

STACD: STAC Extension with DAGs for Geospatial Data and Algorithm Management

Saharsh Laud
IIT Delhi
India
mcs242002@cse.iitd.ac.in

Saurabh Joshi
IIT Delhi
India
mcs242003@cse.iitd.ac.in

Tarun Mangla
IIT Delhi
India
tmangla@cse.iitd.ac.in

Abhilash Jindal
IIT Delhi
India
ajindal@cse.iitd.ac.in

Aaditeshwar Seth
IIT Delhi
India
aseth@cse.iitd.ac.in

Abstract

Geospatial datasets have complex lineages that are crucial for reproducibility and understanding data provenance, yet current metadata standards like STAC (SpatioTemporal Asset Catalog) provide limited support for capturing complete processing workflows. We propose STACD (STAC extension with DAGs), an extension to STAC specifications that incorporates Directed Acyclic Graph (DAG) representations along with defining algorithms and version changes in the workflows. We also provide a reference implementation on Apache Airflow to demonstrate STACD capabilities such as selective recomputation when some datasets or algorithms in a DAG are updated, complete lineage construction for a dataset, and opportunities for improved collaboration and distributed processing that arise with this standard.

CCS Concepts: • **Information systems** → **Geographic information systems**; • **Theory of computation** → *Data provenance*; • **Software and its engineering** → *Software libraries and repositories*; • **Computer systems organization** → *Distributed architectures*.

Keywords: Geospatial Workflows, STAC, Data Provenance, Workflow Management, Reproducibility, Metadata Standards

ACM Reference Format:

Saharsh Laud, Saurabh Joshi, Tarun Mangla, Abhilash Jindal, and Aaditeshwar Seth. 2025. STACD: STAC Extension with DAGs for Geospatial Data and Algorithm Management. In *Proceedings of the 2nd ACM SIGPLAN International Workshop on Programming for the Planet (PROPL '25)*, October 12–18, 2025, Singapore, Singapore. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3759536.3763803>



This work is licensed under a Creative Commons Attribution 4.0 International License.

PROPL '25, Singapore, Singapore

© 2025 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2161-8/25/10

<https://doi.org/10.1145/3759536.3763803>

1 Introduction

The process of geospatial data analysis uses complex workflows that operate on multiple datasets through long pipelines of several algorithms and processing steps, all of which create new data products. These workflows are essential for environmental monitoring, climate analysis, urban planning, and disaster response. Understanding how these workflows create multi-level dependencies is extremely important for reproducible science and efficient data management.

Recent work on planetary computing has highlighted critical challenges in maintaining such workflows, particularly regarding data, algorithm, and library “uncertainty” that significantly impact their reproducibility and extensibility [8]. These uncertainties arise from the absence of adequate maintenance of the complex interdependencies that exist between datasets, processing algorithms, and computational environments that characterize modern geospatial data analysis pipelines.

Consider the geospatial workflow shown in Figure 1 which is an actual part of an open-source climate adaptation technology stack called the CoRE stack [2, 13]. We can see that multiple input sources flow through different processing algorithms to create intermediate datasets, which then undergo further processing to produce final outputs. This workflow illustrates the complex multi-level lineage that is a key characteristic of modern geospatial data processing. When changes occur, such as updating the underlying machine learning model or switching from SRTM DEM to the newer FABDEM product [11], the impact should propagate through the processing chain. A model update would require recomputation of the entire LULC processing branch, while a DEM change would only affect the terrain branch.

The current metadata standards, especially the SpatioTemporal Asset Catalog (STAC) specification [17], provide excellent support for describing individual geospatial assets. STAC has become the leading standard for geospatial metadata,

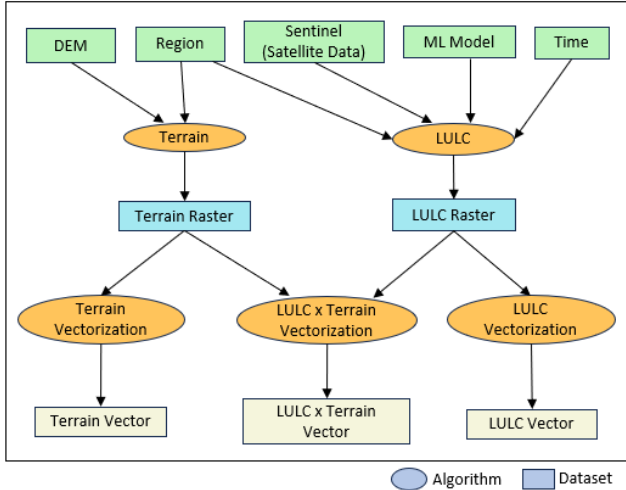


Figure 1. Multi-level lineage workflow example showing how different input sources (Digital Elevation Maps (DEM), Region of Interest, Satellite data from systems like Sentinel-1/2 and Landsat, Machine learning models, Time duration) flow through geospatial algorithms (Terrain classification [14], Land-use and Land-cover (LULC) classification [5]) to create intermediate datasets (Terrain raster @ 30m resolution, LULC raster @ 10m resolution), which then undergo vectorization processes to produce final vector outputs (% of area on flat mountain tops, % area under grasslands, % of flat mountain tops under grasslands, etc.). This shows the web of dependencies where changes at any level should propagate to downstream outputs and it should be possible to relate any output with the underlying algorithms, input parameters, and input datasets from which it was derived.

enabling data discovery and sharing. However, STAC’s approach to lineage tracking through simple parent-child relationships, and not documenting the algorithm that produced a dataset, cannot capture complex multi-level dependencies.

We propose STACD (STAC Extension with DAGs), an extension to STAC that incorporates Directed Acyclic Graph (DAG) representations for complete workflow lineage tracking. Our approach preserves STAC’s existing capabilities while adding the ability to represent complex multi-level dependencies and algorithm versioning. We first examine the motivation for DAG-based workflow representation in geospatial processing and analyze why current STAC lineage capabilities are insufficient. We then move ahead to explore the current STAC specification and present our proposed extensions which include some new class definitions and enhanced lineage representation. Through a reference implementation using Apache Airflow, we demonstrate how

STACD enables comprehensive workflow lineage maintenance. Finally, we discuss wider applications including distributed processing opportunities and collaboration benefits.

2 Why DAGs For Geospatial Data

2.1 Limitations of Current STAC Lineage Representation

The STAC specification is a key standard for geospatial metadata, helping researchers discover and share data. However, STAC has several important limitations when tracking complex workflow lineage like the one shown in Figure 1.

Shallow Lineage Tracking: The STAC standard incorporates a simple `derived_from` field which can only capture the immediate parent-child relationship, thus creating a limited view of data provenance. In our example workflow, the LULC vector dataset has a dependency on LULC raster, but the current STAC specification fails to capture that the raster was actually created by the LULC Algorithm with specific regional parameters, model configurations, and some temporal context. When researchers are required to trace dependencies back through multiple processing levels, it is not feasible to track the complete computational history due to this shallow lineage.

No Algorithm Representation: STAC has no class for describing processing algorithms, their versions, or parameter specifications. Therefore, when algorithms are updated or some new parameters added, there is no mechanism to track these changes and understand how this will effect the existing datasets. For instance, if the Terrain Algorithm is updated, STAC provides no mechanism to identify which downstream datasets require reprocessing.

2.2 DAG-Based Workflow Representation

Scientific workflows are subject to continuous evolution as algorithms improve and new data sources become available [9]. Dependency relationships are therefore an important part of scientific workflows, and DAG structures naturally show these kinds of relationships [7]. DAGs capture the entire dependency history of data products, which is lacking in simple lineage chains and allows for sophisticated reasoning about data relationships. This dependency information supports advanced queries such as “find all datasets that would be affected by updating algorithm X” or “identify the root cause of quality issues in dataset Y”, which are essential for maintaining data integrity.

Geospatial datasets are also updated regularly at varying frequencies: ingestion of new satellite imagery every couple of days, weather data hourly, census data decadal updates, etc. When new temporal data becomes available (for example, to extend change-detection analysis from 2017-2024 to incorporate data from 2025), DAG-based representations can

⁰STACD reference implementation and example workflows are available at: <https://github.com/SaharshLaud/STACD-Airflow>

help automatically identify which downstream datasets need reprocessing for the new temporal coverage while preserving existing results for unchanged time periods.

Literature on scientific workflow provenance has also consistently supported DAG-based approaches for complex dependency management [6, 7, 18]. We next show how STAC’s current lineage capabilities can be enhanced with DAGs.

3 STAC Specification and Required Extensions

3.1 Current STAC Architecture

The STAC specification provides a standardized framework for describing geospatial assets through three object types that work together to create a hierarchical structure.

Catalog: Provides a flexible hierarchical organization for easier search and discovery, such as to group together different types of datasets relevant for a particular use-case, E.g. planning watershed development activities requires LULC and terrain datasets.

Collection: Groups related Items with shared characteristics, such as putting together all Terrain datasets in one collection or all LULC datasets in one collection.

Item: Represents individual geospatial assets with spatial geometry, temporal information, asset references, and basic lineage through `derived_from` fields, such as a Terrain or LULC dataset for a particular region and time-period.

For the multi-level workflow shown in Figure 1, different STAC Items would be created for each geographic region and time period. There would be separate Collections for LULC raster products, terrain raster products, and vector assets, all organized within a hierarchical Catalog structure.

3.2 Proposed STACD Extensions

We first present the original STAC class definitions followed by our proposed extensions.

3.2.1 Original STAC Class Definitions. STAC defines three core classes with the following abstract structure:

STAC Item Class Definition

1: Class Item:	
2: <code>stac_version: String</code>	▸ STAC specification version
3: <code>stac_extensions: [String]</code>	▸ Extension identifiers ▸ Must be "Feature"
4: <code>type: String</code>	▸ Unique item identifier
5: <code>id: String</code>	▸ Spatial extent
6: <code>geometry: GeoJSON</code>	▸ Bounding box
7: <code>bbox: [Number]</code>	▸ Asset metadata
8: <code>properties: Object</code>	▸ Temporal information
9: <code>datetime: Timestamp</code>	▸ Asset title
10: <code>title: String</code>	▸ Asset description
11: <code>description: String</code>	▸ Links to data files
12: <code>assets: Object</code>	▸ Links to other STAC entities
13: <code>links: [Object]</code>	▸ Basic parent-child lineage
14: <code>derived_from: [String]</code>	

STAC Collection Class Definition

1: Class Collection:	
2: <code>stac_version: String</code>	▸ STAC specification version
3: <code>stac_extensions: [String]</code>	▸ Extension identifiers ▸ Must be "Collection"
4: <code>type: String</code>	▸ Unique collection identifier
5: <code>id: String</code>	▸ Collection title
6: <code>title: String</code>	▸ Collection description
7: <code>description: String</code>	▸ Data license
8: <code>license: String</code>	▸ Spatial and temporal bounds
9: <code>extent: Object</code>	▸ Spatial bounds
10: <code>spatial: Object</code>	▸ Temporal bounds
11: <code>temporal: Object</code>	▸ Common asset definitions
12: <code>item_assets: Object</code>	▸ Data provider information
13: <code>providers: [Object]</code>	▸ Links to other STAC entities
14: <code>links: [Object]</code>	

STAC Catalog Class Definition

1: Class Catalog:	
2: <code>stac_version: String</code>	▸ STAC specification version
3: <code>stac_extensions: [String]</code>	▸ Extension identifiers ▸ Must be "Catalog"
4: <code>type: String</code>	▸ Unique catalog identifier
5: <code>id: String</code>	▸ Catalog title
6: <code>title: String</code>	▸ Catalog description
7: <code>description: String</code>	▸ Links to other STAC entities
8: <code>links: [Object]</code>	

Note: Here, [String] denotes a list of strings, as used in the STAC specification.

3.2.2 STACD Extensions. STACD extends the existing STAC framework with five key classes: New classes for Algorithm Type and Dataset Type, a DAG class which links the Algorithm Type and Dataset Type classes, an Algorithm Implementation class which indicates the specific implementation of an Algorithm Type, and an extension to the STAC Item class for a particular Dataset Instance linked with the Algorithm Instance that produced it.

DAG Class:

STACD DAG Class Definition

1: Class DAG:	
2: <code>id: String</code>	▸ Unique DAG identifier
3: <code>name: String</code>	▸ DAG name
4: <code>version: String</code>	▸ DAG version
5: <code>description: String</code>	▸ Workflow description
6: <code>params: [Object]</code>	▸ DAG parameters
7: <code>alg_type_nodes: [Algorithm_Type]</code>	▸ Node list
8: <code>dataset_type_nodes: [Dataset_Type]</code>	▸ Node list

Dataset Type Class:

STACD Dataset Type Class Definition

1: Class Dataset_Type:	
2: <code>id: String</code>	▸ Dataset identifier
3: <code>name: String</code>	▸ Dataset name

Algorithm Type Class:

STACD Algorithm Type Class Definition

1: Class Algorithm_Type:	
2: <code>id: String</code>	▸ Algorithm identifier
3: <code>name: String</code>	▸ Algorithm name
4: <code>inputs:</code>	▸ Input specifications
5: <code>params: [Object]</code>	▸ Algorithm parameters
6: <code>input_datasets: [Dataset_Type]</code>	▸ Input datasets
7: <code>outputs: [Dataset_Type]</code>	▸ Output datasets

Algorithm Instance Class:

STACD Algorithm Instance Class Definition

1: Class <code>Algorithm_Instance</code> :	
2: <code>version</code> : String	▸ Algorithm version
3: <code>type</code> : <code>Algorithm_Type</code>	▸ Refers to algorithm type
4: <code>assets</code> : Object	▸ Links to source code

Extended STAC Item Class with Enhanced Lineage:

STACD Dataset Instance Class Definition

1: Class <code>Dataset_Instance</code> extends <code>Item</code> :	
2: <code>// Inherits all STAC Item properties</code>	
3: <code>version</code> : String	▸ Dataset version
4: <code>type</code> : <code>Dataset_Type</code>	▸ Refers to dataset type
5: <code>params</code> : [Object]	▸ Dataset parameters
6: <code>alg_name</code> : <code>Algorithm_Instance</code>	▸ Creating algorithm
7: <code>alg_inputs</code> :	▸ Creating algorithm inputs
8: <code>params</code> : [Object]	▸ Processing parameters used
9: <code>input_datasets</code> : [<code>Dataset_Instance</code>]	▸ Input dataset

These extensions enable STACD to capture the complete workflow context that the current STAC `derived_from` field cannot adequately represent. While `derived_from` in STAC could be used to list multiple parent entries, there are no specifications to define what details should be included here. STACD essentially standardises this by also recording the producing `Algorithm_Instance` and its version, all input `Dataset_Instance`s with their versions, and the processing parameters, turning a free-form field into a structured workflow description. A complete example is shown in Appendix A.

4 Reference Implementation In Apache Airflow

We carried out a reference implementation of the STACD concept on Apache Airflow [1] to validate the feasibility of our proposed extensions. Airflow is a workflow scheduler that defines task execution order as a DAG, maintains execution logs, supports re-triggering after errors, sub-DAG computation, and similar capabilities.

4.1 System Architecture

YAML-based Workflow Definition: We defined DAGs using YAML specifications identical to the proposed STACD extensions. While STAC uses JSON, we chose YAML for easier editing and maintenance of workflow specs.

Airflow Scheduler: We wrote a plugin on Airflow that parses the given YAML specifications and converts them into executable Airflow DAGs. Airflow picks up a DAG definition and begins task execution according to the dependency graph. The scheduler monitors task dependencies and ensures that upstream tasks complete successfully before downstream tasks are initiated.

Comprehensive Database Schema: We further use a SQLite database to maintain the equivalent of `Algorithm_Instance` and `Dataset_Instance` records that are used as inputs or produced as outputs when a DAG is executed. The database can thus be queried to build the lineage for any

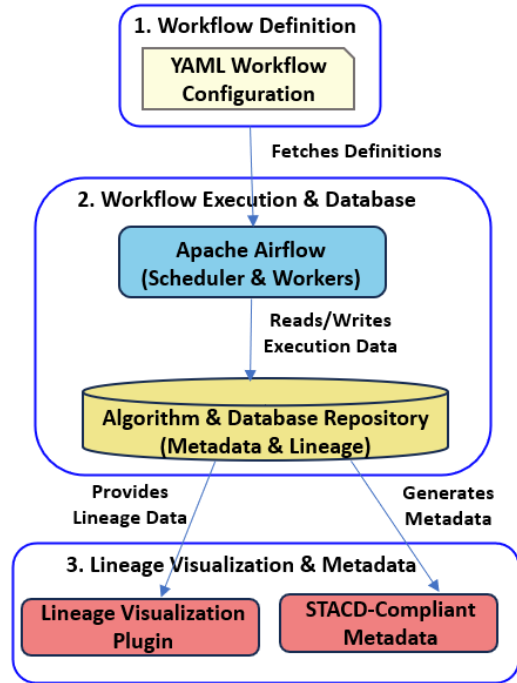


Figure 2. Overall system architecture showing the integration between YAML workflow definitions, DAG generation, Airflow execution engine, database storage, and lineage visualization components.

Dataset Instance. The database schema is shown in Figure 3 and includes three main tables:

- **Algorithm Instances:** Stores Algorithm Instance information including the name, version, creation date, and other metadata. This maintains a complete history of algorithm versions to allow tracking of which specific version was used to create a particular dataset.
- **Dataset Instances:** Stores information about datasets created during execution, including the algorithm that created it and the complete list of input datasets. This enables tracing the complete upstream lineage of any dataset.
- **Execution Logs:** Records task execution details including timestamps, status, and error messages. This provides the execution history required for debugging and reproducibility.

Automated Lineage Capture: During workflow execution, the database records can be used to automatically construct the lineage for any dataset instance. We built a visualization plugin on Airflow which for any target dataset conducts a recursive trace to discover the entire upstream lineage for the dataset. Figure 4 shows a screenshot of the lineage visualization, starting from root datasets and parameters (such as the region of interest, the machine learning

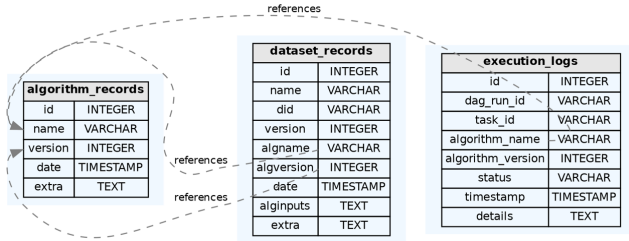


Figure 3. Database schema with tables for algorithm instances, dataset instances, and execution context, in line with the proposed STACD metadata proposal.

model, and year) which are used in various algorithms (terrain classification and LULC classification) to produce intermediate datasets, and which then undergo further processing to generate the final target dataset.

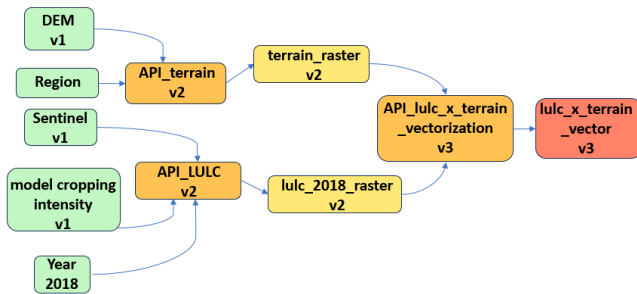


Figure 4. Lineage visualization showing complete upstream dependencies.

4.2 Workflow Execution Primitives

We further defined certain primitives which we anticipate will be used frequently.

Full Execution (full_exec): This executes the entire workflow from input datasets to final outputs, recording the complete execution context for all generated datasets.

full_exec

- 1: **Input:** DAG specification, execution parameters
- 2: **Output:** All datasets generated with complete lineage
- 3: Identify all nodes in topological order
- 4: For each node in execution order:
- 5: Execute algorithm with specified inputs
- 6: Record execution context and lineage information
- 7: Generate output datasets with STACD specs

Algorithm Update (update_alg): Does selective recomputation when a new algorithm version is defined, identifying downstream datasets for reprocessing.

update_alg

- 1: **Input:** Updated algorithm identifier
- 2: **Output:** Recomputed downstream datasets
- 3: Identify all nodes using the specified algorithm
- 4: Recursively identify downstream nodes
- 5: Execute all dependent nodes in topological order
- 6: Generate new versions of the datasets

Dataset Update (update_dataset): Similar to the above, this does a selective re-computation when a new dataset version is defined such as a new DEM version.

update_dataset

- 1: **Input:** Updated dataset identifier
- 2: **Output:** Recomputed downstream datasets
- 3: Identify nodes that depend on the updated dataset
- 4: Recursively identify downstream dependent nodes
- 5: Execute all affected nodes in topological order

Resume Execution (resume_exec): This is required for error recovery in case some intermediate algorithm failed and the computation needs to be re-triggered from that point onward after the error is fixed.

resume_exec

- 1: **Input:** Failed algorithm identifier
- 2: **Output:** Recomputed downstream datasets
- 3: Resume execution from failure point onwards

Update DAG (update_dag): This is required for new versions of the DAG when additional algorithms may be added and new datasets might get produced.

update_dag

- 1: **Input:** Updated DAG specification
- 2: **Output:** Recomputed or newly computed downstream datasets
- 3: Compare old and updated DAG structures
- 4: Identify newly added nodes and dependencies
- 5: Execute only new processing components

We are currently in the process of porting existing CoRE stack pipelines to be triggered via Airflow, using the STACD specification.

4.3 Execution Demonstration

To show how STACD works in practice, we demonstrate using the DAG in Figure 1 an execution sequence of full_exec followed by update_dataset and then update_algorithm. Figure 5 shows how STACD selectively recomputes parts of the workflow.

Complete Execution: When we run the complete workflow with full_exec, the Terrain Algorithm processes DEM data, the LULC Algorithm processes Sentinel data using the ML Model, and the vectorization steps process the outputs of these algorithms. Figure 5 shows that all the nodes get executed.

Updating Sentinel Data: If a previous run encountered missing Sentinel data and higher quality data is now available, invoking update_dataset triggers execution of algorithms downstream from the Sentinel dataset node. The terrain processing branch of the workflow is skipped in this case.

Updating Terrain Algorithm: Similarly, if an improved version of the Terrain Algorithm is available then an invocation of update_alg executes the Terrain algorithm and downstream nodes.

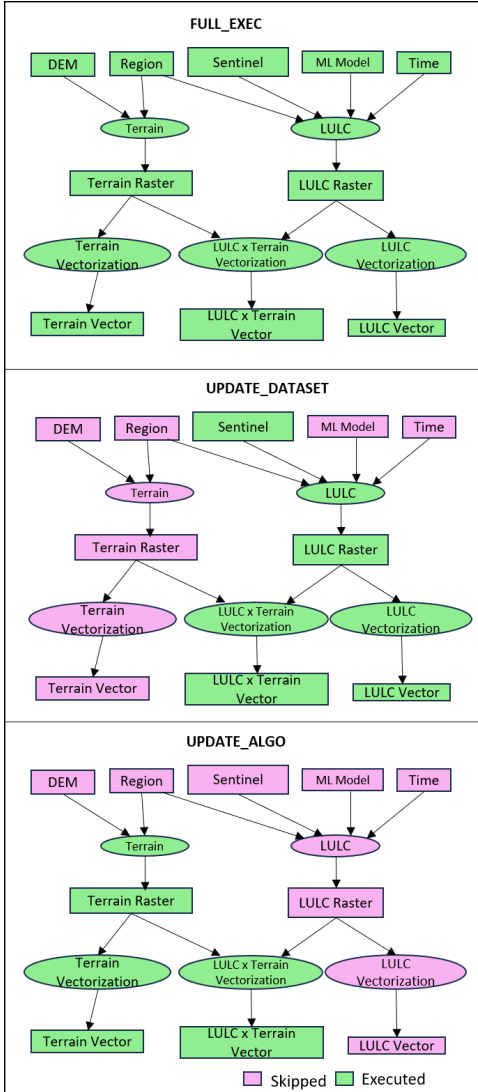


Figure 5. Demonstration of selective recomputation. Nodes in green indicate computations and new outputs.

This selective recomputation can save resources while keeping complete records of different dataset versions that were generated. The reference Apache Airflow implementation for STACD, with example workflows, is available at our Github repository [12].

5 Related Work

Scientific workflow provenance has been extensively studied across multiple domains. Davidson and Freire [7] provide a comprehensive overview of provenance challenges in scientific workflows, emphasizing the importance of DAG-based structures for capturing causality relationships. Lineage tracking in scientific data processing has been surveyed by Bose and Frew [6], who identify key requirements for

provenance systems including dependency tracking, selective recomputation, and workflow reproducibility. Simmhan et al. [18] survey data provenance approaches in e-science, highlighting the need for standardized metadata that can capture complete workflow context, manage versions, and enable selective recomputation. While their work identifies these requirements in general, our contribution is to put them into practice for the geospatial domain through STACD and provide a working reference implementation.

The need for large-scale environmental computing infrastructure has been highlighted by Ferris et al. [8], who advocate for “planetary computing” systems that can handle global-scale environmental data processing. Their work emphasizes the critical importance of traceability and reproducibility in environmental science workflows, particularly for policy-making applications. Our proposal to extend STAC with full lineage tracking is inspired by this work.

Current geospatial metadata standards, particularly STAC, have focused primarily on spatial and temporal discovery capabilities. While STAC provides excellent support for individual asset description, it lacks the comprehensive workflow representation capabilities required for scientific workflow systems. Our approach bridges this gap by incorporating proven DAG-based provenance techniques into the established STAC ecosystem, enabling workflow-aware geospatial data management.

6 Future Work

STACD’s DAG-based approach enables workflow reusability across different regions and time periods and supports lineage queries. We next discuss future extensions.

Non-DAG Workflows and Feedback Loops: Our current model handles only acyclic workflows. Some workflows may involve conditional cycles like iterative re-training of ML models or error corrections [4]. Dynamic dataflow systems like Naiad [15] and CIEL [16] provide techniques for handling such cyclic computations in distributed execution environments that could inspire our extensions. Future extensions can allow such workflows to be specified and keep a track of execution traces to ensure reproducibility.

Execution Environment Extensions: The algorithm description standard can be extended to include execution environment details and compute resource requirements, potentially contingent on the size of the datasets to be processed. This would enable more sophisticated workflow management where algorithms specify their computational needs (CPU cores, memory, GPU requirements) and execution environments (Docker containers, library versions, etc.). This could support further distributed computing scenarios where different algorithms might be optimally executed on different types of compute nodes based on their resource requirements and compute availability. Recording details of the execution may also be crucial since differences in library versions or random seeds (such as for clustering or machine learning

training workflows) can lead to different outputs. To address this, the Algorithm Instance class can be extended to record more details of the execution environment such as container IDs and fixed random seeds.

Conditional Propagation of Changes: The algorithm specification can also include conditional hooks to perform diff operations that compare a new dataset output with the previous version and determine if the changes are substantial enough to merit continued downstream execution of the workflow. These hook functions can be custom implementations for each output and reference implementations can be provided with a library of common utility comparison operations for raster and vector outputs. Incremental computing systems like Adapton [10] provide inspiration for efficient change propagation techniques that could inform our conditional update mechanisms.

Comprehensive Guidelines: To support a detailed specification of STACD workflows, we will develop a set of guidelines to define how expressive the DAGs should be, with reproducibility and reusability as the main criteria. These guidelines will explain through examples different kinds of execution environment specifications that should be provided, algorithm parameters including random seeds and sample selection, and considerations to keep in mind to define intermediate outputs.

Distributed Computation and Collaboration: We finally share some thoughts to enable networked setups where DAGs can be reused, datasets can be pooled, and computation resources can be shared. The STAC index [3] which already serves as a global directory listing many geospatial datasets can also serve as a registry for DAGs and algorithms. For readers interested in exploring STAC further, the complete specification and community resources are available at <https://stacspec.org/>. Dataset creators can deploy local Airflow instances to execute existing DAGs pulled from the STAC index. A shared compute registry can further list computational resources available to execute different types of algorithms, and individual Airflow workflow schedulers can query this registry and trigger computations at appropriate nodes. Entirely distributed versions of the registries can also be envisioned, or pull-based architectures of compute nodes that can pull tasks queued for execution can help with cooperative computation of datasets. Such a setup can enable a decentralized infrastructure for planetary computing, anchored especially at academic institutions, to work cooperatively to share, cross-utilize, and improve upon compute resources, datasets, and algorithms.

Acknowledgments

We sincerely thank Professor Anil Madhavapeddy from the University of Cambridge for early discussions that led us to think along these lines. We also acknowledge funding support for this project from Tower Research and the CSE Research Acceleration Fund of IIT Delhi.

A STACD Implementation Examples

This appendix provides implementation examples of STACD extensions for the sample workflow shown in Figure 1.

A.1 DAG Definition

The workflow can be specified as a STACD DAG:

Listing 1. STACD DAG Definition

```
- !DAG
  id: "terrain_and_lulc_workflow"
  name: "Terrain and LULC Processing"
  version: "1.0"
  description: "Workflow for processing terrain
    ↪ and LULC data with vectorization"
  params:
    - region
    - year
    - model
  alg_type_nodes:
    - LULC_Algorithm
    - Terrain_Algorithm
    - LULC_Vectorization
    - Terrain_Vectorization
    - LULC_x_Terrain_Vectorization
  dataset_type_nodes:
    - DEM
    - Sentinel
    - Model
    - LULC_Raster
    - Terrain_Raster
    - LULC_Vector
    - Terrain_Vector
    - LULC_x_Terrain_Vector
```

Listing 2. Dataset Type Definitions

```
- !Dataset_Type
  id: "DEM"
  name: DEM

- !Dataset_Type
  id: "COPERNICUS/S2_HARMONIZED"
  name: Sentinel

- !Dataset_Type
  id: "IndiaSAT_CI"
  name: Model

- !Dataset_Type
  id: "IndiaSAT_LULC_Raster"
  name: LULC_Raster

- !Dataset_Type
  id: "CoRE_Terrain_Raster"
  name: Terrain_Raster

- !Dataset_Type
```

```

id: "IndiaSAT_LULC_Vector"
name: LULC_Vector

- !Dataset_Type
id: "CoRE_Terrain_Vector"
name: Terrain_Vector

- !Dataset_Type
id: "CoRE_LULCxTerrain_Vector"
name: LULC_x_Terrain_Vector

```

Listing 3. Algorithm Type Definitions

```

- !Algorithm_Type
id: LULC_Algorithm
name: LULC_Algorithm
params:
  - region
  - model
  - year
input_datasets:
  - COPERNICUS/S2_HARMONIZED
  - IndiaSAT_CI
outputs:
  - IndiaSAT_LULC_Raster

- !Algorithm_Type
id: Terrain_Algorithm
name: Terrain_Algorithm
params:
  - region
input_datasets:
  - DEM
outputs:
  - CoRE_Terrain_Raster

- !Algorithm_Type_Node
id: LULC_Vectorization
name: LULC_Vectorization
params:
  - region
input_datasets:
  - IndiaSAT_LULC_Raster
outputs:
  - IndiaSAT_LULC_Vector

- !Algorithm_Type_Node
id: Terrain_Vectorization
name: Terrain_Vectorization
params:
  - region
input_datasets:
  - CoRE_Terrain_Raster

```

```

outputs:
  - CoRE_Terrain_Vector

- !Algorithm_Type_Node
id: LULC_x_Terrain_Vectorization
name: LULC x Terrain Vectorization
params:
  - region
input_datasets:
  - IndiaSAT_LULC_Raster
  - CoRE_Terrain_Raster
outputs:
  - CoRE_LULCxTerrain_Vector

```

A.2 DAG Execution with Specific Parameters

Input Dataset Instances: To execute the DAG with parameters of region = "Jharkhand_Dumka_Masalia", year = "2018", and model = "IndiaSAT_CI_PANINDIA", the following root dataset instances are needed:

Listing 4. Input Dataset Instances

```

- !Dataset_Instance
version: 1
type: DEM
params:
  - region: "Jharkhand_Dumka_Masalia"
  - source: "FABDEM_V1-2"
alg_name: null
alg_inputs:
  params: []
  input_datasets: []

- !Dataset_Instance
version: 1
type: COPERNICUS_S2_HARMONIZED
params:
  - region: "Jharkhand_Dumka_Masalia"
  - year: "2018"
alg_name: null
alg_inputs:
  params: []
  input_datasets: []

- !Dataset_Instance
version: 1
type: IndiaSAT_CI
params:
  - model: "IndiaSAT_CI_PANINDIA"
alg_name: null
alg_inputs:
  params: []
  input_datasets: []

```

Algorithm Instances: Similarly, the following algorithm instances are needed.

Listing 5. Algorithm Instance Examples

```
- !Algorithm_Instance
version: 1
type: LULC_Algorithm_Type
assets:
  code: "https://github.com/geo-algs/lulc-v1"

- !Algorithm_Instance
version: 1
type: Terrain_Algorithm_Type
assets:
  code: "https://github.com/geo-algs/terrain-v1"

- !Algorithm_Instance
version: 1
type: LULC_Vectorization_Type
assets:
  code: "https://github.com/geo-algs/lulc-vec-v1"

- !Algorithm_Instance
version: 1
type: Terrain_Vectorization_Type
assets:
  code:
    ↪ "https://github.com/geo-algs/terrain-vec-v1"

- !Algorithm_Instance
version: 1
type: LULC_x_Terrain_Vectorization_Type
assets:
  code:
    ↪ "https://github.com/geo-algs/combined-vec-v1"
```

Generated Dataset Instances: The DAG execution creates the following Dataset Instances.

Listing 6. Generated Dataset Instances

```
- !Dataset_Instance
version: 1
type: IndiaSAT_LULC_Raster
params:
  - region: "Jharkhand_Dumka_Masalia"
  - year: "2018"
alg_name: LULC_Algorithm_v1
alg_inputs:
  params:
    - region: "Jharkhand_Dumka_Masalia"
    - model: "IndiaSAT_CI_PANINDIA"
    - year: "2018"
  input_datasets:
    - [COPERNICUS_S2_HARMONIZED, 1]
    - [IndiaSAT_CI_PANINDIA, 1]

- !Dataset_Instance
version: 1
```

```
type: CoRE_Terrain_Raster
params:
  - region: "Jharkhand_Dumka_Masalia"
alg_name: Terrain_Algorithm_v1
alg_inputs:
  params:
    - region: "Jharkhand_Dumka_Masalia"
  input_datasets:
    - [FABDEM_V1-2, 1]

- !Dataset_Instance
version: 1
type: CoRE_LULCxTerrain_Vector
params:
  - region: "Jharkhand_Dumka_Masalia"
  - year: "2018"
alg_name: LULC_x_Terrain_Vectorization_v1
alg_inputs:
  params:
    - region: "Jharkhand_Dumka_Masalia"
  input_datasets:
    - [IndiaSAT_LULC_Raster, 1]
    - [CoRE_Terrain_Raster, 1]
```

References

- [1] 2025. Apache Airflow. <https://airflow.apache.org/>.
- [2] 2025. CoRE Stack. <https://core-stack.org/>.
- [3] 2025. STAC Index. <https://stacindex.org/>.
- [4] A. S. Ahmed, Abhilash Jindal, and Karan Beedkar. 2025. Popper: A Dataflow System for In-Flight Error Handling in Machine Learning Workflows. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE, Hong Kong, 4596–4599. <https://doi.org/10.1109/ICDE65448.2025.00361>
- [5] Chahat Bansal et al. 2021. IndiaSat: A Pixel-Level Dataset for Land-Cover Classification on Three Satellite Systems - Landsat-7, Landsat-8, and Sentinel-2. In *Proceedings of the 4th ACM SIGCAS Conference on Computing and Sustainable Societies (Virtual Event, Australia) (COMPASS '21)*. Association for Computing Machinery, New York, NY, USA, 147–155. <https://doi.org/10.1145/3460112.3471953>
- [6] Rajendra Bose and James Frew. 2005. Lineage retrieval for scientific data processing: a survey. *Comput. Surveys* 37, 1 (2005), 1–28. Available at CiteSeerX.
- [7] Susan B Davidson and Juliana Freire. 2008. Provenance and scientific workflows: challenges and opportunities. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM, 1345–1350. <https://doi.org/10.1145/1376616.1376772>
- [8] Patrick Ferris, Michael Dales, and et al. 2023. Planetary computing for data-driven environmental policy-making. *arXiv preprint arXiv:2303.04501* (2023). <https://doi.org/10.48550/arXiv.2303.04501>
- [9] Juliana Freire, David Koop, Emanuele Santos, and Cláudio T Silva. 2008. Provenance for computational tasks: A survey. *Computing in Science & Engineering* 10, 3 (2008), 11–21. <https://doi.org/10.1109/MCSE.2008.79>
- [10] Matthew A Hammer, Jana Dunfield, Michael Hicks, and Jeffrey S Foster. 2014. Adapton: composable, demand-driven incremental computation. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 156–166.
- [11] Laurence Hawker, Peter Uhe, Luntadila Paulo, Jeison Sosa, James Savage, Christopher Sampson, and Jeffrey Neal. 2022. A 30m global map of elevation with forests and buildings removed. *Environmental Research Letters* 17, 2 (2022), 024016. <https://doi.org/10.1088/1748-9326/ac4d4f>

- [12] Saharsh Laud. 2025. STACD-Airflow: Reference Implementation of STACD on Apache Airflow. <https://github.com/SaharshLaud/STACD-Airflow>. Accessed: 2025-08-14.
- [13] Shivani A. Mehta et al. 2025. *CoRE Stack: Technology Architecture for Distributed Geospatial Computing*. Technical Report. Indian Institute of Technology Delhi. <https://www.cse.iitd.ernet.in/~aseth/core-stack-layers-mar-2025.pdf>
- [14] Shivani A. Mehta et al. 2025. Initial Observations from Field Testing of a Digital Participatory Tool to Improve Water Security in Rural India. In *Proceedings of the 13th International Conference on Information & Communication Technologies and Development (Nairobi, Kenya) (ICTD '24)*. Association for Computing Machinery, New York, NY, USA, 337–364. <https://doi.org/10.1145/3700794.3700816>
- [15] Derek G Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martin Abadi. 2013. Naiad: a timely dataflow system. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. 439–455.
- [16] Derek G Murray, Malte Schwarzkopf, Christopher Smowton, Steven Smith, Anil Madhavapeddy, and Steven Hand. 2011. CIEL: a universal execution engine for distributed data-flow computing. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*.
- [17] Radiant Earth Foundation. 2024. STAC Specification. <https://github.com/radiantearth/stac-spec/tree/master> Accessed: 2025.
- [18] Yogesh L Simmhan, Beth Plale, and Dennis Gannon. 2005. A survey of data provenance in e-science. *ACM SIGMOD Record* 34, 3 (2005), 31–36. <https://doi.org/10.1145/1084805.1084812>

Received 2025-07-02; accepted 2025-08-11