AtlasHDF: An Efficient Big Data Framework for GeoAI

Martin Werner
Professorship of Big Geospatial Data Management,
Technical University of Munich
Munich, Germany
martin.werner@tum.de

Hao Li
Professorship of Big Geospatial Data Management,
Technical University of Munich
Munich, Germany
hao_bgd.li@tum.de

ABSTRACT

The last decade witnesses a fast development in geospatial application of artificial intelligence (GeoAI). However, due to the misalignment with wider computer science progresses, the geospatial community, for a long time, keeps working with powerful and over-sophisticated tools and software, whose functionality goes far beyond the actual basic need of GeoAI tasks. This fact, to a certain extent, hinders our steps towards establishing future sustainable and replicable GeoAI models. In this paper, we aim to address this challenge by introducing an efficient big data framework based on the modern HDF5 technology, called AtlasHDF, in which we designed lossless data mappings (immediate mapping and analysis-ready mapping) from OpenStreetMap (OSM) vector data into a single HDF5 data container to facilitate fast and flexible GeoAI applications learnt from OSM data. Since the HDF5 is included as a default dependency in most GeoAI and high performance computing (HPC) environments, the proposed AtlasHDF provides a cross-platformm and single-techonology solution of handling heterogeneous big geodata for GeoAI. As a case study, we conducted a comparative analysis of the AtlasHDF framework with three commonly-used data formats (i.e., PBF, Shapefile and GeoPackage) using the latest OSM data from the city of Berlin (Germany), then elaborated on the advantages of each data format w.r.t file size, querying, rending, dependency, data extendability. Given a wide range of GeoAI tasks that can potentially benefit from our framework, our future work will focus on extending the framework to heterogeneous big geodata (vector and raster) to support seamless and fast data integration without any geospatial software dependency until the training stage of GeoAI. A reference implementation of the framework developed in this paper is provided to the public at: https://github.com/tumbgd/hdf4water.

CCS CONCEPTS

• **Information systems** → **Geographic information systems**; • **Computing methdologies** → **Artificial intelligence**; • **Theory of computation** → **Data compression**.

BigSpatial'22, November 01, 2022, Seattle, WA, USA

ACM ISBN 978-1-4503-9531-1/22/11…$15.00
https://doi.org/10.1145/3557917.3567615

KEYWORDS

GeoAI, Hierarchical Data Format, Immediate mapping, Big Data, OpenStreetMap

ACM Reference Format:
Martin Werner and Hao Li. 2022. AtlasHDF: An Efficient Big Data Framework for GeoAI. In *The 10th ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data (BigSpatial '22) (BigSpatial '22), November 1, 2022, Seattle, WA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3557917.3567615

1 INTRODUCTION

Modern methods of data analysis including data mining, machine learning, and artificial intelligence have made significant progress over the last decades. While there has been a significant amount of time where the progress was most visible in the computer vision and natural language processing domain (e.g., convolutional neural networks), geospatial application of artificial intelligence (GeoAI) has seen a rising interest in the last years [5] though the intersection of AI and geographical information systems (GIS) has a rather long history back to 1990s [7, 10].

Due to the special needs and as well due to some misalignment with the wider computer science progress, the geospatial community, however, keeps working with quite complex and powerful tools whose functionality goes way beyond the need of basic GeoAI projects.

For instance, the Copernicus program offers a globally strategic remote sensing (RS) data acquisition scheme, where the RS imagery was provided in a fixed grid system [2] and as a ZIP file in which many individual GeoTIFF files provide meaningful geospatial information. However, the grid is kept constant and new products are generated even in cases where the satellite just "touches" the spatial footprint of such a pre-defined geospatial object. Furthermore, the collection of data in a ZIP file can fail in supporting random access such that applications that rely only on a subset of the provided information can easily query the exact imagery patch that is needed.

Furthermore, for good reasons, the optical products from the Sentinel-2 mission are generated, processed and provided in varying map projections in order to keep distortions manageable. Though this is effective for the data generation, from a cartography perspective, this practice results in a common limitation, since in any large-scale mapping project, one needs to reproject almost all data files into another given map projection. Although such data formats makes sense from a precise geographic and geodetic perspective, it puts a difficult-to-manage burden for swift and fine-scale GeoAI applications, such as object detection or land-cover-land-use (LULC) classification, where all the advantages of aforementioned sensing techniques are hindered to a great extent [11].

But the consequence of the current geospatial big data landscape is that all GeoAI applications either transform data into the preferred representation of the computer vision domain (e.g., folders of PNG files) with a significant loss in terms of metadata and quality, or they use the preferred geospatial domain representation like GeoTIFF for holding sub-patches of RS imagery ready to be fed to deep neural networks for training [12]. While the latter keeps metadata information intact and does not immediately imply data loss, it puts a strong burden on the researchers and data scientists in a situation that they have to deal with huge and powerful environments such as GDAL and all dependencies it relies on. This means intensive and time-consuming manual work in practice, as most pre-configured and maintained deep learning (DL) environments (e.g., NVIDIA's DL container registry, TensorFlow or PyTorch project docker container) are missing these libraries and their dependencies.

To take a step back and rethink from the opposite perspective to this issue, all of these pre-configured DL environments have many things in common, one of which is often overlooked in terms of its potentials: all reasonable AI environments bring Hierarchical Data Format 5 (HDF5) [4] support as a default dependency.

It is worthy to note for those that are not familiar with HDF5 that it is a widely-adopted technology in industry, such that almost all supercomputing systems have decent support for it. For example, the basic implementation of HDF5 includes parallel IO support for supercomputers through the MPI-IO interface, something that would have to be implemented with considerable efforts for other environments. This is especially important when global or time-series oriented GeoAI tasked are considered, where the data footprint can reach multiple petabytes easily.

Moreover, all the developments of this paper are possible with any decent group-based data infrastructure, for instance the ZARR project (https://github.com/zarr-developers/zarr-python) which is more like a clone of HDF5 written in modern languages. People should have a close look at both, because they provide virtually the same set of functions whereas HDF5 has a wider adoption in industry and is pre-installed in all relevant DL environments whereas ZARR comes as a "modern" implementation of the same functionality and might, therefore, can be easier to modify and to fully understand.

In this context, this paper introduces an efficient big data framework, namely AtlasHDF, in which we define lossless data mappings from OpenStreetMap (OSM) into HDF5 files to facilitate fast and flexible GeoAI models learnt from OSM data. The concept is called AtlasHDF as an HDF file now resembles the concept of an atlas: a book in which each page can show a different map in a different projection, but still, the user can access all the information without caring about the projections with the user's eyes.

With this setup, we demonstrate that there is a quite simple possibility to close the domain gap between spatial computing and GIScience relying a lot on historic data formats and bundling of data and metadata with modern machine learning in which most problems are processed in terms of only technical specifications such as pixel sizes or tile sizes. Further, we note that the existence and accessibility of geospatial metadata inside the HDF containers can allow to easily include some spatial methodology into deep learning-centric software, which is one of the prerequisites of future sustainable and replicable GeoAI. To demonstrate the advantage of our framework, we present the design of AtlasHDF framework with a case study of representing OSM data in an analysis-ready format for different GeoAI applications, where we report preliminary results and findings in the city of Berlin, Germany.

A reference implementation of the developments of this paper is provided to the public at https://github.com/tumbgd/hdf4water which is supported from the National Research Data Infrastructure for the Geospatial Sciences (NFDI4Earth) (see https://www.nfdi4earth.de/).

2 THE ATLASHDF REPRESENTATION OF OPENSTREETMAP

OSM is a high-value and large database of volunteered geographic information following a very special way of data modeling optimized for very low barriers for contributors. In a nutshell, the data model defines three types: nodes, ways, and relations. Each of these have a unique integer identifier, hereafter called OSM Ids, and can have an arbitrary number of tags associated following a key-value pattern. In addition to these attributes, nodes contain information on coordinates given as latitude and longitude in EPSG:4326, ways contain ordered sequences of nodes typically modeling linestrings and parts thereof, and relations can contain ordered sequences of multiple ways to model polygons consisting of one outer ring and zero or more inner rings (e.g., holes in the polygons). Note that the meaning of nodes, ways, and relations can only be understood from the set of attributes such that relations are not always modeling polygons, they are also used to bind together elements of other semantic relation.

The semantics have been defined first by the main purpose of OSM: to generate a usable world map. Best practice in terms of tag usage combined with the default style file for the cartographic rendering have established a best practice of how entities are represented in order to get the right appearance on the map. For example, buildings are polygons tagged with something with a key "building". Lots of buildings contain building="yes", but in some areas of the world, you will also find building="residential" as a more detailed information. The expectation should, however, be that data quality of OSM is mainly defined by the cartographic use [1].

The following subsections explain our approach to transforming native OSM data into data that can be used without dependencies to special software from the OSM community or the GIS community. The advantage of HDF5 is that it is already a default dependency in all major deep learning frameworks like TensorFlow or PyTorch which themselves rely on HDF5 to store weights and model information.

2.1 Immediate Mapping of OpenStreetMap Data

As a first step, we read an OSM file in the protocol buffer format which provides us with an ordered stream of OSM objects of the classes node, way, or relation. Whenever objects are modeled by referring to other objects, this reference is made using the globally unique OSM Id. In traditional OSM software (look at, for example, the osmosis library), this leads to a two-phase parsing: in a first phase, objects of interest are found (e.g., all buildings) and in a

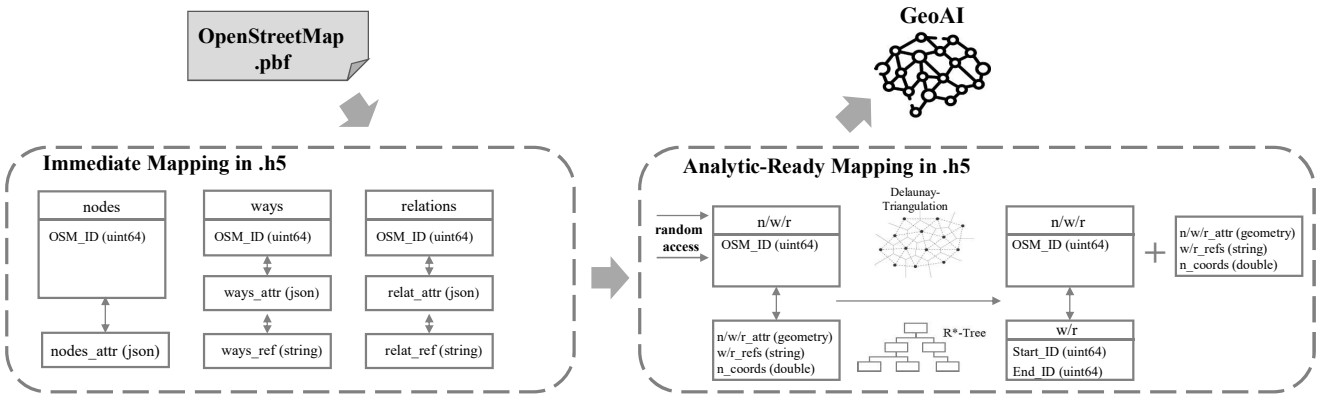

Figure 1: The overview of AtlasHDF4Water, which consists of two main modules: (1) Immediate mapping of OSM data representation; (2) Analysis-ready Mapping for GeoAI

second phase, all references are resolved. In order to avoid this two-phase parsing, our immediate mapping provides random access to tables of ways, nodes, and relations by their OSM IDs.

This is accomplished rather straightforward following the paradigm of a column store: we create a data table for all OSM IDs. This is small enough to be held in main memory even for the planet, hence, this is expected to be indexed in main memory. For each row of this column, the same row of another column table holds coordinate information and a third column holds attributes. That is, we have a primary key column holding OSM IDs expected to be held in main memory for quick access and we have multiple tables of the same number of rows mapping by the row index to the OSM Id. That is, the k-th row of the ID column, the k-th row of the coordinates table, and the k-th row of the attributes table make up each OSM object.

Spelling this out for N nodes, M ways and O relations, the three classes provides the schema for the immediate mapping given in Table 1.

This mapping of OSM is called immediate mapping as we just replace the data containers and representations with HDF5 and JSON for usage of built-in facilities.

2.2 Analysis-Ready Mapping of OpenStreetMap Data

The immediate mapping is a generic representation of OSM data in which the use of standard data representations like JSON allows intuitive access to the raw data. But the pointer-intensive modeling of OpenStreetMap data in which a polygon resolves to multiple linestrings which themselves resolve to multiple points leads to challenging demand for random access. In fact, this heavy use of references in data modeling is excellent for enforcing consistency (moving a point in this representation moves it for all entities derived from this point), but as soon as the data is considered immutable, one can do better. Our mapping shall facilitate the following key features:

- Data read from the HDF container can be immediately used by libraries like OpenGL or TensorFlow
- Data should be understandable by users neither experienced with OGC Simple Features nor with the quite different geometry notions of OpenStreetMap
- Using the data shall be efficient

The following three chapters describe the decisions we took for representing the three main geometric entities points, linestrings and polygons in AtlasHDF.

2.2.1 Point Representation. The point representation of the immediate mapping already fulfills all our requirements. We can materialize a continuous memory area holding just coordinate values. This is compatible with important APIs such as OpenGL for visualization or Proj for projecting points. Furthermore, numpy, TensorFlow, and other python extensions can make use of such data without copying using the Python Buffer Protocol which exchanges only memory location information between different libraries while interoperating instead of communicating data. Finally, when compression is disabled, the HDF5 libraries can provide the data mapped into virtual memory while residing on disk such that only the kernel of the operating system mediates between the program working on the data and the devices the data is stored on. With chunking and compression enabled, however, the disk utilization can be optimized fully transparent to the application.

2.2.2 Linestrings. In order to achieve the same performance and efficiency for the linestrings, we define the analytics-ready representation with two tables. One table holds the coordinates avoiding duplicate values based on a threshold distance (e.g., machine precision) and a second table consists of indices. More concretely, all linestring coordinates are stored in a dataset linestrings_coords with two columns. The analysis-ready linestring information is given as a second table holding all row numbers of linestrings. For compatibility with OpenGL, we use the highest index value (e.g 0xFFFFFFFF for unsigned integer of 32 bits) as a marker to restart a primitive. This mapping is called the *in-memory mapping* of linestrings as its efficiency depends on holding all coordinates in main memory (or other memory suitable for efficient random access).

Dataset	Shape	Data Type	Map on	Semantics
/nodes	(N,1)	uint_64	-	A table of OSM IDs for each node
/nodes_attr	(N,1)	string	/nodes	Attributes as JSON objects
/nodes_coords	(N,2)	double	/nodes	Coordinates
/ways	(M,1)	uint_64	-	A table of OSM IDs for each way
/ways_attr	(M,1)	string	/ways	Attributes as JSON objects
/ways_refs	(M,1)	string	/ways	Referenced nodes as a CSV string
/relations	(O,1)	uint_64	-	A table of OSM IDs for each way
/relations_attr	(O,1)	string	/relations	Attributes as JSON objects
/relations_refs	(O,1)	string	/relations	A JSON object referring to OSM objects mentioning their role like inner or outer ring of polygon

Table 1: The Schema of the OSM Immediate Mapping

As an alternative, especially for big data situations where not all corodinates can be held in main memory, we omit the index table entirely and store coordinates in the order of their occurence in linestirngs.

Then, linestrings are modeled indirectly by storing the start and end index of each linestring. More concretely, the linestring information is given as a second table of shape (M,2) holding the start and end index of each line string. In this way, materializing a single (or multiple consecutive) linestring is a serial read operation without reading unused data.

This model is called the *consecutive mapping* and is preferred for parallel applications due to avoiding the need for materializing all coordinates or random access over the table of coordinates. The redundancy of repeated coordinates does not play a big role in this case.

2.2.3 Polygon Representation. An analysis-ready mapping of polygons to HDF containers is very challenging. First of all, polygons according to the OGC Simple Feature Specification contain one outer ring and zero or more inner rings with a handful of constraints disallowing problematic cases. For example, inner rings have to lie within the outer ring. OGC defines multiplygons just as multiple polygons.

For the polygons, we face a more challenging problem: a polygon contains one outer ring of a variable number of points and zero or more inner rings of a variable number of points. Mapping this to an efficient memory representation is not possible as the multiple levels of variability in length leads to a per-item processing which is neither compatible with pipelines nor directly supported by HDF5. In this context, we propose to replace all polygons (e.g., general polygons with/without holes) with just simple polygons (e.g., polygons without holes). This is always possible and efficient algorithms for such a task are known. Again, this is not ideal for modeling, but simplifies data analytics as now each polygon is modeled as a linestring internally. That is, the same representation as the one designed for linestrings can now be used for polygons.

As a final step, we propose to add a triangulated data structure optionally. You might have noticed that the previous construction turns a two-times variable-length data structure into a one-time variable length data structure. If the significant increase in the number of objects is tolerable, we propose to triangulate all polygons (but maybe not mapping attributes to the resulting many triangles) and to store these triangles in a six-column dataspace. In this representation, hardware acceleration of rendering becomes very easy as OpenGL can use these buffers directly.

As a summary, we can say that the AtlasHDF immediate mapping is augmented with additional columns for fast and memory-contiguous access to linestrings. By replacing polygons with multiple simple polygons, the linestring technique can be extended to polygons finalizing the representation.

2.3 Querying and Visualizing OpenStreetMap Data in AtlasHDF

Querying OSM can now be implemented in any technology that is suitable including, but not limited to state-of-the-art in-memory R*-tree implementations, out-of-memory R-trees embedded into the HDF5 file, JSON queries based on the JQ query language, or JSON queries based on a small python function which given a JSON object as a string returns True or False depending on whether or not the element is relevant.

Due to the column store nature, three type of queries can be distinguished: (1) queries involving only spatial information (e.g., using the coords fields), (2) queries involving only attribute information (e.g., relying only on the attribute tables), and (3) mixed queries. For all queries, due to the strict column store model, the query result will be a set of integers describing the row numbers.

If this query result is sorted by row number, it is very easy to materialize search results as a consequence of HDF5 providing reading for any set of rows that are monotonously ordered. Hence, in python, a simple slicing on the file data spaces can be used. It is worth noting that query results can be stored easily in the HDF5 file itself as a table of integers.

For the purpose of this paper, it is sufficient to understand that a single scan over the data can be used to implement all types of queries by giving the rows one after another (or in parallel) to a predicate function. We give more details on query implementation and optimization in Section 2.4.

For visualization, the coordinate tables can be immediately materialized to main memory and transmitted to GPU memory such that OpenGL can use them directly in functions like glDrawPrimitives. After materializing the coordinate arrays, advanced OpenGL techniques such as "Primitive Restart" can be used and allow for further compression of the geometry data, which is then beyond

the scope of this single paper. Herein, we focus on the potential computational cost of querying and visualizing OSM data in AtlasHDF, and a detailed analysis of the cost of converting OSM data into AtlasHDF format will be included in future works.

2.4 An AtlasHDF Query for GeoAI Target Objects

As a demonstration, we show how one can implement - based on the aforementioned AtlasHDF data representation, a subsetting operation which queries OSM features relevant for GeoAI application, for instance automatic surface water mapping with OSM and Sentinel-2 satellite data [6]. Herein, a foremost important while computational-expensive step is to generate training labels for GeoAI by combining surface water vector data from OSM and multispectral satellite raster imagery from Sentinel-2. More specifically, we will have many patches of satellite data, which is straightforward as HDF5 provide efficient data presentation of all kinds of satellite imagery as raster-based datasets, from a range of different areas of interest. In this context, the actual challenge is how to conduct an efficient and customized (e.g., surface water) query over the OSM vector data representation without including a long list of package dependencies (e.g., GDAL, osmosis, Overpass). In AtlasHDF, we decide to first compute the query towards the Surface water attributes and materialize this query then as an ordered set of indices. In a second step, for each of these indices, a minimum bounding box is computed and stored as a map to the query result. An in-memory R-tree will give us quick access to spatial subsets of the interesting features (e.g., lakes, rivers, ponds) in terms of the indices in the original dataset of relevant primitives. In this way, queries can be very efficiently computed on-the-fly.

For such an attribute query, we provide a very simple implementation showing the gain of the given approach. Without any requirements beyond HDF5 (which is implicitly available when using TensorFlow or PyTorch for GeoAI applications), we implement a kernel as follows:

```
# Define the query
def is_water(x):
  obj = json.loads(x)
  # JQ query on key/value
  [...]
  return ("water" in obj)

vpredicate = np.vectorize(is_street)
```

Next, we map it over the HDF5 attributes as follows:

```
# Create query indices
query_indices = vpredicate(attributes)
                .nonzero()[0]
```

Finally, the result can be realized in main memory immediately using the slicing features of HDF5:

```
# Materialize only relevant
lsi = f["osm"]["linestrings_idx"]
      [query_indices,:]
# Store indices into water
f["osm"].create_dataset("water_idx",lsi)
```

Now, the new dataset "water_idx" contains indices into the shared point array, which can be later combined with satellite image patches to generate GeoAI training labels. In principle, it is also possible to compactify the dataset now by creating a new pair of point data with index data, but for our application, it is not worth it due to the excellent performance of HDF5 managing partial datasets.

3 CASE STUDY

3.1 Preliminary Result

OSM data is licensed under the Open Database License (ODbL), thus is freely available in vector format comprising mainly point features as nodes, polyline features as ways and relations, and polygon features as ways and relations [8]. There are various existing data formats applied in OSM community, among which the following three types are considered to be commonly used:

Protocolbuffer Binary Format (PBF) - As a preliminary format, OSM data is stored in a low-level encoded PBF file (.osm.pbf), which was designed as an alternative to the XML format to support extensibility and flexibility. The advantages of distributing OSM data in PBF is also intuitive, as it needs only half of the size and 5 times faster to write comparing to a gzipped planet. Though random access is available at the "file-block" level, it can be difficult to work directly with OSM data in PBF for GeoAI applications. Mostly, one relies on open-source software packages for parsing or converting OSM data in PBF format into another vector format or a spatial database for further analysis, such as Osmium, osm-read, Osm4j, etc. The latest OSM .pdf file can be assessed at https://planet.osm.org/pbf/.

ESRI Shapefile (shp) - Unlike the PBF, the shp format is a dedicated geospatial vector data format designed by ESRI for GIS software. OSM data is storied as primitive geometric shapes (e.g., points, lines, and polygons) together with their associated attributes. In this context, OSM data stored in shp format can be much easier to work with in combination with other geospatial data (e.g., satellite imagery) for diverse GeoAI applications. However, the term "shapefile" by default refers to a collection of files (i.e., .shp, .shx, and .dbf) to make it complete for storage and distribution. In addition, it is important to notice that storing topological information is not supported in the shp format, therefore one may end up with difficult cases such as inner polygons and mixed shape types. Another limitation of OSM data in shp lies in the fact that each component files (i.e., .shp, .shx, and .dbf) could not be bigger than 2GB, which refers to roughly 70 million points.

GeoPackage (gpkg) - The gpkg is based on an extended SQLite 3 database file specifically for geospatial data, thus supports both raster and vector data at the same time. In fact, the proposed AtlasHDF data representation shares a similar idea of open-source and platform-independent data container (raw data and metadata) with the gpkg format. Moreover, OSM vector data storied in the gpkg format can be packaged into a single file without size limits. Though the gpkg can be extended by creating spatial indexes to speed up intensive spatial queries compared to traditional OSM data format (PBF and shp), it still lacks the capacity of fast random accessing and effective handling of geometric issues (e.g., inner

Table 2: Comparative analysis of selected data representations of OSM data in Berlin.

Representation	Structure	Size	Files	Querying	Rendering	QueryDependency	Extendable	Raster-Support	Ready-GeoAI
PBF	Encoded	67.7 (MB)	1	✗	✗	-	✗	✗	✗
Shapefile	Geometry	361.5 (MB)	91	Slow	Slow	GDAL, PostgreSQL, HDF5, etc	✓	✗	✗
Geopackage	Container	287.3 (MB)	1	Medium	Slow	GDAL, PostgreSQL, HDF5, etc	✓	✓	✗
AtlasHDF	Container	637.5 (MB)	1	Fast	Fast	HDF5	✓	✓	✓

polygons), which poses new challenges coming to advanced GeoAI applications.

To demonstrate the advantage of OSM data representation in AtlasHDF, we conducted a comparative analysis in the city of Berlin, Germany. More specifically, we downloaded the latest .osm.pbf file for Berlin from the Geofabrik download server, then mapped the OSM data into AtlasHDF format (Table 1), which resulted in more than 7 million points and 0.9 million linestrings in Berlin. Figure 2 shows an OpenGL-ready GUI (DSLAB) in materializing OSM geometries with immediate mapping into main memory, then rendering them via OpenGL drawing functions with GPU memory. Herein, the rendering for around 8 million geometries is about 200 fps with a common gaming GPU. Moreover, the powerful aspect is that AtlasHDF can load the whole OSM data for Berlin into visualizable forms within 2 seconds from the hard drive.

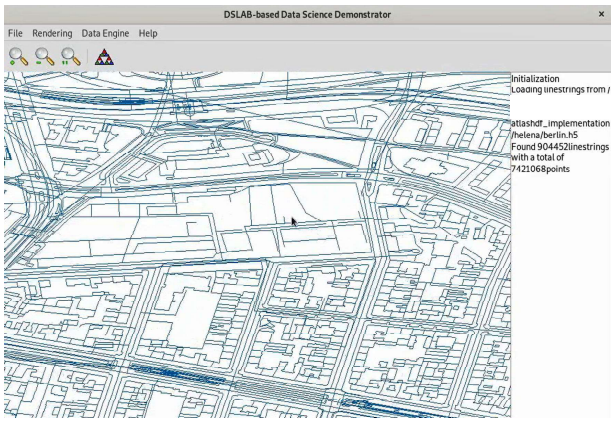

Figure 2: The interface of rendering OSM data stored in AtlasHDF with an OpenGL-ready GUI (DSLAB).

Besides the aforementioned features of different date formats used in the OSM community, we parsed the latest OSM data of Berlin into three common formats (i.e., PBF, Shapefile, and GeoPackage), and compared their storage sizes as well as file numbers. In this context, Table 2 elaborates on the main features of three common formats and our proposed AtlasHDF by focusing on their capability of handling intensive data queries and integrating with raster-based satellite data for GeoAI applications. One can notice the following key findings:

- Both GeoPackage and AtlasHDF store OSM in data containers, thus leads to a single file, which can be a big advantage to more than 90 files in Shapefile. Though PBF has also just a single file, OSM data is structured in low-level encoded data, which hinders direct querying and rendering over the data.
- Given the fact that Shapefile skips the topological relations between geometries, intensive queries can be relatively slower than other container-based format. However, the querying with GeoPackage is still suboptimal as its missing feature of data random access. This is also the reason that AtlasHDF can support fast rendering than other data formats.
- Regarding data extendability, Shapefile can only support vector data extension, while GeoPackage and AtlasHDF can support both raster and vector data.
- Last but not the least, the proposed AtlasHDF format has the most concise dependency of only HDF5, which is even a built-in dependency of GDAL and more importantly in modern DL libraries (e.g., TensorFlow and PyTorch), which are mostly used in building GeoAI models. This unique feature of the AtlasHDF data representation make it ready for a wide range of GeoAI applications.

3.2 Towards Facilitating GeoAI Applications

Due to the development of big data and crowdsourcing technology, OSM has recently become a promising source of massive, free training labels together with rich and detailed semantic information for the emerging GeoAI applications, ranging from automatic surface water mapping [6], Open LULC classification [9] to 3D city modelling (e.g., via estimating OSM building height) [3], as shown in Figure 3. According to the specific task, it is possible and intuitive to extend the existing AtlasHDF data framework and map required geospatial data, such as satellite imagery, environmental data, and point clouds data, into a similar analysis-ready format within a single AtlasHDF data container.

In this context, the excellent features of querying and rending (as shown in Table 2) of AtlasHDF would be even more helpful when scaling up the GeoAI model globally. Therefore, the lessons learned in this paper could encourage future work in this direction of closing the gap between intensive training data preparation and large-scale GeoAI model training, especially in an HPC environment.

Though stimulating GeoAI applications were explored, one of the major limitation of harvesting OSM data as GeoAI training data is still the lack of an efficient and dependency-optimized data representation, which can be easily deployed across different platforms and programming languages. Aiming at release this limitation, the proposed AtlasHDF framework provides a cross-platform single-techonology solution for GeoAI using heterogeneous big geodata (e.g., vector and raster) with a minimum library dependency on only HDF5, which is usually a default dependency in reasonable AI and HPC environments.

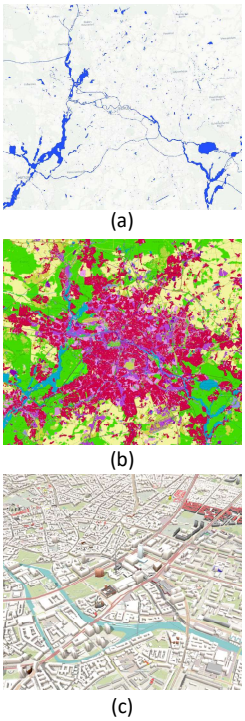

Figure 3: GeoAI applications that can benefit from the AtlasHDF data representation. (a) Automatic surface water mapping; (b) Open LULC classification; (c) 3D city modelling by estimating OSM building height (i.e., https://osmbuildings.org/).

4 CONCLUSION

In this paper, we proposed an efficient and single-technonolgy data framework for storing, processing, and querying big geospatial data, namely AtlasHDF, in which we define lossless data mapping from OpenStreetMap vector data into HDF5 files to empower fast and flexible GeoAI applications. The framework takes advantage of the modern HDF5 technology and the concept of data container to first conducted an immediate mapping of the OSM data into main memory and support random access in feature-level; then we designed an analysis-ready mapping for three main geometric entities (points, linestrings, and polygons) of OSM data and make the data ready for GeoAI applications. Moreover, we elaborated on the advantage of our AtlasHDF framework regarding data querying and visualizing, as well as from an implementation perspective, regarding the existing dependences in modern DL libraries (e.g., TensorFlow and PyTorch).

As a case study, we selected the city of Berlin (Germany) and downloaded the latest OSM data from the Geofabrik server, we then compared the AtlasHDF implementation with three common data formats used by OSM community, specifically Protocolbuffer Binary Format (PBF), ESRI Shapefile (shp), and GeoPackage (gpkg), by evaluating their corresponding features w.r.t size, file number, querying, rending, dependency, and data extendability. Based on our findings in the comparison, we discussed the potential applications of GeoAI model learnt from OSM data, which can benefit from our AtlasHDF framework. In future, more comparative analysis will be conducted to further examine the benefits of AtlasHDF w.r.t storage, scalability, efficiency, especially when extending the proposed AtlasHDF to a global OSM dataset.

This paper sheds a light into closing the domain gap of spatial computing and GIScience communities relying on sub-optimal data formats and over-complex software dependencies with a cross-platform and single-technology solution based on HDF5. In future research endeavours, we plan to extend the AtlasHDF framework by including more types of heterogeneous big geodata (both vector and raster) so that different GeoAI models can be trained without any geospatial software dependency until the training stage of the GeoAI. This will allow for fast, immediate adoption of new updates from the AI community and increases the ability to re-use infrastructures (e.g., implementations from papers that come from a non-spatial community) without the cumbersome need of "spatializing" them by installing all missing dependencies and even compiling some geoprocessing scripts.

ACKNOWLEDGMENTS

We acknowledge the support from NFDI Consortium Earth System Sciences (NFDI4Earth), which is funded by Deutsche Forschungsgemeinschaft (DFG), project number: 460036893.

REFERENCES

[1] Christopher Barron, Pascal Neis, and Alexander Zipf. 2014. A comprehensive framework for intrinsic OpenStreetMap quality analysis. *Transactions in GIS* 18, 6 (2014), 877–895.

[2] Yves-Louis Desnos, Maurice Borgeaud, Mark Doherty, Michael Rast, and Volker Liebig. 2014. The European Space Agency's Earth Observation Program. *IEEE Geoscience and Remote Sensing Magazine* 2, 2 (2014), 37–46. https://doi.org/10.1109/MGRS.2014.2319270

[3] Hongchao Fan, Gefei Kong, and Chaoquan Zhang. 2021. An Interactive platform for low-cost 3D building modeling from VGI data using convolutional neural network. *Big Earth Data* 5, 1 (2021), 49–65.

[4] Mike Folk, Gerd Heber, Quincey Koziol, Elena Pourmal, and Dana Robinson. 2011. An Overview of the HDF5 Technology Suite and Its Applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases* (Uppsala, Sweden) *(AD '11)*. Association for Computing Machinery, New York, NY, USA, 36–47. https://doi.org/10.1145/1966895.1966900

[5] Krzysztof Janowicz, Song Gao, Grant McKenzie, Yingjie Hu, and Budhendra Bhaduri. 2020. GeoAI: spatially explicit artificial intelligence techniques for geographic knowledge discovery and beyond. , 625–636 pages.

[6] Hao Li, Johannes Zech, Christina Ludwig, Sascha Fendrich, Aurelie Shapiro, Michael Schultz, and Alexander Zipf. 2021. Automatic mapping of national surface water with OpenStreetMap and Sentinel-2 MSI data using deep learning. *International Journal of Applied Earth Observation and Geoinformation* 104 (2021), 102571. https://doi.org/10.1016/j.jag.2021.102571

[7] Stan Openshaw and Christine Openshaw. 1997. *Artificial intelligence in geography*. John Wiley & Sons, Inc.

[8] OpenStreetMap Wiki. 2020. Elements — OpenStreetMap Wiki,. https://wiki.openstreetmap.org/w/index.php?title=Elements&oldid=2056268 [Online; accessed 24-July-2022.

[9] Michael Schultz, Janek Voss, Michael Auer, Sarah Carter, and Alexander Zipf. 2017. Open land cover from OpenStreetMap and remote sensing. *International Journal of Applied Earth Observation and Geoinformation* 63 (2017), 206–213. https://doi.org/10.1016/j.jag.2017.07.014

[10] Terence R Smith. 1984. Artificial intelligence and its applicability to geographical problem solving. *The Professional Geographer* 36, 2 (1984), 147–158.

[11] Martin Werner. 2021. GloBiMapsAI: An AI-Enhanced Probabilistic Data Structure for Global Raster Datasets. *ACM Transactions on Spatial Algorithms and Systems* 7, 4 (2021), 1–24.

[12] Martin Werner, Gabriel Dax, and Moritz Laass. 2021. Computational challenges for artificial intelligence and machine learning in environmental research. *INFORMATIK 2020* (2021).