is an ongoing National Institute of Standards and Technology (NIST) TrojAI competition [https://pages.nist.gov/trojai/].
- See this recent review on attacks and counter-measures:
  - Gao, Y., Doan, B. G., Zhang, Z., Ma, S., Fu, A., Nepal, S., & Kim, H. (2020). Backdoor Attacks and Countermeasures on Deep Learning: A Comprehensive Review. arXiv preprint arXiv:2007.10760. https://arxiv.org/pdf/2007.10760.pdf



- Tensorflow Extended has a data validation module to check for anomalies: https://colab.research.google.com/github/tensorflow/tfx/blob/master/docs/tutorials/data_validation/tfdv_basic.ipynb

12.5. Is it necessary to have analysis ready data (ARD) for ML?

12.5.1. Additional related questions

- Can ML help ARD development?

12.5.2. Response

- [CRIM] Given a large and diverse enough training set, deep learning models could potentially capture data variability and therefore be more robust to pre-processing
- In the near future, ML could potentially replace some part of the ARD pre-processing in particular for the radiometric corrections.
- Minimally, preprocessing should be done in order to apply all necessary sensor-specific corrections.
- Any following preprocessing operations could reduce the variability of data and should most probably be avoided.

12.6. What is the value of datacubes for ML?

- The temporal aspect captured by datacubes could facilitate the development of multi-temporal models.
- It could also facilitate the development of unsupervised ML approaches (i.e. without annotations)

12.7. How do we address interoperability of distributed datacubes maintained by different organizations?

- [CRIM]:
  - OGC API - Coverage services could be deployed on top of datacubes to facilitate data sharing
  - It is important to facilitate the replication of datacubes, for example as a workflow specification from raw datasets.
- In case a computing infrastructure is available remotely in the same environment as the targeted datacube, we can then deploy OGC API - Processes to produce and distribute derived products

12.8. What is the potential of MapML in the context of ML?

12.8.1. Additional related questions.

- Where does it need to be enhanced?

12.8.2. Response

See Standardizing Web Maps to Increase Adoption among Browser Vendors.

12.9. How to discover and run an existing ML model?

- [CRIM] An existing ML model (i.e. its architecture and parameters) must be integrated into a ML pipeline or workflow in order to be production ready . This pipeline must include at least:
  - Handling functions for pre-processing and post-processing
  - Semantic information about the output
  - A description of the expected input
  - A way to verify the model output given a known input
  - Relevant metadata about the source of the model (what it was trained with/for, reference article if any, revision number if re-trained with more recent data. etc.)
- [52north] From an ADES perspective, an ML model can be a process on an ADES. It can be

technically discovered by the means of ADES (~ OGC API Processes). Though, it is still the same as an arbitrary process in ADES - not particularly highlighted or promoted as an ML model. Running the model must ensure that the input data (e.g. raster GeoTIFF) is pre-processed in the same way as the training data. There is currently no concept/approach on describing the pre-processing (e.g. S1 SNAP-based was used in Testbed-16).

- Probably also need to include details about the source of the model (what it was trained with/for, reference article if any, revision number if re-trained with more recent data. etc.)

Chapter 13. Issues

13.1. Overview

The work of the Testbed-16 ML thread was facilitated using a private GitLab repository. One feature of the repository is an issue tracker where a number of issues that arose during the course of the Testbed were discussed. The most important of these issues are summarized here.

13.2. Persisting ML model results (issue #19)

13.2.1. Overview

The ML models developed by testbed participants were deployed to an ADES which provides a standard interface for executing and retrieving results. Execution results, however, do not persist for any length of time which is an issue for any downstream actors that may want to use those results for further processing. Currently the only option is that the client retrieves the results and arranges for their persistent storage.

The discussion between thread participants consistently proposed alleviating the client of this burden and to have the ADES arrange for the persistent storage of results.

13.2.2. Solution 1 - stateful container

This solution involves having the ADES mount a persistent volume (e.g. Amazon's EFS [https://aws.amazon.com/efs/]) onto the Docker container into which the results of ML processing are copied. When execution of the Docker container terminates, the results of processing would persist on the mounted volume and would be accessible from the host machine. The workflow would proceed as follows:

1. The ML Tool is deployed to an ADES. The Application Package that defines the process interface, refers to the Docker image containing the bundled/trained ML tool.
2. The ML Tool is executed via the ADES interface (i.e. POST of execute request to the /jobs endpoint).
3. The ADES launches the Docker container. This involves, among other things, arranging to mount a host volume into the Docker container for the persistent storage of results.
4. The tool executes the ML model within the Docker container and produces some output (e.g. a GeoTIFF) that is written to this mounted volume.
5. Once the tool completes execution the Docker container is shutdown but the processing result are still available to the ADES on the host volume that was mounted to the container.
6. The ADES can now provide the results via the `/jobs/{jobId}/results` endpoint.
7. The host-persisted results can also be fed manually to the GeoServer [http://geoserver.arg] MapML Community Model which then feeds the results onward to the MapML client.

13.2.3. Solution 2 - object store

This solution for persisting the results of executing an ML model via the ADES involves pushing the results to an object store (e.g. Amazon's S3 [https://aws.amazon.com/s3/] or Google Cloud Storage [https://cloud.google.com/storage]). The results can be pushed to the object store from within the Docker container or it can be done by the ADES after ML processes has completed. The results can then be pulled from the object store and fed into the GeoServer [http://geoserver.arg] or directly to the MapML client.

The primary issue with the approach concerns the mechanism of interacting with the object store to retrieve the results. In most cases the job of forming an access URL involves credentials and signed HTTP requests and can be quite onerous. However, vendor-provided and open-source libraries are available that may mitigate this problem.

13.2.4. Solution 3 - OGC API

This solution makes use of an extended OGC API - Coverages [https://github.com/opengeospatial/ogc_api_coverages] interface that offers a `/images` endpoint. When processing results (e.g. GeoTIFF) are available they can be HTTP POST'ed to the `/collections/{coverageId}/images` endpoint and then subsequently retrieved using the OGC API - Coverages [https://github.com/opengeospatial/ogc_api_coverages] coverage access interface. An example of the images endpoint can be found here: http://test.cubewerx.com/cubewerx/cubeserv/demo/ogcapi/Daraa/collections/Daraa_mosaic_2019/images.

The primary issue with this approach, at least in the case of the CubeWerx server, is that the GeoTIFF results need to be orthorectified (see clause: D016 Machine Learning Training Data ER [https://portal.ogc.org/files/?artifact_id=95717], Sentinel-1). However, assuming the results can be pushed to a coverage server, a client (MapML or otherwise) can simply make a standard OGC API - Coverages [https://github.com/opengeospatial/ogc_api_coverages] (or map [https://github.com/opengeospatial/OGC-API-Maps] request, or tiles [https://github.com/opengeospatial/OGC-API-Tiles] request for that matter) to retrieve the results for display.

13.3. MapML Client Prototype based on Web-Map-Custom-Element

13.3.1. Discussion overview

This issue was concerned with how to parse MapML documents available from the GeoServer module.

In principle, the HTML parser should be able to be used in JavaScript, but for the time being, the XML parser built into browsers provides a reasonable API that browser-based clients can rely on to parse MapML.

Ideally, Progressing Web Apps [https://web.dev/progressive-web-apps/] (PWA) could actually use the Web-Map-Custom-Element and its somewhat obscure JavaScript API. It remains to be specified what this API should be in the HTML standard and thread participants experimented with that.

It was felt by thread participants that the API should be similar to the Leaflet API [https://leafletjs.com/], but there are obviously elements of the latter API which shouldn't be specified by HTML, but by libraries built on top (i.e. like Leaflet itself).

Web-Map-Custom-Element, being HTML DOM based, presents a clean and straightforward API.

Because Safari doesn't support customized built-in elements (<map is=web-map...>), another regular custom element called <mm-mapp> has been provided which works in a similar manner but does not support the <area is=map-area> element).

The idea is to eventually load the ML output into GeoServer, perhaps via an OGC API, and then GeoServer could automatically publish that data as MapML for consumption by the client.

While developing a MapML client based on the WebMap Custom Element framework an alternative way was found to expose the layer metadata provided by the framework. Some simple information such as title and legend URL can be acquired in a straightforward manner (by short text). However, the complex information such as layer extent is wrapped in a MicroXML which requires a standard manner to extract all effective information from it. Below is an example of a MicroXML of layer extent:

```
<extent units="OSMTILE">
   <input name="z" type="zoom" value="18" min="0" max="18"/>
   <input name="xmin" type="location" rel="map" position="top-left" axis="easting"
units="pcrs" min="-8240672.435241637" max="-8227290.1626559235"/>
   <input name="ymin" type="location" rel="map" position="bottom-left" axis="northing"
units="pcrs" min="4965875.319295468" max="4994389.518266143"/>
   <input name="xmax" type="location" rel="map" position="top-right" axis="easting"
units="pcrs" min="-8240672.435241637" max="-8227290.1626559235"/>
   <input name="ymax" type="location" rel="map" position="top-left" axis="northing"
units="pcrs" min="4965875.319295468" max="4994389.518266143"/>
   <input name="w" type="width" min="1" max="10000"/>
   <input name="h" type="height" min="1" max="10000"/>
   <link tref="http://cici.lab.asu.edu/geoserver217/tiger/wms?version=1.3.0
&amp;service=WMS&amp;request=GetMap&amp;crs=urn:x-ogc:def:crs:EPSG:3857
&amp;layers=tiger_roads&amp;styles=line&amp;bbox={xmin},{ymin},{xmax},{ymax}&amp;forma
t=image/png&amp;transparent=true&amp;width={w}&amp;height={h}" rel="image"/>
   <input name="i" type="location" axis="i" units="map"/>
   <input name="j" type="location" axis="j" units="map"/>
   <link tref="http://cici.lab.asu.edu/geoserver217/tiger/wms?version=1.3.0
&amp;service=WMS&amp;request=GetFeatureInfo&amp;FEATURE_COUNT=50&amp;crs=urn:x-
ogc:def:crs:EPSG:3857&amp;layers=tiger_roads&amp;query_layers=tiger_roads&amp;styles=l
ine&amp;bbox={xmin},{ymin},{xmax},{ymax}&amp;width={w}&amp;height={h}&amp;info_format=
text/mapml&amp;transparent=true&amp;x={i}&amp;y={j}" rel="query"/>
</extent>
```

13.3.2. Open questions

The sought-after feedback about Web-Map-Custom-Elements is:

- What should the API for this element suite look like?
- In what way should Web authors be able to customize the user experience for this element suite?
- What facilities could the element provide that would support its use in a PWA?
- What facilities should the element provide that you would expect to be built into a browser?

The answers to these open questions can be found in the MapML Client 1 (D130 - ASU) and Standardizing Web Maps to Increase Adoption among Browser Vendors clauses.

13.4. Processing Sentinel-1 Data using SNAP

This issue was concerned with the orthorectification of Sentinel images since that is a requirement for loading them into the CubeWerx OGC API - Coverages [https://github.com/opengeospatial/ogc_api_coverages] server.

The Sentinel-1 images accessible from the Registry of Open Data on AWS [https://registry.opendata.aws/sentinel-1/] are not orthorectified preventing them from being loaded into the CubeWerx server.

To support the first pass at resolving this problem, NRCan has provided a SNAP [https://step.esa.int/main/toolboxes/snap/] graph to process GRD files along with terrain correction to create GeoTIFF images as output. The source of that graph is provided here:

```
<graph id="Graph">
  <version>1.0</version>
  <node id="Read">
    <operator>Read</operator>
    <sources/>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
      <file>D:\Sentinel-
1_GRD\GRD\S1A_IW_GRDH_1SDV_20200705T014935_20200705T015000_033313_03DC13_49ED.zip</fil
e>
    </parameters>
  </node>
  <node id="Calibration">
    <operator>Calibration</operator>
    <sources>
      <sourceProduct refid="Read"/>
    </sources>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
      <sourceBands/>
      <auxFile>Product Auxiliary File</auxFile>
      <externalAuxFile/>
      <outputImageInComplex>false</outputImageInComplex>
      <outputImageScaleInDb>false</outputImageScaleInDb>
      <createGammaBand>false</createGammaBand>
      <createBetaBand>false</createBetaBand>
      <selectedPolarisations/>
      <outputSigmaBand>true</outputSigmaBand>
```

```
      <outputGammaBand>false</outputGammaBand>
      <outputBetaBand>false</outputBetaBand>
    </parameters>
  </node>
  <node id="Speckle-Filter">
    <operator>Speckle-Filter</operator>
    <sources>
      <sourceProduct refid="Calibration"/>
    </sources>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
      <sourceBands/>
      <filter>Boxcar</filter>
      <filterSizeX>7</filterSizeX>
      <filterSizeY>7</filterSizeY>
      <dampingFactor>2</dampingFactor>
      <estimateENL>true</estimateENL>
      <enl>1.0</enl>
      <numLooksStr>1</numLooksStr>
      <windowSize>7x7</windowSize>
      <targetWindowSizeStr>3x3</targetWindowSizeStr>
      <sigmaStr>0.9</sigmaStr>
      <anSize>50</anSize>
    </parameters>
  </node>
  <node id="Terrain-Correction">
    <operator>Terrain-Correction</operator>
    <sources>
      <sourceProduct refid="Speckle-Filter"/>
    </sources>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
      <sourceBands/>
      <demName>CDEM</demName>
      <externalDEMFile/>
      <externalDEMNoDataValue>0.0</externalDEMNoDataValue>
      <externalDEMApplyEGM>true</externalDEMApplyEGM>
      <demResamplingMethod>BILINEAR_INTERPOLATION</demResamplingMethod>
      <imgResamplingMethod>BILINEAR_INTERPOLATION</imgResamplingMethod>
      <pixelSpacingInMeter>30.0</pixelSpacingInMeter>
      <pixelSpacingInDegree>2.6949458523585647E-4</pixelSpacingInDegree>
      <mapProjection>GEOGCS[&quot;WGS84(DD)&quot;, &#xd;
DATUM[&quot;WGS84&quot;, &#xd;
  SPHEROID[&quot;WGS84&quot;, 6378137.0, 298.257223563]], &#xd;
PRIMEM[&quot;Greenwich&quot;, 0.0], &#xd;
UNIT[&quot;degree&quot;, 0.017453292519943295], &#xd;
AXIS[&quot;Geodetic longitude&quot;, EAST], &#xd;
AXIS[&quot;Geodetic latitude&quot;, NORTH]]</mapProjection>
      <alignToStandardGrid>false</alignToStandardGrid>
      <standardGridOriginX>0.0</standardGridOriginX>
      <standardGridOriginY>0.0</standardGridOriginY>
      <nodataValueAtSea>true</nodataValueAtSea>
      <saveDEM>false</saveDEM>
```

```
      <saveLatLon>false</saveLatLon>
      <saveIncidenceAngleFromEllipsoid>false</saveIncidenceAngleFromEllipsoid>
      <saveLocalIncidenceAngle>false</saveLocalIncidenceAngle>
      <saveProjectedLocalIncidenceAngle>false</saveProjectedLocalIncidenceAngle>
      <saveSelectedSourceBand>true</saveSelectedSourceBand>
      <outputComplex>false</outputComplex>
      <applyRadiometricNormalization>false</applyRadiometricNormalization>
      <saveSigmaNought>false</saveSigmaNought>
      <saveGammaNought>false</saveGammaNought>
      <saveBetaNought>false</saveBetaNought>
      <incidenceAngleForSigma0>Use projected local incidence angle from
DEM</incidenceAngleForSigma0>
      <incidenceAngleForGamma0>Use projected local incidence angle from
DEM</incidenceAngleForGamma0>
      <auxFile>Latest Auxiliary File</auxFile>
      <externalAuxFile/>
    </parameters>
  </node>
  <node id="Write">
    <operator>Write</operator>
    <sources>
      <sourceProduct refid="Terrain-Correction"/>
    </sources>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
      <file>D:\Sentinel-
1_GRD\GRD\S1A_IW_GRDH_1SDV_20200705T014935_20200705T015000_033313_03DC13_49ED.tif</fil
e>
      <formatName>GeoTIFF</formatName>
    </parameters>
  </node>
  <applicationData id="Presentation">
    <Description/>
    <node id="Read">
            <displayPosition x="61.0" y="236.0"/>
    </node>
    <node id="Calibration">
      <displayPosition x="55.0" y="146.0"/>
    </node>
    <node id="Speckle-Filter">
      <displayPosition x="44.0" y="58.0"/>
    </node>
    <node id="Terrain-Correction">
      <displayPosition x="246.0" y="62.0"/>
    </node>
    <node id="Write">
            <displayPosition x="278.0" y="233.0"/>
    </node>
  </applicationData>
</graph>
```

A graph was also provided for SLC files from the Boreal Cloud for terrain correction and generating GeoTIFF:

```
<graph id="Graph">
  <version>1.0</version>
  <node id="Read">
    <operator>Read</operator>
    <sources/>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">

<file>K:\SENTINEL1\S1B_IW_SLC__1SDV_20170620T143323_20170620T143353_006135_00AC6F_505C
.zip</file>
    </parameters>
  </node>
  <node id="Calibration">
    <operator>Calibration</operator>
    <sources>
      <sourceProduct refid="Read"/>
    </sources>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
      <sourceBands/>
      <auxFile>Latest Auxiliary File</auxFile>
      <externalAuxFile/>
      <outputImageInComplex>false</outputImageInComplex>
      <outputImageScaleInDb>false</outputImageScaleInDb>
      <createGammaBand>false</createGammaBand>
      <createBetaBand>false</createBetaBand>
      <selectedPolarisations/>
      <outputSigmaBand>true</outputSigmaBand>
      <outputGammaBand>false</outputGammaBand>
      <outputBetaBand>false</outputBetaBand>
    </parameters>
  </node>
  <node id="TOPSAR-Deburst">
    <operator>TOPSAR-Deburst</operator>
    <sources>
      <sourceProduct refid="Calibration"/>
    </sources>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
      <selectedPolarisations/>
    </parameters>
  </node>
  <node id="Multilook">
    <operator>Multilook</operator>
    <sources>
      <sourceProduct refid="TOPSAR-Deburst"/>
    </sources>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
      <sourceBands/>
      <nRgLooks>6</nRgLooks>
```

```
      <nAzLooks>2</nAzLooks>
      <outputIntensity>true</outputIntensity>
      <grSquarePixel>true</grSquarePixel>
    </parameters>
  </node>
  <node id="Speckle-Filter">
    <operator>Speckle-Filter</operator>
    <sources>
      <sourceProduct refid="Multilook"/>
    </sources>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
      <sourceBands/>
      <filter>Boxcar</filter>
      <filterSizeX>5</filterSizeX>
      <filterSizeY>5</filterSizeY>
      <dampingFactor>2</dampingFactor>
      <estimateENL>true</estimateENL>
      <enl>1.0</enl>
      <numLooksStr>1</numLooksStr>
      <windowSize>7x7</windowSize>
      <targetWindowSizeStr>3x3</targetWindowSizeStr>
      <sigmaStr>0.9</sigmaStr>
      <anSize>50</anSize>
    </parameters>
  </node>
  <node id="Terrain-Correction">
    <operator>Terrain-Correction</operator>
    <sources>
      <sourceProduct refid="Speckle-Filter"/>
    </sources>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
      <sourceBands>Sigma0_VH,Sigma0_VV</sourceBands>
      <demName>CDEM</demName>
      <externalDEMFile/>
      <externalDEMNoDataValue>0.0</externalDEMNoDataValue>
      <externalDEMApplyEGM>true</externalDEMApplyEGM>
      <demResamplingMethod>BILINEAR_INTERPOLATION</demResamplingMethod>
      <imgResamplingMethod>BILINEAR_INTERPOLATION</imgResamplingMethod>
      <pixelSpacingInMeter>30.0</pixelSpacingInMeter>
      <pixelSpacingInDegree>2.6949458523585647E-4</pixelSpacingInDegree>
      <mapProjection>GEOGCS[&quot;WGS84(DD)&quot;, &#xd;
DATUM[&quot;WGS84&quot;, &#xd;
  SPHEROID[&quot;WGS84&quot;, 6378137.0, 298.257223563]], &#xd;
PRIMEM[&quot;Greenwich&quot;, 0.0], &#xd;
UNIT[&quot;degree&quot;, 0.017453292519943295], &#xd;
AXIS[&quot;Geodetic longitude&quot;, EAST], &#xd;
AXIS[&quot;Geodetic latitude&quot;, NORTH]]</mapProjection>
      <alignToStandardGrid>false</alignToStandardGrid>
      <standardGridOriginX>0.0</standardGridOriginX>
      <standardGridOriginY>0.0</standardGridOriginY>
      <nodataValueAtSea>true</nodataValueAtSea>
```

```
      <saveDEM>false</saveDEM>
      <saveLatLon>false</saveLatLon>
      <saveIncidenceAngleFromEllipsoid>false</saveIncidenceAngleFromEllipsoid>
      <saveLocalIncidenceAngle>false</saveLocalIncidenceAngle>
      <saveProjectedLocalIncidenceAngle>false</saveProjectedLocalIncidenceAngle>
      <saveSelectedSourceBand>true</saveSelectedSourceBand>
      <outputComplex>false</outputComplex>
      <applyRadiometricNormalization>false</applyRadiometricNormalization>
      <saveSigmaNought>false</saveSigmaNought>
      <saveGammaNought>false</saveGammaNought>
      <saveBetaNought>false</saveBetaNought>
      <incidenceAngleForSigma0>Use projected local incidence angle from
DEM</incidenceAngleForSigma0>
      <incidenceAngleForGamma0>Use projected local incidence angle from
DEM</incidenceAngleForGamma0>
      <auxFile>Latest Auxiliary File</auxFile>
      <externalAuxFile/>
    </parameters>
  </node>
  <node id="Write">
    <operator>Write</operator>
    <sources>
      <sourceProduct refid="Terrain-Correction"/>
    </sources>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">

<file>K:\SENTINEL1\S1B_IW_SLC__1SDV_20170620T143323_20170620T143353_006135_00AC6F.tif<
/file>
      <formatName>GeoTIFF</formatName>
    </parameters>
  </node>
  <applicationData id="Presentation">
    <Description/>
    <node id="Read">
            <displayPosition x="63.0" y="258.0"/>
    </node>
    <node id="Calibration">
      <displayPosition x="60.0" y="155.0"/>
    </node>
    <node id="TOPSAR-Deburst">
      <displayPosition x="41.0" y="43.0"/>
    </node>
    <node id="Multilook">
      <displayPosition x="251.0" y="42.0"/>
    </node>
    <node id="Speckle-Filter">
      <displayPosition x="411.0" y="41.0"/>
    </node>
    <node id="Terrain-Correction">
      <displayPosition x="402.0" y="139.0"/>
    </node>
```

```
    <node id="Write">
            <displayPosition x="428.0" y="251.0"/>
    </node>
  </applicationData>
</graph>
```

CRIM provided the following orthorectification SNAP [https://step.esa.int/main/toolboxes/snap/] graph:

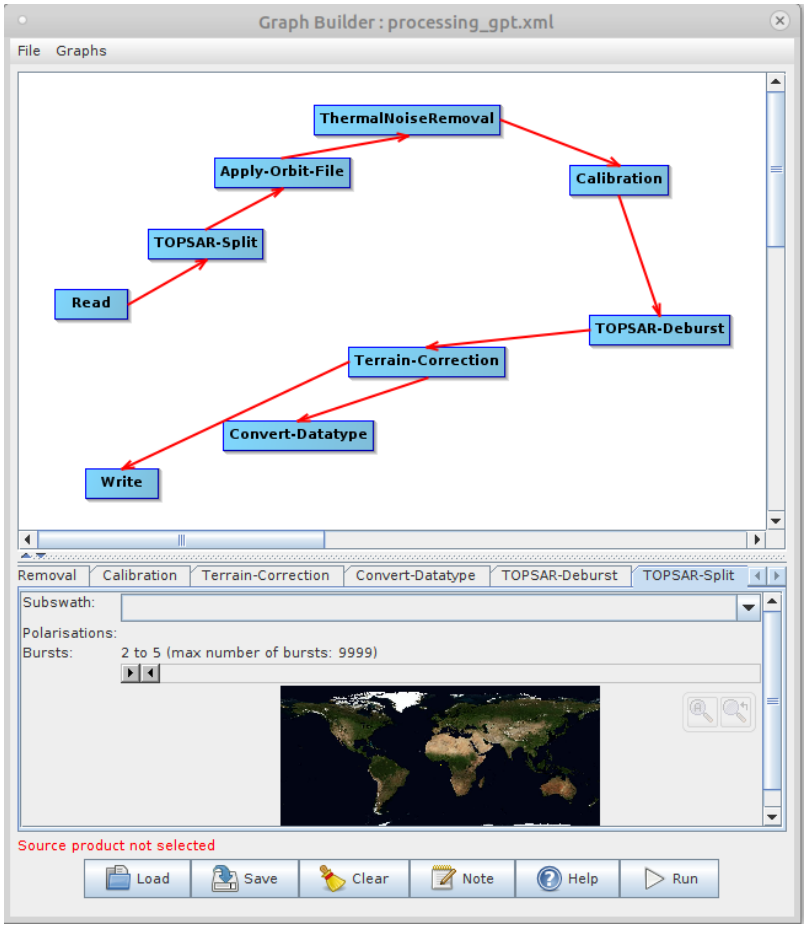


Figure 26. CRIM orthorectification graph

It is assuming a S1 image in IW mode and SLC format (S1A_IW_SLC_XXX.zip), the processing is only for the VV band, the IW1 subswath and the Burst Index between 2 and 5, you can modify the values in the xml or load the graph in SNAP: `

```
<operator>TOPSAR-Split</operator>
<sources>
  <sourceProduct refid="Read"/>
</sources>
<parameters class="com.bc.ceres.binding.dom.XppDomElement">
  <subswath>IW1</subswath>
  <selectedPolarisations>VV</selectedPolarisations>
  <firstBurstIndex>2</firstBurstIndex>
  <lastBurstIndex>5</lastBurstIndex>
  <wktAoi/>
</parameters>
```

```
<graph id="Graph">
  <version>1.0</version>
  <node id="Read">
    <operator>Read</operator>
    <sources/>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">

<file>C:/DATA/MUSE/S1/S1A_IW_SLC__1SDV_20190408T225217_20190408T225244_026705_02FF7D_A
279.zip</file>
    </parameters>
  </node>
  <node id="Apply-Orbit-File">
    <operator>Apply-Orbit-File</operator>
    <sources>
      <sourceProduct refid="TOPSAR-Split"/>
    </sources>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
      <orbitType>Sentinel Precise (Auto Download)</orbitType>
      <polyDegree>3</polyDegree>
      <continueOnFail>false</continueOnFail>
    </parameters>
  </node>
  <node id="ThermalNoiseRemoval">
    <operator>ThermalNoiseRemoval</operator>
    <sources>
      <sourceProduct refid="Apply-Orbit-File"/>
    </sources>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
      <selectedPolarisations>VV</selectedPolarisations>
      <removeThermalNoise>false</removeThermalNoise>
      <reIntroduceThermalNoise>false</reIntroduceThermalNoise>
    </parameters>
  </node>
  <node id="Calibration">
    <operator>Calibration</operator>
    <sources>
      <sourceProduct refid="ThermalNoiseRemoval"/>
    </sources>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
      <sourceBands/>
      <auxFile/>
      <externalAuxFile/>
      <outputImageInComplex>false</outputImageInComplex>
      <outputImageScaleInDb>false</outputImageScaleInDb>
      <createGammaBand>false</createGammaBand>
      <createBetaBand>false</createBetaBand>
      <selectedPolarisations>VV</selectedPolarisations>
      <outputSigmaBand>false</outputSigmaBand>
      <outputGammaBand>false</outputGammaBand>
      <outputBetaBand>false</outputBetaBand>
    </parameters>
```

```
  </node>
  <node id="Terrain-Correction">
    <operator>Terrain-Correction</operator>
    <sources>
      <sourceProduct refid="TOPSAR-Deburst"/>
    </sources>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
      <sourceBands/>
      <demName>SRTM 3Sec</demName>
      <externalDEMFile/>
      <externalDEMNoDataValue>0.0</externalDEMNoDataValue>
      <externalDEMApplyEGM>true</externalDEMApplyEGM>
      <demResamplingMethod>BILINEAR_INTERPOLATION</demResamplingMethod>
      <imgResamplingMethod>BILINEAR_INTERPOLATION</imgResamplingMethod>
      <pixelSpacingInMeter>13.94</pixelSpacingInMeter>
      <pixelSpacingInDegree>1.252251506062613E-4</pixelSpacingInDegree>
      <mapProjection>GEOGCS[&quot;WGS84(DD)&quot;,
  DATUM[&quot;WGS84&quot;,
    SPHEROID[&quot;WGS84&quot;, 6378137.0, 298.257223563]],
  PRIMEM[&quot;Greenwich&quot;, 0.0],
  UNIT[&quot;degree&quot;, 0.017453292519943295],
  AXIS[&quot;Geodetic longitude&quot;, EAST],
  AXIS[&quot;Geodetic latitude&quot;, NORTH]]</mapProjection>
      <alignToStandardGrid>false</alignToStandardGrid>
      <standardGridOriginX>0.0</standardGridOriginX>
      <standardGridOriginY>0.0</standardGridOriginY>
      <nodataValueAtSea>true</nodataValueAtSea>
      <saveDEM>false</saveDEM>
      <saveLatLon>false</saveLatLon>
      <saveIncidenceAngleFromEllipsoid>false</saveIncidenceAngleFromEllipsoid>
      <saveLocalIncidenceAngle>false</saveLocalIncidenceAngle>
      <saveProjectedLocalIncidenceAngle>false</saveProjectedLocalIncidenceAngle>
      <saveSelectedSourceBand>true</saveSelectedSourceBand>
      <outputComplex>false</outputComplex>
      <applyRadiometricNormalization>false</applyRadiometricNormalization>
      <saveSigmaNought>false</saveSigmaNought>
      <saveGammaNought>false</saveGammaNought>
      <saveBetaNought>false</saveBetaNought>
      <incidenceAngleForSigma0>Use projected local incidence angle from
DEM</incidenceAngleForSigma0>
      <incidenceAngleForGamma0>Use projected local incidence angle from
DEM</incidenceAngleForGamma0>
      <auxFile>Latest Auxiliary File</auxFile>
      <externalAuxFile/>
    </parameters>
  </node>
  <node id="Convert-Datatype">
    <operator>Convert-Datatype</operator>
    <sources>
      <sourceProduct refid="Terrain-Correction"/>
    </sources>
```

```
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
      <sourceBands/>
      <targetDataType>uint8</targetDataType>
      <targetScalingStr>Linear (slope and intercept)</targetScalingStr>
      <targetNoDataValue/>
    </parameters>
  </node>
  <node id="TOPSAR-Deburst">
    <operator>TOPSAR-Deburst</operator>
    <sources>
      <sourceProduct refid="Calibration"/>
    </sources>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
      <selectedPolarisations/>
    </parameters>
  </node>
  <node id="TOPSAR-Split">
    <operator>TOPSAR-Split</operator>
    <sources>
      <sourceProduct refid="Read"/>
    </sources>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
      <subswath>IW1</subswath>
      <selectedPolarisations>VV</selectedPolarisations>
      <firstBurstIndex>2</firstBurstIndex>
      <lastBurstIndex>5</lastBurstIndex>
      <wktAoi/>
    </parameters>
  </node>
  <node id="Write">
    <operator>Write</operator>
    <sources>
      <sourceProduct refid="Terrain-Correction"/>
    </sources>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
      <file>C:/DATA/MUSE/S1/stacker_output.tif</file>
      <formatName>GeoTIFF</formatName>
    </parameters>
  </node>
  <applicationData id="Presentation">
    <Description/>
    <node id="Read">
      <displayPosition x="29.0" y="176.0"/>
    </node>
    <node id="Apply-Orbit-File">
      <displayPosition x="159.0" y="69.0"/>
    </node>
    <node id="ThermalNoiseRemoval">
      <displayPosition x="240.0" y="26.0"/>
    </node>
    <node id="Calibration">
```

```
      <displayPosition x="448.0" y="75.0"/>
    </node>
    <node id="Terrain-Correction">
      <displayPosition x="268.0" y="223.0"/>
    </node>
    <node id="Convert-Datatype">
      <displayPosition x="166.0" y="283.0"/>
    </node>
    <node id="TOPSAR-Deburst">
      <displayPosition x="464.0" y="197.0"/>
    </node>
    <node id="TOPSAR-Split">
      <displayPosition x="105.0" y="127.0"/>
    </node>
    <node id="Write">
      <displayPosition x="54.0" y="322.0"/>
    </node>
  </applicationData>
</graph>
```

The following graph converts the image to 8-bit output:

```
<graph id="Graph">
  <version>1.0</version>
  <node id="Read">
    <operator>Read</operator>
    <sources/>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">

<file>C:/DATA/MUSE/S1/S1A_IW_SLC__1SDV_20190408T225217_20190408T225244_026705_02FF7D_A
279.zip</file>
    </parameters>
  </node>
  <node id="Apply-Orbit-File">
    <operator>Apply-Orbit-File</operator>
    <sources>
      <sourceProduct refid="TOPSAR-Split"/>
    </sources>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
      <orbitType>Sentinel Precise (Auto Download)</orbitType>
      <polyDegree>3</polyDegree>
      <continueOnFail>false</continueOnFail>
    </parameters>
  </node>
  <node id="ThermalNoiseRemoval">
    <operator>ThermalNoiseRemoval</operator>
    <sources>
      <sourceProduct refid="Apply-Orbit-File"/>
    </sources>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
```

```
      <selectedPolarisations>VV</selectedPolarisations>
      <removeThermalNoise>false</removeThermalNoise>
      <reIntroduceThermalNoise>false</reIntroduceThermalNoise>
    </parameters>
  </node>
  <node id="Calibration">
    <operator>Calibration</operator>
    <sources>
      <sourceProduct refid="ThermalNoiseRemoval"/>
    </sources>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
      <sourceBands/>
      <auxFile/>
      <externalAuxFile/>
      <outputImageInComplex>false</outputImageInComplex>
      <outputImageScaleInDb>true</outputImageScaleInDb>
      <createGammaBand>false</createGammaBand>
      <createBetaBand>false</createBetaBand>
      <selectedPolarisations>VV</selectedPolarisations>
      <outputSigmaBand>false</outputSigmaBand>
      <outputGammaBand>false</outputGammaBand>
      <outputBetaBand>false</outputBetaBand>
    </parameters>
  </node>
  <node id="Terrain-Correction">
    <operator>Terrain-Correction</operator>
    <sources>
      <sourceProduct refid="TOPSAR-Deburst"/>
    </sources>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
      <sourceBands/>
      <demName>SRTM 3Sec</demName>
      <externalDEMFile/>
      <externalDEMNoDataValue>0.0</externalDEMNoDataValue>
      <externalDEMApplyEGM>true</externalDEMApplyEGM>
      <demResamplingMethod>BILINEAR_INTERPOLATION</demResamplingMethod>
      <imgResamplingMethod>BILINEAR_INTERPOLATION</imgResamplingMethod>
      <pixelSpacingInMeter>13.94</pixelSpacingInMeter>
      <pixelSpacingInDegree>1.252251506062613E-4</pixelSpacingInDegree>
      <mapProjection>GEOGCS[&quot;WGS84(DD)&quot;,
  DATUM[&quot;WGS84&quot;,
    SPHEROID[&quot;WGS84&quot;, 6378137.0, 298.257223563]],
  PRIMEM[&quot;Greenwich&quot;, 0.0],
  UNIT[&quot;degree&quot;, 0.017453292519943295],
  AXIS[&quot;Geodetic longitude&quot;, EAST],
  AXIS[&quot;Geodetic latitude&quot;, NORTH]]</mapProjection>
      <alignToStandardGrid>false</alignToStandardGrid>
      <standardGridOriginX>0.0</standardGridOriginX>
      <standardGridOriginY>0.0</standardGridOriginY>
      <nodataValueAtSea>true</nodataValueAtSea>
      <saveDEM>false</saveDEM>
```

```
      <saveLatLon>false</saveLatLon>
      <saveIncidenceAngleFromEllipsoid>false</saveIncidenceAngleFromEllipsoid>
      <saveLocalIncidenceAngle>false</saveLocalIncidenceAngle>
      <saveProjectedLocalIncidenceAngle>false</saveProjectedLocalIncidenceAngle>
      <saveSelectedSourceBand>true</saveSelectedSourceBand>
      <outputComplex>false</outputComplex>
      <applyRadiometricNormalization>false</applyRadiometricNormalization>
      <saveSigmaNought>false</saveSigmaNought>
      <saveGammaNought>false</saveGammaNought>
      <saveBetaNought>false</saveBetaNought>
      <incidenceAngleForSigma0>Use projected local incidence angle from
DEM</incidenceAngleForSigma0>
      <incidenceAngleForGamma0>Use projected local incidence angle from
DEM</incidenceAngleForGamma0>
      <auxFile>Latest Auxiliary File</auxFile>
      <externalAuxFile/>
    </parameters>
  </node>
  <node id="Convert-Datatype">
    <operator>Convert-Datatype</operator>
    <sources>
      <sourceProduct refid="Terrain-Correction"/>
    </sources>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
      <sourceBands/>
      <targetDataType>uint8</targetDataType>
      <targetScalingStr>Linear (slope and intercept)</targetScalingStr>
      <targetNoDataValue/>
    </parameters>
  </node>
  <node id="TOPSAR-Deburst">
    <operator>TOPSAR-Deburst</operator>
    <sources>
      <sourceProduct refid="Calibration"/>
    </sources>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
      <selectedPolarisations/>
    </parameters>
  </node>
  <node id="TOPSAR-Split">
    <operator>TOPSAR-Split</operator>
    <sources>
      <sourceProduct refid="Read"/>
    </sources>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
      <subswath>IW1</subswath>
      <selectedPolarisations>VV</selectedPolarisations>
      <firstBurstIndex>2</firstBurstIndex>
      <lastBurstIndex>5</lastBurstIndex>
      <wktAoi/>
    </parameters>
```

```
  </node>
  <node id="Write">
    <operator>Write</operator>
    <sources>
      <sourceProduct refid="Terrain-Correction"/>
    </sources>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
      <file>C:/DATA/MUSE/S1/stacker_output.tif</file>
      <formatName>GeoTIFF</formatName>
    </parameters>
  </node>
  <applicationData id="Presentation">
    <Description/>
    <node id="Read">
      <displayPosition x="29.0" y="176.0"/>
    </node>
    <node id="Apply-Orbit-File">
      <displayPosition x="159.0" y="69.0"/>
    </node>
    <node id="ThermalNoiseRemoval">
      <displayPosition x="240.0" y="26.0"/>
    </node>
    <node id="Calibration">
      <displayPosition x="448.0" y="75.0"/>
    </node>
    <node id="Terrain-Correction">
      <displayPosition x="268.0" y="223.0"/>
    </node>
    <node id="Convert-Datatype">
      <displayPosition x="166.0" y="283.0"/>
    </node>
    <node id="TOPSAR-Deburst">
      <displayPosition x="464.0" y="197.0"/>
    </node>
    <node id="TOPSAR-Split">
      <displayPosition x="105.0" y="127.0"/>
    </node>
    <node id="Write">
      <displayPosition x="54.0" y="322.0"/>
    </node>
  </applicationData>
</graph>
```

13.5. Download S1 data from Amazon S3

This issue was concerned with downloading Sentinel products once discovered in the catalogue. Each Sentinel catalogue record includes a download reference but that reference is an Amazon S3 [https://aws.amazon.com/s3/] reference. An Amazon S3 [https://aws.amazon.com/s3/] reference is not a URL.

The considered solution was to convert the S3 reference into a URL and store it in the catalogue

record. Conversion of the S3 reference into a URL requires that a signature [https://docs.aws.amazon.com/general/latest/gr/sigv4_signing.html] be generated according to an algorithm defined by Amazon and then appended to the URL or specified in the request header. The signature is meant to "prove" your identity and makes use of the caller's Amazon login credentials. The following fragment illustrates the type of information that is hashed into the signature:

```
  runHeaders = [
    {
      "Host": "sentinel-s1-l1c.s3.amazonaws.com"
    },
    {
      "x-amz-date": "20200819T132258Z"
    },
    {
      "x-amz-content-sha256":
"e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855"
    },
    {
      "x-amz-request-payer": "requester"
    },
    {
      "Authorization": "AWS4-HMAC-SHA256 Credential=xxxxxxxxxxxxxxxx/20200819/eu-
central-1/s3/aws4_request,SignedHeaders=host;x-amz-content-sha256;x-amz-date;x-amz-
request-
payer,Signature=5bf598d3b99f025765ead5db4711d5fe0bf37a65740e99c5331927f84d918f61"
    }
]
```

One of the components, x-amz-date, is a time stamp which means the signature need to be recomputed each time a download request is made. So, without the catalogue having a built-in capability to dynamically compute a signature putting a signed download URL into the catalogue record is a futile exercise.

The ultimate workaround/hack agreed upon by the thread participants was to host the Sentinel data earmarked for use during the testbed on the same host as the catalogue and include a link (rel="enclosure") in each catalogue record pointing to the corresponding local copy of the Sentinel product.

13.6. Records for model description

The purpose of this issue was to try to address how ML models can be described so that they can then be harvested into a catalogue and made discoverable.

The ONNX format only contains the model (as a graph) and its parameters. There is no additional information about pre-processing, output description, semantic information, etc.

MXNet has a richer model that includes:

1. Pre-trained MXNet Model (it can be an ONNX file).

2. A 'signature.json' describing the inputs and outputs of the models.
3. Semantic information as a 'synset.txt' containing the class names and synonims (a Synset is a special kind of a simple interface that is used to look up words in WordNet. Synset instances are the groupings of synonymous words that express the same concept. Some of the words have only one Synset and some have several).
4. Custom model service files for pre-processing.

Metadata was harvested from a model repository containing MXNet models and located at:

https://github.com/awslabs/multi-model-server/blob/master/docs/model_zoo.md

Queries that could be useful:

1. Query if the models has the relevant semantic outputs
2. The machine learning task (classification, semantic segmentation, instance recognition, etc.)
3. The size of the model (number of parameters)
4. Pre-processing or post-processing that must be applied
5. Performance on some known benchmark (e.g. ImageNet)

13.7. Metadata Extraction for LiDAR Datasets

This issue is concerned with how metadata can be extracted from LiDAR tiles (.laz) in order to identify relevant information in ML.

Details of the issue are presented here, however OGC Members can access the original content at: https://gitlab.ogc.org/ogc/t16-d015-machine-learning-er/-/issues/33

Data:

- Source: https://geonb.snb.ca/downloads2/lidar/2018/snb/aoi1/laz/nb_2018_2489000_7421000.laz
- General metadata: https://geonb.snb.ca/downloads2/lidar/2018/snb/aoi1/meta/meta_2018_aoi1.html

LiDAR tiles (.las), metadata can be extracted using LASTools:

```
lasinfo -i nb_2018_2489000_7421000_predictions.las
lasinfo (200304) report for 'nb_2018_2489000_7421000_predictions.las'
reporting all LAS header entries:
  file signature:             'LASF'
  file source ID:             0
  global_encoding:            1
  project ID GUID data 1-4:   00000000-0000-0000-0000-000000000000
  version major.minor:        1.2
  system identifier:          'LAStools (c) by rapidlasso GmbH'
  generating software:        'las2las (version 181108)'
  file creation day/year:     20/2019
  header size:                227
```

```
  offset to point data:       331
  number var. length records: 1
  point data format:          1
  point data record length:   28
  number of point records:    22404360
  number of points by return: 11826714 6626147 2971595 830861 135248
  scale factor x y z:         0.01 0.01 0.01
  offset x y z:               0 0 0
  min x y z:                  2489000.00 7421000.00 8.95
  max x y z:                  2489999.99 7421999.99 63.33
variable length header record 1 of 1:
  reserved             0
  user ID              'LASF_Projection'
  record ID            34735
  length after header  48
  description          'by LAStools of rapidlasso GmbH'
    GeoKeyDirectoryTag version 1.1.0 number of keys 5
      key 1024 tiff_tag_location 0 count 1 value_offset 1 - GTModelTypeGeoKey:
ModelTypeProjected
      key 3072 tiff_tag_location 0 count 1 value_offset 2953 - ProjectedCSTypeGeoKey:
NAD83(CSRS) / New Brunswick Stereographic
      key 3076 tiff_tag_location 0 count 1 value_offset 9001 - ProjLinearUnitsGeoKey:
Linear_Meter
      key 4099 tiff_tag_location 0 count 1 value_offset 9001 - VerticalUnitsGeoKey:
Linear_Meter
      key 4096 tiff_tag_location 0 count 1 value_offset 1127 - VerticalCSTypeGeoKey:
VertCS_Canadian_Geodetic_Vertical_Datum_2013
the header is followed by 2 user-defined bytes
reporting minimum and maximum for all LAS point record entries ...
  X           248900000  248999999
  Y           742100000  742199999
  Z                 895       6333
  intensity           0          0
  return_number       0          0
  number_of_returns   0          0
  edge_of_flight_line 0          0
  scan_direction_flag 0          0
  classification      0          8
  scan_angle_rank     0          0
  user_data           0          0
  point_source_ID     0          0
  gps_time 0.000000 0.000000
number of first returns:        22404360
number of intermediate returns: 0
number of last returns:         22404360
number of single returns:       22404360
WARNING: for return 1 real number of points by return (0) is different from header
entry (11826714).
WARNING: for return 2 real number of points by return (0) is different from header
entry (6626147).
WARNING: for return 3 real number of points by return (0) is different from header
```

```
entry (2971595).
WARNING: for return 4 real number of points by return (0) is different from header
entry (830861).
WARNING: for return 5 real number of points by return (0) is different from header
entry (135248).
WARNING: there are 22404360 points with return number 0
WARNING: there are 22404360 points with a number of returns of given pulse of 0
histogram of classification of points:
             490  never classified (0)
         4423717  unclassified (1)
        17817147  ground (2)
            4120  low vegetation (3)
            1899  medium vegetation (4)
           10282  high vegetation (5)
             596  building (6)
             703  noise (7)
          145406  keypoint (8)
```

For the purposes of training and testing, the relevant information could be:

```
histogram of classification of points:
             490  never classified (0)
         4423717  unclassified (1)
        17817147  ground (2)
            4120  low vegetation (3)
            1899  medium vegetation (4)
           10282  high vegetation (5)
             596  building (6)
             703  noise (7)
          145406  keypoint (8)
```

It gives the number of point per classes which can be useful to build a training/testing dataset. Also, the general metadata contains class definitions:

```
"eainfo": {
                "detailed": [
                    {
                        "enttyp": {
                            "enttypl": "LiDAR Point Cloud",
                            "enttypd": "ASPRS Codes used for this classification
\nASPRS 1 = Unclassified \nASPRS 2 = Ground \nASPRS 3 = Low Vegetation \nASPRS 4 =
Medium Vegetation \nASPRS 5 = High Vegetation \nASPRS 6 = Buildings \nASPRS 7 = Low
Noise \nASPRS 8 = Model Key-Point \nASPRS 9 = Water \nASPRS 17 = Bridge \nASPRS 18 =
High Noise",
                            "enttypds": "LAS Specification Version 1.2 Approved by
ASPRS Board 09/02/2008 http://www.asprs.org/wp-
content/uploads/2010/12/asprs_las_format_v12.pdf"
                        }
                    }
                ]
          }
```


Below is an attempt to save this information as a json file:

Python script: https://gist.github.com/sfoucher/94ae0f0438e1d5d335d39ed829c82357

and the json output for one tile:


```
{
    "lasinfo": {
        "file signature": "'LASF'",
        "file source ID": "0",
        "global_encoding": "1",
        "project ID GUID data 1-4": "00000000-0000-0000-0000-000000000000",
        "version major.minor": "1.2",
        "system identifier": "'LAStools (c) by rapidlasso GmbH'",
        "generating software": "'las2las (version 160329)'",
        "file creation day/year": "58/2013",
        "header size": 227,
        "offset to point data": 229,
        "number var. length records": 0,
        "point data format": 1,
        "point data record length": 28,
        "number of point records": 17924782,
        "number of points by return": [
            9596534,
            4570242,
            2475718,
            963955,
            261676
        ],
        "scale factor x y z": [
            0.01,
```

```
            0.01,
            0.01
        ],
        "offset x y z": [
            0,
            0,
            0
        ],
        "min x y z": [
            2480000,
            7436000,
            17.58
        ],
        "max x y z": [
            2480999.99,
            7436999.99,
            239.46
        ],
        "LASzip compression (version 2.4r2 c2 50000)": "POINT10 2 GPSTIME11 2",
        "number of first returns": 9596534,
        "number of intermediate returns": 3758885,
        "number of last returns": 9594332,
        "number of single returns": 5024969,
        "WARNING1": "there are 49406 points with return number 6",
        "WARNING2": "there are 7251 points with return number 7",
        "overview over number of returns of given pulse": [
            5024969,
            4187662,
            4535642,
            2810571,
            1062064,
            257021,
            46853
        ],
        "histogram of classification of points": [
            {
                "name": "ground",
                "label": 2,
                "count": 1414759
            },
            {
                "name": "low vegetation",
                "label": 3,
                "count": 3926298
            },
            {
                "name": "medium vegetation",
                "label": 4,
                "count": 484228
            },
            {
```

```
                "name": "high vegetation",
                "label": 5,
                "count": 12088992
            },
            {
                "name": "noise",
                "label": 7,
                "count": 128
            },
            {
                "name": "keypoint",
                "label": 8,
                "count": 10377
            }
        ]
    },
    "meta_2018_aoi1": {
        "metadata": {
            "idinfo": {
                "citation": {
                    "citeinfo": {
                        "origin": "Government of New Brunswick (comp.)",
                        "pubdate": "20181206",
                        "title": "SNB 2018 LiDAR \nAOI 1",
                        "edition": "001",
                        "geoform": "remote-sensing LiDAR",
                        "onlink": "https://geonb.snb.ca/li/index.html"
                    }
                },
                "descript": {
                    "abstract": "Aerial LiDAR Data",
                    "purpose": "To provide accurate and dense elevation data for the
terrain and the natural and man-made features on and above the terrain"
                },
                "timeperd": {
                    "timeinfo": {
                        "rngdates": {
                            "begdate": "20180611",
                            "enddate": "20180825"
                        }
                    },
                    "current": "publication date"
                },
                "status": {
                    "progress": "Complete",
                    "update": "None planned"
                },
                "spdom": {
                    "bounding": {
                        "westbc": "-67.20518808",
                        "eastbc": "-64.90306242",
```

```
                    "northbc": "46.15928300",
                    "southbc": "44.81861194"
                }
            },
            "keywords": {
                "theme": {
                    "themekt": "none",
                    "themekey": [
                        "lidar",
                        "aerial lidar",
                        "point cloud",
                        "classified point cloud",
                        "elevation"
                    ]
                },
                "place": {
                    "placekt": "Canadian Geographical Names Data Base (CGNDB)",
                    "placekey": [
                        "Saint John",
                        "Fredericton Junction",
                        "Saint George",
                        "New Brunswick"
                    ]
                },
                "temporal": {
                    "tempkt": "none",
                    "tempkey": "Summer"
                }
            },
            "accconst": "As outlined in the GeoNB Open Data Licence -
http://geonb.snb.ca/documents/license/geonb-odl_en.pdf",
            "useconst": "As outlined in the GeoNB Open Data Licence -
http://geonb.snb.ca/documents/license/geonb-odl_en.pdf",
            "ptcontac": {
                "cntinfo": {
                    "cntorgp": {
                        "cntorg": "Service New Brunswick",
                        "cntper": "GeoNB"
                    },
                    "cntpos": "LiDAR specialist",
                    "cntaddr": [
                        {
                            "addrtype": "mailing address",
                            "address": [
                                "Service New Brunswick",
                                "P.O. Box 1998"
                            ],
                            "city": "Fredericton",
                            "state": "NB",
                            "postal": "E3B 5G4",
                            "country": "Canada"
```

```
                        },
                        {
                            "addrtype": "physical address",
                            "address": [
                                "Service New Brunswick",
                                "985 College Hill Road"
                            ],
                            "city": "Fredericton",
                            "state": "NB",
                            "postal": "E3B 4J7",
                            "country": "Canada"
                        }
                    ],
                    "cntvoice": "(506) 457-3581",
                    "cntfax": "(506) 453-3898",
                    "cntemail": "geonb@snb.ca",
                    "hours": "0815 - 1630 AST (GMT - 0400), Monday to Friday"
                }
            },
            "browse": {
                "browsen":
"https://geonb.snb.ca/documents/lidar_browse_graphic/2018_LiDAR_AOI1.jpg",
                "browsed": "LiDAR project footprint displayed on map of New
Brunswick",
                "browset": "JPEG"
            },
            "secinfo": {
                "secsys": "None",
                "secclass": "Unclassified",
                "sechandl": "None"
            }
        },
        "dataqual": {
            "logic": "Unknown",
            "complete": "The LiDAR point cloud includes all hits",
            "posacc": {
                "horizpa": {
                    "horizpar": "LiDAR Horizontal Accuracy Statement",
                    "qhorizpa": {
                        "horizpav": "0.20",
                        "horizpae": "This data set was produced to meet ASPRS
Positional Accuracy Standards for Digital Geospatial Data (2014) for a 20 cm
RMSEx/RMSEy Horizontal Accuracy Class which equates to Positional Horizontal Accuracy
= +/- 49.0 cm at a 95% confidence level."
                    }
                },
                "vertacc": {
                    "vertaccr": "LiDAR NVA/VVA Vertical Accuracy Statements",
                    "qvertpa": [
                        {
                            "vertaccv": "0.105",
```

```
                                "vertacce": "This data set was tested to meet ASPRS
Positional Accuracy Standards for Digital Geospatial Data (2014) for a 10 cm RMSEz
Vertical Accuracy Class on RTK measured points. Actual NVA accuracy was found to be
RMSEz = 10.5 cm, equating to +/- 20.6 cm at 95% confidence level"
                            },
                            {
                                "vertaccv": "0.327",
                                "vertacce": "Actual VVA accuracy was found to be 32.7
cm at the 95th percentile."
                            }
                        ]
                    }
                },
                "lineage": {
                    "procstep": {
                        "procdesc": "The point cloud is initially produced using the
latest boresight values for the sensor. Preliminary quality assurance steps are taken
to ensure data integrity. A series of off the shelf software and proprietary tools are
utilized throughout the LiDAR data processing procedures.\n\nThe calibrated point
cloud strips are then processed into 1km tiles in the New Brunswick Double
Strereographic projection. Next, a combination of automatic and manual classification
is done separating points into ground (class 2), non ground (class 1), low noise
(class 7) and high noise (class 18). The manual and visual inspection plays a major
role in improving the classification accuracy. \n\nOnce the classification of the
ground class is deemed final, automatic classification is run on all non-ground
points.  Project specifications for vegetation classification is: above ground points
to 50cm - low vegetation (class 3), above 50cm to 2m - medium vegetation (class 4),
above 2m - high vegetation (class 5). A manual/visual QC pass is made to fine tune the
classification of points, including the manual classification of buildings (class 6)
and bridges (class 17).  Ground above culverts is left in the ground class. Further
classification of water and water bodies is done with the following parameters: water
bodies and water courses meeting specifications of greater than 3600 m2 (water
bodies), greater than 10 m nominal width (water courses) and at least 220 m in length
are delineated. LiDAR points falling within these areas are classified the water class
(class 9).  Islands greater than 100m2 are delineated within any water feature and not
classified as water.\n\nThe final 1m Bare Earth DEMs are produced from the TIN of the
\"ground\" and \"model keypoint\" classes (2 & 8). Similarly, Full Feature grids are
produced by using all non-noise points.",
                        "procdate": "Unknown",
                        "proccont": {
                            "cntinfo": {
                                "cntperp": {
                                    "cntper": "Data Processing Manager",
                                    "cntorg": "Airborne Imaging"
                                },
                                "cntaddr": {
                                    "addrtype": "mailing and physical address",
                                    "address": "2700 61 Avenue SE",
                                    "city": "Calgary",
                                    "state": "Alberta",
                                    "postal": "T2C 4V2",
```

```
                                "country": "Canada"
                            },
                            "cntvoice": "403 215 2960",
                            "cntfax": "403 258 3189",
                            "hours": "0900 - 1500 MST",
                            "cntinst": "All questions should first be directed to
GeoNB (geonb@snb.ca)"
                        }
                    }
                }
            }
        },
        "spref": {
            "horizsys": {
                "planar": {
                    "gridsys": {
                        "gridsysn": "other grid system",
                        "othergrd": "PROJCS[\"NAD83(CSRS) / New Brunswick Stereo
\", GEOGCS[\"NAD83(CSRS)\", DATUM[\"D_North_American_1983_CSRS98\", SPHEROID[
\"GRS_1980\", 6378137, 298.257222101]], PRIMEM[\"Greenwich\",0], UNIT[\"Degree
\",0.0174532925199433]], PROJECTION[\"Double_Stereographic\"], PARAMETER[
\"Latitude_Of_Origin\",46.5], PARAMETER[\"central_meridian\",-66.5], PARAMETER[
\"scale_factor\",0.999912], PARAMETER[\"false_easting\",2500000], PARAMETER[
\"false_northing\",7500000], UNIT[\"Meter\",1]]\n\nEPSG 2953"
                    },
                    "planci": {
                        "plance": "coordinate pair",
                        "coordrep": {
                            "absres": "0.01",
                            "ordres": "0.01"
                        },
                        "plandu": "metres"
                    }
                },
                "geodetic": {
                    "horizdn": "NAD83 (CSRS)",
                    "ellips": "Geodetic Reference System 80",
                    "semiaxis": "6378137.0",
                    "denflat": "298.257222101"
                }
            },
            "vertdef": {
                "altsys": {
                    "altdatum": "Canadian Geodetic Vertical Datum of 2013
(GCVD2013)",
                    "altres": "0.0001",
                    "altunits": "metres",
                    "altenc": "Explicit elevation coordinate included with
horizontal coordinates"
                }
            }
```

```
            },
            "eainfo": {
                "detailed": [
                    {
                        "enttyp": {
                            "enttypl": "LiDAR Point Cloud",
                            "enttypd": "ASPRS Codes used for this classification
\nASPRS 1 = Unclassified \nASPRS 2 = Ground \nASPRS 3 = Low Vegetation \nASPRS 4 =
Medium Vegetation \nASPRS 5 = High Vegetation \nASPRS 6 = Buildings \nASPRS 7 = Low
Noise \nASPRS 8 = Model Key-Point \nASPRS 9 = Water \nASPRS 17 = Bridge \nASPRS 18 =
High Noise",
                            "enttypds": "LAS Specification Version 1.2 Approved by
ASPRS Board 09/02/2008 http://www.asprs.org/wp-
content/uploads/2010/12/asprs_las_format_v12.pdf"
                        }
                    },
                    {
                        "enttyp": {
                            "enttypl": "1m 32bit GeoTIFF Digital Elevation Model
(DEM)",
                            "enttypd": "Bare Earth Digital Elevation Model created
from ASPRS Classes 2 and 8",
                            "enttypds": "LAS Specification Version 1.2 Approved by
ASPRS Board 09/02/2008 http://www.asprs.org/wp-
content/uploads/2010/12/asprs_las_format_v12.pdf"
                        }
                    },
                    {
                        "enttyp": {
                            "enttypl": "1m 32bit GeoTIFF Digital Elevation Model
(DSM)",
                            "enttypd": "Full Feature Digital Elevation Model created
from ASPRS Classes 1,2,3,4,5,6,8,9 and 17 and creating a grid surface of the highest
elevations within each cell",
                            "enttypds": "LAS Specification Version 1.2 Approved by
ASPRS Board 09/02/2008 http://www.asprs.org/wp-
content/uploads/2010/12/asprs_las_format_v12.pdf"
                        }
                    }
                ]
            },
            "distinfo": {
                "distrib": {
                    "cntinfo": {
                        "cntorgp": {
                            "cntorg": "Service New Brunswick",
                            "cntper": "GeoNB"
                        },
                        "cntpos": "LiDAR specialist",
                        "cntaddr": [
                            {
```

```
                            "addrtype": "mailing address",
                            "address": [
                                "Service New Brunswick",
                                "P.O. Box 1998"
                            ],
                            "city": "Fredericton",
                            "state": "NB",
                            "postal": "E3B 5G4",
                            "country": "Canada"
                        },
                        {
                            "addrtype": "physical address",
                            "address": [
                                "Service New Brunswick",
                                "985 College Hill Road"
                            ],
                            "city": "Fredericton",
                            "state": "NB",
                            "postal": "E3B 4J7",
                            "country": "Canada"
                        }
                    ],
                    "cntvoice": "(506) 457-3581",
                    "cntfax": "(506) 453-3898",
                    "cntemail": "geonb@snb.ca",
                    "hours": "0815 - 1630 AST (GMT - 0400), Monday to Friday"
                }
            },
            "resdesc": "classified LiDAR point cloud",
            "distliab": "As outlined in the GeoNB Open Data Licence -
http://geonb.snb.ca/documents/license/geonb-odl_en.pdf"
        },
        "metainfo": {
            "metd": "20190308",
            "metc": {
                "cntinfo": {
                    "cntorgp": {
                        "cntorg": "Service New Brunswick",
                        "cntper": "GeoNB"
                    },
                    "cntpos": "LiDAR Specialist",
                    "cntaddr": [
                        {
                            "addrtype": "physical address",
                            "address": [
                                "Service New Brunswick",
                                "985 College Hill Road"
                            ],
                            "city": "Fredericton",
                            "state": "NB",
                            "postal": "E3B 4J7",
```

```
                            "country": "Canada"
                        },
                        {
                            "addrtype": "mailing address",
                            "address": [
                                "Service New Brunswick",
                                "P.O. Box 1998"
                            ],
                            "city": "Fredericton",
                            "state": "NB",
                            "postal": "E3B 5G4",
                            "country": "Canada"
                        }
                    ],
                    "cntvoice": "(506) 457-3581",
                    "cntfax": "(506) 453-3898",
                    "cntemail": "geonb@snb.ca",
                    "hours": "0815 - 1630 AST (GMT - 0400), Monday to Friday"
                }
            },
            "metstdn": "FGDC Content Standards for Digital Geospatial Metadata",
            "metstdv": "FGDC-STD-001-1998",
            "mettc": "local time"
        }
    }
},
"links": [
    {
        "title": "LAZ Source Data",
        "rel": "data",
        "href": "nb_2015_2480000_7436000"
    },
    {
        "title": "LAZ Metadata",
        "rel": "meta",
        "href":
"https://geonb.snb.ca/downloads2/lidar/2018/snb/aoi1/meta/meta_2018_aoi1.xml"
    }
]
}
```


13.7.1. OGC API - Records

First attempt using pygeoapi based on this geojsonfile:


```
{
    "type": "FeatureCollection",
    "features": [
        {
            "type": "Feature",
```

```
            "properties": {
                "id": "nb_2015_2480000_7436000",
                "name": "nb_2015_2480000_7436000",
                "featureclass": "LIDAR",
                "LAZ Metadata":
"https://geonb.snb.ca/downloads2/lidar/2018/snb/aoi1/meta/meta_2018_aoi1.xml",
                "onlink": "https://geonb.snb.ca/li/",
                "histogram of classification of points": [
                    {
                        "name": "ground",
                        "label": 2,
                        "count": 1414759
                    },
                    {
                        "name": "low vegetation",
                        "label": 3,
                        "count": 3926298
                    },
                    {
                        "name": "medium vegetation",
                        "label": 4,
                        "count": 484228
                    },
                    {
                        "name": "high vegetation",
                        "label": 5,
                        "count": 12088992
                    },
                    {
                        "name": "noise",
                        "label": 7,
                        "count": 128
                    },
                    {
                        "name": "keypoint",
                        "label": 8,
                        "count": 10377
                    }
                ],
                "class_names": [
                    "ground",
                    "low vegetation",
                    "medium vegetation",
                    "high vegetation",
                    "noise",
                    "keypoint"
                ],
                "class_labels": [
                    2,
                    3,
                    4,
```

```
                    5,
                    7,
                    8
                ],
                "class_count": [
                    1414759,
                    3926298,
                    484228,
                    12088992,
                    128,
                    10377
                ],
                "class_frequency": [
                    7.89275428844825,
                    21.904299868193654,
                    2.7014442909263834,
                    67.44289553981744,
                    0.0007140951560805593,
                    0.05789191745818722
                ],
                "bbox": [
                    248000000,
                    248099999,
                    743600000,
                    743699999
                ]
            },
            "geometry": {
                "type": "Polygon",
                "coordinates": [
                    [
                        [
                            248000000,
                            743600000
                        ],
                        [
                            248099999,
                            743600000
                        ],
                        [
                            248099999,
                            743699999
                        ],
                        [
                            248099999,
                            743600000
                        ]
                    ]
                ]
            }
        }
```

```
    ]
}
```

Screenshot of the item in pygeoapi:

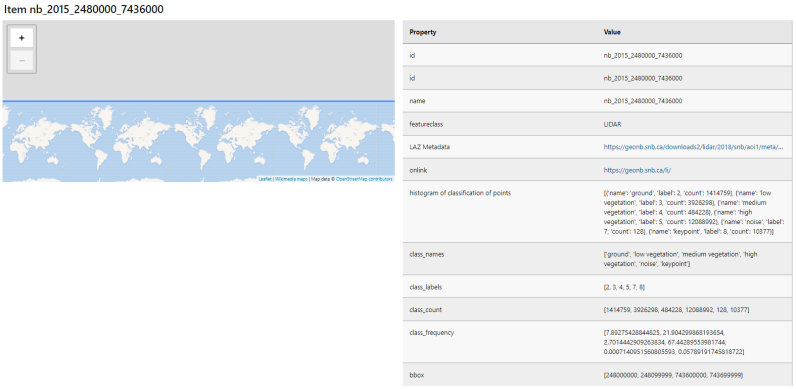


Figure 27. pygeoapi item sample

The next question is how can queries be formulated on the properties "histogram of classification of points", class_count or class_names?

One possibility would be to use CQL queries. Possible queries for a ML engineer:

- "I want all the tiles with a given set of classes."
- "I want all of tile with a minimum number of points for a given class."

13.7.2. Experiments using QGIS:

- An API record was created using pygeoapi:

```
    nb_lidar:
        type: collection
        title: NB LiDAR metadata record
        description: NB LiDAR Data
        keywords:
            - LiDAR
        links:
            - type: text/html
              rel: canonical
              title: information
              href: https://geonb.snb.ca/li/
              hreflang: en-US
        extents:
            spatial:
                bbox: [-69.05, 44.56, -63.7, 48.07]
                crs: http://www.opengis.net/def/crs/EPSG/9.8.15/2953
            temporal:
                begin: 2011-11-11
                end: null  # or empty (either means open ended)
        providers:
            - type: feature
              name: GeoJSON
              data: tests/data/nb_lidar.json
              id_field: id
```

A Vector layer is then directly added in QGIS based on the API record URL (http://localhost:5000/collections/nb_lidar/items?f=json)

QGIS has a powerful filter engine: https://docs.qgis.org/testing/en/docs/user_manual/working_with_vector/expression.html

Here is a classification map produced by lasgrid:

```
lasgrid -i nb_2018_2489000_7421000_predictions.las -o test.tif -step 0.5
-classification -false
```

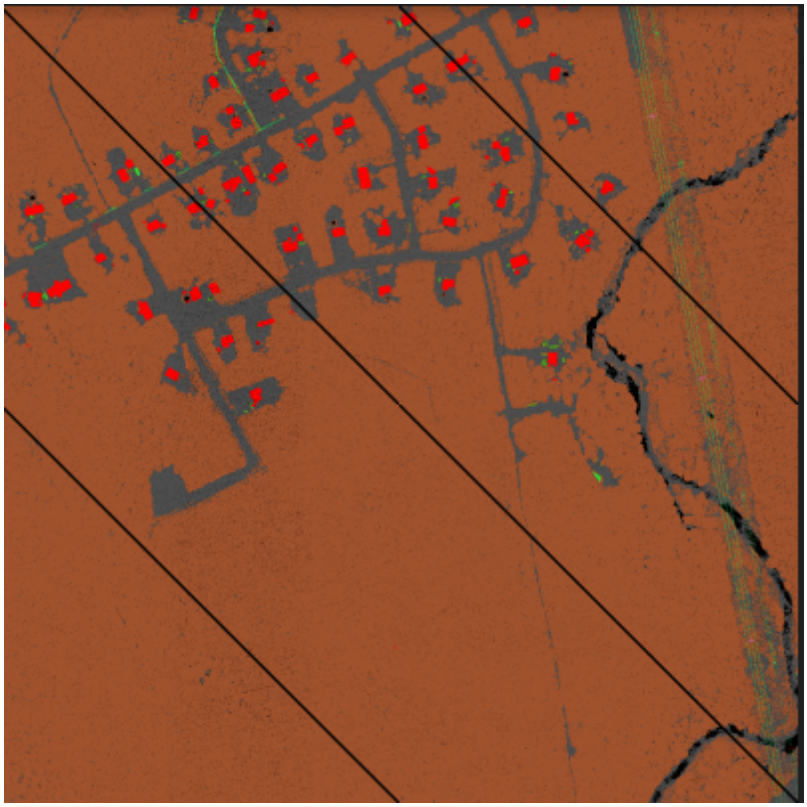


Figure 28. lasgrid output image

The issue also included a file, nb_lidar.json, that is meant to be loaded into a catalogue in order to test CQL queries. CQL could query this data one loaded into a catalogue but not without the server supporting JSON path expression so that predicates could be formulated.

Here are some sample CQL queries. These queries rely on the server exposing the following paths as queryables:

- class_histograms[*].name as class_name
- class_histograms[*].count as clause_count
- "I want all the tiles with a given set of classes."

```
cql=class_name in ('high vegetation','low vegetation')
```

- "I want all of tile with a minimum number of points for a given class".

```
cql=class_name='medium vegetation' and class_count>150000
```

```
...
"class_histograms": [{
   "name": "unclassified",
   "label": 1,
   "count": 91
   }, {
   "name": "ground",
   "label": 2,
   "count": 2185856
   }, {
   "name": "low vegetation",
   "label": 3,
   "count": 1631259
   }, {
   "name": "medium vegetation",
   "label": 4,
   "count": 1651525
   }, {
   "name": "high vegetation",
   "label": 5,
   "count": 17728529
   }, {
   "name": "noise",
   "label": 7,
   "count": 3497
   }, {
   "name": "keypoint",
   "label": 8,
   "count": 10768
   }],
...
```

Appendix A: Revision History

Table 11. Revision History

Date	Editor	Release	Primary clauses modified	Descriptions
May, 2020	P. Vretanos	.1	all	IER version
October, 2020	P. Vretanos	.2	all	draft DER version
January 4, 2021	S. Serich	.3	all	top-to-bottom feedback from NRCan sponsor

Appendix B: Bibliography

[1] Robert Linder, E.P., Joan Masó: Map Markup Language. HTML Community Group, https://maps4html.org/MapML/spec/ (2020).

[2] GeoServer MapML Plug-in. GeoServer, https://docs.geoserver.org/latest/en/user/community/mapml/index.html (2020).