Earth Observation & Weather Data Federation with AI Embeddings

D3.2:

Data Access for AI Compressors

Revision: v 1.1

D3.2: Data Access for AI compressors

Work package	3
Task	3.3
Due date	31/12/2024
Submission date	19/12/y2024
Deliverable lead	WWU
Version	1.0
Authors	Jonas Hurst
Reviewers	Romeo Kienzler (IBM), Conrad Albrecht (DLR)
Abstract	To enable data access for the AI compressors, we are using STAC for cataloging available data and providing metadata for it, and openEO for processing of the data.
Keywords	STAC, openEO

Document Revision History

Version	Date	Description of change	List of contributor(s)
V1.0	28/11/2024	1st edit	Jonas Hurst
V1.1	17/12/2024	Incorporate reviewer feedback	Jonas Hurst
V1.2	11/03/2025	Incorporate external reviewer feedback	Jonas Hurst
v1.3	24/03/2025	Incorporate internal reviewer feedback	Jonas Hurst

Funded by the European Union

Grant Agreement No: 101131841 | **Topic:** HORIZON-EUSPA-2022-SPACE-02-55

Call: HORIZON-EUSPA-2022-SPACE | **Type of action:** HORIZON-RIA

DISCLAIMER

Project funded by

Swiss Confederation

Embed2Scale (Earth Observation & Weather Data Federation With AI Embeddings) project is funded by the EU's Horizon Europe program under Grant Agreement number 101131841. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them. This work has also received funding from the Swiss State Secretariat for Education, Research and Innovation (SERI) and the UK Research and Innovation (UKRI).

COPYRIGHT NOTICE

Project funded by the European Commission in the Horizon Europe Programme		
Nature of the deliverable:	R	
Dissemination Level		
PU	Public, fully open, e.g. web (Deliverables flagged as public will be automatically published in CORDIS project's page)	☐
SEN	Sensitive, limited under the conditions of the Grant Agreement	
Classified R-UE/ EU-R	*EU RESTRICTED under the Commission Decision No2015/ 444*	
Classified C-UE/ EU-C	*EU CONFIDENTIAL under the Commission Decision No2015/ 444*	
Classified S-UE/ EU-S	*EU SECRET under the Commission Decision No2015/ 444*	

* *R: Document, report (excluding the periodic and final reports)*
DEM: Demonstrator, pilot, prototype, plan designs
DEC: Websites, patents filing, press & media actions, videos, etc.
DATA: Data sets, microdata, etc.
DMP: Data management plan
ETHICS: Deliverables related to ethics issues.
SECURITY: Deliverables related to security issues
OTHER: Software, technical diagram, algorithms, models, etc.

TABLE OF CONTENTS

1. Introduction **7**
2. STAC-API **8**
2.1 STAC with pgSTAC 8
2.2 STAC-API with stac-fastapi-pgstac 8
2.3 Deployment 9
3. openEO **10**
3.1 tensorlakehouse-openeo-driver 11
3.2 Authentication 12
3.3 Example: Connecting to a backend 12
3.4 Headless openEO 13
3.5 Deployment 14
4. Conclusions **15**

EXECUTIVE SUMMARY

The Embed2Scale project utilizes diverse datasets hosted across multiple data stores and platforms. To facilitate seamless access for project partners, a unified and simplified API is required.

To achieve this, critical datasets relevant to the project have been consolidated and stored at the Forschungszentrum Jülich data center. These datasets are to be cataloged using the Spatio-Temporal Asset Catalog (STAC) framework, ensuring comprehensive metadata management and discoverability.

Additionally, openEO has been chosen to provide standardized access and processing capabilities for the datasets (and eventually, also the embeddings) hosted in the different partner data centers, streamlining analysis workflows and promoting interoperability and federation.

Funded by the European Union

LIST OF FIGURES

FIGURE 1 Interaction between pgSTAC container and stac-fastapi-pgstac container to enable data access through STAC-API 9

FIGURE 2 Schematic showing how openEO clients interacting with openEO-compliant backends 10

FIGURE 3 Components and their interaction of tensorlakehouse-openeo-driver 11

FIGURE 4 Python notebook showcasing how to invoke processing on an openEO-compliant backend, exemplified by computing an NDVI over a time series of Sentinel-2 images. 12

FIGURE 5 Python notebook showcasing headless openEO computations, exemplified by computing an NDVI for a Sentinel-2 image. 14

Funded by the European Union

ABBREVIATIONS

API	Application Programming Interface
COS	Cloud Object Storage
CDSE	Copernicus Dataspace Ecosystem
HTTP	Hypertext Transfer Protocol
ML	Machine Learning
NDVI	Normalized Difference Vegatation Index
STAC	SpatioTemporal Asset Catalog
TCP	Transmission Control Protocol
VM	Virtual Machine

1. INTRODUCTION

Different kinds of data used within the Embed2Scale project lie in different storage systems and are available through a wide range of application specific APIs. For the simplified development of AI compressors, it is important for project partners to have unified access and discoverability to the data, both for pre-training of the AI compressors as well as for compressing after pre-training. Accessing this data shall be possible through simple APIs based on open standards. To decide which API standard we are going to use to access the data, we have evaluated different API standards commonly used in the geospatial domain in Task T1.2 and reported on them this project's deliverable D1.1.

We concluded to use a combination of SpatioTemporal Asset Catalog (**STAC**) and **openEO** to access the data. While details on how we fixed this decision can be found in D1.1, a short summary on the reasons is given in the following:

- **Well-established:**
  Both STAC and openEO are well-established and well-known within the EO community
- **Modern (RESTful):**
  The APIs follow a RESTful, resource-oriented approach, which at the time of writing is state-of-the-art. They rely on JSON for client-server communication instead of XML, which is known for entailing a large overhead. JSON further aligns well with data structures in Python, a programming language commonly used by deep learning scientists.
- **Management of metadata:**
  STAC allows cataloging its assets with metadata that are needed to make sense of the data (and embeddings) it serves.
- **Extensibility:**
  Both STAC and openEO are open-source community standards, so extending it can be done relatively easily, without going through a lengthy bureaucratic process. STAC can be extended through STAC-Extensions. openEO relies on a set of granular processes which when chained together will create an analysis workflow.
- **Queryable:**
  The API must allow querying the data it gives access to by metadata to allow consumers to easily find data that fit given metadata criteria.

In the following, we are going to present the undertakings we have done within T3.3 to allow access to EO data through openEO and STAC.

All components we use within our deployment are open-source with their code available on GitHub [1], a platform commonly used to host code for open-source projects.

In the following, we are first going to report on the STAC deployment (Chapter 2), then we will report on the openEO deployment (Chapter 3), before drawing a conclusion and outlining further steps (Chapter 4).

2. STAC-API

STAC is an established data access API in the EO community for cataloging available data in a standardized way while also providing metadata about the data that has been cataloged in the STAC. STAC is a community standard, meaning that no governing body exists to make decisions about its development. Rather, all decisions and developments are made by the community of its users. Consequently, a large number of open-source components are available to facilitate the deployment of a STAC and a STAC-API. This chapter lists the open-source components, which we are basing our STAC infrastructure upon to allow access to data needed during model training.

This report aims to describe how STAC-API will be utilized within the Embed2Scale project. Detailed information on how users can interact with it can be found in its specification [2].

2.1 STAC WITH PGSTAC

The pgSTAC project [3] has developed a data model optimized for storing STAC collections and items in a database. The project consists of a collection of SQL scripts to set up this data model in a Postgres database with PostGIS extension. It is highly performant and provides STAC filtering and CQL2 search.

For simple deployment, prebuilt docker images are available on GitHub's container registry [4]. These images contain the postgres database with the pgSTAC schema applied. It uses user-defined functions (UDF) written in RUST to further accelerate performance.

PyPgSTAC [5] is a python-based tool to simplify the interaction with STAC databases powered by the pgSTAC schema. It can either be used through the command-line or it can be called directly from within a python script. PyPgSTAC allows adding new STAC collections to the pgSTAC database, and adding STAC items to existing collections. It further contains useful tools for managing the pgSTAC database and metadata, e.g. for updating the spatial extent of a collection in case this extent has changed due to items being added to or removed from said collection.

2.2 STAC-API WITH STAC-FASTAPI-PGSTAC

The stac-fastapi project [6] implements the endpoints that are required by the STAC-API specification [7]. It is written in Python and uses the FastAPI web framework [8].

The gap between the API endpoint definitions with stac-fasatpi and the STAC database model with pgSTAC is bridged with stac-fastapi-pgstac [9]. It accepts HTTP requests using the endpoints defined in stac-fastapi, queries the pgSTAC database schema to respond to calls to the STAC-API appropriately, serializes the data it has queried from the database as JSON, then responds to the HTTP request by sending this JSON back to the requesting client.

For easy deployment of a pgSTAC powered STAC-API with stac-fastapi-pgstac, a docker-compose.yaml file is provided. It creates two docker containers: Firstly, a container hosting the postgres database is created with the pgSTAC schemas applied. A second container hosts the STAC-API endpoint implementation from stac-fastapi to serve the content of the pgSTAC database in a STAC-compliant manner. The two containers are connected through a docker network. This interaction between the containers is illustrated in Figure 1.

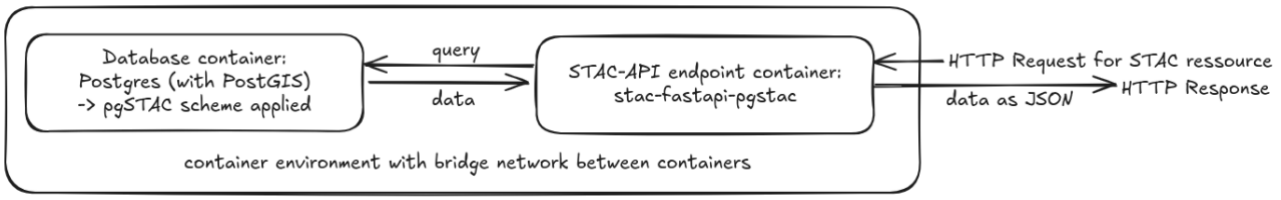

Figure 1: Interaction between pgSTAC container and stac-fastapi-pgstac container to enable data access through STAC-API

2.3 DEPLOYMENT

Image data needed for pre-training of the AI compressors has already been transferred to the data center at Forschungszentrum Jülich (FZJ), which is part of the Embed2Scale consortium and which is the location where AI compressor model pre-training will take place. Specifically, 9 million samples of the SSL4EO dataset [10] and the MajorTOM [11] dataset for multiple modalities (Sentinel2-L1C, Sentinel2-L2A, Sentinel1, DEM) have been transferred to the data center

For deployment of the STAC-API, we plan to create a virtual machine (VM) on the Copernicus Dataspace Ecosystem (CDSE) running Linux.

On this VM, we set up stac-fastapi-pgstac inside a docker environment as described above. Within the pgSTAC database, we aim to create a STAC collection for the images which have already been transferred to the FZJ data center. Consequently, URLs of all STAC-assets will point to their location at FZJ, so that when accessing the data from within FZJ (e.g. for pre-training), latency is minimized and they will not be transferred to the data center a second time. This ensures that efficient pre-training of AI compressor models will be possible.

3. OPENEO

To enable interaction with the data and to process it directly in the data center without needing to download it to a local machine first, we are opting to use openEO [12] as a processing engine for the imagery. OpenEO is a standardized API to interact with backends offering image processing capabilities. OpenEO client libraries exist for many programming languages (e.g. Python [13] and R [14]) giving EO analysts the freedom to choose a programming language of their liking. Figure 2 demonstrates how different clients are able to interact with openEO-compliant backends through HTTP requests.

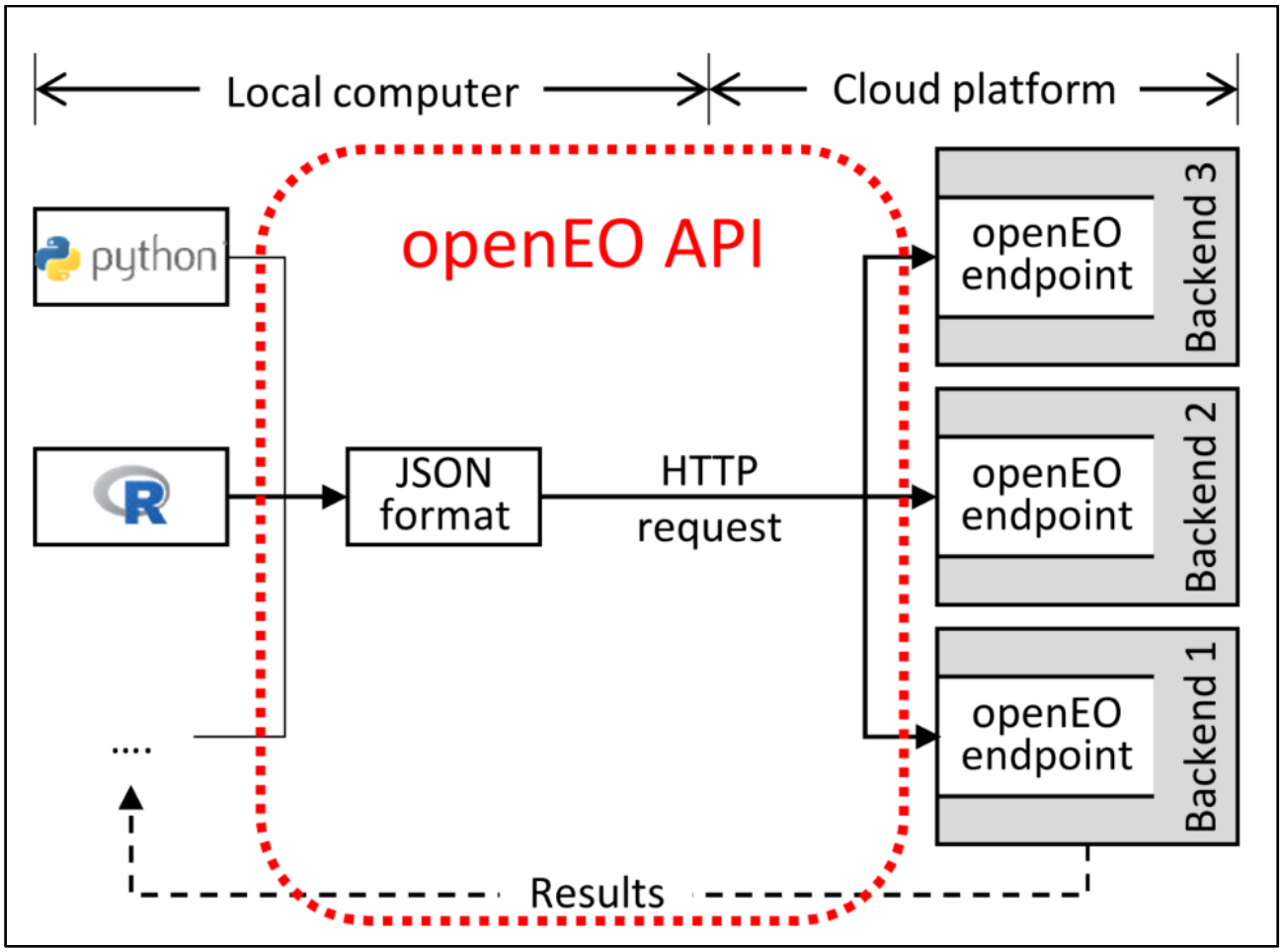

Figure 2: Schematic showing how openEO clients interact with openEO-compliant backends [12]

OpenEO integrates well with STAC, as openEO queries local or remote STACs to load data into an openEO workflow. Likewise, results of an openEO workflow are provided as a STAC collection and could therefore serve as input to a new openEO workflow.

This report does not aim to explain to readers how they can interact with openEO-compliant backends to invoke remote processing of earth observation data. For this, please refer to its official documentation [15]

3.1 TENSORLAKEHOUSE-OPENEO-DRIVER

IBM, one of the Embed2Scale consortium partners, is maintainer of an open-source openEO backend implementation called tensorlakehouse-openeo-driver [16].

It is written in Python and relies on open-source components to offer an openEO-compliant API interface. Most importantly, it relies on an openEO process implementation called openeo-processes-dask [17], which itself relies on xarray to perform EO datacube processing. Xarray itself is built on top of dask, which enables computation of xarray operations in parallel, not only locally but also across HPC clusters. The schematic depicted in Figure 3 shows the different components tensorlakehouse-openeo-driver is composed of, and how they interact with each other.

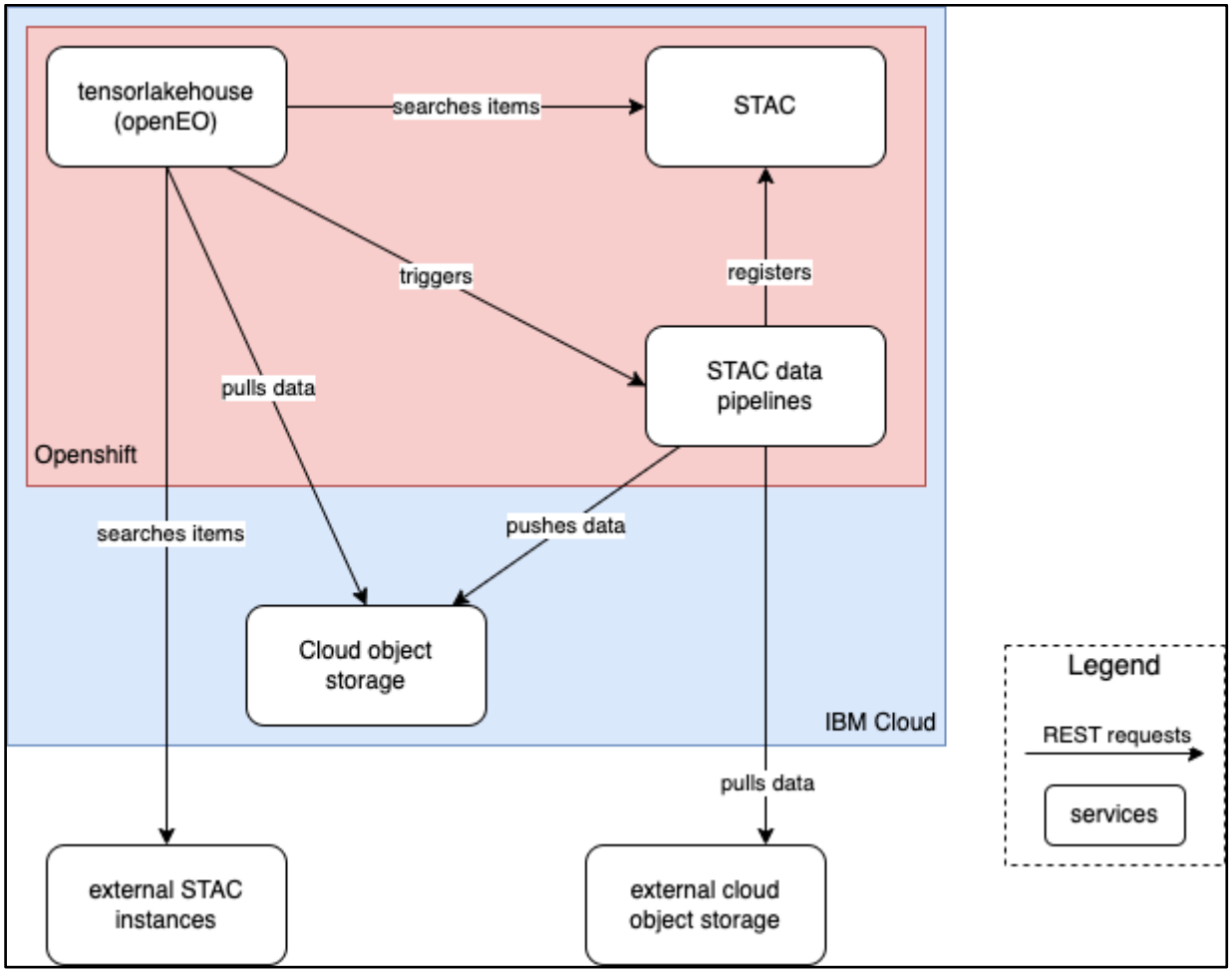

Figure 3: Components and their interaction of tensorlakehouse-openeo-driver

To briefly summarize, the tensorlakehouse-openeo-driver is able to offer EO processing capabilities through an openEO compliant interface, while being able to scale computations across an HPC cluster.

3.2 AUTHENTICATION

As the openEO-API is designed to invoke processing on remote machines, a mechanism to control access to the compute resources is essential. This ensures that only users who are authenticated and authorized to invoke computation are able to do so, thereby protecting the compute resources

against bad actors with malicious intent. OpenEO supports two different methods by which users can authenticate against the openEO-API [18]. Tensorlakehouse-openeo-driver supports both methods of authentication.

When authenticating using the *HTTP Basic* authentication scheme, username and password are transmitted to the backend by using the HTTP Authroization header. This method is however discouraged, as credentials are transmitted in clear text when the connection is unencrypted.

Rather, it is recommended to *OpenID Connect*, where users authenticate against an independent identity provider. When a user successfully authenticates against the identity provider, it will then issue a token to be validated by the openEO-backend to grant access to the resources.

We are also going to benefit from using an independent identity provider later in the project, when multiple openEO instances are going to be used to facilitate data federation: in this scenario, a single user will have to authenticate with multiple openEO instances as described in [19], which can then all rely on this single independent identity provider.

3.3 EXAMPLE: CONNECTING TO A BACKEND

The following Python code snippet in Figure 4 shows how a user can connect to an openEO compliant service, authenticate with OpenID Connect, invoke the computation of a Normalized Difference Vegetation Index (NDVI) on the backend and download the results to the local machine.

```python
import openeo
con = openeo.connect("https://link-to-backend.org/openeo")
con.authenticate_oidc()

cube = con.load_collection(
    "SENTINEL2_L2A",
    spatial_extent={"west": 8.327804, "east": 8.492775, "north": 49.041750, "south": 48.986638},
    temporal_extent=["2024-08-01", "2024-08-31"],
    bands=["B04", "B08"]
)

red = cube.band("B04")
nir = cube.band("B08")
ndvi = (nir - red) / (nir + red)

result = ndvi.save_result()
job = result.create_job()
job.start_and_wait()
job.get_results().download_files("out")
```

Figure 4: Python notebook showcasing how to invoke processing on an openEO-compliant backend, exemplified by computing an NDVI over a time series of Sentinel-2 images.

3.4 HEADLESS OPENEO

Alternatively, the openeo python client library [13], usually used to connect to openEO compliant backends, also offers headless execution of openEO python scripts: In this case, no backend service is needed, all computations happen within a local Python environment. This is beneficial in case we cannot deploy the tensorlakehouse-openeo-driver backend from IBM in the HPC cluster (e.g. the

cluster does not support Kubernetes environments). In this case, we can simply run the Headless openEO script on the HPC cluster.

This option of headless execution of openEO python scripts also uses the openeo-processes-dask [17] implementation of openEO processes. Therefore, it also has the advantage of xarray and dask, to scale horizontally across large HPC clusters by parallelizing computations.

The following Python code snippet in Figure 4 exemplifies briefly how such a headless openEO deployment is able to access locally hosted data. In this simple example, we use the headless openEO instance to compute a simple NDVI across an image saved on the local machine.

The downside of using openEO headlessly is that simply splitting the OpenEO execution graph and sending the splits to different OpenEO endpoints residing at different data centers for data and processing federation as envisioned in [19] is not possible. Instead, data federation using headless OpenEO can be achieved by efficiently applying the Hollywood principle (*"don't call us, we'll call you"*). In this setup, workers within distributed data centers independently pull tasks from a centralized task queue, execute them, and push the results to a centralized object storage (COS). This approach simplifies network and firewall configurations, as outbound TCP connections from the data center to the internet are typically allowed, whereas allowing inbound TCP connections to services running within the data center are often restricted. By leveraging this architecture, automated data and processing federation becomes seamless, eliminating the need for manual invocation across multiple data centers and streamlining the entire workflow.

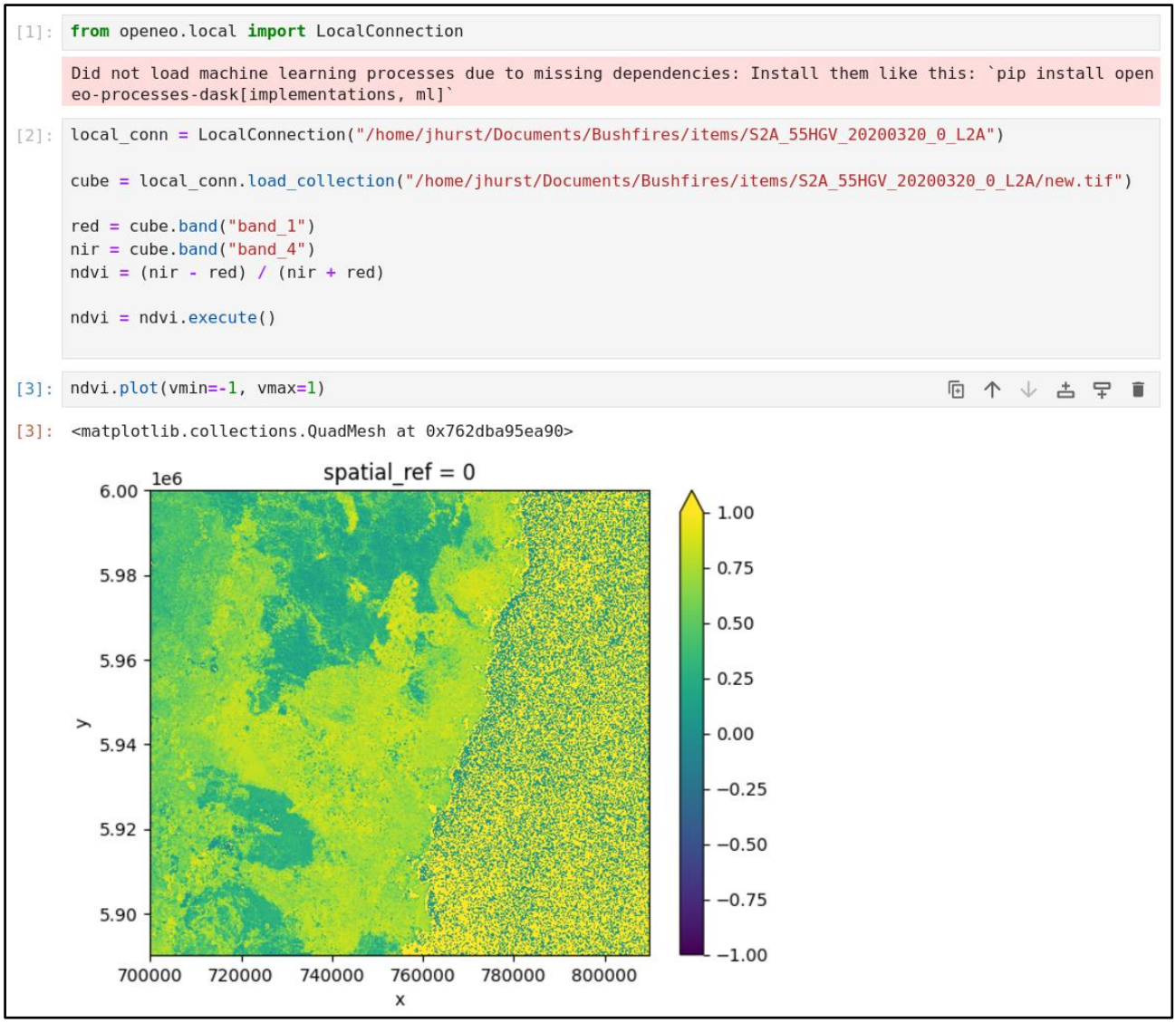

Figure 5: Python notebook showcasing headless openEO computations, exemplified by computing an NDVI for a Sentinel-2 image.

3.5 DEPLOYMENT

It is our aim to deploy the tensorlakehouse-openeo-driver openEO backend at the Jülich data center. By connecting it to the STAC deployment outlined in Chapter 2, we can invoke EO processing scripts at the data center easily to perform computations on the data that already reside there.

In case it is not possible to deploy the backend, the goal is to set up an openEO headless environment, so that the openEO processing scripts can still be executed, even though it may not be as simple as sending HTTP requests to the data center.

4. CONCLUSIONS

By deploying a STAC and an openEO instance, we can create an efficient infrastructure to not only access the data, but to also process it at large scales in an HPC environment.

In its current form, openEO supports machine learning (ML) methods only on a prototypical level. In a next step (Task T1.3), we are going to extend the specification to support ML on a much more elaborate level, such that openEO can be used to not only generate embeddings from foundation models, but also compress, process and analyze them within an openEO process graph.

After specifying the interfaces for both the openEO API as well as openEO-Processes needed to perform ML within openEO, we are going to implement them into the openeo-processes-dask collection of openEO process implementations. As IBM's tensorlakehouse-openeo-backend already relies on openeo-processes-dask, deploying the implemented processes should then be relatively straightforward.

As a general roadmap, openEO must be extended in the following ways to be able to handle embeddings and AI compressors:

- **Load embeddings into an openEO workflow:**
  If the embeddings have been pre-computed on another machine, it must be possible to load them into an openEO workflow. However, it shall also be possible to compute the embeddings within the openEO workflow.
- **Export embeddings from openEO workflow:**
  After computing the embeddings, it must be possible to write them to file for downloading, using an appropriate data format. Using AI-compressors in this step is crucial.
- **Process embeddings within openEO Workflow**
  It must be possible to process and analyze the embeddings within the openEO workflow. This requires more advanced ML capabilities within openEO beyond what is currently available.

To foster re-use of trained machine learning models across openEO workflows, exporting them from one workflow and importing them in another is crucial. To manage metadata of the saved models, STAC and the recently published Machine Learning Model (MLM) extension [20] can be used.

These measures are going to enable openEO to work with AI-compressed embeddings, and will thereby contribute to make openEO a modern analysis platform for EO data.

REFERENCES

[1] https://github.com

[2] https://api.stacspec.org/v1.0.0/

[3] https://github.com/stac-utils/pgstac

[4] https://github.com/stac-utils/pgstac/pkgs/container/pgstac

[5] https://stac-utils.github.io/pgstac/pypgstac/

[6] https://github.com/stac-utils/stac-fastapi

[7] https://github.com/radiantearth/stac-api-spec

[8] https://fastapi.tiangolo.com/

[9] https://github.com/stac-utils/stac-fastapi-pgstac

[10] Wang, Y., Braham, N.A.A., Xiong, Z., Liu, C., Albrecht, C.M., Zhu, X.X., 2023. SSL4EO-S12: A large-scale multimodal, multitemporal dataset for self-supervised learning in Earth observation [Software and Data Sets]. IEEE Geosci. Remote Sens. Mag. 11, 98–106. https://doi.org/10.1109/MGRS.2023.3281651

[11] Francis, A., Czerkawski, M., 2024. Major TOM: Expandable Datasets for Earth Observation.

[12] Schramm, M., Pebesma, E., Milenković, M., Foresta, L., Dries, J., Jacob, A., Wagner, W., Mohr, M., Neteler, M., Kadunc, M., Miksa, T., Kempeneers, P., Verbesselt, J., Gößwein, B., Navacchi, C., Lippens, S., Reiche, J., 2021. The openEO API–Harmonising the Use of Earth Observation Cloud Services Using Virtual Data Cube Functionalities. Remote Sensing 13, 1125. https://doi.org/10.3390/rs13061125

[13] https://github.com/Open-EO/openeo-python-client

[14] https://github.com/Open-EO/openeo-r-client

[15] https://openeo.org/

[16] https://github.com/IBM/tensorlakehouse-openeo-driver

[17] https://github.com/Open-EO/openeo-processes-dask

[18] https://openeo.org/documentation/1.0/authentication.html

[19] Mohr, M., Pebesma, E., Dries, J., Lippens, S., Janssen, B., Thiex, D., Milcinski, G., Schumacher, B., Briese, C., Claus, M., Jacob, A., Sacramento, P., Griffiths, P., 2025. Federated and reusable processing of Earth observation data. Sci Data 12, 194. https://doi.org/10.1038/s41597-025-04513-y

[20] Charette-Migneault, F., Avery, R., Pondi, B., Omojola, J., Vaccari, S., Membari, P., Peressutti, D., Yu, J., Sundwall, J., 2024. Machine Learning Model Specification for Cataloging Spatio-Temporal Models (Demo Paper), in: Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Searching and Mining Large Collections of Geospatial Data.

Presented at the SIGSPATIAL '24: The 32nd ACM International Conference on Advances in Geographic Information Systems, ACM, Atlanta GA USA, pp. 36–39. https://doi.org/10.1145/3681769.3698586