Final Report prepared for:
NOAA/NESDIS

Contract Number: 1332KP22CNEEP0013
Report Number: 0002 (Final)
September 25, 2024

One-Stop Digital Space for Earth Observation Processing and Analysis

Prepared By:
Orion Space Solutions, an Arcfield Company
282 Century Place
Suite 1000
Louisville, CO 80027
Home Page: https://orion.arcfield.com

Principal Investigators:
Jeff Steward, Ph.D. & Patrick McBride, Ph.D.

Contributors:
Ryan Nguyen, Ph.D., Jeremy Highley, Rachel Stutz, John Furlong, Ryan Kelly, Joel Nyquist

Program Manager:
Jeff Campbell

Note: This report was edited by NOAA after delivery to make it more accessible.

This material is based upon work supported by the Dept of Commerce under Contract No. 1332KP22CNEEP0013. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Dept of Commerce.

Table of Contents

TABLE OF CONTENTS 3
1 INTRODUCTION 6
1.1 DIGITAL TWINS 7
1.2 DIGITAL TWIN BENEFITS TO EARTH OBSERVATIONS SYSTEMS 7
1.3 EFFECTIVE DIGITAL TWIN DESIGN 8
2 SOFTWARE CAPABILITIES AND INFRASTRUCTURE 8
2.1 DEVELOPMENT AND DEPLOYMENT ENVIRONMENT 9
2.1.1 Containerization 9
2.2 PROJECT GOALS AND FEATURES 9
2.3 SOFTWARE ARCHITECTURE 10
2.4 DATA PROCESSING OVERVIEW 12
2.4.1 Data Sources 12
2.4.2 Data Processing Pipeline 14
2.4.3 Data Governance and Standards 16
3 ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING 17
3.1 ANOMALY DETECTION 17
3.1.1 Supervised CNN 17
3.1.2 Unsupervised F-AnoGAN 19
3.2 COMPRESSION 21
3.3 LOSSLESS COMPRESSION 21
3.4 ENCODING 23
3.4.1 ATMS Encoding 24
3.5 DATA FUSION 27
3.5.1 Fusing ATMS and CrIS L1b Data to Extract Sea Ice Concentration 28
3.5.1.1 SIC Model Development – Phase I 30
3.5.1.2 SIC Model Development – Phase II 32
3.5.2 Retrieval Augmented Generation 38
3.5.2.1 Knowledge Base Construction 39
3.5.2.2 RAG Pattern Overview 40
3.5.2.3 Selected RAG Pattern 42
3.5.2.4 Evaluation Methods 44
3.5.3 Kolmogorov-Arnold Networks 46
3.5.3.1 Cloud Masks from CrIS L1b Data 48
3.5.3.2 Sea Ice Concentration from ATMS SDR 49
3.6 ML INFRASTRUCTURE AND MLOPS 53
3.6.1 Data Management 53
3.6.2 Model Training 55
3.6.3 MLOps Platform And Deep Learning Infrastructure 57
4 UI VISUALIZATION 59
4.1 VISUALIZATION TOOL 59
4.1.1 Data Visualization Tool Selection 59
4.1.1.1 NASA Web WorldWind 59
4.1.1.2 Alternative Visualization Tools Explored 60
4.1.1.3 CesiumJS 61

4.1.2 UI Visualization Data Layers and Techniques 61
4.1.2.1 Map and Satellite Imagery 61
4.1.2.2 Grid Lines 62
4.1.2.3 Bathymetry 62
4.1.2.4 Administrative Borders 63
4.1.2.5 Volumetric Vector Field Representations 64
4.1.2.6 Terrain Data Visualization 66
4.1.2.7 Additional Data Formats and Techniques 68
4.2 DATA VISUALIZATION FORMAT 70
4.2.1 Hierarchical/Variable Resolution Gridding 70
4.2.2 Global Gridding Challenges 72
4.2.2.1 Oblate Spheroid Representations 72
4.2.2.2 Mercator Projection Gridding 72
4.2.2.3 Alternate Projection Grids 73
4.2.3 Volumetric Data Visualization 77
4.2.3.1 Volumetric Gridding 77
4.2.3.2 Hierarchical Levels of Detail 78
4.2.3.3 Boxes vs Point Clouds Rendering 79
4.2.4 Data Visualization Format Performance Testing 82
4.2.5 UI Platform Design and Architecture 84
4.2.6 UI Platform Services & Features 85
4.2.6.1 Datasets Service 85
4.2.6.2 Data Slicing Feature 86
4.2.6.3 Data Iso-surface Selection Feature 88
4.2.6.4 Point Rendering Techniques 89
4.2.6.5 Models Service 92
4.2.6.6 RAG LLM Chatbot Service 95
4.2.6.7 Orbits Service 97
4.2.6.8 Tileset Deltas Service 100
4.2.7 Opportunities for Improvement 102
4.2.7.1 Geometric Error and Screen Error Issues 102
4.2.7.2 Custom Shader to Engine Shader Interactions 102
4.2.7.3 Generalized Processing 103
4.2.8 Future Development 103
4.2.8.1 OpenEO 103
4.2.8.2 OGC API 103
5 RECOMMENDATIONS SUMMARY 105
5.1 SPECIFIC DIGITAL TWIN RECOMMENDATIONS: 105
5.2 ML RECOMMENDATIONS 105
6 PROGRAM BAA FULFILLMENT 107
7 PROGRAM MANAGEMENT AND EXECUTION 109
7.1 NOAA/OSS ENGAGEMENT 109
8 REFERENCES 109
9 APPENDIX 111
9.1 GLOSSARY OF TERMS 111
9.2 GRIDDING/TILE STATISTICS TABLES 111
9.2.1 Google S2 Cell Table 111
9.2.2 Standard Latitude Longitude Grid Table 112
9.2.3 Quantized Mesh Terrain Tiles (ALOS ~30m) 113

9.3 MICROSERVICE SOFTWARE ARCHITECTURE DIAGRAM..... 114

1 Introduction

NOAA collects observations, forecasts, and advisories from diverse sources. The size of this data reaches terabytes per day, but this economically important data often does not make it into the hands of the companies, organizations, and people who could use them. NOAA's mission includes seeking innovative methods to disseminate data in order to address actual, real-world problems so that NOAA's insights can become action for the government, industry, and general public.

Through the Earth Observation Digital Twin (EO-DT) Broad Agency Announcement (BAA) NOAA NESDIS Office of Architecture and Advanced Planning, Joint Venture Partnerships program, Orion Space Solutions, an Arcfield Company (OSS) and other partners have demonstrated that the digital twin paradigm provides a compelling ecosystem for enabling the kind of last-mile insights NOAA's diverse user base requires. Digital twins were originally developed in the manufacturing sector to provide a basis for testing and exploration a digital replica of a real, physical system. While it is often easier and cheaper to work with a simulation than a real physical version, that will only be valuable in as much as insights from one system will transfer to the other. Assuming this simulation/physical linkage is useful and well understood, applying the same concept of a digital twin to the Earth helps users interact with NOAA's enormous data catalog. In this project, OSS aims to prototype how such digital twins can indeed provide valuable real-world insights.

The value of a digital twin revolves around allowing the user to understand what is happening now on the Earth ("what now"), what is happening in the future ("what next"), and what could happen under various hypothetical scenarios ("what if"). Through a sophisticated visualization engine and the ability to run, manage, and visualize complex processes using High-Performance Computing (HPC) in the cloud, the OSS solution successfully prototyped how digital twins could be used for a variety of purposes including space domain awareness, sea ice retrieval, weather forecast dissemination, and helping train advanced machine learning algorithms. Representing the entire Earth system as a whole from the ocean to the biosphere and near space environment and everything in between is a daunting NOAA agency-wide task, but this project and final report provides NOAA with recommendations on how to proceed should it decide such an effort is worthwhile.

In this project, advanced software solutions and physics-informed machine learning (ML) technologies have been used to prototype a seamless and automated Earth observational data processing, analysis & visualization system. OSS showcased how this can be integrated with NOAA's Unified Earth System Modeling Framework for scientists, engineers, forecasters, and the public to better visualize, understand, and predict the past, present, and future of the Earth environment.

The goal of this digital engineering platform is to extend and automate data processing and analytics as a next-generation ground processing enterprise. The demonstrated EO-DT solution continuously ingests and decodes real-time spaceborne, airborne, and ground-based observations and numerical simulations. AI technologies (such as computer vision, data fusion, and Autoencoder) implemented within the EO-DT have shown the prototype capabilities to process, fuse, harness, and analyze the data, while also providing a flexible environment for researchers to

develop, apply, visualize and test their own algorithms. The digital system embeds a modern visualization toolkit, and a built-in data lake hosts the processed data products in proper formats for disseminating to physics-based, empirical, and machine-learned models. Orion's EO-DT enables users to visualize, understand, and predict the earth's environment's past, present, and future in one place with a few clicks. While more work is needed to bring this system to full operational relevance, the platform demonstrates the exciting promise of digital twins for NOAA's needs.

1.1 Digital Twins

Digital twins are intended to be a synthetic replica mirroring the behavior of a physical system to the extent possible necessary to make informed decisions. Digital twins have their roots in manufacturing and have been used in the Earth observation context from the early NASA space era where ground-based representations of spacecraft and equipment were fed inputs from previous and current missions to test and evaluate commands to evaluate and troubleshoot the actual equipment. Digital twins have progressed to create detailed software representations which can be utilized to explore a variety of scenarios and to better understand very complex interactions. In fact, most extant NOAA physics-based and empirical models fit well into this concept, but a key facet of these digital systems is the ability to "play" in ways that would be physically impossible or at least prohibitively expensive with the physical analog. Through the visualization and job submission engine, digital twins can enable users to study the evolution of behaviors of concern to them that may be rare or not yet encountered in the real system such as black swan events. The digital twin can be informed from sensors, or from simulated physics-based models, or from a hybrid approach of both. These inputs can be varied to simulate a vast array of possible situations. For environmental digital twins, events such as geomagnetic storms, extreme weather events, or equipment malfunction allow analysts to investigate and prepare potential responses and plan accordingly. The EO-DT program is therefore a step toward exploring effective design and use of digital twins in the Earth observing context.

1.2 Digital Twin Benefits to Earth Observations Systems

Earth observation systems benefit from the introduction of digital twin technologies due to many factors – including complex coupling, visualization, and nowcast/forecast/scenario exploration. Because of the extremely complex nature of Earth observations, many organizations, technologies, and teams have developed virtual representations to capture pieces of the complex puzzle. A digital twin can combine those representations and through their interaction can move closer to realizing a viable representation of the system for the task at hand. For example, a digital twin representation of a coastal ecosystem may need input from satellite observations, ocean and wave models, numerical weather prediction, biological feedback, and data assimilation/fusion. While each of these pieces may initially have either too fine or too coarse data for the task at hand, by stringing these components together pain points can be quickly identified, and the digital twin can be iteratively improved until it meets the user needs in a timely fashion. While each user will have different needs from a digital twin system, reusable visualization and user interface/experience within the system allows for more rapid understanding of the complex behaviors and interactions that occur in these simulations. Such views may include 2D surface plots, wind/current vector plots, volumetric 3D representations, and so on. Visualizing scientific data in the context of a sphere is itself a worthwhile endeavor, as seen from the enduring value of the Science on a Sphere

program. With a computer with moderate specifications (such as some sort of a GPU) in a browser environment, and without additional software needed, the OSS EO-DT provides interactive visualization so that the diverse user base is able to quickly understand the value of the digital representation and extract relevant information in the nowcast/ forecast/ scenario exploration mode. With the best estimate of the current environment, users can analyze and monitor the situation on the ground to the extent possible, which naturally depends on the availability of observations and the applicability of the relevant models. Uncertainty and lack of information needs to be clearly communicated so that decision makers base their decisions on appropriate insights. Training is also important for this purpose. Forecasting future results likewise can help with planning, again with the caveats that the uncertainty in the forecast must be properly quantified, understood, and taken seriously. Digital twins truly shine in the ability to utilize scenarios and hypotheticals, again with relevant uncertainty, to understand the bounds of environmental events and inform additional decisions such as city/state/federal government planning, satellite mission design, sea ice danger to oil exploration, and many additional use cases too numerous to list here. While each use case requires careful attention to detail and extensive concerted effort to ensure applicability, a Earth observation digital twin platform and ecosystem can help enable and accelerate such developments.

1.3 Effective Digital Twin Design

As popularized by the talks of Dr. Jacqueline Le Moigne and others, OSS followed the guiding principle for our digital twin to help answer the questions: “What Now?”, “What Next?”, “What If?” The idea is that an effective digital twin should help the user find relevant answers to those questions. “What Now?” concerns itself with representing the state of the system as it exists currently. This involves the incorporation of measured or modelled data and placing it into a 2D, 3D, 3D+time, etc. digital representation to allow the user to work with the data as appropriate for the given use case. The higher the resolution and accuracy of the data, the more closely the digital twin can represent the system, but since this comes at a higher computational cost, finding the appropriate trade-offs is a crucial step the digital twin should assist with. “What Next?” addresses how the model will evolve into the future. Results from this step typically take the form of a forecast. Models answering this question in digital twins are typically created using either physics-based models, empirical models, or AI/ML learned models of the systems involved. The ability to swap models and compare/combine results is one of the most attractive applications of digital twins. Finally, “What if?” explores using digital twins to change initial conditions, drivers, or parameters of the modelled system under study to produce varying predictions of a system. Exploring these variations can lead to better understanding of the system(s) and help drive informed decision making.

2 Software Capabilities and Infrastructure

Software infrastructure is crucial to the creation and delivery of an effective digital twin user experience. Earth observations in particular entail the ingestion, processing, and delivery of huge datasets. Further, these datasets may also entail several stages of processing moving from raw sensor data to delivered data product and then further to the assimilation and fusion of data products. The section discuses these software components along with recommendations.

2.1 Development and Deployment Environment

The software developed by OSS for the EO-DT program utilized a modern software stack containing a data processing pipeline to deliver effect data products for visualization and further processing in standards-based formats and transports. Most of the backend code is written in Python due to the prevalence of scientific and numeric processing libraries. OSS utilized a GitLab source repository implementing continuous integration/continuous deployment (CI/CD). Git is a software tool developed for the Linux kernel development and used extensively to allow for parallel development of co-mingled software products by multiple developers. Using git for source control allows an organization to develop, enhance, and stabilize sections of code at any stage of deployment with a reliable and repeatable process. CI/CD patterns ensure the latest code is delivered upon completion and that resulting deployments are consistent and stable. This CI/CD includes automated testing, code quality checking, and deployment. OSS recommends having such tools in place with the caveat that all such tools must ultimately add value without distracting from the ultimate goal of usable software. CI/CD can be useful to help find software issues early but if overdone can potentially become a shackle that discourages or even prevents real innovation; for example, with low-level unit tests and the requirement to test every single line of code with 100% coverage, changes and refactoring may become difficult without real benefit to system stability. It is recommended for a centralized internal development platform team to offer a relatively light touch set up as a default with the potential to add increasingly sophisticated features if-and-only-if the development team agrees such CI/CD tools are actually helping, not hurting, the development process. This ensures that the CI/CD is enabling high-quality and rapid development.

2.1.1 Containerization

Core components of the EO-DT code base are written for cloud platform deployment with a microservice architecture to allow for flexible deployment and scaling of resources using containerized and stateless components. These services can be composited together to form a scalable application but retain the flexibility to swap services to meet the request or performance desires of the end users. OSS initially explored delivering components using Apptainer (formally Singularity) containers, but ultimately used docker components primarily because of the simplicity and widespread support for docker toolchains. It should be straightforward to port components between these frameworks as they are similar but have some structural differences during configuration and deployment. While Apptainer containers are not Open Container Initiative (OCI) compliant, it maintains extensive interoperability with OCI containers and can build, host and run OCI containers from standard repositories. Apptainer itself is free and open source and falls under the BSD license. In general, OSS recommends Apptainer for HPC deployments due to their isolation in multi-user restricted environments, while for general purpose containers the simplicity of docker and community support is difficult to beat.

2.2 Project Goals and Features

Early in the program OSS developed a series of features to encompass our expected deliverables for the project and to focus our efforts on those specific goals. The goals fall broadly into two categories: 1) AI/ML processing of the data and 2) UI/UX visualizations. AI/ML features are meant to address enhanced processing capabilities that serve to advance the quality, accuracy, and efficiency of the scientific data. The UI/UX use cases are focused on displaying the data sources

in a modern web-based 3D/4D visualization platform to maximize the accessibility of high density volumetric scientific data.

Feature #	Name	Category	Objective / Description	Actors	Benefits/Purpose
1	Data assimilation of latent space data	AI/ML	Test assimilation of ML optimally compressed data	Forecasting	Improved data assimilation techniques
2	Data Fusion of Level 1b Data	AI/ML	Use multiple sensors collecting data in the similar regions with various levels of fidelity to get a more accurate representation of the L1b data	Fusion	Higher resolution data
3	Anomaly Detection	AI/ML	Remove and resolve issues in the fused data products	Anomalies	Increased trustworthiness, improve accuracy of higher-level products
4	Lossless Compression	AI/ML	Lossless compression for increased data storage	Lossless Compression	Increased data storage
5	Co-locating Satellites	AI/ML	two satellites passing close to each other	Data Fusion	Can improve calibration, our nowcasting accuracy, confidence in our observations, etc.
6	Time Slider	UI	View geophysical and/or radiometric data over time (as in an animation).	Visualizer	Visualize the time dependence of the data.
7	Altitude Slider	UI	View horizontal geophysical data at multiple levels of the atmosphere controlled by a slider.	Visualizer	Visualize the horizontal structure of the data at a particular vertical level.
8	Dataset Comparison	UI	Display two forecast models, or look at two similar datasets such as VIIRS+GOES, or compare pre/post processed data	Visualizer	Visualize the model differences.
9	Cross Section View	UI	Show a (vertical) cross section of the volumetric data	Visualizer	Visualize the vertical structure of the data at a particular horizontal location.

Table 1. Overview of program features.

2.3 Software architecture

The software architecture OSS employed follows a standard processing pipeline for satellite data and validation shown in Figure 1. It begins in the upper left with the ingestion of level 1 and level 2 observation and retrieval data directly from the satellite data sources. OSS used NOAA's Open Data Dissemination Program (NOAA, 2019) to gain access to nearly all the data used. An extract, translate, load (ETL) pipeline was created in Mage.ai to select, transfer, convert, and then later purge data in a local data lake. Mage provides the ability to trigger based on scheduled times or completion of other steps. OSS configured this ETL pipeline to periodically pull the latest information. The data was kept in its original format as well as being tiled for visualization as discussed below.

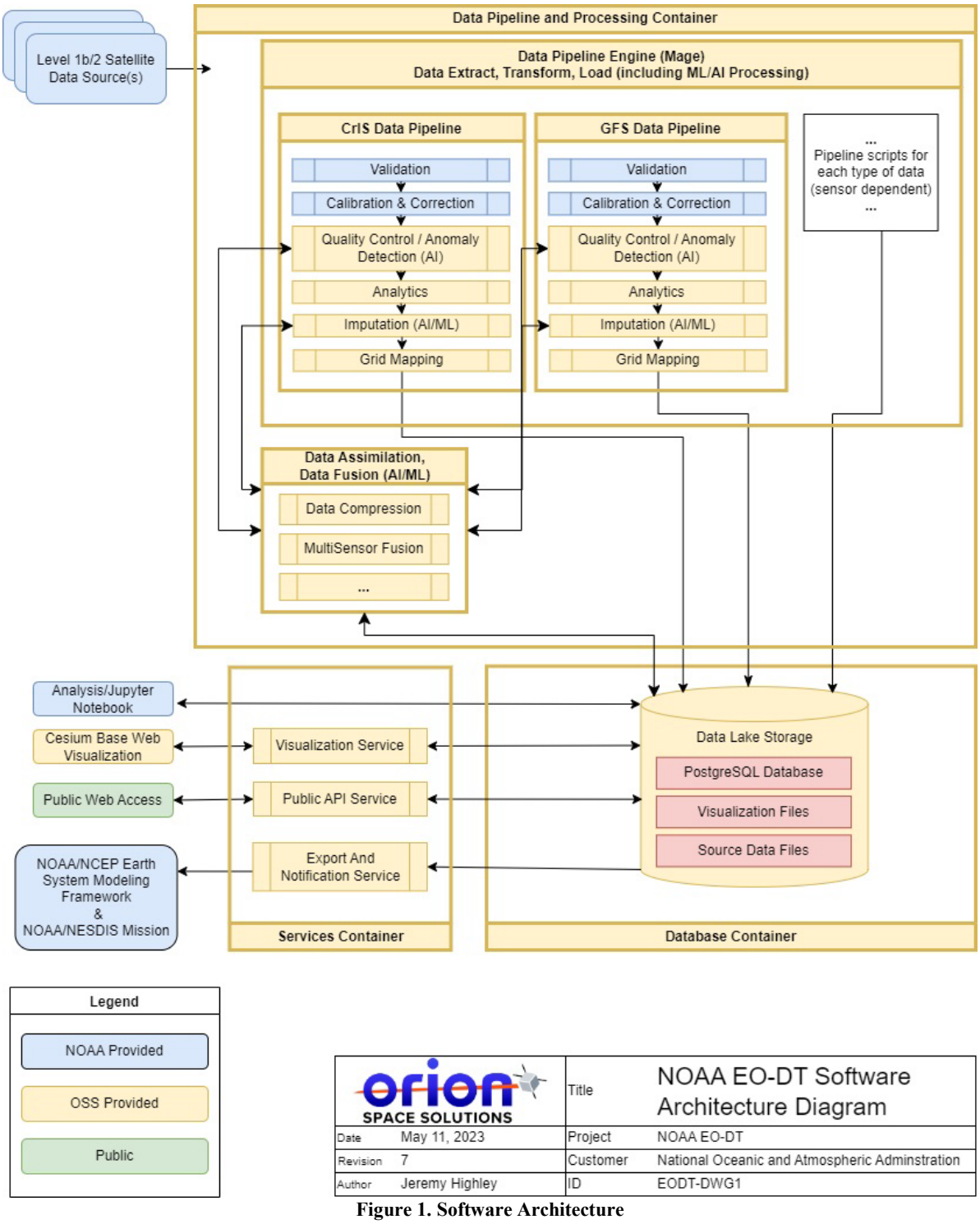

Figure 1. Software Architecture

The data was kept both in its native format as well as being tiled for visualization as discussed below. Services were created to expose these processed data sources so that they could be used in the visualization or other API services that may need to access the information.

2.4 Data Processing Overview

The first step to any digital twin is the effective consumption of data for use within the digital representation. This requires a scalable infrastructure to ingest and perform any needed processing of the data. Raw data, especially from satellite instruments, can be difficult for humans to interpret, so multi-stage processing of the data is necessary to convert from observation instruments to physical values. OSS chose to utilize a tool called Mage.ai to perform the initial query, download, and processing. The AI/ML tools were developed using standard AI Python tools and not integrated into the pipeline due to their prototype and experimental nature of design - but could be added to the continually running processing chain for an operational environment. NOAA's NESDIS data, hosted through the Open Dissemination Program on AWS, acted as the primary data source and was utilized heavily in the data processing chain.

2.4.1 Data Sources

OSS identified numerous data products that could be used in our prototype digital twin, listed below, most of which are provided from NOAA. These sources represent a variety of data formats, instrument technologies, and environmental regimes. OSS set up data download pipelines for the sources listed in **bold**, and generated tilesets for visualization for those in ***bold italics***. Manually processed data sets are shown in *italics* only. Since this was a prototype, OSS selected only some of the datasets from the entire list to showcase the capabilities of our digital twin. For visualization in particular, OSS limited some of datasets that involved less intensive processing to visualize as a tileset in CesiumJS. This included CrIS radiances (which could be roughly converted to brightness temperatures) as well as further processed products such as GFS and space environment models, as opposed to, for example, imagery data from GOES.

- Joint Polar Satellite System (JPSS)
    - **Advanced Technology Microwave Sounder (ATMS)**
    - ***Cross-track Infrared Sounder (CrIS)***
    - Visible Infrared Imaging Radiometer Suite (VIIRS)
        - Sea Surface Temperature product
        - Snow Cover product
        - *Sea Ice Cover product*
- Deep Space Climate Observatory (DISCOVR)
    - Faraday Cup (speed and direction of positively charged solar wind)
- GOES Satellites
    - **Advanced Baseline Imager (ABI)**
        - **Radiance Mesoscale (RadM)**
        - **Radiance Full Disk (RadF)**
        - **Radiance CONUS (RadC)**
    - **Space Environment In-Situ Suite (SEISS)**
        - **Solar and Galactic Proton Sensor (SGPS)**
        - **Energetic Heavy Ion Sensor (EHIS) (heavy ion fluxes)**
        - **Magnetospheric Particle Sensor High (MPS-H) (electron and proton fluxes)**
        - **Magnetospheric Particle Sensor Low (MPS-L) (electron and proton fluxes)**
    - Magnetometer (multi-axis measurement of Earth's magnetic field)
- Global Data Assimilation System (GDAS) product

- ***Global Forecast System (GFS) (1° and 0.25° grids) product***
- NCEP ADP Global Upper Air and Surface Weather Observations (PrepBUFR) product
- Aerosol Optical Depth (AOD) product
- Active Fire products from NOAA STAR
- *Ocean Surface Current Analysis Real-Time (OSCAR)*
- Space Environment Models (non-NOAA)
    - ***Thermosphere-Ionosphere Electrodynamics General Circulation Model (TIE-GCM)***
    - ***NRL Mass Spectrometer Incoherent Scatter radar model (MSIS)***

The download of the implemented data sources had a variety of throughputs that were limited primarily by download speed from Amazon S3 which hosted the NESDIS Open dissemination program data to the OSS processing computers and the platform visualization processing times.

Data Set	Availability Frequency	Download Frequency	Processing Rate	Spatial Resolution	Temporal Resolution
ATMS (antenna temp and radiance), 3 satellites	~142KB /~300s /satellite	All new files every hour	60s per file (~20k scans)	16-75km	~1hr revisit
CrIS (full science data), 3 satellites	~15MB /~300s /satellite	"	60s per file (~20k scans)	14km hor., 1km vert.	~1hr revisit
ABI-RadM, 3 satellites	~0.1-4MB /60s /satellite	"	n/a	0.5-2km	60s
ABI-RadC, 3 satellites	~0.1-32MB /5m /satellite	"	n/a	0.5-2km	5m
ABI-RadF, 3 satellites	~4-240MB /10min /satellite	"	5m	0.5-2km	10m
SGPS	323KB /1min /satellite	"	n/a	n/a	1m
EHIS	220KB /5min /satellite	"	n/a	n/a	5m
MPS-H	397KB /45s /satellite	"	n/a	n/a	45s
MPS-L	1MB /30s /satellite	"	n/a	n/a	30s
GFS	4MB /6hr	All new files every 6 hours	5m per vertical level	0.25°	6hr

Data Set	Availability Frequency	Download Frequency	Processing Rate	Spatial Resolution	Temporal Resolution
TIE-GCM (model)	n/a	Run on demand	15s	5° hor., 29 vertical pressure levels between ~90-600km	15min
MSIS (model)	(run on demand)	Run on demand	60s	variable, processed at 3° hor., 200 km vert.	3hr

Table 2. Data Throughputs for Implemented Data

2.4.2 Data Processing Pipeline

ETL pipelines assist in data processing by providing automated, defined workflows. They are a critical aspect of many scientific and engineering projects which involve automatically extracting data from various sources, transforming it into a format that can be used for modeling and analysis, and loading it into a data storage system. ETL workflows allow for multiple, complex processing steps which can handle all common data formats. OSS decided to use the containerized Mage.ai open-source data processing pipeline to perform ETL workflows from various NOAA data sources. By packaging the necessary software and dependencies into a containerized ETL image, the pipeline can be run consistently, regardless of the host system's configurations. This ensures that the data is processed and loaded correctly, and that the results are consistent and accurate. The Mage.ai pipeline uses python scripts for processing at any step of the data pipeline and serves as the scheduler and executer of functions to move data from source to destination. Mage also supports scripts in other languages such as SQL and R.

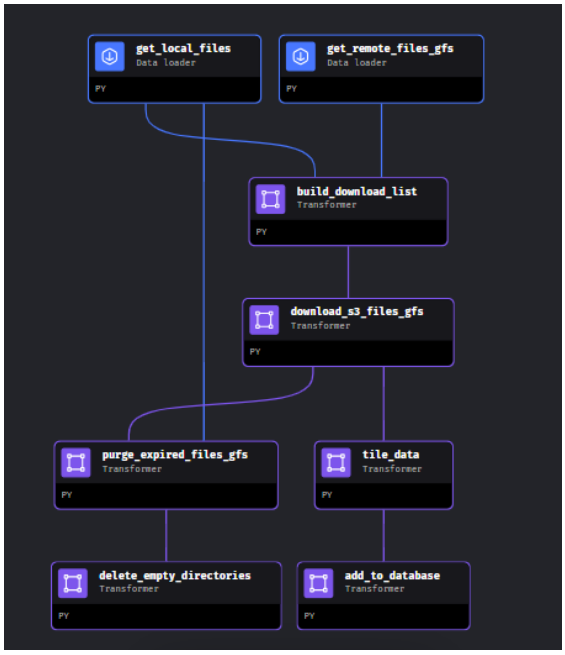

Figure 2. GFS data processing pipeline in Mage.

The OSS pipelines perform four main processes: data download, rolling window maintenance, tileset creation, and database updates. Since OSS did not have access to NOAA on-premise storage for datasets, data was downloaded from AWS S3 through NOAA's Open Data Dissemination program. For storage reasons, OSS maintained a rolling window of data for each of these datasets, with a minimum of 24 hours and maximum of 30 days depending on dataset size. Many of these were NOAA Level 1b data products which were then directly processed for visualization in the CesiumJS engine. OSS decided to use a separate format (3DTiles) for the visualization (referred to as tilesets) as well as a PostgreSQL database to store metadata for each generated tileset. The Mage pipelines were scheduled based on data availability; for example, NOAA's Global Forecast System (GFS) products are published at a 6-hour cadence, so the pipeline for GFS data download and processing was set to run every 6 hours.

To connect to the containerized "tiling service", which generates tilesets from a downloaded data file, the scheduling service sends an API request with the path to a file to be visualized and a few additional parameters that allow the tileset to be created with the right metadata. The parameters for a request to the tiling service are as follows:

host: machine where the dataset to be tiled is hosted
input_path: path to the dataset to be tiled (on the host machine)
output_path: path to the output directory where the tileset should be generated
iso_start_date: start date for the dataset
iso_start_time: start time for the dataset
input_file_type: input file type/extension (e.g. "anl", "nc", "csv")
is_time_series: boolean indicating whether dataset includes multiple time steps
data_source_name: data source name as it is stored in the database (e.g. "gfs")
product_id: product within a data source (e.g. 580 for GFS surface temperature)

Without knowing the full details of NOAA's data pipeline service, for digital twin applications OSS recommends a service such as Mage, Apache Airflow, Apache DataBricks, etc. to help schedule ETL workflows for any processing steps of the digital twin that can be automated, deployed to a distributed environment (such as cloud), and scaled to the need of the data being delivered.

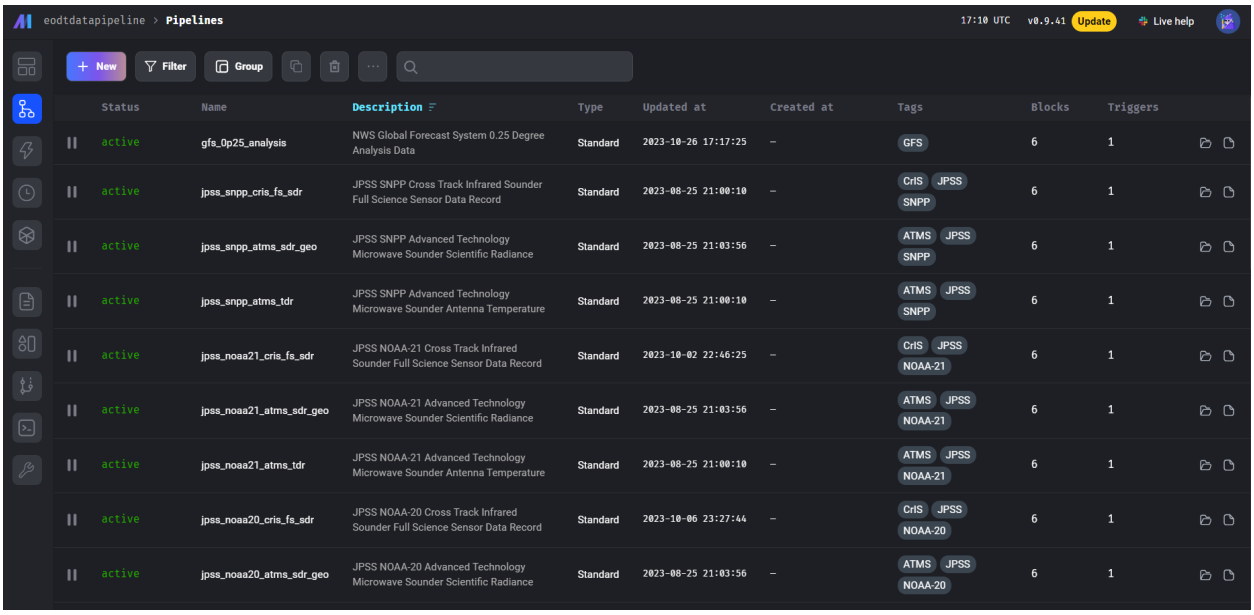

Figure 3. Screenshot of several of the currently active Mage pipelines, including data from GFS and the JPSS satellites. Each pipeline is configured to maintain a rolling window of 1-30 days of data, depending on dataset size.

2.4.3 Data Governance and Standards

Most scientific data sets for this program are distributed as single files for small segments of time that often require further processing and analysis to be merged, visualized, or maintained. OSS chose a data processing pipeline that pulled from our known sources hourly and kept the data for a 2-week rolling window to limit disk usage since we were not intending to setup an operational data processing operation. However, proper data governance would be required to ensure the quality, security, visibility and timeliness of the data provided. For an operational system, the data lineage would be of high scientific importance to ensure provenance and repeatability of the higher-level products delivered. Our data stores were isolated from external networks since our processing could be performed on internal servers, but proper security would be needed to isolate the data from manipulation or corruption. In operations, cloud storage of the data could enable redundant backups and multi-region copies for disaster recovery and faster regional availability. Standards around data preservation times could be implemented to reduce the long-term data storage cost or alternatively movement of data from high cost "hot" fast storage to slower "cold" storage. Cloud storage also provides the high bandwidth and network connectivity to distribute large volumes of data quickly from time of capture to public release – allowing faster "time to consumer" of high value NOAA data.

Open source and standard data formats proved important for the processing chains since the variety and complexity of the data dictates a vast toolchain. Files delivered in the open HDF5 and GRIB formats allow for flexible scientific structure while maintaining consistent readability and tool availability. The ability to find data tools in a variety of computer languages is critical to the use of the data in the broader scientific community since computer language adoption varies drastically from field to field. We relied on NASA's Panoply data tool for data exploration and visual spot checking of GRIB and netCDF files and HDFViewer for HDF5 file formats. Python was our primary data processing language for the pipelines. For non-standard data types, toolsets available in a broadest set of computer languages proves useful, but specifically C++ and Python seem to

have the most popular adoption in the data science community. ETL data pipeline futher isolates the development and deployment of non-standard data types to components which can be isolated and copied as needed.

Documentation for the file formats was sometimes difficult to find, but generally existed – having consistent filename and format documentation linked at the root description of the data product would be highly valuable for data reuse. For visualization, we relied heavily on OGC's 3D Tiles format which provided maximum flexibility in delivering scientific data over the web and in a manner that limited client resource and network bandwidth. There did not appear to be a standard for time dynamic file formats for visualization in standard toolsets.

Standards for metadata have been established in some fields to harmonize the reuse of data between providers, but the vast array and use of these data challenges any global harmonization efforts. Within specific fields, metadata standards can enforce metadata formats (dates, times, units, etc.) and those standards should be encouraged when distributing data at-large. We used standard ISO formats where available in our metadata and data formats (i.e. ISO 8601 for datetimes) and there are numerous other common standards that could be adopted in an operational setting (ISO 19115 for geographic data, National System for Geospatial Intelligence/NATO for mapping symbology for geospatial data, etc.)

3 Artificial Intelligence and Machine Learning

Artificial Intelligence (AI) and Machine Learning (ML) can help enable the goals of EO-DT to provide users with nowcast, forecast, or hypothetical information. This project's ML efforts focused on the exploration of a collection of use cases that were identified to serve the digital twin platform user's needs. OSS experimented with a collection of AI based techniques to automatically detect the quality of observations, reduce the required dataset storage size, estimate geophysical parameters from Level 1 and Level 2 data, and seamlessly serve data to users for exploration and visualization.

3.1 Anomaly Detection

OSS implemented two Deep Learning (DL) based anomaly detection methods to identify anomalies within CrIS data: a supervised CNN approach and an Unsupervised F-AnoGAN approach.

3.1.1 Supervised CNN

Convolutional Neural Networks (CNNs) are very effective at detecting features within images as they function to extract features over continuous blocks of pixels. OSS implemented a CNN based anomaly detection algorithm to classify a single channel of CrIS data as having an anomaly or not having an anomaly. In initial testing, OSS utilized a swath of CrIS data along a single channel. OSS decided to simulate anomalies to make this process easier and simulated potential observation errors by adding noise to individual pixels within a patch of CrIS data along one channel and hypothesized that in practice an entire patch would most likely not be noisy, only a few pixels. OSS created data to handle this case by adding Gaussian noise with 0.5-sigma to the original patches. OSS shuffled the data and created a train, validation, and test set. The train set consisted

of 20,000 samples and the validation and test sets were each 5,000 samples. The synthetic observation data used a standard scaler which normalizes each pixel's value based on the overall distribution. The batch size used was 1000 and the network was composed of a linear layer with 100 nodes followed by a 1D batch normalization layer and a leaky ReLU activation with a slope of 0.2. This fed into a second linear layer with 400 nodes with the same normalization and activation. The data was reshaped to the size (batch size, 4, 10, 10) which was passed to 3 convolution blocks consisting of a convolution layer, 2D batch normalization, and a leaky ReLU with a slope of 0.2 (Figure 4).

OSS tested with various choices for sigma to find the lowest noise level where the model began to perform at 100% accuracy (shown in Table 3). Being able to achieve convergence (Figure 5) and 100% accuracy with a noise level of 0.5-sigma for 25 random points in a 20x20 patch shows promise for using this method to identify noise for the incoming data.

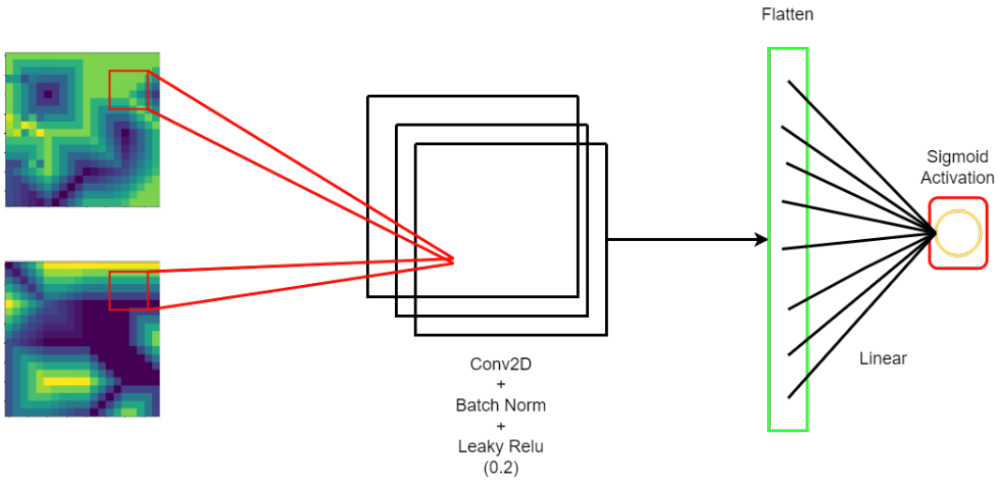

Figure 4. Visualization of CNN architecture.

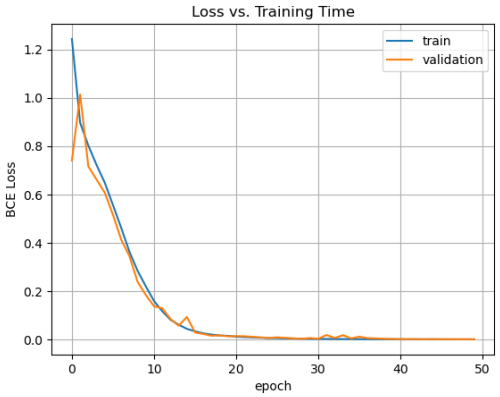

Figure 5. CNN Training Loss.

	Precision	Recall	F1-Score
Nominal	1.000	1.000	1.000
Anomalous	1.000	1.000	1.000
	Value		
Accuracy	1.000		

Table 3. CNN classification metrics.

This work identified anomalies that may exist in a single channel. A potential avenue of improvement is to leverage the ability to identify anomalies using multiple channels, which the next section describes.

3.1.2 Unsupervised F-AnoGAN

The second anomaly detection algorithm used the f-AnoGAN architecture (Schlegl, Seeböck, Waldstein, Langs, & Schmidt-Erfurt, f-AnoGAN: Fast unsupervised anomaly detection with generative adversarial networks, 2019) to identify anomalies within 10 by 10 patches of CrIS data along all 2223 channels. This architecture is a variation of the Generative Adversarial Network (GAN). A GAN consists of two models: a generator and discriminator which is sometimes called the critic. The goal of the generator is to generate realistic samples when given a vector of random noise from a latent distribution. The objective of the discriminator is to discern which samples are created by the generator and which samples are taken from the true set of samples. The Anomaly GAN (AnoGAN) architecture (Schlegl, Seeböck, Waldstein, Schmidt-Erfurth, & Langs, Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery, 2017) extends on the GAN architecture by defining an anomaly score. The anomaly score of a particular sample is calculated as a weighted sum of the discriminator's loss and a residual loss. The discriminator loss term scores its confidence that the image is real, while the residual loss term is calculated as the distance between the optimally generated sample, and the sample itself. To find the optimally generated sample, an iterative root finding algorithm is implemented to find the latent space vector that optimally reconstructs the original sample when given to the generator. The Anomaly GAN (AnoGAN) algorithm was shown to be effective at identifying anomalies within medical images. The Fast AnoGAN (f-AnoGAN) architecture builds off the AnoGAN by adding an encoder model to act as the inverse of the generator, eliminating the need for a root finding algorithm, and by implementing advanced training techniques. Note that with the addition of the generator model, the two components of an autoencoder are present within this architecture, highlighting the potential to use an autoencoder for anomaly detection.

The implementation of the f-AnoGAN utilized a collection of normalization, convolutional, and leaky ReLU activation layers in sequence, in each of the generator, discriminator and encoder models. Additionally, OSS leveraged the Wasserstein Loss with Gradient Penalty to train our model, in an effort to alleviate some common difficulties in training GANs (Arjovsky, Chintala, & Bottou, 2017). A high-level diagram of our f-AnoGAN architecture is shown in Figure 6.

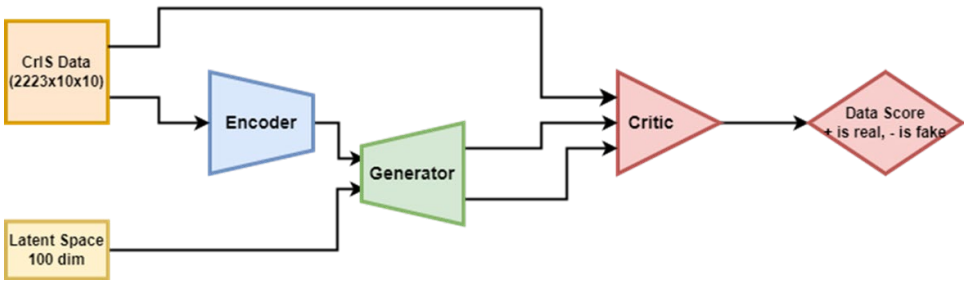

Figure 6. F-AnoGAN High Level Structure

OSS qualitatively evaluated our model by calculating anomaly scores on all the samples in the training and holdout testing set, and by reconstructing the original images after passing them through the encoder and decoder. Figure 7 displays a few of the CrIS samples as RGB images by

selecting three channels to display, in addition to their reconstructed counterpart, and the mean reconstruction error for each pixel across all 2223 channels. Qualitatively, the encoder and decoder provide reasonable reconstructions of the original images.

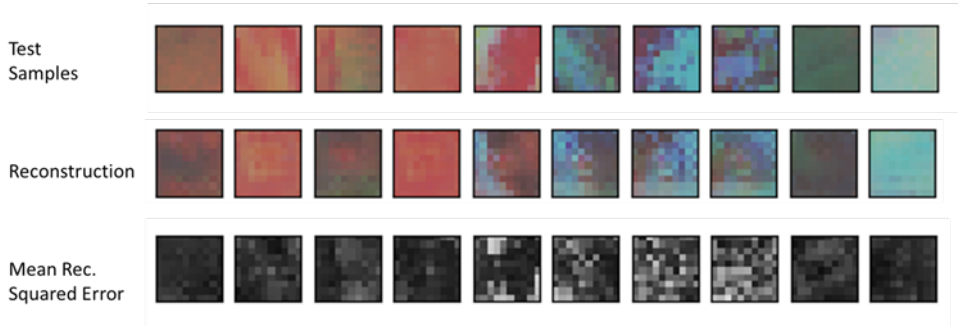

Figure 7. Example f-AnoGAN reconstructed test images.

The samples that were not effectively reconstructed by the model, as seen in the Figure 8, were identified as anomalous and achieved the highest anomaly scores of all the samples in our dataset.

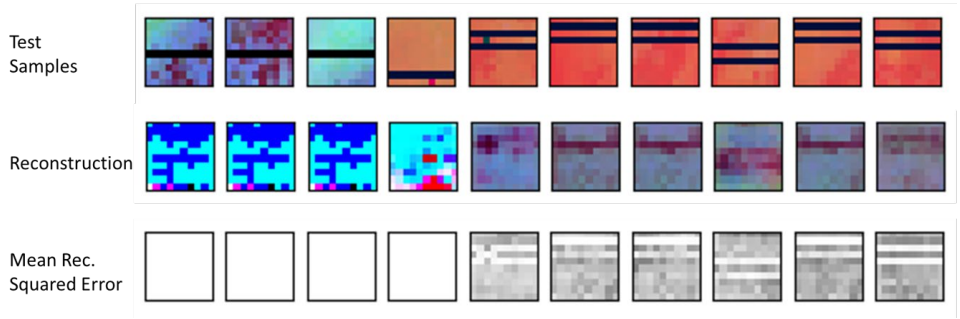

Figure 8. Example f-AnoGAN reconstructed test images with strips of missing data.

OSS additionally evaluated this model by adding noise to real images at a random collection of pixels. The model could almost perfectly identify images with noisy pixels when the number of pixels with 3 sigma noise added was at least three, or when the number of pixels with 1 sigma noise was 30 or larger. Figure 9 shows that the model can additionally identify the individual pixels that had noise added to them by locating individual pixels with large reconstruction error.

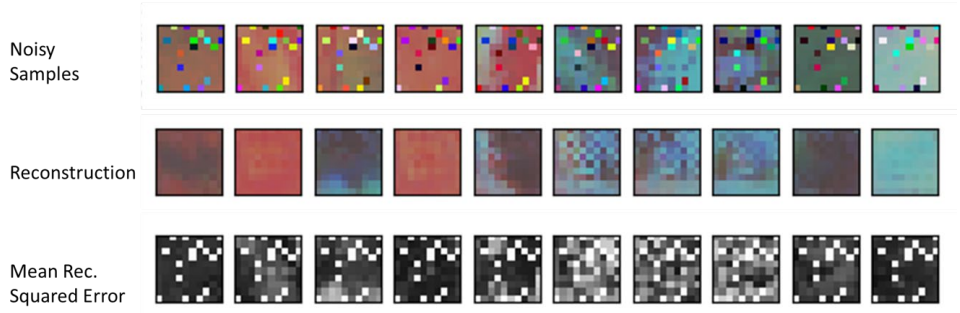

Figure 9. Example f-AnoGAN reconstructed testing images with artificially added noise.

Overall, this model was effective at detecting artificially noisy samples. It allows for the detection of anomalies at multiple scales, either at the entire “image” or granule level, or for each individual

pixel by looking at the reconstruction error of each pixel. However, there were several limitations to this study and to the development of the model. First, OSS utilized a very limited amount of data, consisting of samples established from a small number of CrIS granules over a few days. This implementation of the model took a long time to train to convergence, even with a small architecture and small dataset, so additional computational power should be utilized. Finding hyperparameter configurations that lead to model convergence is also a challenge. Finally, OSS did not begin from a large number of actually representative anomalous samples to evaluate the model against. Each of these issues should be addressed in order to make f-AnoGAN anomaly detection methods applicable to NOAA's needs.

3.2 Compression

For AI/ML purposes, compression is an underlying technology that is crucial for efficient data storage, transmission, and machine learning. OSS investigated both lossless and lossy compression.

3.3 Lossless Compression

As the name implies, lossless compression reduces the size of data without loss to ensures that the original data can be perfectly reconstructed. This compression technique offers several key advantages. Firstly, to the extent possible, it optimizes storage space, enabling more efficient use of storage devices and reducing costs. Secondly, it facilitates faster data transmission over networks with limited bandwidth, as smaller compressed files require less time to transmit. Thirdly, lossless compression preserves the integrity of sensitive or critical data, such as scientific datasets where loss of information may lead to undesirable artifacts or noise characteristics.

OSS prototyped implementation of an effective ML based lossless compression approach coined DeepZip (Goyal, Tatwawadi, Chandak, & Ochoa, 2018). The DeepZip method uses recurrent neural networks (RNNs), a type of deep learning model that can capture sequential dependencies in data as the crucial component of its architecture which is responsible for encoding the next value in a sequence, given the previous values. Traditional lossless compression algorithms, such as Huffman coding or arithmetic coding, rely on handcrafted rules and statistics to encode and compress data. In contrast, DeepZip aims to leverage the learning capabilities of RNNs to automatically learn and exploit patterns and correlations in the input data. The architecture consists of an encoder and a decoder, both implemented using RNNs. The encoder takes an input sequence and compresses it into a compact representation, while the decoder reconstructs the original sequence from the compressed representation. The RNN-based architecture allows the model to capture long-range dependencies and encode them efficiently. The architecture of DeepZip consists of two main components: the RNN-based model and the arithmetic encoder. The RNN is responsible for capturing the patterns and dependencies in the input data, while the arithmetic encoder encodes the output probabilities of the RNN into a compressed representation.

The RNN is trained to predict the next symbol in the input sequence based on the previous symbols. DeepZip uses a cross-entropy loss function, which encourages the model to output accurate probability distributions over the symbol space. The RNN's output probabilities represent the model's confidence in predicting each symbol. These probability distributions are then passed to the arithmetic encoder. The arithmetic encoder assigns shorter codewords to symbols with higher

probabilities and longer codewords to symbols with lower probabilities. This coding scheme takes advantage of the RNN's learned probabilities to assign more efficient codes to frequently occurring symbols and less efficient codes to infrequent symbols. When decoding, the arithmetic decoder reverses the compression process. It uses the compressed representation and the RNN's output probabilities to reconstruct the original sequence. The decoder uses the same arithmetic coding principles to decode the symbols based on their assigned codewords and the probability distribution.

The experimental results presented in their paper demonstrate that DeepZip achieves competitive compression ratios compared to traditional compression algorithms like gzip and bzip2, and in some cases, it even outperforms them. The authors also show that DeepZip performs well across different types of data, indicating its versatility and potential for real-world applications. To this end, OSS attempted to further compress binary filetypes using the DeepZip model. Initial experiments showed that DeepZip did not perform well for our hdf5 and grib file types as OSS saw an increase in file size, rather than a decrease. This is to be expected since hdf5 and grib are already efficient binary formats widely utilized by NOAA and other organizations.

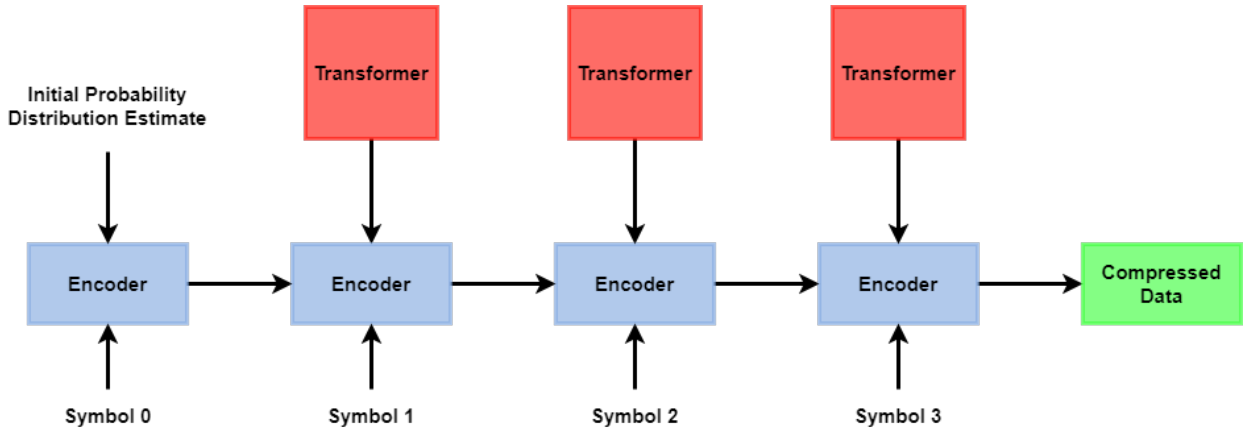

Figure 10. Transformer-based DeepZip compressor, where the encoder is an arithmetic encoder.

However, OSS believed there was room for improvement by replacing the RNN with a Transformer. Transformers have revolutionized the field of Natural Language Processing (NLP) and convincingly outperform RNNs in various tasks. Their ability to process sequences in parallel yields tremendous computational efficiency and faster training. Moreover, transformers utilize a self-attention mechanism that captures dependencies between words in a sequence, allowing for a more comprehensive understanding of global relationships. This attention mechanism significantly outshines the sequential processing of RNNs, which can lead to information loss over long sequences. Additionally, transformers excel at generating contextual word embeddings, enabling them to capture nuanced meaning and context within sentences. The superior performance of Transformers over RNNs led OSS to replace the RNN layers within DeepZip a Transformer. A high-level architecture diagram is shown in Figure 10.

From the binary hdf5 and grib file types the approach was able to achieve average compression ratios of 1.427 and 1.333, respectively (shown in Table 1). The model was successfully able to compress the given files and ideally could be used in ETL pipelines to decrease storage required and subsequently increase the size of the moving window of stored data.

File Type	Compression ratio ($R_c = \frac{S_{UC}}{S_C}$)	Example Original File Size	Example Compressed Size
Hdf5	1.427 ± 0.134	236.50MB	156.07MB
Grib	1.333 ± 0.011	510.91MB	383.18MB

Table 4. Compression ratios and example compressed sizes of hdf5 and grib file types, averaged over 5 files for each data type.

However, while these results were able to decrease the overall file sizes using the modified DeepZip algorithm, the compression and decompression times were infeasible. Both compression and decompression times were around 3 days, largely due to the model's size and the computational expense of the arithmetic encoder/decoder. OSS made some efforts to reduce the compression and decompression times by changing the model's hyperparameters with minimal success. Therefore, while these results show there is potential value in this approach, due to these practical constraints additional deep compression is not recommended at the current time for binary files such as hdf5 and grib.

3.4 Encoding

Throughout the Deep Learning (DL) experiments on this project, OSS consistently faced the challenge of managing the volume of available satellite data provided by NOAA. OSS built and trained DL models that use data from multiple instruments on the various satellites. However, these instruments, as well as those available on the GOES satellites, record substantial amounts of data; for instance, CrIS produces around 25 GB per day and ABI over 100 GB per day.

While having copious amounts of data is often necessary for a DL model to train and perform well, it is also necessary that the training data is representative of the data that will be seen at inference. Although it is not immediately clear how much data is necessary for a given model architecture, to fully capture seasonal and daily dependencies, a robust model would ideally be trained on the majority of historical data. This could result in the need for hundreds of days' worth of data, and hence on the order of terabytes of training data. Additionally, it is usually necessary to iteratively tune model architectures and hyperparameters, and it is infeasible to train every iteration of a deep learning model on this amount of data. Hence the task of hyperparameter tuning and architecture design can be difficult on such large datasets. OSS recommends performing initial architecture design and hyperparameter experimentation on smaller subsets of data to accelerate training.

To potentially alleviate some of the difficulties associated with the amount of available data, in some areas OSS decided to explore the use of "lossy compression," or encoding, algorithms to reduce the overall data footprint. In principle, if high compression ratios with minimal loss of information is possible, the amount of data can be increased during training and training time reduced without sacrificing model performance by training the models on the encoded data. However, it is also important to preserve the scientific integrity of the original data. OSS reports the findings here.

3.4.1 ATMS Encoding

The goal of an encoding algorithm is to compress a sample of data while retaining as much of the information of the original sample as possible with the use of decoder. A simple example of an encoding algorithm is Principal Component Analysis (PCA) which is a linear algorithm that represents a collection of data by extracting the most prevalent linear dependencies within the dataset (the principal components). The Autoencoder (AE) can be seen as a natural extension of PCA that learns a nonlinear representation of the original dataset by training a NN Encoder model to encode a given sample in a smaller dimensional latent space, and a NN Decoder model to reconstruct the original sample from the smaller dimensional representation. AEs often learn latent representations of the original dataset that are poorly conditioned and can thus be difficult to interpret or may not generalize to unseen data. An extension of the AE model is the Variational Autoencoder (VAE) which adds variation (or random noise) to the encoded sample before it is passed to the decoder, and subsequently adds KL-Divergence training loss term that conditions the model to learn a gaussian representation of the training set.

OSS trained a VAE on one day's worth of ATMS data from the SNPP satellite and then compared the model to the Discrete Cosine Transform. After training the VAE, additional evaluation was performed using a separate day of testing data. To prepare the data, one granule of L1b SDR data with shape (22, 12, 96), as a sample, was utilized and normalized the values according to the training set's distribution along each of the 22 channels. Within each day 2700 samples (granules) were obtained. For the VAE encoder and decoder architecture, OSS tested several architectures, but ultimately the optimal architecture found utilized Densely Connected Convolutional Layers (Huag, 2017). This architecture is a variation of the CNN model that uses batches of normalization, leaky ReLU and convolutional layers, with skip connections between each of the batches, as visualized in Figure 11. The skip connections within this architecture are similar to the residual skip connections within ResNet, and can be implemented as residual connections, or as concatenated connections; in this case, OSS concatenated the outputs of initial layers to subsequent layers. By providing the outputs of initial layers to subsequent layers, the later layers do not need to learn how the information extracted from the initial layers is abstractly represented by the intermediary layers. This allows later layers to leverage previously learned features explicitly and in combination with the features of all the other previous layers. It is hypothesized that this reduces the overall number of parameters that are needed within the model.

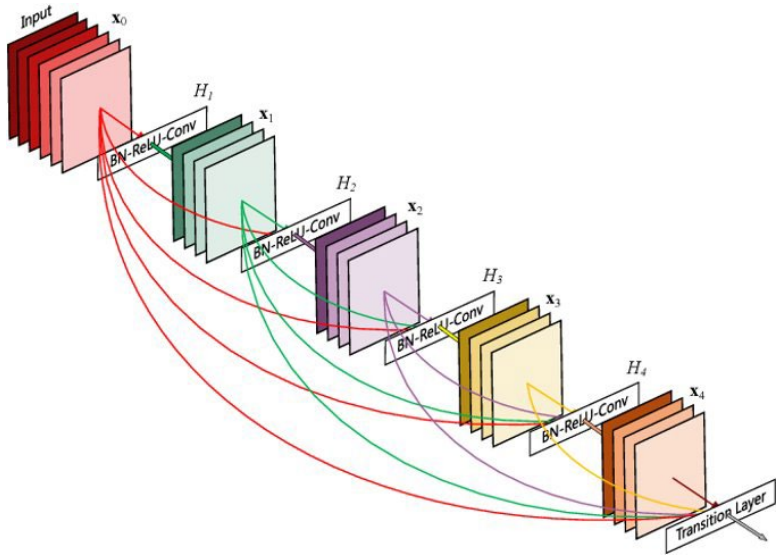

Figure 11. Densely Connected Convolutional Layers.

OSS tested several hyperparameter configurations with densely connected layers and report the results from two of them. The first model had 3 convolutional layers, a 9-by-9 kernel size, and a channel growth factor of 30, and was trained for 3 hours. The second model had 10 layers, a 5-by-5 kernel size, and a channel growth factor of 10, and was trained for 13 hours. Each model was trained with the Adam optimizer and used a latent dimension of 1200, representing a roughly 21x compression ratio.

The Mean Absolute Error (MAE) for each of the reconstructed samples in both the training and testing sets are shown in Figure 12, Figure 13, and Figure 14. Because the samples were normalized, a mean absolute reconstruction error of 1 would indicate that each reconstructed pixel was on average within 1 standard deviation of the original pixel value, according to the channel-wise distributions. The following figures show the distribution of sample MAE for the training and testing sets, along with the distribution of error for the DCT algorithm, which reconstructs samples using the 1100 lowest frequency terms. The n-dimensional DCT provided by the scikit-learn (Pedregosa, et al., 2011) Python package was utilized.

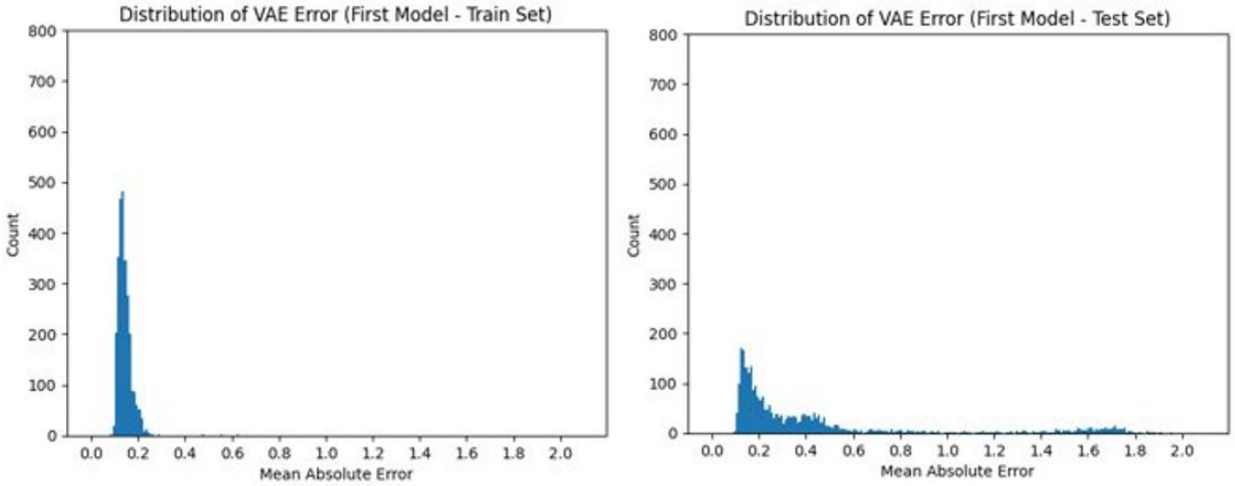

Figure 12. VAE first model distribution of mean absolute error. Left: training set. Right: testing set.

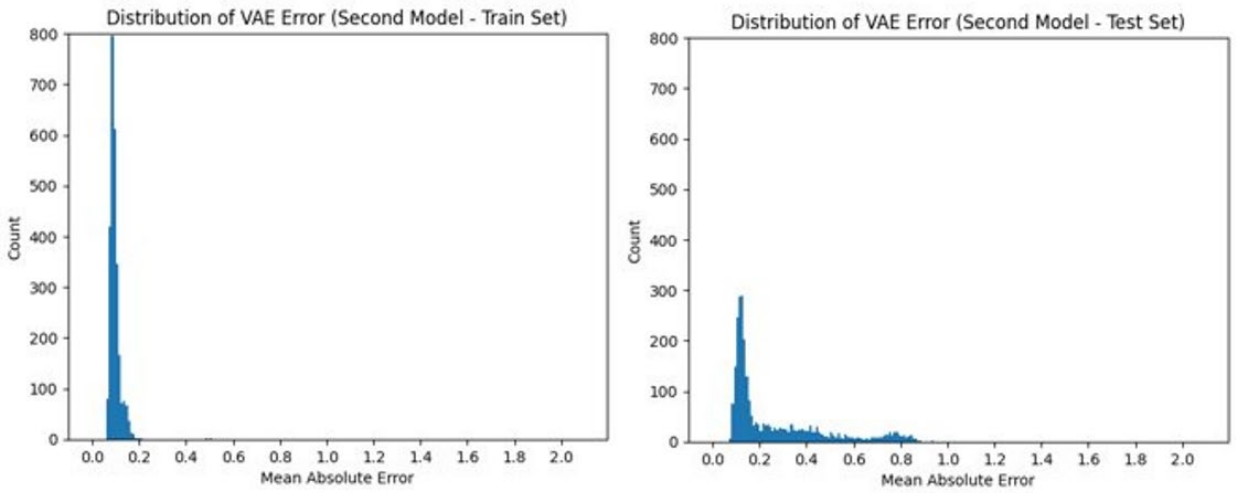

Figure 13. VAE second model distribution of mean absolute error. Left: training set. Right: testing set.

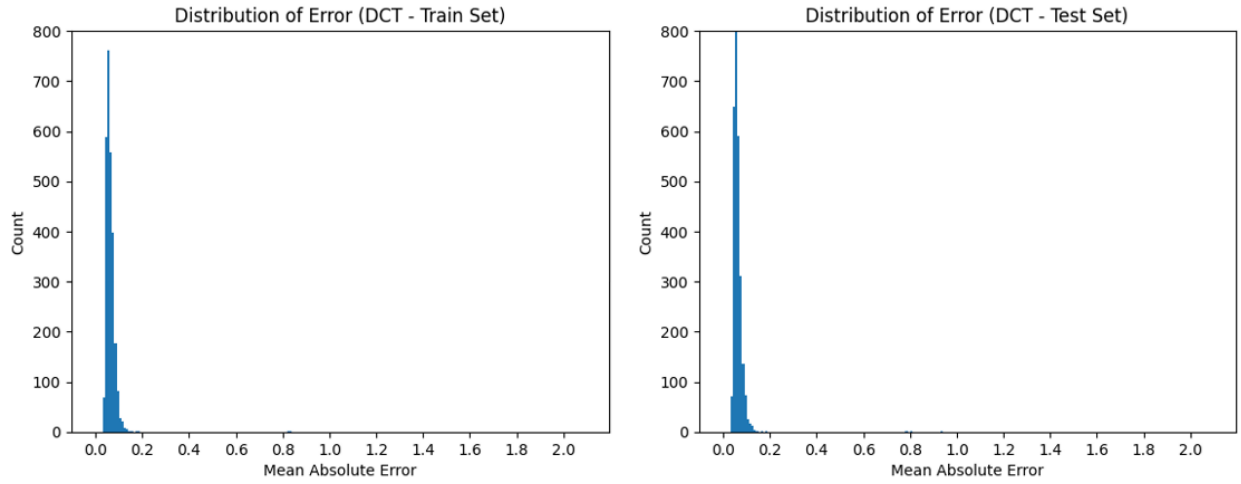

Figure 14. DCT distribution of mean absolute error. Left: training set. Right: testing set.

Figure 12, Figure 13, and Figure 14 show that each method was able to achieve a typical MAE of less than about 0.2 on the training set. However, the VAE models did not generalize very well to the testing set, as seen by the right skew in the testing set distribution of error. Moreover, the DCT was able to achieve a consistently smaller reconstruction error than each of the VAE models, with the second VAE model significantly outperforming the first, highlighting the importance of model architecture and training time. The lack of generalization capacity of the VAE models can be partially attributed to the training set not representing the testing set, as they were from entirely separate days.

To qualitatively verify each of the model's performance, an example reconstructed image for the Second VAE model is shown in Figure 15 and the DCT with 1100 frequency components retained in Figure 16. Qualitatively, the VAE reconstructions can capture some of the higher frequency components that the DCT ignores.

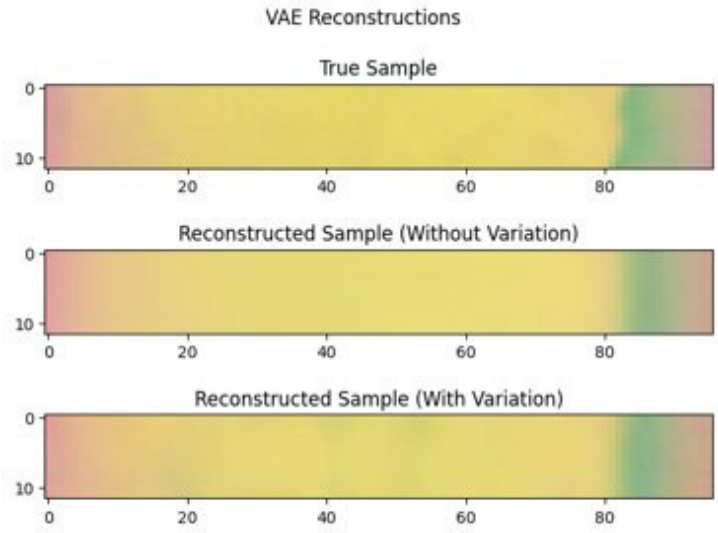

Figure 15. VAE second model, example normalized sample reconstruction. Channels 3, 5, and 11 are shown as RGB values respectively.

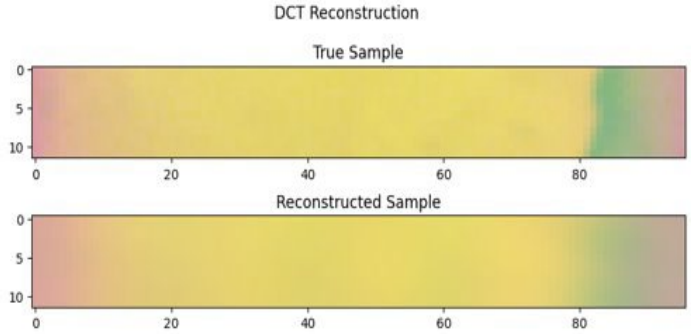

Figure 16. DCT example normalized sample reconstruction. Channels 3, 5, and 11 are shown as RGB values respectively.

These results might lead one to believe that the classical compression DCT algorithm is overall better than a VAE. The DCT requires no training, is straightforward to implement and configure, and achieved lower reconstruction errors than our VAE model. However, the VAE models were only trained on one day of data, which is not enough to be conclusive as the model has likely overfit to our training data. If given few hundred days' worth of training data and a more complex architecture with more tunable parameters, it is possible a VAE could learn a near optimal representation of ATMS data. Nonetheless, the DCT is recommended as an initial baseline for DL encoding.

3.5 Data Fusion

Digital twins benefit greatly from fusing multiple sources of data together. This can either be done externally as a product to be visualized or within the platform itself. In this project, OSS prototyped innovative ML methods for data fusion to leverage multiple data sources to gain insights that would not be available from a single data source. Data fusion can occur at different areas in the data processing chain. Early data fusion is the process of combining datasets before any other processing steps. Intermediate fusion combines multiple data sources after first processing them to be within a similar domain, but before finalizing the output data product. Late data fusion combines data after the full processing lifecycle has occurred.

3.5.1 Fusing ATMS and CrIS L1b Data to Extract Sea Ice Concentration

To prototype using Earth observations for the purpose of a digital twin, OSS trained a model to simultaneously ingest data from the ATMS and CrIS instruments on board the NOA20 satellite. It is reasonable to pair these datasets, as they both observe roughly the same geographic region, and their information content will likely be complementary. The machine learning data set was curated by pairing 12 ATMS scans with 4 CrIS scans, assuming that the scans have sufficient overlap between them. This pairing requires finding the subset of scan indices for each of ATMS and CrIS for which the difference in scan time is sufficiently small, and for which the next collection of scans are continuous, i.e. there are no gaps between their scans. OSS then labeled each of the ATMS and CrIS pixels with the Sea Ice Concentration (SIC) value obtained from interpolating the *NOAA/NSIDC Climate Data Record of Passive Microwave Sea Ice Concentration, Version 4* (Meier, Fetterer, Windnagel, & Stewart, 2021) and *Version 2* (Meier, et al., 2013) in both the northern and southern hemispheres. Below this SIC dataset is visualized in the northern hemisphere on January 21st, 2023, in Figure 17 and Figure 18.

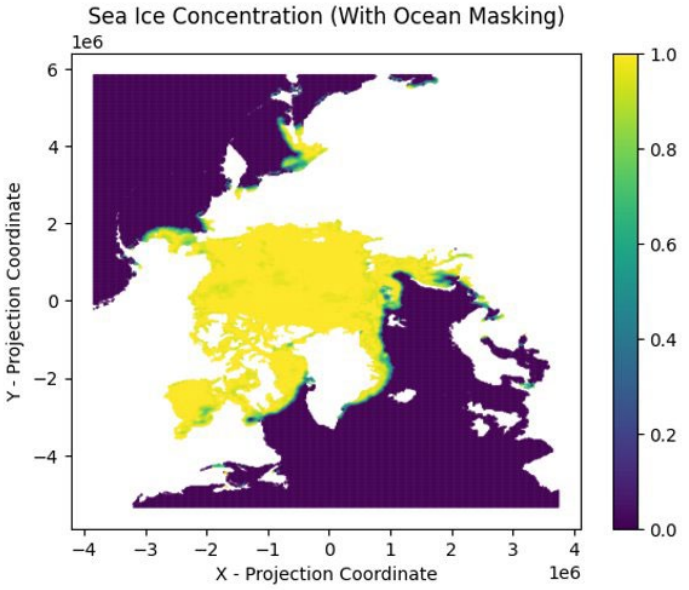

Figure 17. Sea ice concentration on January 1st, 2023 in the northern hemisphere (Meier, Fetterer, Windnagel, & Stewart, 2021); visualized in the dataset native stereographic projection coordinates.

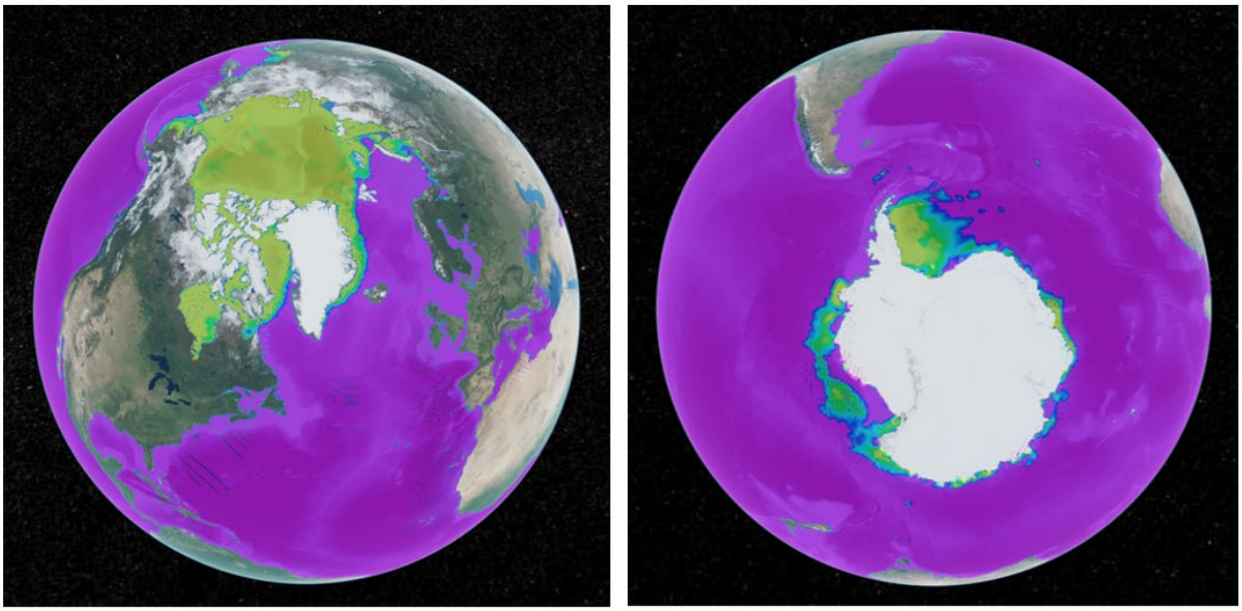

Figure 18. Sea ice concentration on January 1st, 2023 (Meier, Fetterer, Windnagel, & Stewart, 2021)**. Visualized in the OSS EO-DT Cesium toolkit. Yellow-Green color indicates high sea ice concentration, while purple color indicates zero sea ice concentration. (Left) Northern Hemishpere. (Right) Southern hemisphere.**

To accomplish this labeling, OSS first transformed the latitude and longitude satellite geo information into the Earth Centered Earth Fixed (ECEF) coordinate system, and similarly transform the SIC dataset's observations from stereographic projection coordinates into ECEF coordinates. This transformation to the ECEF coordinate system allows for easy extraction of the nearest sea ice datapoint where the surface arclength can be accurately approximated as a distance in three dimensions. Ultimately, each sample in the dataset is a labeled pairing of a CrIS array of size (2223, 4, 30, 9) and an ATMS array of size (22, 12, 96), and the labels of one such sample obtained in the northern hemisphere in the native SIC projection coordinates is shown in Figure 19.

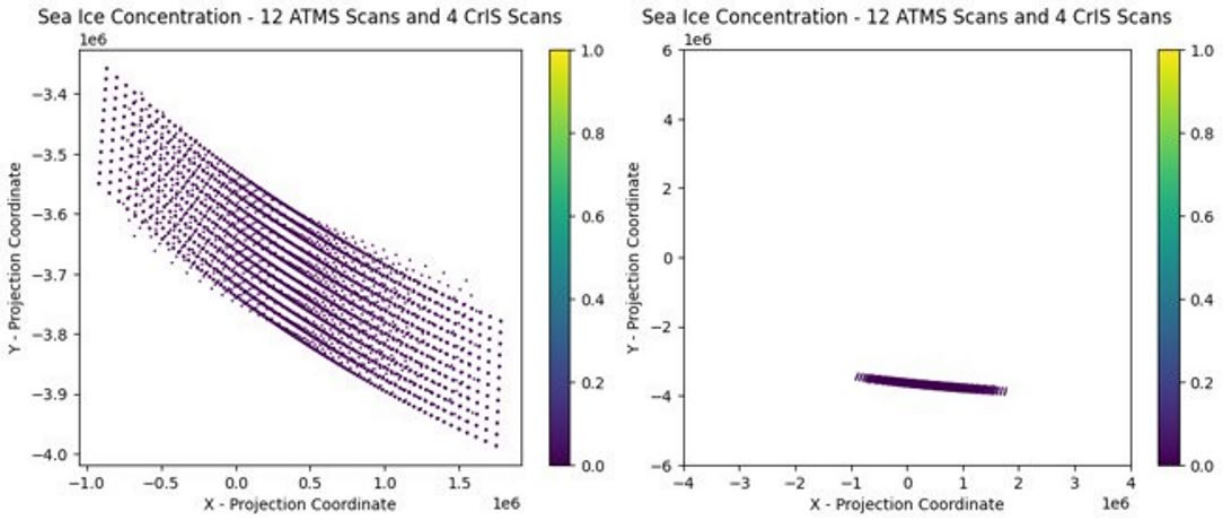

Figure 19. Example joining of ATMS and CrIS data, labeled by sea ice concentration and plotted in stereographic projection coordinates at two zoom levels.

To create a model that leverages both the ATMS and CrIS datasets with different sizes, OSS utilized an intermediate fusion model architecture. This architecture, show in Figure 20, used a collection of convolutional, normalization and activation layers on the CrIS and ATMS datapoints separately, followed by flattening and then a collection of fully connected layers. After each sample's size was reduced and made one dimensional, the ATMS and CrIS outputs were concatenated. OSS then applied another collection of fully connected layers whose goal is to learn the relationships between the two datasets. Finally, two fully connected layers separately resize the encoded information to match the SIC labels generated for each of the CrIS and ATMS instruments. This architecture allows for simultaneous leverage of the information within both datasets and obtain estimates at each of their coordinates.

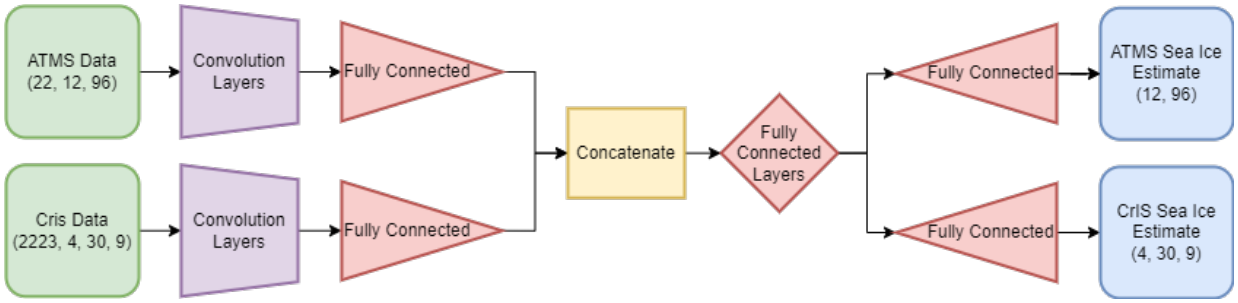

Figure 20. Overview of sea ice model design with full intermediate data fusion.

This data fused SIC model was developed in two phases. In the first phase, OSS explored the validity of such an architecture by training and testing the model on one days' worth of data. OSS additionally examined the impact of utilizing the DCT to reduce the dimensionality of CrIS and ATMS instead of convolutional layers. In the second phase, OSS trained and evaluated the Data Fusion model on a large collection of data using an on-prem server grade GPU.

3.5.1.1 SIC Model Development – Phase I

In the Phase I efforts, OSS trained and tested each model on one day's worth of data in the northern hemisphere. These experiments were performed on a laptop with 64 GB of RAM and an NVIDIA RTX A4500 Laptop GPU. Additionally, at this point in the development cycle, OSS considered using dimensionality reduction techniques to reduce the amount of data required, and as such experimented with using the Discrete Cosine Transform on ATMS and CrIS data.

OSS presents some of the results of three different initial models. All three of these models employ Architecture 2 as shown in Figure 20. The first model had minimal overall complexity, having 3 convolutional layers—each performing down sampling, and one fully connected layer in each block. The second model had more convolutional layers, without down sampling at every step, as well as more complex fully connected layers. And finally, the third model replaced each of the convolutional blocks with a Discrete Cosine Transform (DCT) that extracted the lowest frequency components; ATMS utilized 198 DCT coefficients, and CrIS utilized DCT 3960 coefficients, with significant down sampling along the channel dimension. Note that it is not computationally optimal to embed the DCT as a layer within the model architecture, as this requires computation of DCT coefficients at every pass through the model; this embedding was used only to make the implementation and direct comparison with the other two models easier. Figure 21, Figure 22, and Figure 23 show the results of these three different models. Note that within the plots showing the estimated sea ice concentration (on the left), the data does not cover the entire region of interest;

this is due to the requirement that each sample fall entirely within the domain of the ground truth dataset, a condition relaxed in our Phase II efforts.

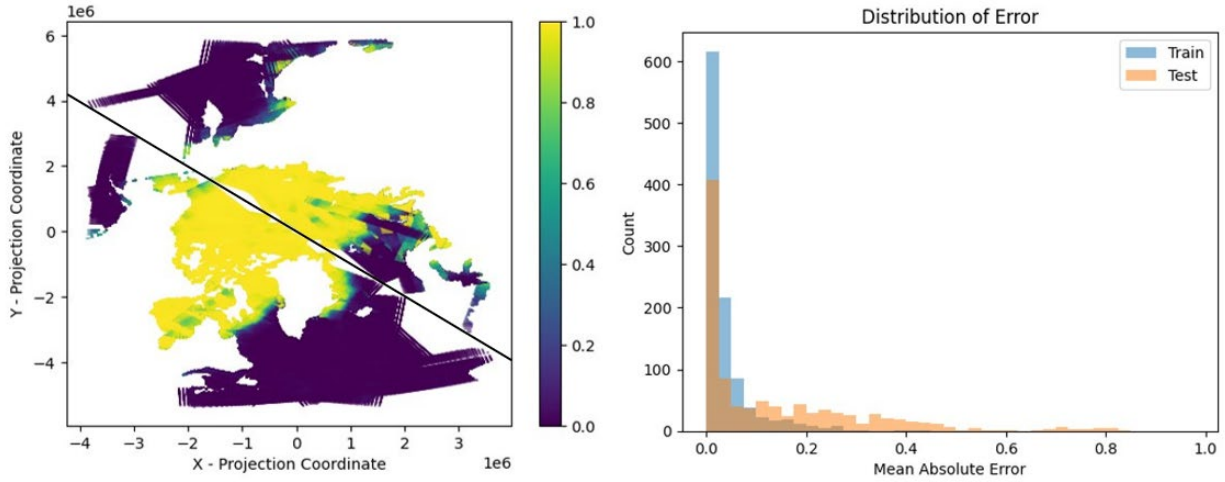

Figure 21. First model—lower complexity with convolutional layer. Left: Estimated sea ice concentration with training data below the black line, and testing data above the black line. Right: Distribution of mean absolute error for each sample; combined average between CrIS and ATMS error per each sample.

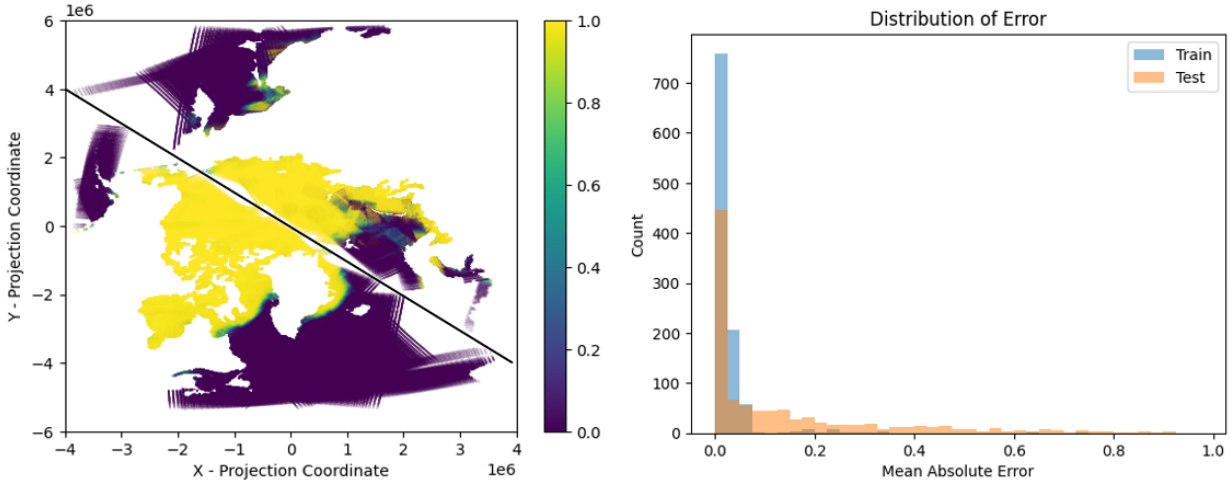

Figure 22. Second model—higher complexity with convolutional layers. Left: Estimated sea ice concentration with training data below the black line, and testing data above the black line. Right: Distribution of mean absolute error for each sample; combined average between CrIS and ATMS error per each sample.

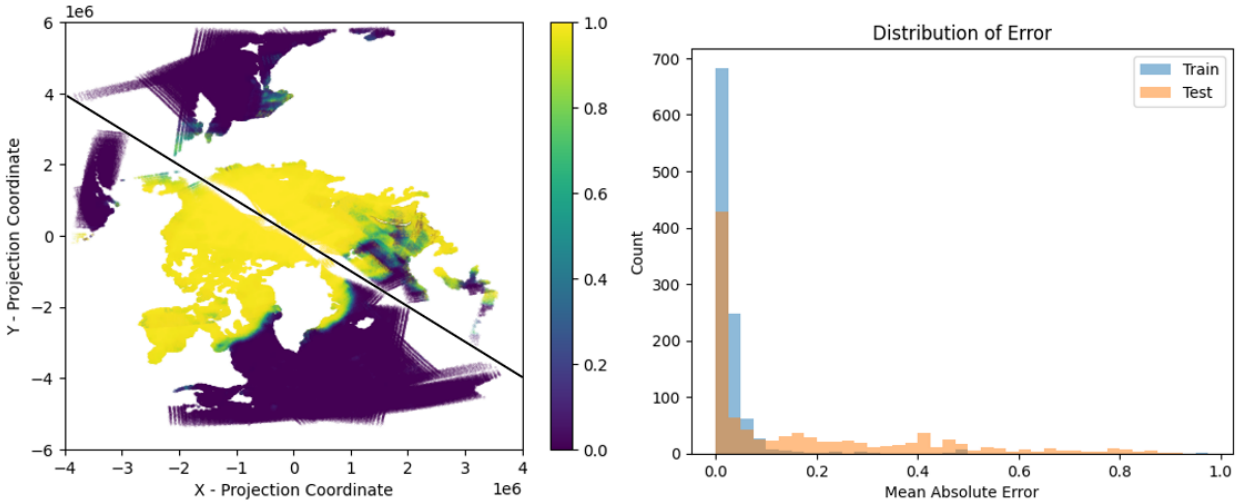

Figure 23. Third model—DCT instead of convolutional layers. Left: Estimated sea ice concentration with training data below the black line, and testing data above the black line. Right: Distribution of mean absolute error for each sample; combined average between CrIS and ATMS error per each sample.

From the distribution of error plots above it is apparent that all three models suffer from overfitting on the training set, as there is a strong right skew in the testing distributions that is not present in the training distributions. This overfitting can be mitigated by training over several days' worth of data, adjusting hyperparameters, or utilizing some form of early stopping criterion with a validation set.

While the results presented within Phase I are not comprehensive, they illustrate at least two points. First, a deep learning model can combine CrIS and ATMS data to estimate sea ice concentration, although more training data and better evaluation methods are required. Second, it might be feasible to significantly down sample using a lossy compression algorithm before training without a significant loss in performance.

3.5.1.2 SIC Model Development – Phase II

The second phase of SIC model development utilized an on-prem server to train and evaluate more sophisticated models. This machine has 1 TB of local storage, 1 TB of RAM, and two sever grade NVIDIA A30 Tensor Core GPUs with 24 GB throughput each. While the increase in computational resources allows training more robust models on a significant amount of data, there were several challenges encountered along the way.

To effectively make use of the available computing power, it is vital that all components of the ML development are effective. This requires a re-evaluation of the data downloading, data ingestion, model architecting, model evaluation, and model maintenance scripts and processes. Additionally, all of these considerations that are aimed at efficiently utilizing compute resources are crucial to managing cloud-based computation costs. While OSS has successfully addressed some of these challenges within this project, others still require attention and would benefit from continued iteration.

Data downloading and ingestion: As mentioned previously, the CrIS instrument produces about 24 GB worth of L1b SDR netcdf files per day. However, since only a portion of scans that fall

within the extent of the sea ice dataset are retained, only a total of about 9GB of data is required per day, including CrIS, ATMS and SIC labels. As such, OSS was able to process and store 60 days of data within year 2023 (5 from each month) resulting in about 500 GB of total data. Data from the northern hemisphere was included in the training set, and data in the southern hemisphere was used for the testing set. Figure 24 shows the distribution of sea ice concentration at each ATMS and CrIS observation. After downloading and processing, only the labeled data for every scan within the region of interest remains other than scan times and geo information such as Latitude and Longitude. However, the decision was made to leave this pairing to be performed at data ingestion. While this allowed for flexibility in implementation, it resulted in a computational overhead that is larger than initially expected. Due to the volume of data, this paring and loading process takes approximately 40 minutes for the entire 500 GB of input data.

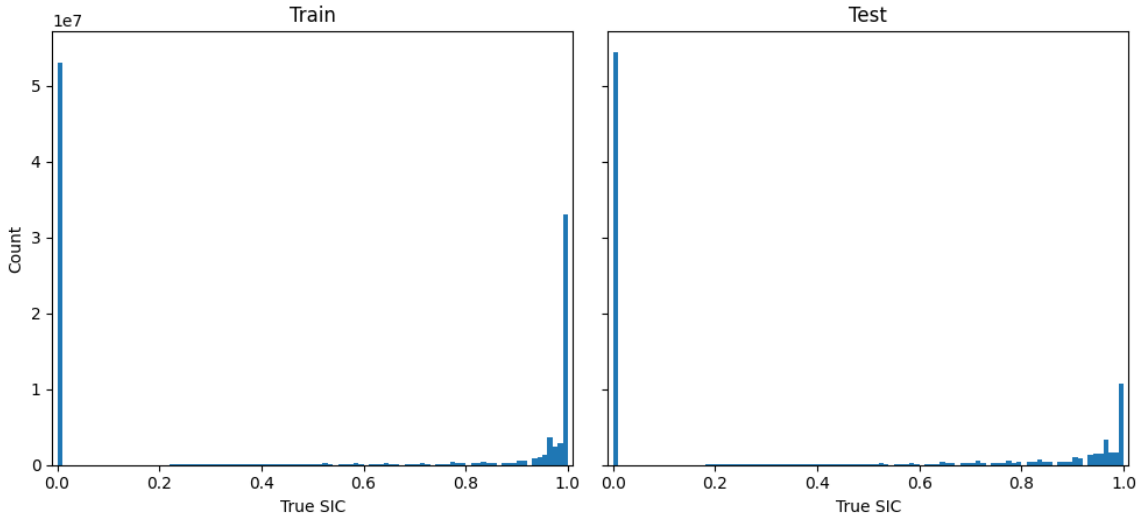

Figure 24. Count plot of true sea ice concentration, calculated over every pixel in every sample.

Model evaluation: Once a model is trained, it can be used to perform inference on the training and testing sets. When performing inference on the entire training and testing sets, only the necessary evaluation information such as the scan geographic information, the model outputs, and original SIC label values are retained, leaving nearly 20 GB of evaluation data. To efficiently calculate evaluation metrics over these evaluations sets, custom evaluation functions iterate over the datasets and calculate the necessary cumulative statistics to establish the desired metrics such as MAE or R2-score. While it was feasible in this case to load the entire 20 GB of evaluation data as a single tensor in RAM while working on the server and calculate the metrics directly, there were three main reasons it became necessary to write cumulative statistical algorithms. The first is that training and evaluating a model took place on the server, and the evaluation data was copied onto OSS scientists' machines to gain more insight in a Jupyter notebook environment. In this case, all of the evaluation data could not be loaded into their local RAM. The second is that OSS considered training multiple models simultaneously on the server with a significant quantity of training data, so even when computing evaluation metrics on that machine might lead to memory issues. Finally, computing these evaluation metrics cumulatively could potentially optimize the process to run in parallel. These concerns highlight the importance of implementing an efficient computational architecture for training and evaluation of models.

Model training: In phase II, OSS established more robust training algorithms. PyTorch as the OSS ML framework. While PyTorch is highly customizable and easy to use, it does not have a built in robust training API, as is provided by other frameworks such as TensorFlow. So far, OSS have implemented the following features within model training scripts (the majority of which are handled by classes built into PyTorch):

- Adam Optimizer with an initial learning rate at 1e-4.
- Learning rate factor decay of 0.5 after 1000 to 2000 iterations without improvement, depending on batch size
- Batch size ranging from 64 to 256. To best make use of the GPUs, one should try to use the largest possible batch size; however, the literature is not conclusive about the optimal batch size for the optimum overall model accuracy and it can sometimes be benefited to use an adaptive batch size (Devarakonda, Naumov, & Garland, 2018).
- Mean Squared Error loss.
- Train for at most 15 epochs.
- Save training information to a file. This includes the learning rate and loss at each iteration.

For easier model training implementations, it is possible to utilize the Keras API (Chollet, 2015) with PyTorch as the backend. With these training specifications, each model takes roughly 20 minutes to train over one full epoch, resulting in a total training time of about 5 hours.

Model results: While OSS experimented with multiple architectures, reported here are the results of one model that performed particularly well. As a part of the upscaling process, OSS has written scripts to evaluate the fully trained models on all training and testing data. These outputted are then saved to a file along with the original SIC concentration labels. Finally, the OSS suite automatically generates a collection of metrics and plots to gain insight into the model performance. The following tables and figures display some of the outputs of these evaluation scripts performed on two models. These automatically generated reports allow insight into the model's performance and identify where there are areas for improvement within the model architecture, making hyperparameter tuning an easier process. To illustrate the value of obtaining these metrics, the results of two different models are shown. The first model, referred to as Architecture 1, did not use any data fusion at all and established SIC estimates for each instrument independently. The second, referred to as Architecture 2, utilized the data fusion architecture of Figure 20 above. The results of Architecture 1 give insight into how ATMS and CrIS would individually perform at estimating SIC. On the other hand, one would expect that the evaluation metrics of Architecture 2 are very similar for ATMS and CrIS as they each have access to the other's inputs as part of the concatenated and fused information at the central part of the architecture.

The overall evaluation metrics of each architecture is shown in Table 5 and Table 6 below. Focusing on the R2-score column of Table 5, one can see that ATMS is able to estimate SIC reasonable well, with an R2-score of 0.83 on the testing set, and 0.68 on the training set, while CrIS is no better than random guessing, with negative R2 scores on the training and testing sets. On the other hand, the R2 scores of Table 6 indicate that Architecture 2 is able to estimate SIC very accurately, achieving an R2 score of above 0.94 on the testing and training set. In either architecture, the training score is significantly larger then the testing score, indicating that the

model overfit to the training set, and does not optimally generalize to the testing set, suggesting that the models could benefit from further hyperparameter tuning or early stopping when training.

Architecture 1.	MAE	MSE	RMSE	R2 Score	Precision*	Recall*	Accuracy*	F1 Score*
ATMS – Train Set	0.1254	0.0380	0.1951	0.8335	0.9408	0.9338	0.9387	0.9373
ATMS – Test Set	0.1822	0.0601	0.2451	0.6862	0.8351	0.9337	0.9077	0.8817
CrIS – Train Set	0.3863	0.2387	0.4886	-0.0452	0.7979	0.2683	0.6059	0.4016
CrIS – Test Set	0.3788	0.2578	0.5077	-0.3411	0.3667	0.0260	0.6205	0.0487

Table 5. Architecture 1 evaluation metrics. *Classification metrics are calculated with an artificial cutoff in SIC at 0.5.

Architecture 2.	MAE	MSE	RMSE	R2 Score	Precision*	Recall*	Accuracy*	F1 Score*
ATMS – Train Set	0.0304	0.0055	0.0745	0.9757	0.9894	0.9743	0.9823	0.9818
ATMS – Test Set	0.0455	0.0104	0.1018	0.9458	0.9789	0.9439	0.9718	0.9611
CrIS – Train Set	0.0300	0.0054	0.0735	0.9764	0.9895	0.9746	0.9824	0.9820
CrIS – Test Set	0.0446	0.0100	0.0999	0.9480	0.9791	0.9468	0.9726	0.9626

Table 6. Architecture 2 evaluation metrics. *Classification metrics are calculated with an artificial cutoff in SIC at 0.5.

One might be inclined to believe that the superior performance of architecture 2 (which utilizes data fusion of ATMS and CrIS) over architecture 1 (which does not utilize data fusion) indicates that in this instance, utilizing ATMS in conjunction CrIS significantly increases the ability to estimate SIC. However, one cannot reach this conclusion for two reasons. First, without training every possible configuration of each model's hyperparameters—including number of layers, layer depth, activation functions, etc., one cannot be certain that another model configuration would not have performed better. Second, there are very clear indications our hyperparameter choices for architecture 1 are not optimal. For instance, the training loss curves on the left of Figure 29 indicate that the model of Architecture 1 had a large variability in training loss for each batch. Although this might be expected if there is no learnable information within the training set, the results of architecture 2 indicate that one would expect to see better performance from either the ATMS or CrIS model of architecture 1. Moreover, the Error distributions on the left of Figure 25 and Figure 26 indicate that the models within architecture 1 are severely underfitting to the training data. While it is important to limit the extent of overfitting for models in deployment, it is often good practice to explore what amount of complexity is needed within a model (or which choice of hyperparameters) will lead to overfitting on the training set, so that the upper limits of performance can be explored. In this case, architecture 1 should be retrained with more layers and larger depth in each layer.

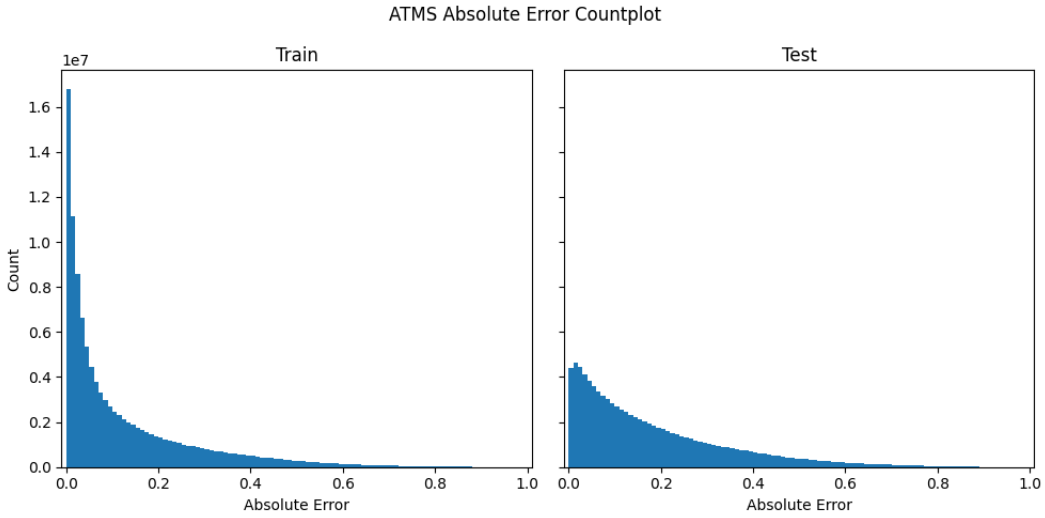

Figure 25. Distribution of absolute error for ATMS SIC predictions for architecture 1.

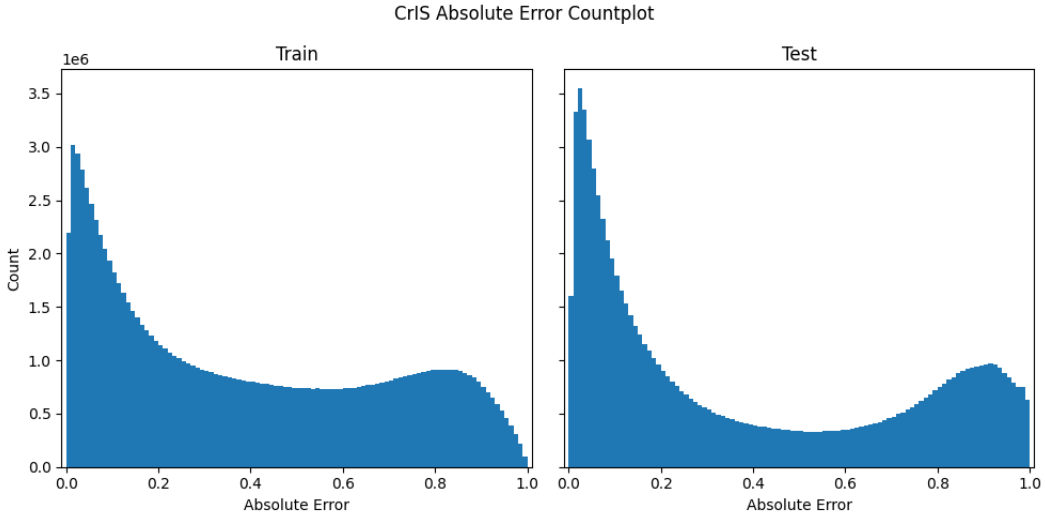

Figure 26. Distribution of absolute error for CrIS SIC predictions for architecture 1.

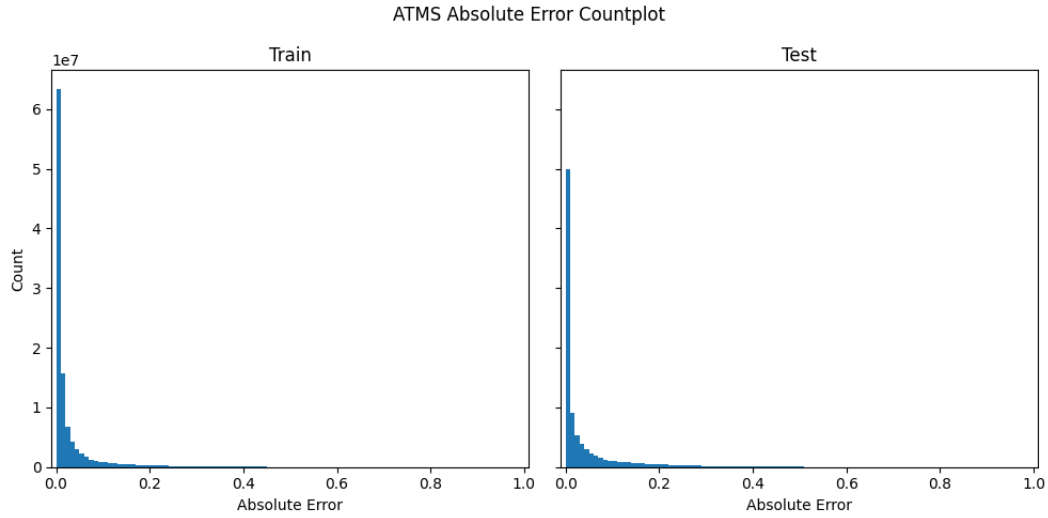

Figure 27. Distribution of absolute error for ATMS SIC predictions for architecture 2.

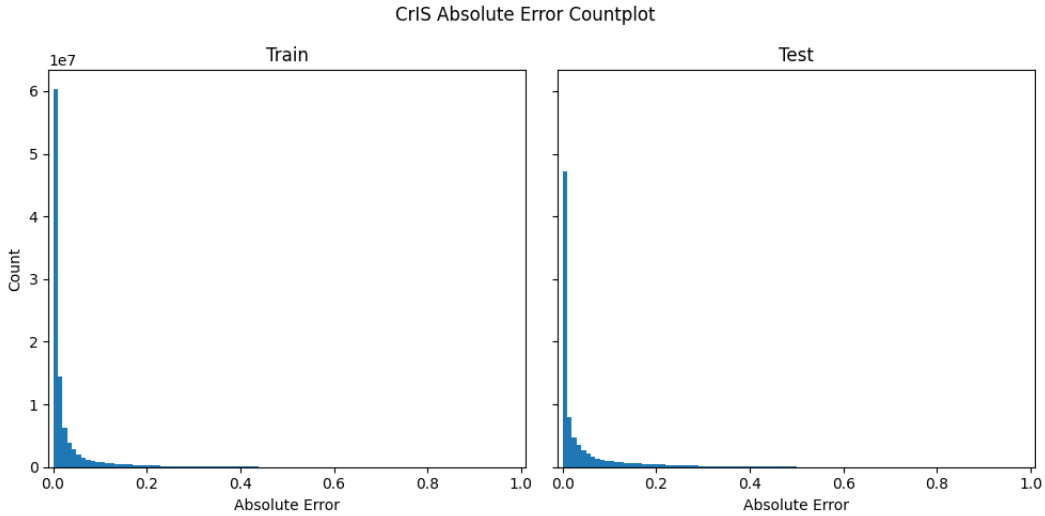

Figure 28. Distribution of absolute error for CrIS SIC predictions for architecture 2.

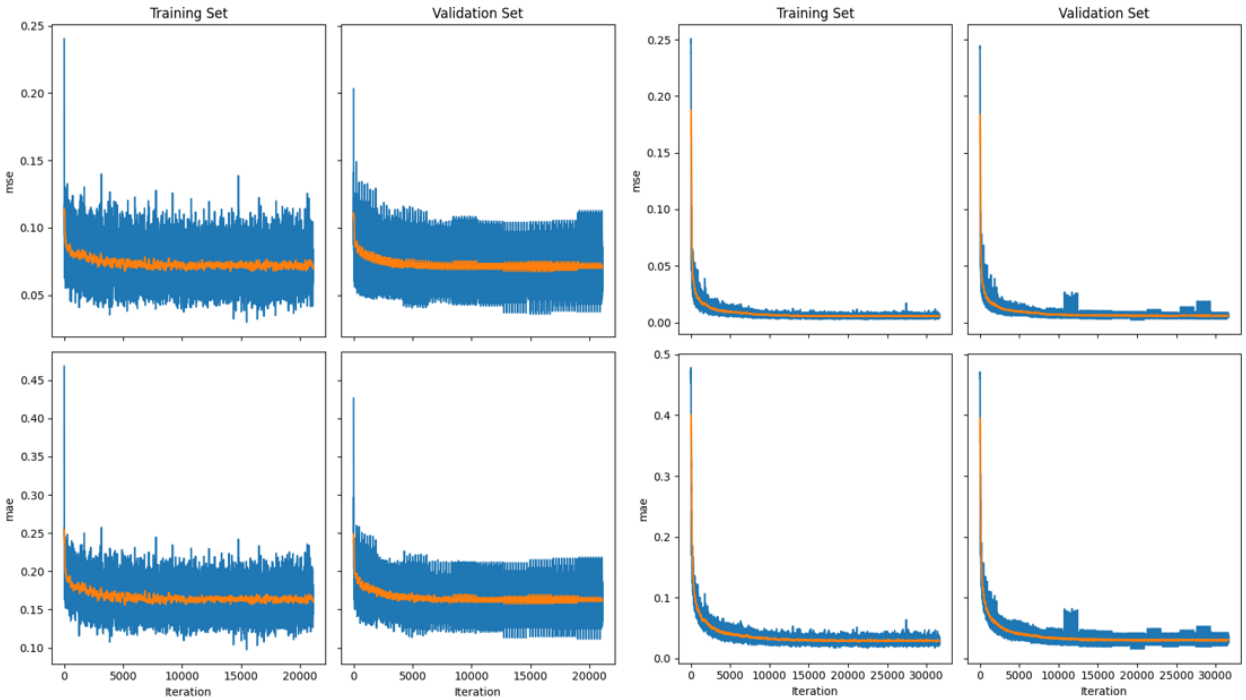

Figure 29. Training loss curves; orange line records the 100 point rolling mean. Left - architecture 1. Right – architecture 2. Top – Mean Squared Error. Bottom – Mean Absolute Error.

OSS had initially planned to extend on the results presented here by training more models with different hyperparameter configurations to test the limits of data fusion and to establish more scientifically grounded results. The goal was to assess the degree to which data fusion of ATMS and CrIS results in better estimates in SIC, and to understand when or why those improvements occur, if at all. Towards this end, OSS had planned to incorporate cloud coverage information to either mask out instances when clouds were presented in our dataset, or to estimate the effects of the presence of clouds on our estimation of SIC for the two different architectures. However, there were two limitations that hindered progress on this front. First, OSS scientists were not able to locate any publicly available CrIS cloud mask products, nor did OSS find any operational open-

source algorithms to collocate the VIIRS L2 cloud mask product with the CrIS observations. OSS ultimately decided to develop our own algorithm to collocate VIIRS pixels with CrIS pixels; however, the algorithm was prohibitively slow for this use, requiring several hours to process one day of ATMS and CrIS data over the polar regions. Second, due to internal IT processes at OSS, the EO-DT team lost access to our on-prem server and thus did not have access to sufficient computation resources. To overcome this, the team had planned to re-implement these models on cloud-based resources, but the process of reworking the code to function efficiently on the cloud took significant effort so the decision was made to focus efforts on other use cases. These challenges highlight the importance setting up sufficient ML Infrastructure and implementing Machine Learning Operations (MLOps) best practices in conjunction with the IT and other relevant teams. Validation and verification of these results will proceed beyond this project as we seek to publish these results in a peer-reviewed journal.

3.5.2 Retrieval Augmented Generation

Due to the popularity of large language models (LLMs) and the rapid development and deployment of LLM-based products, the team decided to explore their use in our EO-DT platform. LLMs are a form of generative artificial intelligence (GenAI) that are used to generate text-based outputs from user queries. This is accomplished with a self-attention mechanism which determines the relative relevance of other words in a sentence or paragraph to the current word being examined. During training, this self-attention mechanism learns relationships between words, which enables LLMs to gain an understanding of how language functions based on the semantic meaning of words. Although LLMs have a deep understanding of language, they have a limited understanding of very particular domains of expertise. That is, LLMs effectively follow the rules of grammar but may struggle to answer questions about El Nino. Unfortunately, LLMs always try to generate text in response to a question they are asked, even if they do not know the answer. This means that rather than simply state that they do not know the answer, they may provide a response that can be misleading, or entirely false. These types of responses are known as hallucinations. Addressing hallucinations requires that LLMs be provided with some amount of domain-specific understanding or knowledge.

While it is possible to train an LLM using this domain specific knowledge (a process known as fine-tuning) it can often be computationally expensive or otherwise impractical due to availability of models. Another common method to ground large language models with domain specific knowledge is known as Retrieval Augmented Generation (RAG). There are many variations of RAG, but at a high level, it is an algorithm to retrieve relevant information from a knowledge base, which contains the domain specific texts, and inject that text into the query that the LLM is given. OSS implemented a RAG based model that enables two features for users of the EO-DT platform. The first feature allows users to ask an LLM about a particular satellite, and the RAG model will retrieve relevant information about the satellite to base its response off. The goal of this RAG pattern is to provide users of the EO-DT platform with additional context when inquiring about satellites and the data that they gather. It is natural to embed this feature within an EO-DT platform so that users who might have limited scientific background associated with each of the available datasets can ask targeted questions about them. The second feature allows users to request a particular dataset to be visualized or described, in which case the RAG model can retrieve appropriate dataset tiles for the user to visualize within the platform, or provide a summary description of the dataset that is visualized. This feature provides for a more seamless visualization

experience, as one does not need to manually sift through all available datasets. Additionally, if this feature were to be expanded upon, it could provide users the ability to query about more complex operations on datasets, such as to visualize the difference of two forecasting model outputs from within an ensemble.

The two most popular python packages that are used to perform RAG are Llama Index (Liu J. , 2022) and LangChain (Chase, 2022). Both Llama Index and LangChain have a variety of integrations with vector database packages, knowledge graph packages, and cloud providers. However, Llama Index currently offers a larger variety of RAG patterns, including myriad advanced RAG patterns. Due to the larger variety of available RAG patterns and the ease with which Llama Index can be connected to RAG and LLM evaluation metrics, OSS chose to use Llama Index over LangChain for managing RAG patterns.

To implement this model, OSS utilized the llama-index python package and experimented with a variety of advanced RAG patterns. Llama Index provides integrations that allow the backend LLM to be almost any publicly available or self-hosted model, and for these experiments, OSS utilized OpenAI's GPT-4 as our primary model, Ada-02 for embedding knowledge docs, and GPT-3.5 to perform evaluation tasks.

At a high level, there are three required steps to implement a RAG model. First, a collection of documents needs to be gathered to create the knowledge base that contains information that the LLM should incorporate in their responses. Second, the knowledge base documents need to be chunked into nodes and stored in a way that allows for effective and efficient retrieval of information—this step includes vectorization of nodes or the graph-based configuration of nodes. Third, an overarching RAG pattern for retrieval and synthesization of node-based information is configured.

3.5.2.1 Knowledge Base Construction

To build the knowledge base that allows one to provide information about satellites, OSS gathered: NOAA User's Guides for ABI, ATMS, CrIS, and VIIRS in PDF format; Algorithmic Theoretical Basis Documents (ATBDs) for AIRS L2 data, ATMS L1b, CMIP, CrIS, and VIIRS in PDF format; and documentation about a variety of satellites from the World Meteorological Organization's (WMO) Observing Systems Capability Analysis and Review Tool (OSCAR) (World Meteorological Organization, n.d.) in JSON format. Before these data sources can be added to the knowledge base, they must be thoroughly cleaned and chunked into smaller pieces. The OSS data cleaning process for the ATBD and user guide pdf documents consisted of removing tables from the documents and omitting irrelevant pages including the table of contents, appendices, lists of acronyms, and other information which is not relevant to the satellite being discussed in the ATBD or user guide. Omitting irrelevant information is important because it makes the retrieval portion of RAG faster and more accurate, since RAG cannot accidentally match to a keyword to a chunk of text in an appendix which contains less relevant information than text earlier in the document. Tables were omitted from these documents because the tables in these documents either contain irrelevant information such as the revision history of the document or contain non-LLM readable text such checkmarks or "X"s to denote the presence or absence of data or sensors on a satellite. For cleaning the WMO data, a separate document needed to be created for each portion of the JSON element which corresponds to a single satellite. Since the information about each satellite

consists of key-value pairs, the portion of the JSON corresponding to each satellite was converted to a text string that was pre-defined to include the satellite's name, acronym, description, space agency, status, orbit, launch date, end of life date, longitude, altitude, ECT, data access link, a concise list of the names of its instruments, and the documentation link for that satellite. An additional document was created for each instrument which stores its name, acronym, instrument type, providing agency, and a link to documentation about the instrument.

To construct the knowledge base for the retrieval of datasets within the platform, OSS established a collection of example descriptors of available datasets, and their format to query from a PostgreSQL database that contains information about available tile sets to visualize, including the source, product ID, product name, url, start datetime, and end datetime for each dataset. After preliminary testing, the JSON file for the datasets was updated to include text strings providing a longer form of the start and end date and time as well as a description of the dataset product, its starting and ending time, and its url as a text string to facilitate easier searching over the datasets.

Each of these knowledge bases needed to be configured to use the Llama-Index native Document or TextNode classes. Once the Llama Index document class instances are defined, they need to be split into smaller pieces using a chunking process. Chunking is essential when using RAG because it is easier for the retrieval process to find matches between a user's query and small portions of text than it is to find matches between a user's query and several pages of text. There are a variety of different chunking strategies, each with pros and cons, so experimentation is required to determine the optimal chunking strategy for each use case. Several of the most common chunking strategies include token splitting, sentence splitting, and semantic splitting. In token splitting, the text is split on a predefined token such as the newline character, which can be useful for splitting the text into chunks based on its formatting. However, token splitting requires that all documents in the knowledge base can be split according to the same token and that the selected token will result in relatively uniform-length chunk, which is difficult to guarantee, especially with the variety of data sources considered for EO-DT. Sentence splitting is more robust than token splitting and splits the document at the end of sentences such that each chunk has the same predefined size and the same predefined overlap with its adjacent chunks. The downside of sentence splitting is that it does not consider the semantic meaning of the text, just the ends of sentences which means that sentences conveying similar ideas or about similar topics may be split into different chunks, making accurate retrieval more difficult. Semantic splitting, on the other hand, does consider the semantic meaning of sentences when deciding where to split the text. In the experimentation and prototyping for this project, OSS found that semantic chunking consistently yielded better results than this resulting from sentence splitting or token splitting. The chunks that result from chunking the larger documents are referred to as nodes.

3.5.2.2 RAG Pattern Overview

After the documents have been converted to nodes, the RAG pattern can be determined and set-up. The basic RAG patterns are full-text search, vector search, and graph RAG. Full-text search searches the knowledge base for matches to the user's query based on keywords. Vector search, on the other hand, uses an embedding model to produce a vector representation of the semantic meaning of the chunks in the knowledge base as well as the user's query. The idea behind vector search is that chunks in the knowledge base that are relevant to the user's query will have vectors that are close in distance and/or direction to the vector representing the user's query so the closest

knowledge base vectors to the user's query are returned from vector RAG. Graph RAG involves converting the knowledge base into a knowledge graph which captures both the existing information in the knowledge base within vertices and the relationships between chunks in the knowledge base, which can yield improved results. Graph RAG is most effective when there are clear entities, attributes, and relationships between the two contained within the data sources. Additionally, graph RAG produces the best results when the relationships between pieces of the data are at least as important as the pieces of data themselves.

Some more advanced RAG techniques include hybrid search, HyDE, hypothetical questions, sub-queries or multi-step queries, sentence-window retrieval, auto-merging retrieval, query routing, and agents or agentic systems. Hybrid search combines vector search and full-text search which allows retrieval to consider both keyword matching and matching by semantic meaning, which provides improved results. HyDE and hypothetical questions are twin methods for query transformation with HyDE using an LLM to generate a hypothetical response to a user's query and searching for matches to the hypothetical response. Hypothetical questions, on the other hand, uses an LLM to generate questions that each chunk in the knowledge base can answer and searches for matches between the questions and the user's query. The idea behind both of these methods is that it is easier to match responses than user queries to chunks in the knowledge base and it is easier to match user queries to questions than chunks, which provides better retrieved results. Sub-queries or multi-step queries are used to address complex user queries which require multiple steps to complete such as asking to compare or describe a large number of satellites or to perform data analysis. Since these complex user queries cannot be addressed directly, sub-queries or multi-step queries are used to break these complex user queries into a sequence of simpler steps, perform the steps, and synthesize the results into a final response. Sentence-window retrieval and auto-merging retrieval both help when text chunks come from long documents since they enable searching across small chunks of text, then replace the small chunk of text with a larger chunk of text which contains more context during retrieval. In sentence-window retrieval, each sentence in the document is embedded separately and a window of 5 sentences around and including the embedded sentence are stored as metadata for that sentence and the window replaces the sentence at retrieval. Similarly, in auto-merging retrieval, the document is decomposed into child chunks which reference larger parent chunks and if greater than k of the n total retrieved chunks reference the same parent chunk, all the child chunks referencing that parent chunk are replaced with the parent chunk. Both these methods allow the LLM to see more context than it would have seen previously, which enables closer matching within long-form text while preserving the context included in the long-form text. Query routing is useful when there are disparate sources of information, such as in our EO-DT set-up, or when different RAG patterns are best suited to different sources of information in the knowledge base such as when the knowledge base contains both long-form text and short pieces of text. During query routing, the LLM selects one or more RAG patterns or knowledge sources to be used to answer the user's question and the results are combined to form a single context for the LLM during response generation. Finally, agents and agentic systems are useful when the LLM should be able to perform a variety of tasks since agents can autonomously decide which other agent(s) should address the user's query and how to handle each step of a complex query to produce a final answer, which is a more complex and autonomous version of query routing.

3.5.2.3 Selected RAG Pattern

When developing the RAG pattern for this prototype, OSS performed extensive experimentation to determine the optimal RAG pattern for this our case. Since the knowledge base includes long-form ATBD and user guide documents as well as short text pieces from the WMO and the datasets for the visualization studio, OSS needed to leverage different RAG patterns to search these different knowledge sources effectively. For the long-form ATBD and user guide documents, both sentence-window retrieval and auto-merging retrieval were considered, and it was determined that auto-merging retrieval provided slightly better results with a slightly shorter runtime. When searching over the WMO satellite information, OSS considered full-text search, vector search, hybrid search, and HyDE/hypothetical questions. Similarly, OSS considered full-text search, vector search, hybrid search, and HyDE/hypothetical questions for searching over the datasets for the EO-DT visualization studio. For searching over both the WMO satellite information and the datasets, OSS found that hybrid search and HyDE/hypothetical questions significantly outperformed both full-text search and vector search. Unfortunately, due to conflicts between the embeddings used for hybrid search and the embeddings used for HyDE/hypothetical questions, only one of these methods can be used per knowledge source. Note that the embedding conflict occurs because hybrid search embeds the semantic meaning of each chunk in the knowledge base and HyDE/hypothetical questions re-embeds the chunks in the knowledge base which means there are two matching processes occurring in two different vector spaces which cannot be easily reconciled to rerank the final results by similarity. After comparing the performance of HyDE/hypothetical questions and hybrid search on both the WMO satellite information and the datasets, OSS found that hybrid search outperformed HyDE/hypothetical questions in both knowledge sources.

Additionally, OSS compared the aforementioned vector RAG patterns to graph RAG since the WMO data was provided in JSON form which has clear entities, attributes, and relationships between entities and attributes. Since there are clear relationships between satellites and their instruments in the WMO data, the ATBDs and user guides provide additional information about satellites and instruments contained within the WMO data, and datasets are collected by satellites and their instruments, the relationships between these entities provide at least as much information as the entities themselves. In fact, due to the clear relationships between satellites, instruments, satellite/instrument documentation, and EO-DT datasets, one can imagine an idealized graph structure based on these data, as shown in Figure 30.

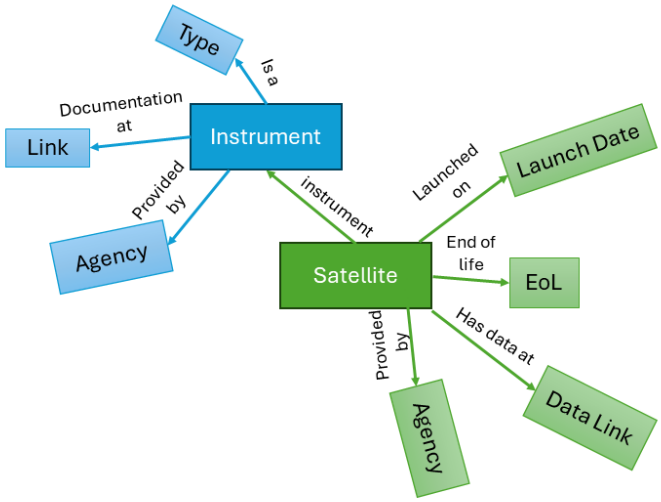

Figure 30. A diagram of the idealized graph structure for the EO-DT knowledge base.

Notice that this idealized graph structure contains the relationships between satellites, instruments, and datasets that NOAA scientists would likely generate if asked to replicate this task by hand. Observe that there are two primary methods used to generate knowledge graphs from the information in a knowledge base: using LLMs and using graph databases. When using an LLM, the LLM extracts entities, attributes, and their relationships from the data in the knowledge base and uses these extracted entities, attributes, and relationships to generate the structure of the knowledge graph. On the other hand, a graph database can be constructed from the data in the knowledge base using a pre-defined schema. This schema includes the names of each of the entities and attributes in the knowledge base data along with the types of relationships that exist between them. There are tradeoffs to using both these methods since LLMs can generate the knowledge graph much faster than graph databases can be constructed but graph databases enable more control over the structure of the knowledge graph, tend to provide improved graph RAG results, and eliminate the potential for the LLM to focus on entities, attributes, or relationships that are not important to the use case(s). Due to the preference for breadth over depth in our EO-DT, OSS decided to use GPT-4 to extract the entities, attributes, and their relationships from the knowledge base and generate the graph structure from these extracted values due to the reduction in time needed to generate the knowledge graph. The LLM-generated knowledge graph for the EO-DT knowledge base data is shown in Figure 31.

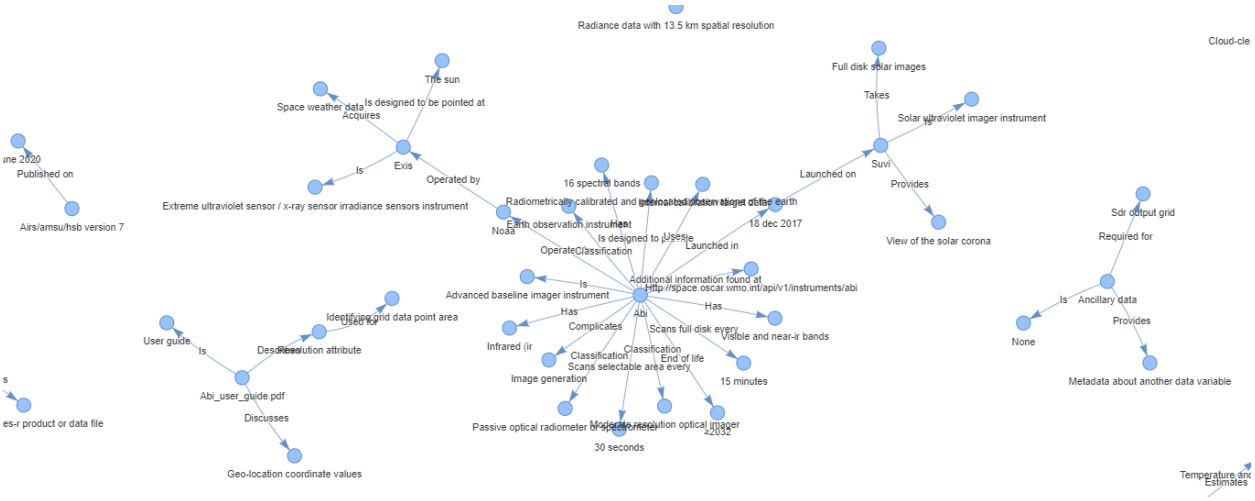

Figure 31. A screenshot of the EO-DT knowledge graph generated by GPT-4 which includes one satellite, ABI, whose attributes and relationships align with the idealized graph structure shown previously. Notice however that the majority of satellites, instruments, and datasets were not included in this knowledge graph.

While a portion of this knowledge graph matches our idealized graph structure, the majority of the satellites, instruments, and datasets were not included in the knowledge graph. Consequently, the LLM responses resulting from the use of graph RAG lacked detail, were extremely short, and frequently contained hallucinations. This brief case study into graph RAG for EO-DT highlights the need for graph databases during knowledge graph construction since this allows customization of the graph structure based on the use cases and will improve the LLM results.

Due to the limitations of graph RAG, OSS returned to vector RAG and advanced vector RAG methods when selecting our optimal RAG pattern for the EO-DT LLM use cases. Since it is desirable to use auto-merging retrieval to search over the ATBD and user guide documents and hybrid search to search over the WMO satellite information and the datasets for the visualization studio, one needs to use either query routing or agents to coordinate retrieval across these RAG patterns and knowledge sources. Agents take longer to initialize than query routing and OSS saw no clear benefit when using an agentic system compared to query routing, hence OSS used query routing for EO-DT due to our focus on breadth over depth. It is worth noting that agents and agentic systems are particularly important when a large number of RAG patterns or knowledge sources are under consideration or when agents need to be able to call existing functions in order to perform tasks. Hence, for EO-DT, OSS used query routing to coordinate using auto-merging retrieval over the ATBD and user guide documents, hybrid search over the WMO satellite information, and hybrid search over the datasets for the visualization studio as our final RAG pattern. OSS then developed a simple chat interface within the EO-DT frontend to allow users to query against this RAG pattern.

3.5.2.4 Evaluation Methods

Although using RAG substantially decreases the risk of LLM hallucinations, this risk can never be eliminated. Therefore, the development of evaluation metrics for LLM and RAG responses has become a crucial area of study. Unfortunately, since LLMs are a rapidly developing field, LLM evaluation metrics are an active field of research and there is no community consensus on the best evaluation metrics for the various LLM use cases.

OSS explored the use of multiple evaluation metrics for our implemented RAG patterns. Our initial efforts focused in on two elements of DeepEval's LLM evaluation metrics Python package: faithfulness/groundedness and hallucination. The faithfulness/groundedness metric measures the alignment between the LLM's response and the retrieved context from the RAG pattern. Similarly, the hallucination metric measures the likelihood that the LLM response contains one or more hallucinations. Each of these evaluation metrics provide a numeric score between 0 and 1 using a cosine similarity measure to determine the alignment between the LLM's response and the retrieved context from the RAG pattern. Additionally, these metrics provide a text-based justification for the numeric score. When implementing these evaluation metrics, a second LLM is used to evaluate the response from the first LLM according to a collection of example responses and their expected scores. For the faithfulness/groundedness metric, a perfect score is 1 which indicates that the LLM's response is perfectly aligned with the retrieved context from the RAG pattern. On the other hand, the perfect score for the hallucination metric is 0 which indicates that the LLM's response does not contain any hallucinated content.

After implementing both the faithfulness/groundedness and hallucination metrics from DeepEval, OSS explored the use of rubric-based LLM evaluations which provide a more comprehensive assessment of LLM response quality. For rubric-based LLM evaluation, a second LLM is used to evaluate the response from the original LLM according to a set of criteria. These criteria are defined in a text-based rubric which defines the features of an LLM's response across each of the criteria which would merit the LLM response being assigned a specific numeric score. Notice that rubric-based evaluation allows greater flexibility when evaluating LLM responses in a specific domain because we can define the facets of the LLM response that we care most about and have the evaluator LLM consider only those facets when providing its evaluation. Additionally, rubric-based evaluation provides greater control over how each criterion is assessed numerically by defining what aspects of that criteria merit each score value. Furthermore, the score ranges themselves can be changed to enable coarser or finer-grained evaluation depending on the use case.

For all of the evaluation methods, OSS used GPT-3.5-turbo to evaluate the responses from GPT-4 according to both completeness and accuracy criteria. GPT-3.5-turbo was used to provide a numeric score between 0 and 5 along with a text-based justification for its score. Observe that 5 is the best possible numeric score and 0 is the worst possible numeric score. When defining the rubric used for these evaluations, we provide GPT-3.5-turbo with a single example user prompt and LLM response along with the score for that response. Additionally, we describe what aspects of both completeness and accuracy would correspond to each of the scores between 0 and 5 and then ask the LLM to evaluate a new response based on the rubric and the prompt used to generate the response. Upon implementing the rubric-based evaluation metric, OSS found that the numeric scores and their justifications were more detailed than when using DeepEval metrics and gave a better indication of response quality in both use cases for EO-DT. Both the numeric score and its justification resulting from rubric-based LLM evaluation are displayed to the user for each LLM response using the EO-DT LLM frontend GUI, as shown in Figure 32.

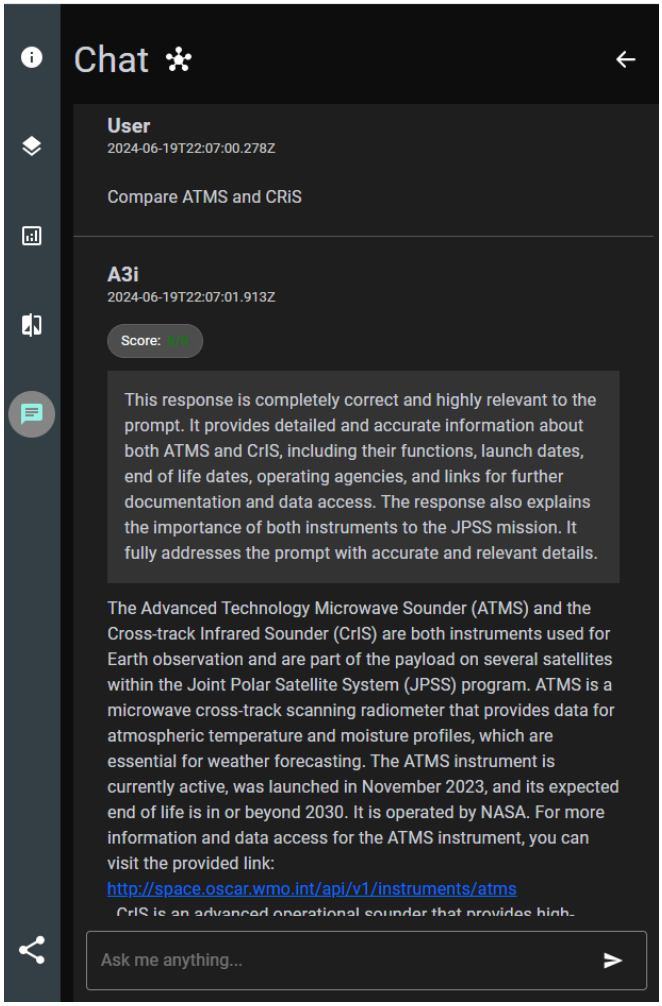

Figure 32. A screenshot of the LLM GUI showing a user query, compare ATMS and CRIS, the LLM's response, and the LLM response evaluation including the numeric score out of 5, which is shown in green to indicate that the LLM response has high quality, and the text

When displaying the numeric score, its text is given a different color based on the score value where green text indicates a good score for the LLM response, orange indicates a mediocre score for the LLM response, and red indicates a low quality score.

3.5.3 Kolmogorov-Arnold Networks

Kolmogorov-Arnold Networks (KANs) are a novel form of explainable AI, that provide developers with direct access to the model's learned function (Liu, et al., 2024). KANs seek to learn a closed form mapping between in input and output variable by leveraging the Kolmogorov-Arnold Representation Theorem, which states that any multivariate continuous function can be written as a finite addition and composition of continuous univariate functions, as expressed in Equation 1. The goal of a KAN is to learn this closed form representation by approximating the function as a composition of splines, then by fitting each spline to a scaled and shifted version of a known function such as a power, exponential, or trigonometric function. Finally, the function can be written down explicitly.

$$f(x_1, x_2, \dots, x_n) = \sum_{q=0}^{2n} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right)$$

where $\phi_{q,p}: [0,1] \to \mathbb{R}$ and $\Phi_q: \mathbb{R} \to \mathbb{R}$.

Equation 1. Kolmogorov-Arnold Representation of the function f.

The KAN architecture design was inspired by neural networks and differs from them by changing how activation functions are treated. In a neural network, each layer learns a collection of weights which are applied linearly to the previous layer, and the subsequently passed through fixed nonlinear activation functions. On the other hand, KANs remove weights from the edges and replace them with continuous, univariate functions. These continuous, univariate functions are learned during training and symbolic Python packages are used to correlate the learned representations of these continuous, univariate functions with polynomial, exponential, or sinusoidal functions. This allows KANs to explain each level of transformation from the input data to the output in a closed-form by describing the composition of univariate functions applied to each input variable. The result a successfully trained KAN is an equation that shows how each input variable is transformed using a composition of continuous, univariate functions to produce the output variable.

The KAN training process requires additional steps to learn the optimal closed form equation that doesn't have unnecessary components with negligible effects. First, the number of layers and their depth are chosen, which decides the initial composition and functional structure. Similar to a neural network, a KAN can implement any depth and with of its intermediary layers. For instance, a model with layer widths of [2, 3, 1] indicates that we are representing the KAN function f with two input variables and one output variable in the form of Equation 2,

$$f(x_1, x_2) = \Phi_1 \left(\phi_{1,1}(\mathrm{x}_1) + \phi_{1,2}(\mathrm{x}_2) \right) + \Phi_2 \left(\phi_{2,1}(\mathrm{x}_1) + \phi_{2,2}(\mathrm{x}_2) \right) + \Phi_3 \left(\phi_{3,1}(\mathrm{x}_1) + \phi_{3,2}(\mathrm{x}_2) \right)$$

Equation 2. The [2, 3, 1] KAN representation structure.

where each of Φ_i and $\phi_{j,k}$ are splines, whose weights are learned from the input and output data. Then after an initial training phase, we might find that an individual function $\phi_{j,k}$ or an entire branch Φ_i makes a small or negligible contribution to the overall function f, in which case we would prune that function. Then, to ensure that the pruning step does not hinder the overall quality of the approximate function f we perform another training step. We repeat the process of training and pruning for a fixed number of steps. Then, finally each component function is approximated by a known function. We utilize the pykan python package to train our KANs, in which each of these steps are readily enabled.

We first explored the use of KANs as a potential method to explicitly extract cloud mask and sea ice concentration labels from CrIS L1b observations. In either case, we utilize a full day's worth of data over the polar regions. After exploring the use of KANs with CrIS observations, with limited success, we developed a KAN model to estimate SIC from ATMS inputs. For all of these architectures, we took one sample of data to consist of one pixel's worth of information. It is possible to implement Convolutional KANs (Bodner, Tepsich, Spolski, & Pourteau, 2024) to learn

explicit formulas to extract features from images using KAN based convolutional kernels, however we did not implement that architecture for our experiments.

Our initial testing indicated that it can often be difficult to successfully train a KAN if there are too many input (or output) features within the training dataset, as each of the training, pruning and symbolically fitting of the learned function take a significant amount of time, or it can require too much RAM. Additionally, if the intermediary layer sizes are chosen to be too large, then similar difficulties apply. Hence, when experimenting with CrIS radiances, we needed to either sub select from the 2223 available channels or perform a dimensionality reduction technique such as PCA.

We provide a brief description of each of our KAN experiments.

3.5.3.1 Cloud Masks from CrIS L1b Data

In Section 3.5.1 we discussed the potential to utilize a cloud mask to better understand the performance of our SIC models. We developed a custom collocation algorithm to collocate VIIRS and CrIS observations on board the NOAA20 satellite; however, our algorithm was too slow for practical use on large datasets. Rather than spending the time to optimize the collocation algorithm we decided to run our collocation script on a day of CrIS and VIIRS data to obtain a sample collection of Cloud mask data for the CrIS instrument. We created the CrIS cloud mask by calculating the percentage of VIIRS pixels within each CrIS FOR for which the VIIRS L2 CloudMaskBinary product detects a cloud. If we can successfully train a model to estimate a CrIS cloud mask from CrIS radiances, we would not need to rely on a collocation algorithm to create the cloud mask for us, dramatically speeding up the calculation. Additionally, by leveraging an explainable AI model such as a KAN, an accurate estimation of the cloud mask could provide scientific insight into the detection of clouds from the CrIS instrument.

To reduce the number of features within our dataset, we first selected the collection of channels within the short wave range of CrIS radiances, and subsequently performed PCA to extract the three most prevalent linear features from those channels. Next, we split the data into 80% training and 20% testing sets. Then to assess the potential ability of a model to identify clouds from CrIS data, we first trained a classical neural network on the dataset which achieved an 84% accuracy on the testing set. This indicated that it might be feasible for a KAN to extract some amount of information from our dataset. However, while our KAN was able to achieve this level of accuracy on the datasets during the training and pruning phases, we were not able to achieve the same level of accuracy after fitting symbolic functions to each of the univariate functions within the KAN. This is likely due to the complex nature of the learned univariate functions, which could not be easily represented by a simple function that is available within the library. The python library that we used to create the KANs currently only allows for the representation of the learned univariate functions with shifted and scaled versions simple base functions such as $x, x^2, e^x, \sin(x)$ or $\tan(x)$. It may be beneficial to utilize linear combinations of such functions, although one could of course represent almost any univariate function to arbitrary accuracy with a sufficiently high degree polynomial, in which case the formula loses its ability to be understood as easily.

3.5.3.2 Sea Ice Concentration from ATMS SDR

OSS had previously trained several models to estimate SIC from ATMS and CrIS radiances, and those models performed quite well—achieving accuracies above 97% on the a large testing set. After failing to effectively generate an equation to estimate a cloud mask from CrIS data, OSS decided to test the KAN architecture on a dataset that OSS was more confident we could find information within.

To prepare the dataset, OSS took one day of L1b ATMS brightness temperature data labeled with the SIC dataset from Section 3.5.1, and split it into an 80% training and 20% testing sets. For this experiment, OSS did not perform any form of normalization of the brightness temperatures, nor did we perform any feature selection. OSS trained several KAN models with this dataset, and for each one we performed 4 stages of training and pruning. OSS trained each KAN model with a batch size of 1024, and each training iteration utilized 1000 steps. We trained our model using the Binary Cross Entropy Loss with logits function—this loss function was used so that the KAN model did not need an additional sigmoid function to its' architecture but could train the model as if it had one.

We present the results of a few of the more interesting training runs. Within the results below, the variable x_i represents the brightness temperature along the i-th ATMS channel. Additionally, the sigmoid functions that are a part of Equation 3, Equation 4, and Equation 5 were not actually learned by the KAN, but are rather an artifact of our training loss function.

KAN initialized with width [22, 45, 1]: The first architecture we tried utilized a width as specified by the Kolmogorov-Arnold Representation Theorem. However, while the model was initialized with an intermediary depth of 45, none of these layers were actually necessary after pruning, and the final expression was—excluding the sigmoid function that is an artifact of our training loss function—a simple linear combination of the brightness temperature along the second channel, as shown in Equation 3.

$$sigmoid(0.119181x_2 - 24.701884)$$

Equation 3. Formula learned by a KAN to represent SIC from ATMS Brightness Temperature.

It is interesting that such a simple formula is able to classify the presence of larger than 50% concentration in sea ice with around 94% accuracy on our small dataset, as noted in Table 7.

	Accuracy	Precision	Recall
Train	0.9398	0.9735	0.8137
Test	0.9567	0.9975	0.9096

Table 7. SIC classification metrics for Equation 3, with a classification cutoff in SIC at 0.5.

Figure 33 shows the SIC estimated by Equation 3 as NOAA20 flies over the south pole on January 1st, 2024.

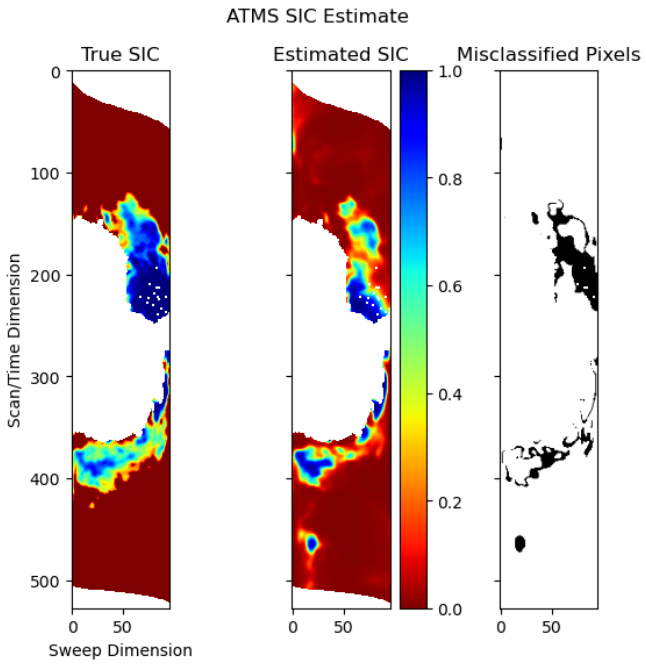

Figure 33, Example swath of ATMS SIC data over Antarctica. Left - True SIC labels. Middle - SIC estimated by Equation 3. Right – Pixels falsely classified with a classification cutoff in SIC at 0.5.

KAN initialized with width [22, 7, 1]: In this configuration, the KAN learned a function involving a term with the sin function, and utilizing three ATMS channels, as shown in Equation 4, and achieved a higher accuracy at 96%.

$$sigmoid(0.002781x_1 - 0.009665x_{17} + 5.148734\sin(0.014677x_1 + 0.00703x_2 - 5.16698) + 5.652251)$$

Equation 4: Formula learned by a KAN to represent SIC from ATMS Brightness Temperature.

	Accuracy	Precision	Recall
Train	0.9645	0.9410	0.9359
Test	0.9836	0.9884	0.9764

Table 8. SIC classification metrics for Equation 4, with a classification cutoff in SIC at 0.5.

However, Figure 34 shows that while Equation 4 achieves a higher classification accuracy than Equation 3, it is not actually able to effectively estimate the concentration of sea ice, as the expression predicts that there is around a .2 concentration level for the majority of pixels that actually have no sea ice at all. This demonstrates that it is important to utilize effective evaluation metrics during training, in our case OSS used classification metrics since the majority of SIC values are either 0 or 1, as shown in Figure 24; however, we should have focused on, or at least included, regression metrics such as mean squared error. Similarly, we should have utilized a regression-based loss function in place of BCE Loss, as BCE Loss is better suited for estimating probabilities in classification tasks.

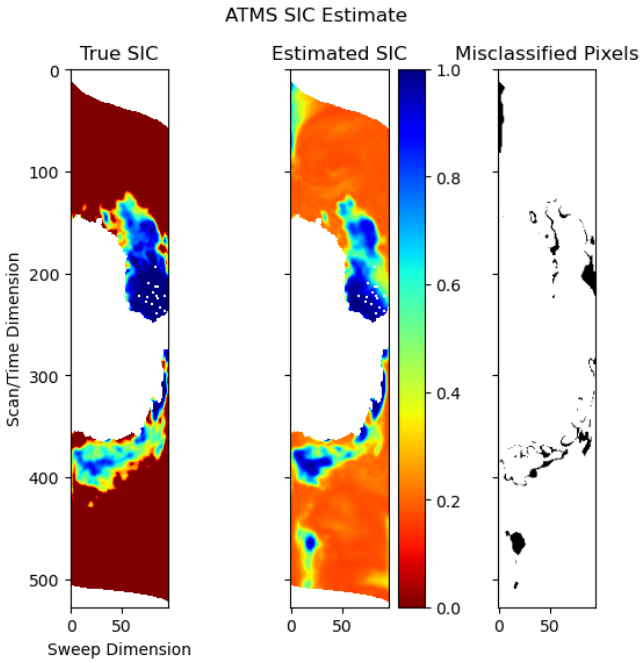

Figure 34. Example swath of ATMS SIC data over Antarctica. Left - True SIC labels. Middle - SIC estimated by Equation 4. Right – Pixels falsely classified with a classification cutoff in SIC at 0.5.

KAN initialized with width [22, 3, 3, 3, 1]: Finally, we present the results of the formula that achieved the highest accuracy of 97%. The learned formula is given by Equation 5, and is—ignoring the sigmoid—ultimately a simple affine combination of the 1st, 2nd, 16th, 17th, 18th, and 19th ATMS channels. It is interesting that the KAN learned such a simple representation considering that the depth of the layer would have allowed for up to four nested compositions of functions. This is an artifact model's pruning during training, and might point to something deeper about the SIC dataset that we used as labels.

$$sigmoid(0.084906x_1 - 0.07627x_{16} - 0.017016x_{17} + 0.000018x_{18} - 0.000025x_{19} + 0.09925x_2 - 14.35171)$$

Equation 5: Formula learned by a KAN to represent SIC from ATMS Brightness Temperature.

	Accuracy	Precision	Recall
Train	0.9763	0.9464	0.9829
Test	0.9758	0.9440	0.9832

Table 9. SIC classification metrics for Equation 5, with a classification cutoff in SIC at 0.5.

As pointed out earlier, we used the BCE Loss function, which is more suited for classification tasks. While the formula attains high classification accuracy, we can see from Figure 35 that the model was unable to effectively actual percentage concentration of sea ice, and more often than not, will estimate SIC to be very close to either 0 or 1. Additionally, it is not clear if the expression is only detecting sea ice, or if it is possibly detecting other features that are present, in Figure 35 we can see a small blip in the middle Estimate SIC plot that is not present within the True SIC plot on the left. It is likely that there is some additional feature the equation detects—possibly clouds contain ice particles. We note that some efforts were made to train a model using MSELoss and MSELoss with a sigmoid applied to model outputs, however those experiments did not successfully converge.

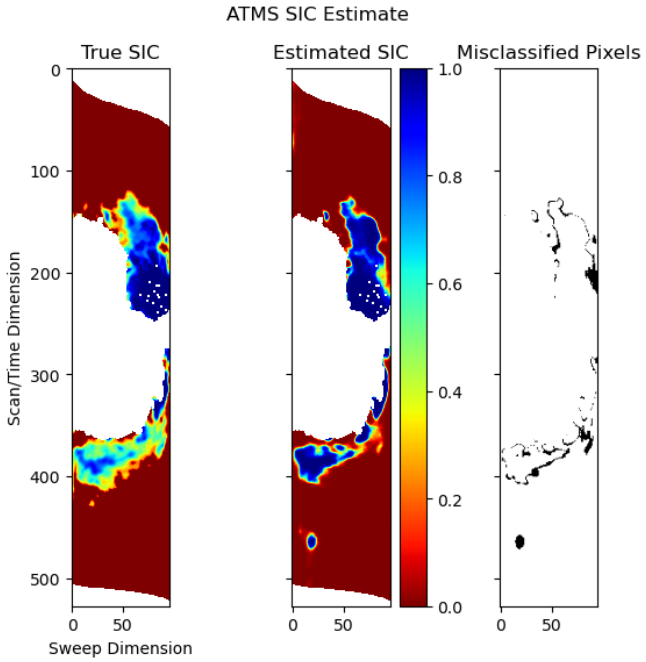

Figure 35. Example swath of ATMS SIC data over Antarctica. Left - True SIC labels. Middle - SIC estimated by Equation 5. Right – Pixels falsely classified with a classification cutoff in SIC at 0.5.

This experiment with KANs validates their ability to extract formulas that estimate geophysical parameters using low level satellite data. However, we can not be certain what information is actually being extracted from each of the learned formulas without a larger study and validation.

3.6 ML Infrastructure and MLOps

When developing a digital twin, it is important to consider to what extent the platform will aid in the development of Machine Learning models, both for internal developers and external developers. An EO-DT can significantly increase the speed of development for nearly every aspect of an ML model, including data preparation, exploratory data analysis, model training, evaluation, inference, and retraining. While the base EO-DT infrastructure required to support real time visualization of data will aid ML developers in downloading and visualizing their desired data sources, there is an additional infrastructural layer that is needed to support the other components of ML development. This additional infrastructural layer can enable ML development by implementing a robust ML based API, providing a suite of ML oriented tools, and implementing components of a Machine Learning Operations (MLOps) framework. There are several important factors to consider when developing this ML framework, including what range of development scales the tools will support, what range of users the toolkit is intended to support, and what computational resources will be used at each step.

3.6.1 Data Management

At the base level, an EO-DT platform will provide a suite of tools download data and process data. Additionally, it can provide a collection of tools and resources for users who are not experts on the datasets provided on the platform to understand and work with the various properties of each one. For instance, there could be a tool for each dataset to extract a binary mask from a collection of data quality flags indicating which observations are generally recommended for users to ignore or mask out. The primary goal should be to enable users of the platform to create ML ready datasets as quickly, and for this process to be easily reproducible. To accomplish this, OSS recommends creating a Python based data processing toolkit for users of the data that is available on the platform. The data processing toolkits should provide interfaces to easily and efficiently

- download data from specific satellites and instruments over specified time intervals, at varying product levels,
- process the data using common scientific processing techniques, such as extraction of principle components,
- concatenate data along the temporal axis for easier,
- record common statistics for datasets within the platform

We only recommend that these tools be created and provided when they are particular to the datasets that are available on the platform, and not where other open-source tools are available to complete the same tasks. By providing and maintaining these data processing tools for users, the users do not need to develop their own data processing scripts. This eliminates redundant effort, as multiple users might write a collection of scripts that are essentially equivalent but must be individually maintained. It enables users to easily reproduce each other's results and ensures the quality of the processing algorithms. Overall, it encourages users to use and interact with the data within the platform. In later sections, we provide a discussion of the OpenEO API, but we note here that part of implementing OpenEO is to enable some of these features on the server using a common API, and thus, these tools can be integrated with that specification.

Typical ML models expect a fixed input and output sample size, and there are a range of potential sample sizes that models can be built to accept. For instance, a standard Neural Network (NN) maps a vector of fixed length to another vector of fixed length (likely of different length), while a Convolutional Neural Network (CNN) can map an image of fixed size to another image of fixed size, where the input and output images can be of different sizes and can have any number of channels. Of course, there are a plethora of other ML models that can work with different data shapes. This variability in the expected data format for ML models—paired with the variability in the provided format of scientific data, makes the implementation of data preparation methods hard to generalize to all potential use cases. An attempt to complete this generalization would need to classify all ML input/output data formats and all data source formats.

There are additional data preparation techniques—such as data fusion, feature selection, collocation, and gap filling that add to this complexity. It is infeasible to support all potential methods, and instead, the platform should implement a collection of tools for common or popular use cases. For instance, it might be feasible to implement a general algorithm that can fuse image-based data from multiple instruments on a single platform of a satellite—and then perform the necessary steps to prepare this data for ML use. Such an algorithm would need to select ranges of data from appropriate time spans for each instrument simultaneously such that the geo spatial coverage of the instruments overlap. Given the variability of ML development, it is unclear at this point which additional advanced data processing features should be prioritized.

Our data preparation efforts for our Sea Ice Concentration ML models are a manifest example of the potential benefits of these data preparation tools being directly supplied to users by the platform. First, we implemented multiple architectures using similar datasets, and it is likely that different users of the platform will implement their own models with different architectures along the same dataset. Providing the data processing scripts seamlessly enables this capacity for users. Second, we sought an algorithm for collocating CrIS and VIIRS pixels, so that we could create a binary cloud mask for CrIS observations. This cloud mask would enable us to more readily explain the performance of our models with respect to the presence of clouds, and additionally to clarify the potential benefits of using a fused CrIS and ATMS dataset to estimate sea ice concentration—as opposed to only using one instrument. We were able to find two open-source implementations of such a VIIRS and CrIS collocation algorithm but could not get either implementation to work successfully with our data sets—either due to dataset versioning differences, or lack of documentation. By providing common data processing tools for users, the community can create open-source tools which directly build off these tools, and thus make the extension of community supported data processing algorithms more reliable. In the end we ended up deciding to develop our own implementation based off the algorithm of Wang, et al., (Wang, Tremblay, Zhang, & Han, 2016), but our implementation was not as computationally efficient, limiting our ability to effectively utilize it. Example outputs from our algorithm are visualized in Figure 36 and Figure 37.

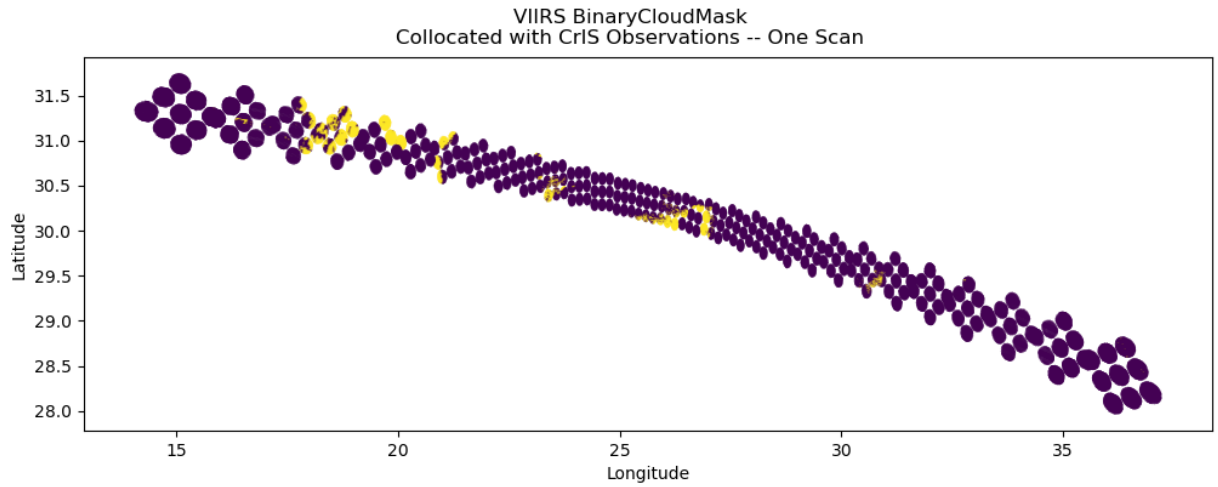

Figure 36. VIIRS BinaryCloudMask collocated with CrIS observations; one CrIS Scan. Yellow indicates the detection of a cloud; purple indicates that a cloud was not detected.

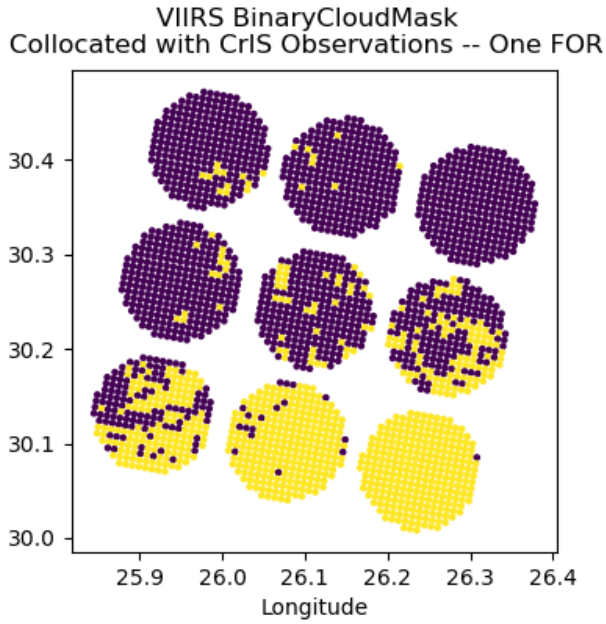

Figure 37. VIIRS BinaryCloudMask collocated with CrIS observations; one CrIS Field of Regard (FOR) at nadir. Yellow indicates the detection of a cloud; purple indicates that a cloud was not detected.

Data is at the foundation of Machine Learning and can often be the ML development task that requires the most effort. Establishing effective data management tools is paramount for both internal ML development and external ML development.

3.6.2 Model Training

The EO-DT platform can enable ML training for internal and external in two parallel ways, by establishing an API for model training, given an already established API for model ingestion, and by providing cloud based computational resources for users to pay for and train on. OpenEO is an open-source API "that connects clients like R, Python and JavaScript to big Earth observation cloud back-ends in a simple and unified way" (About OpenEO, 2024). OpenEO implements a robust framework for handling Earth Observation datasets in a common way using their data cubes,

and an experimental API for training a Random Forest classification or regression model; the API does not currently support any other ML models. This Random Forest API allows one to "fit" a model to input and output DataCubes and then to and "predict" target values from an input DataCube, effectively enabling a simple version of the Scikit learn API (API Reference, 2024). One could imagine extending this ML training API to enable the training of models within the Scikit learn package by adjusting their backend training algorithms to work with DataCubes. Similarly, one could implement an extension of the deep learning KERAS 3 API (Chollet, 2015) that enables the training of deep learning models on OpenEO's DataCubes. Combining these frameworks could potentially enable the training of a robust suite of ML models. Considering the difficulties associated with training large model on large datasets, we do not recommend that NOAA establish and provide users with server based protocols for large scale model training.

However, it will be essential for internal EO-DT developers to implement large scale training algorithms, to train models that they wish to be provided to users. Figure 38 outlines a possible cloud-based ML training workflow. This workflow assumes that training and testing data are too large to store locally, and that several models with different hyperparameter configurations will be trained simultaneously. In this case, it is important to be able to monitor training progress to stop models that are not training as expected, and to manage the costs of transferring data between different nodes.

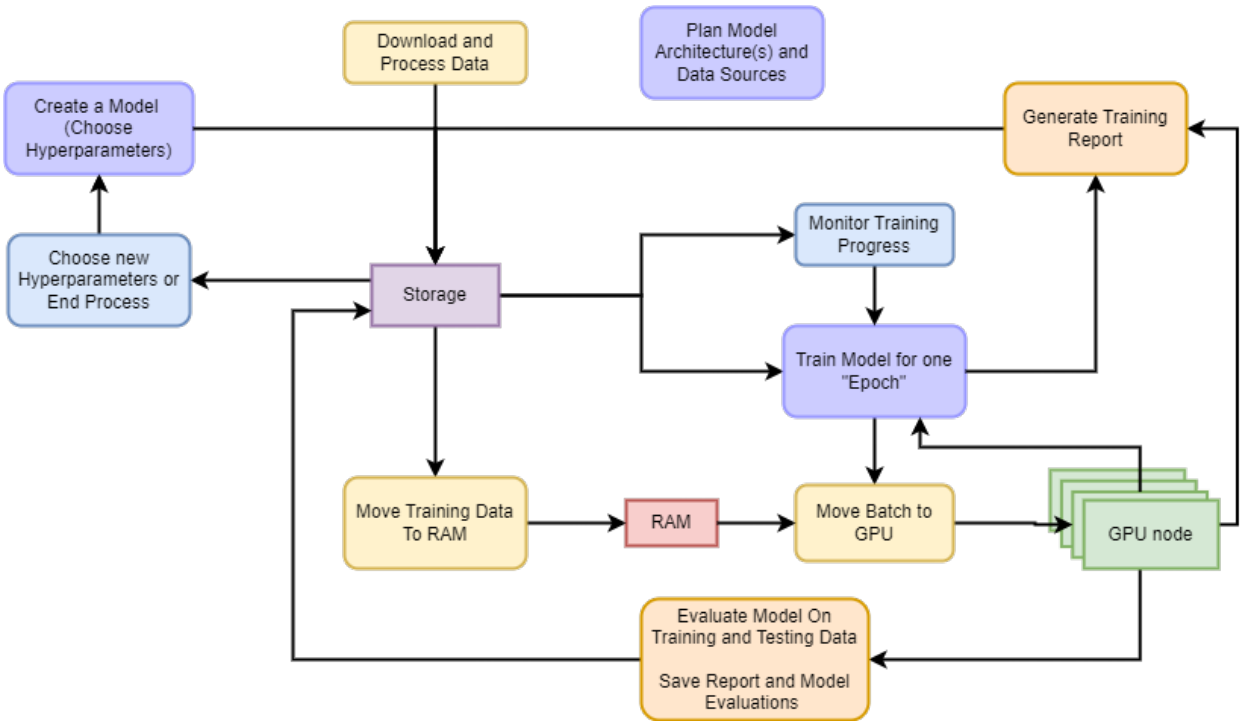

Figure 38. Example Cloud Training Workflow.

In simpler workflows, where the datasets and models are small, models can be trained on individual nodes. In more complex workflows, where the models themselves are prohibitively large or the datasets are enormous, one inference step may require multiple GPU nodes, a Message Passing Interface is required, and the cloud platform will need to enable these features. similar considerations need to be made when hosting models for inference. The majority of these

workflows benefit from some form of a cluster management system such as Kubernetes, or a platform like Parallel Works when the High-Performance Computing framework needs to be used.

3.6.3 MLOps Platform And Deep Learning Infrastructure

Our ML code was predominately written in PyTorch, using Jupyter Notebooks as our development environment. We either saved model weights as .pkl or .pt (PyTorch Pickle) files. Most training and evaluation occurred on an NVIDIA RTX A4500 Laptop GPU or when possible, on an on prem server with two NVIDIA A30 GPUs.

Depending on the anticipated scale of NOAA internal ML development, the adoption or creation of a Machine Learning Operations (MLOps) platform to easily develop and deploy models may be necessary. For smaller tasks with a limited number of developers it is feasible to utilize Jupyter notebooks to train and develop models on their own machines, and to subsequently implement by hand a limited number of models within data processing pipelines or at a REST endpoint for inference. However, when developing a large collection of models for multiple use cases, it is impractical to operate without an MLOps framework.

The goal of MLOps is to allow developers to reliably develop, monitor and deploy machine learning models within production environments. This requires that maintained and consistent methods for data collection, processing, labeling, training, deployment and monitoring are all established. To create an MLOps platform requires a significant amount of infrastructural development, and maintenance.

We recommend that NOAA implements a production ready MLOps platform for their model development, instead of developing one from the ground up. Depending on the development environment, we suggest using SageMaker for integrated AWS development, Azure ML for azure workflows or Kubeflow (Kubeflow) for an open-source self-managed tool that can deployed on prem or on the cloud.

We did not prioritize the deployment or use of such a platform for this project, as even deploying and managing one of these platforms can require substantial efforts. Instead, we highlight some of the core features enabled by the open-source Kubeflow toolkit.

Kubeflow is an ecosystem of Kubernetes components that support each component of ML development. Kubeflow integrates several tools in a microservice architecture, including a Dashboard for an integrated development experience, Pipelines for executing workflows, Notebooks for platform-based development, Katib integration for automated hyperparameter tuning, Kubeflow Training Operators(s) for large scale training jobs, KServe for serving models for inference. A major appeal of the Kubeflow platform is that it is built on top of Kubernetes, a “a portable, extensible, open source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation” (Kubernetes, 2023). Kubernetes is quickly becoming the platform of choice for many ML based operations, and is a great tool to manage the fluid computational demand of machine learning tasks. In Figure 39 below, we show how the Kubeflow components integrate to enable the Machine Learning workflow.

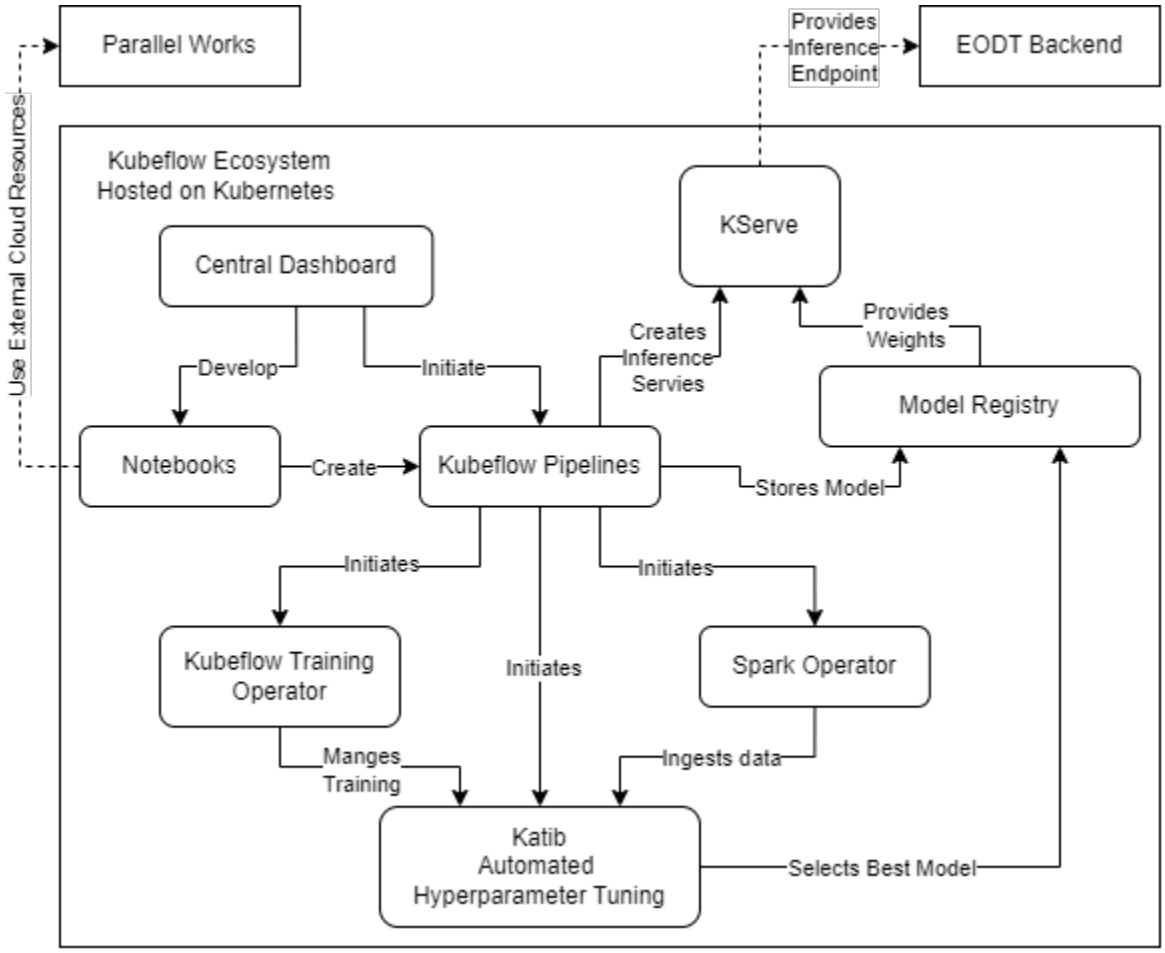

Figure 39. Kubeflow ecosystem integration diagram.

All of these features are a necessary component of ML development, and the continued development, iteration and maintenance of existing platforms should be leveraged by NOAA to speed their ML development process.

4 UI Visualization

Visualization of scientific data products is a cornerstone of the capabilities developed for the EO-DT program and allows for the rapid understanding of globally distributed, complex volumetric data for users of the system. Effective visualization requires a successful collaboration of data formats, transport technologies, and rending capabilities.

We established several goals for the program that we felt would effectively empower the data available from the sources. First, we chose to develop 3D/4D (3D + time) representations of most data shown in the program – allowing full volumetric display of data with ability to scroll through data at available times. The fusion of geospatial and time dynamic earth observation data would allow for an intuitive representation that can give additional insights to multiple dataset interactions. Second, we focused only on open-source, web-based visualization tools to maximize the reach of the visualization capabilities with no software to deploy to clients. We also utilized tools that were capable of GPU based processing for efficient interaction and manipulation of the data real-time – even on modest client machines. Finally, we implemented an efficient streaming and storage scheme for the data using cutting edge data formats so that servers could host a wide variety of data formats and types to many clients with reasonable compute and storage resource both on host and client.

4.1 Visualization Tool

Numerous tools were evaluated to meet the goals of the visualization. Many were quickly eliminated due to their lacking features or capabilities. It also became apparent that some of the data set sizes would require the ability to manipulate and style using the massively parallel processing of the GPU over CPU since the number of data points would greatly affect the performance for even modest data sets when running under CPU alone. 3D volumetric rendering further presented challenges due to the difficulty in showing layers of information that can mask one another – so tools and techniques were needed to highlight or isolate features of interest. Here we explore the findings we made leading to our implementation of the visualization engine.

4.1.1 Data Visualization Tool Selection

Early exploration of available tools for visualizations found numerous tools which could display volumetric data on a 3D view of the Earth. During our work, we evaluated a variety of tools and implemented tests to validate their capabilities and to experience the performance directly. These evaluations quickly led us to CesiumJS for the visualization and we will discuss the capabilities and features that led to the decision and some of the challenges with the other tools available.

4.1.1.1 NASA Web WorldWind

NASA's Web WorldWind (Brovelli, 2016) dynamic web component was first explored to evaluate the capabilities of showing global data in a browser friendly format. This browser component allows for interactive full globe viewing of satellite data that can be rotated and zoomed. Further, it does have the 3D visualization capability, showing data,

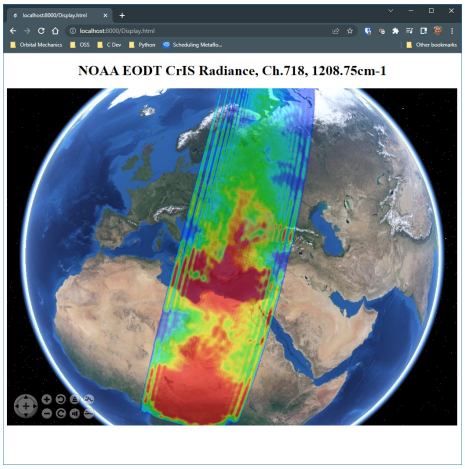

Figure 40. NASA's Web WorldWind Visualization of CrIS channel 1 data from Nov 15, 2022.

line/shapes/polygons in an WGS84 3D globe with realistic background imagery and terrain. Initial results seemed promising (See Figure 40), but after visualizing GFS (1°x1° grid) surface temperature data we encountered performance problems in the browser. This data set consists of approximately 65k data points in the surface layer, which is far smaller than some of the expected data formats. Significant slowdowns were experienced in the browser trying to display the data, increasing the framerate from >10 frames per second (FPS) to under 0.5 FPS. We knew we wanted to use the 0.25°x0.25° grid GFS data which would have just over 1M data points (for each layer) – and this would be untenable using the tool. Further, Web WorldWind seemed to lack advanced data ingest capabilities that would facilitate showing larger-than-memory data sets that could easily be obtained during some of the more advanced data processing situations. It was decided to explore other options that might be better suited for the use case.

4.1.1.2 Alternative Visualization Tools Explored

A survey of additional Earth visualization tools was performed and several candidates were identified, including Google Earth (https://earth.google.com) by Google, NullSchool (https://earth.nullschool.net) by Cameron Beccario, Windy (https://windy.com) by Ivo and the Windy Team, Fluid Earth (https://fluid-earth.byrd.osu.edu) by Ohio State University (Cervenec, 2018), and Polar Globe (https://cici.lab.asu.edu/polarglobe) by Arizona State University (Wenwen, Mar 2017).

Google Earth contains powerful data streaming and client visualization capability as well as a sophisticated 2D hierarchical gridding library (Google S2) but lacked the 3D volumetric display capability. NullSchool introduced us to an implementation of particle system visualizations for vector fields but was not extendable. Windy and Fluid Earth also use particle systems for display and both have sophisticated data layering and data combination capability, but again neither seems to support volumetric display capability. Polar Globe was the closest to the vision we had for visualization and showed some of the capability to do volumetric rendering of GFS style data with 25 layers of information, particle system display, and interpolation on zoom capabilities. However, limited source code was available for Polar Globe and it was not available for distribution or enhancement, nor did they seem to have the facility to do timeline views of the data.

4.1.1.3 CesiumJS

Many other visualization platforms rely on the CesiumJS engine which we explored in detail before choosing it as our visualization tool. CesiumJS is an open-source library originally developed by Analytical Graphics, Inc. for visualizing objects in space, but contains many of the features we were working towards. The CesiumJS team is also actively involved with the standards development community and participates in developing OGC standards. It has time selectable 3D volumetric rendering capabilities and the ability to stream massive data sets efficiently. Further, the layering, dynamic lighting, customizability, and terrain capabilities give it the ability to visualize and analyze data on a high-precision WGS84 globe. Because it implemented a full set of features, was built on in an open-source engine, and has adherence to 3D/4D geospatial standards, we chose CesiumJS as our visualization engine for the EO-DT program.

Figure 41. Minimal CesiumJS view showing Sentinel-2 cloud free basemap and time slider.

4.1.2 UI Visualization Data Layers and Techniques

There are many visualization features and base layers that contribute to the effectiveness of a visualization tool. We explore the various techniques and layering capabilities that are available to use for our EO-DT visual twin and how they may be used in other visualization tools to be developed. Base imagery layers serve as the visual context for the EO-DT platform. These layers are essential for providing users with a geographically accurate and intuitive reference, enabling effective interpretation and analysis of overlaid volumetric data. By integrating various types of imagery and reference data, our system offers a comprehensive platform for exploring and understanding NOAA's scientific data products.

4.1.2.1 Map and Satellite Imagery

Map and satellite imagery is a critical component of EO-DT, providing high-resolution, up to date representations of the Earth's surface. CesiumJS supports the layering of imagery (maps) from a variety of imagery providers including ArcGIS, OpenStreetMaps, BingMaps, Google Earth, custom imagery, and more. Most imagery providers use a REST interface over HTTP(S) to request tiles; however, the request schema may slightly vary depending on the provider. Additionally, the WebMapServiceImageryProvider and the WebMapTileServiceImageryProvider providers enable interfacing with Web Map Service (WMS) and OpenGIS Web Map Tile Service (WMTS) imagery

tiles, respectively. Both WMS and WMTS are OGC standards for requesting map tiles from geospatial databases.

The resolution of satellite imagery varies depending on the provider. The resolution for a given imagery layer may also vary depending on location. While some satellite imagery providers such as Mapbox can provide resolutions up to 5cm per pixel, other satellite imagery such as the Cesium Ion hosted Sentinel-2 imagery only has a 10m per pixel resolution. Similarly, the age of the satellite imagery depends on the provider.

4.1.2.2 Grid Lines

Grid lines serve as a geospatial reference system within the EO-DT platform, helping users orient themselves within the 3D environment. The primary lines indicate the prime meridian and the equator, while the secondary lines indicate longitude and latitude.

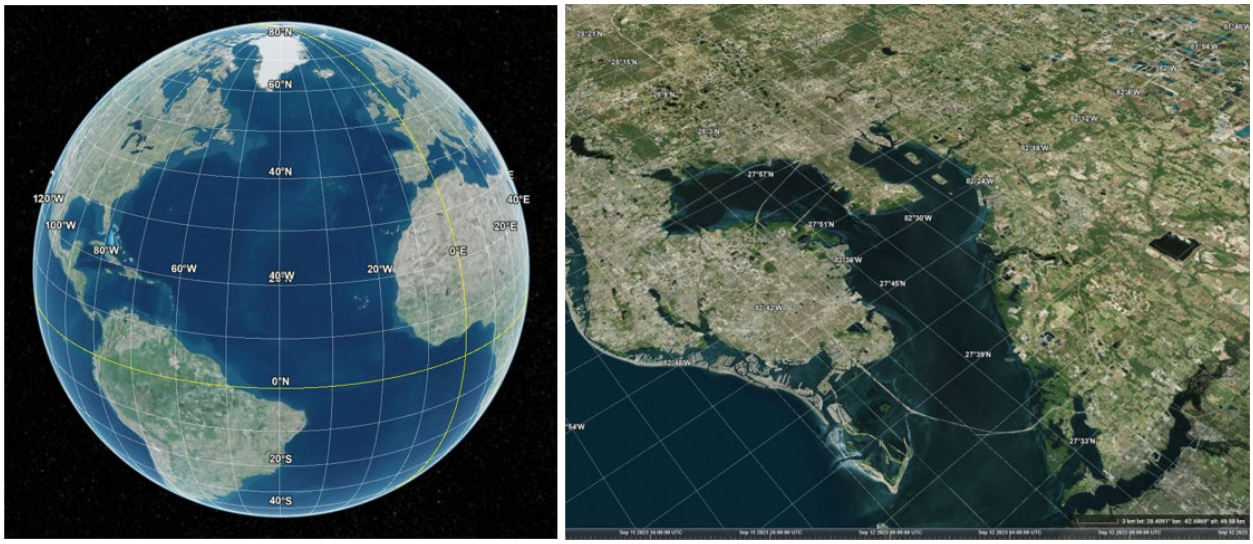

Figure 42. Dynamic grid lines.

The grid itself is dynamic to enhance the usability and functionality of the grid lines. As the user continues to zoom in, the grid intervals will continue to subdivide with updated longitude/latitude labels. This tool has proved to be an effective method for users to interact and understand the spatial characteristics of their data. We implemented dynamically generated grid lines that can be enabled on the EO-DT visualization engine.

4.1.2.3 Bathymetry

Bathymetric data represents the underwater topography of the Earth's surface, capturing the depth and shape of the sea floor and the sea volume above it. This layer is particularly important for NOAA's mission, as it enables the visualization of underwater features such as intertidal zones, tranches, ridges, and plateaus, which are essential for marine navigation, resource exploration, current dynamics, and environmental monitoring. In January 2024, Cesium released their Cesium World Bathymetry global tileset (Cesium, 2024). This 3D dataset fuses multiple open data sources into a single bathymetric and topographic global tileset, which includes land terrain for exploring coastal areas as well as GEBCO's global bathymetry dataset (GEBCO, 2024). To help visualize

and analyze the bathymetry, we have implemented a custom CesiumJS Material which uses the vertex normal from the terrain tiles in combination with custom shaders (GLSL) to dynamically represent the appearance of polygons, polylines, ellipsoids and other objects. Here we can see an example of our bathymetry globe material which indicates elevation and contour lines, which provides an additional level of visual insight to users. In a later section, we will discuss the other techniques we implemented for analyzing 3D terrain data including bathymetry.

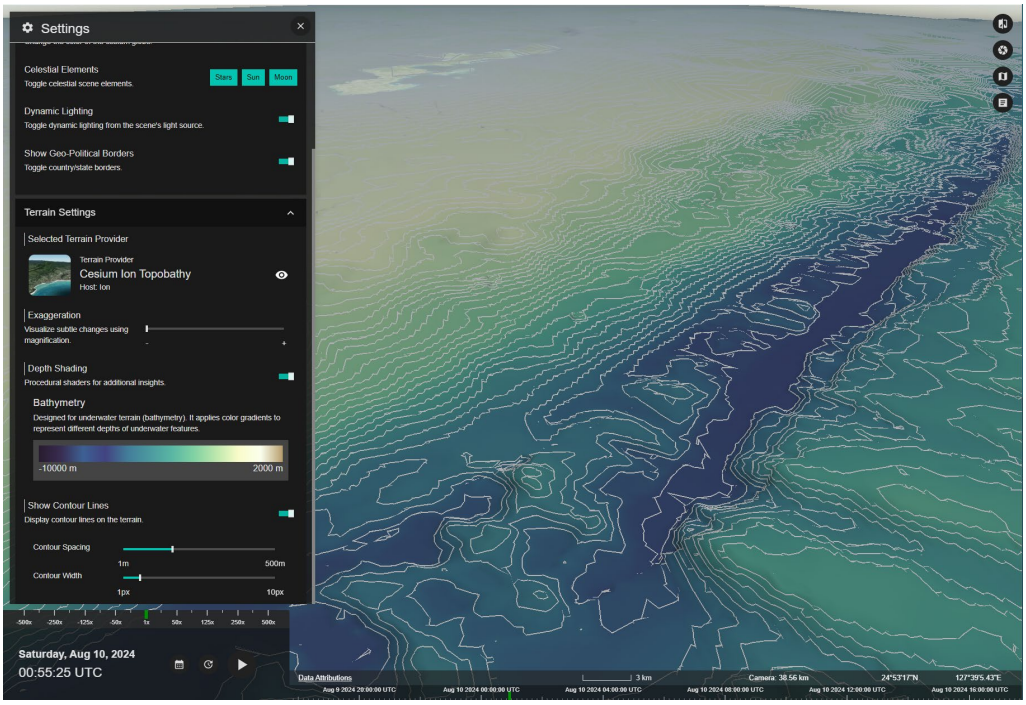

Figure 43. Bathymetry Analysis.

4.1.2.4 Administrative Borders

Administrative borders delineate the boundaries of geopolitical regions such as countries, states, and municipalities. These borders are crucial for contextualizing the data within a socio-political framework, allowing users to analyze how natural phenomena and environmental changes affect different regions and who responsible organizations may be for a given location. In the EO-DT platform, administrative borders are visualized alongside other data layers, providing a clear reference for assessing regional impacts. As with the grid lines, these borders are also dynamic and will progressively render more detailed sections of borders as the camera gets closer and with the ability to further divide regions from country, state, county, city style refinement.

Figure 44. Administrative borders in the Mediterranean.

4.1.2.5 Volumetric Vector Field Representations

Volumetric vector field representations are a powerful tool for visualizing complex, dynamic systems in the EO-DT platform. There are numerous methods to visualize vector fields depending on the intuitive nature of the physical quantity being displayed. For vector fields that represent motion such as wind speeds and vectors or ocean currents, a particle system can simulate blowing/floating particles in the field – following the vector field lines with appropriate speed and direction. We explored the ability to render particle systems using custom WebGL textures to provide better ways to analyze atmospheric and oceanographic phenomena. For vector fields that don't represent motion, such as magnetic fields or gradients in values, additional work could be performed to simulate using arrow arrays or other volumetric visuals. The implementation of these vector fields would likely be similar to the particle systems, e.g. using custom WebGL textures with differen texture coordinates to achieve the desired visual effect.

4.1.2.5.1 Particle Systems

In the EO-DT platform, particle systems were implemented using the low-level CesiumJS Renderer module. We start by creating a custom primitive class where we will customize key rendering procedures like the CesiumJS DrawCommand, ShaderProgram, FrameBuffers, and other WebGL related methods (Cozzi, 2016). All of the computation for a particle system occurs on the GPU, a process known as general purpose GPU (GPGPU) computation. This accelerated parallelization is what enables the browser to compute and render the position and movement of tens of thousands of individual particles simultaneously. In the EO-DT platform, particle systems were implemented using the Global Forecast System (GFS) U and V components of wind at varying altitudes, and Ocean Surface Current Analysis Real-Time (OSCAR) data for ocean currents. These systems allow the visualization of wind and water movement as dynamic particles that flow across the 3D environment, providing a visually engaging and informative representation of these dynamic systems. Particles move according to the direction and magnitude of the underlying data, creating a realistic simulation of natural processes such as wind patterns and ocean currents.

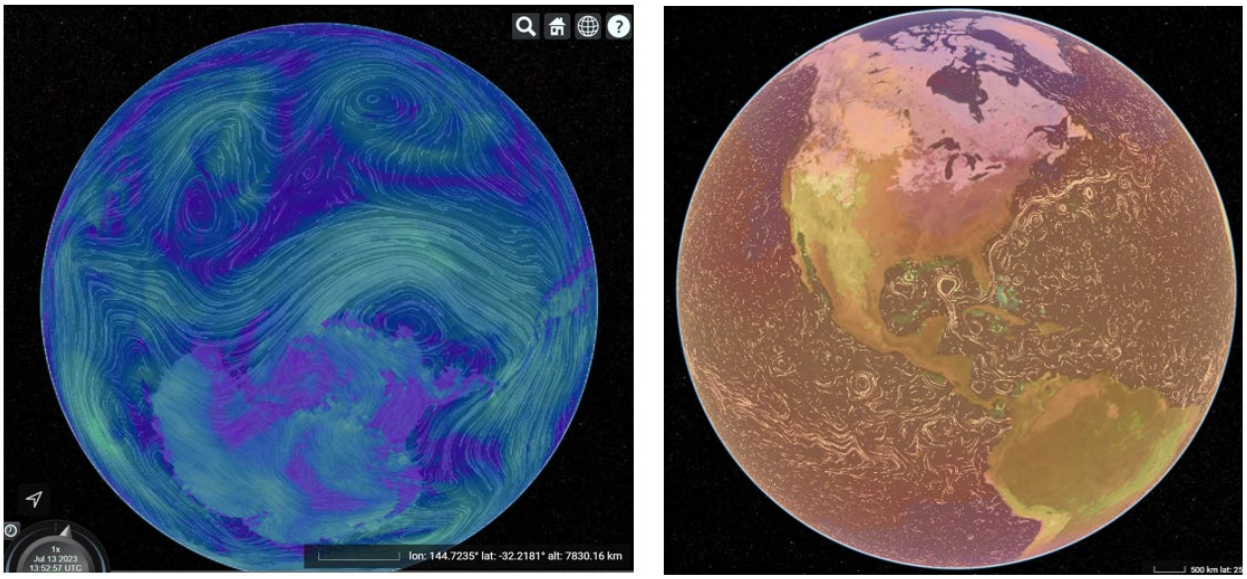

Figure 45. (Left) Particle system representations of winds at 500 hPa. (Right) Particle system representing ocean currents.

4.1.2.5.2 Slow Moving Particles vs. Arrow/Vector Fields

While moving particles provide a dynamic and visually appealing representation, there is some amount of pseudo-randomness in the update position calculation. Arrow or vector fields offer a more traditional approach to visualizing the vector data. Typically, arrow fields represent the direction and magnitude of vectors at discrete points, where the length and orientation of arrows correspond to the strength and direction of the field. Each method has its own strengths: moving particles are excellent for showing continuous, fluid motion while arrow fields are better suited for more static, field-like representations. For EO-DT, we chose to use particle systems since the data of interest for the program tended to be dynamic atmospheric phenomenon at the global scale.

4.1.2.5.3 Other Potential Use Cases for Particle Systems & Vector Fields

Customizing these visualizations is also key to their effectiveness. By adjusting parameters such as particle speed, width, and exaggeration, users can tailor their representation to highlight specific aspects of the data. This flexibility makes particle system visualizations a versatile tool for a wide range of scientific applications. For instance, particle systems might also be used to visualize smoke dispersion in the atmosphere or used to represent global tides in addition to currents.

Future work could also explore some of the paths that our team did not take. As discussed earlier, our team chose to implement particle systems to visualize vector data; however, arrow/vector fields are a visualization that we did not get around to implement that would be a strong candidate for future visualization work. Additionally, the particle systems that we chose to visualize for this project use vector data that is on roughly the same plane (e.g., ocean currents or wind at 500hPa); however, future exploration might include methods that allow individual particle trails to vary in not just longitude and latitude, but also altitude. Visualizing vector fields with these types of "xyz" particles would be useful for applications ranging from tidal analysis to ionospheric studies.

More work is needed to fully operationalize these particle systems and vector field visualizations. The first action item would be integration with our tiling service so that the time series data can be

ingested and processed into the format required by the WebGL textures used for visualizing the data. This would also require creating standardized schemas for updating and querying the basic ontology for these vector data sources in the database. Additionally, there should be efforts for adapting the UI/UX to handle the proposed vector data classes in a consistent manner. This includes not just styling and formatting, but also the interactivity for adjustable settings such as color, speed, and density. Another client-side task would be optimizing the dynamic loading and unloading of these WebGL textures as a function of time, which would depend on a number of parameters including the density of the data source, how many other data products are already in memory, as well as any hardware restrictions on the client's machine. Our implementation has not been optimized which leaves plenty of room for future improvement. We used JSON data to store the vector information rather than our 3D Tiles format; however, there may be some data formats better suited for network transmission while others may be better for rendering.

4.1.2.6 Terrain Data Visualization

In the previous section on bathymetry, we briefly introduced the concept of 3D terrain visualization in CesiumJS, but there are many more sophisticated terrain features. The ability to visualize and analyze 3D terrain data is another critical component of EO-DT, providing a realistic representation of the Earth's surface. Surface structure can affect the atmosphere due orographic forcing or clipping of the dataset due to terrain elevation – so its inclusion can be important to obtain environmental context. Within CesiumJS, configuring a terrain provider is like the other types of layers previously discussed. Terrain data is formatted either as height map tiles or quantized mesh tiles - served over HTTP via a REST interface. Tiles are organized in a hierarchical structure, which each level of detail (LOD) providing a progressively finer resolution so that we minimize network bandwidth and load time.

4.1.2.6.1 Tiling Terrain for CesiumJS

We chose to generate our own tile data set to serve the information more quickly and to have a standalone version that could be deployed to internal disconnected servers. To generate our own terrain data for the EO-DT platform, we began by downloading the open-source "ALOS Global Digital Surface Model 3D – 30m" from JAXA. The dataset is a global digital surface model (DSM) with horizontal resolution of approximately 30m, obtained by the Panchromatic Remote-sensing Instrument for Stereo Mapping (PRISM), which was an optical sensor on board the Advanced Land Observing Satellite (ALOS). The data is normalized, processed for clouds and other artifacts, and then released as a level 2 product.

We chose to create our own terrain tiles using the quantized mesh format which is a multi-resolution quadtree pyramid of meshes. In other words, instead of storing terrain data as a grid of height values (as in the traditional height map format), the quantized mesh format stores the terrain as a collection of triangles which define the shape of a surface within the tile allowing for better capturing of rapidly changing elevations. These triangles are dynamically generated based on the terrain's elevation data, with more triangles used in areas of high detail and fewer triangles in flatter regions. This format allows for less influence of floating- point rounding errors and reduces the size of the data generated.

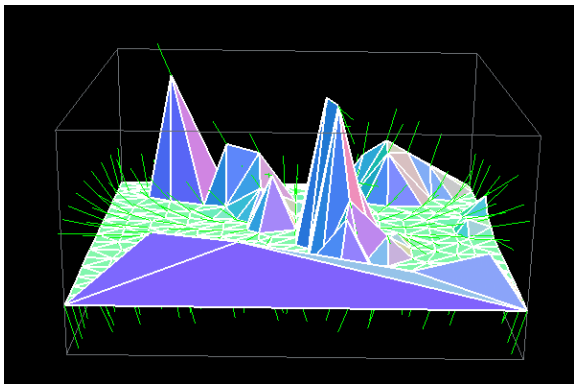

Figure 46. Quantized mesh geometry.

The process of converting global ALOS 30m .tif files into quantized mesh tiles for CesiumJS involves a significant amount of computational effort and a relatively large amount of storage. The maximum level for quantized mesh tiles is ~18, depending on the input dataset. For our dataset, we generated quantized mesh terrain tiles up to level 13 which supports resolutions as fine as 10-15 meters per pixel (this varies in quantized mesh format depending on elevation change density). This level of detail allows for fairly accurate representation of the Earth's surface, making it suitable for most applications that require a balance between detail and performance. The specific resolution can vary, but in general, level 13 provides a highly detailed view that can resolve individual features such as small hills, large buildings, or distinct vegetation patterns.

We began the data processing by using gdalwarp (Project, 2006) to transform the .tif files to WGS84. Then, we used an open-source library called Cesium Terrain Builder (GeoData, 2012) to convert .tif files to one level of quantized mesh terrain tiles. We repeated this process starting from level 0 up to level 13 which required roughly 90GB of storage and a few days of processing (see Quantized Mesh Terrain Tiles in appendix for level sizes). The increasing demands of both time and storage for this process requires careful planning and resource allocation.

4.1.2.6.2 Rendering Terrain in CesiumJS

After selecting a terrain provider in CesiumJS, quantized mesh tiles are loaded and displayed dynamically based on the user's viewpoint and zoom level. The hierarchical structure of the tiles allows CesiumJS to load higher-resolution tiles only when necessary, reducing the amount of data that needs to be transmitted and processed. In addition to the efficient tiling and rendering of 3D terrain data, the EO-DT platform also includes the ability to display terrain with more sophisticated features as we briefly saw in the bathymetry section.

In addition to the custom color ramp for bathymetry, we have also included other custom color ramps for visualizing various terrain attributes such as elevation, slope, and aspect. Through these rendering styles, users can visualize the elevation of terrain, helping identify highs and lows. Additionally, custom slope analysis can be used for studying natural disasters such as landslides and erosion. We also visualized terrain aspect, the cardinal direction that a slope faces, which provides insights into how sunlight affects the landscape – affecting vegetation growth, snow melt, and other regional environmental studies. Lastly, our custom globe materials extend to bathymetry as we saw in the previous section. This material in combination with the dynamically rendered

underwater terrain offers important visualizations to oceanographic research, by clearly depicting underwater features such as trenches, ridges, and continental shelves.

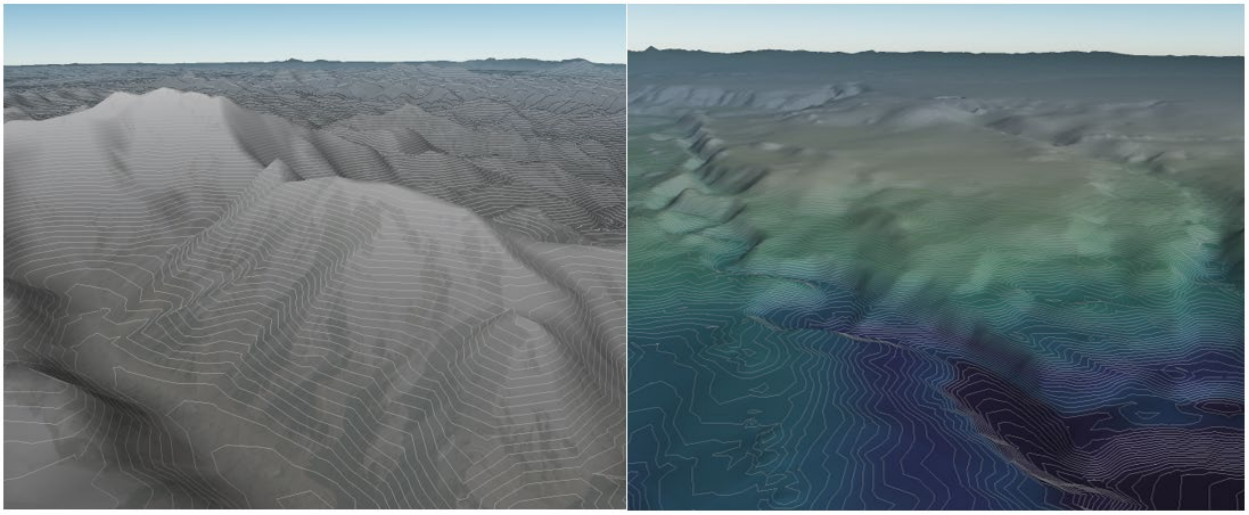

Figure 47. [Left] Aspect Material, [Right] SlopeMaterial.

In addition to custom globe materials, we implemented contour lines as part of the terrain analysis toolkit. Contour lines provide a clear and intuitive way to visualize changes in elevation, helping users understand the gradient of the terrain at a glance. The terrain analysis features implemented in the EO-DT platform makes it easier for users to gain deeper insights into the terrain. By combining efficient terrain rendering with advanced analysis capabilities not only offers visually accurate representations of the terrain, but also ensures that users can derive actionable insights from the data through interacting with these tools.

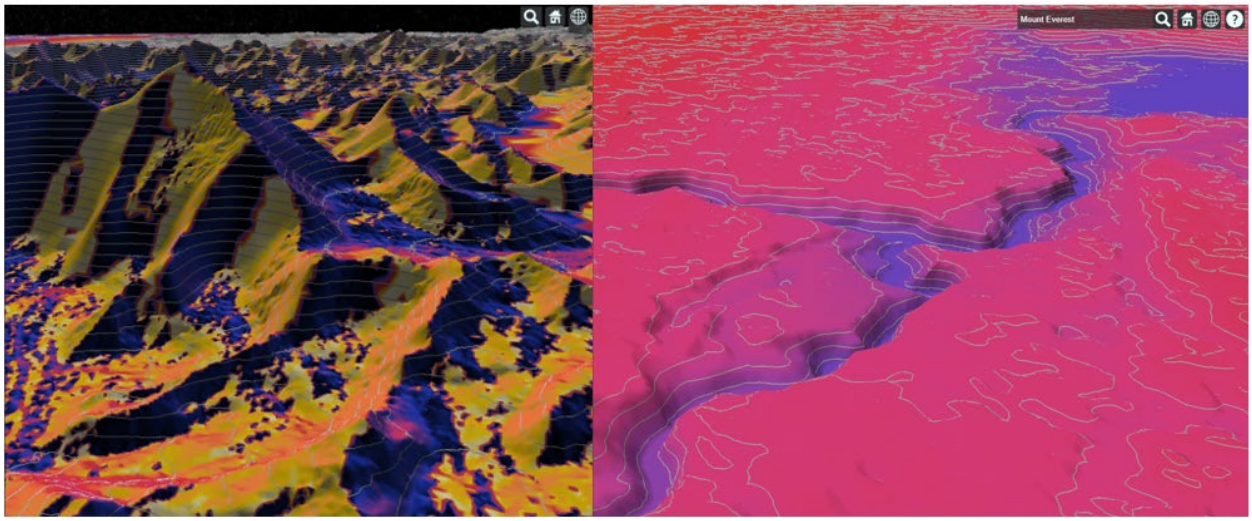

Figure 48. [Left] Aspect dependent styling, [Right] Elevation dependent styling.

4.1.2.7 Additional Data Formats and Techniques

Volume rendering using 3D Tiles constitutes the primary data format utilized in the EO-DT platform; however, we also employed several secondary data formats that play critical roles in the overall functionality and versatility of the system. Among these are terrain and imagery data as we

have previously discussed, but we have also incorporated other significant data formats that enhance capabilities.

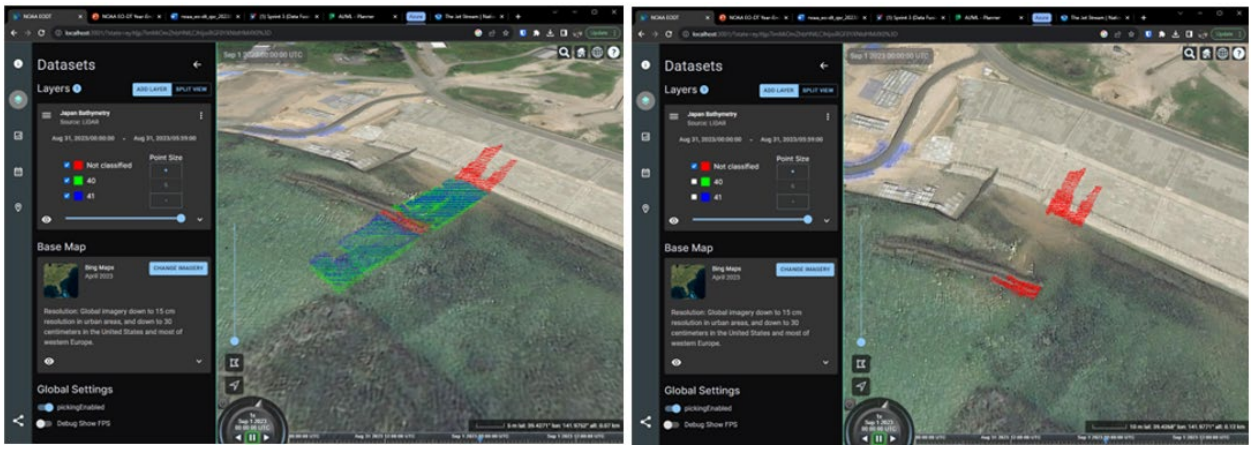

Figure 49. Visualizing a LiDAR scan using 3D Tiles (left: all points, right: only unclassified points and terrain exaggeration).

One of the notable formats that we worked with is LiDAR data, typically stored in either .las or .laz formats. We used our tiling process to convert the data into 3D Tiles point clouds which is a method for rendering volumes which we will expand on in a later section. For our initial exploration with point clouds, we used data from a LiDAR scan obtained by an Orion Space Solution product known as Edge. Edge is a LiDAR equipped drone that performs shallow bathymetric scans, capturing high-resolution (~1cm) 3D point cloud data of shore and subsurface structures. We processed and tiled this sample LiDAR data into the 3D Tiles point cloud in the Figure 49 below showing a Japanese shoreline with floating structure visualized in CesiumJS.

To further enhance the visualization, we utilized the classification values embedded in the .laz file to style the points according to their classifications. We can see that points classified as “41” colored blue (surface), points classified as “40” colored green (sub-surface), and unclassified points colored red. This not only makes it easier for users to distinguish between different types of features within the scan, but users also can hide or show points based on their classification, allowing for focused analysis of specific elements in the scene.

To further showcase the value of this data, we set Bing Maps as our imagery provider alongside the point cloud. This allowed us to directly compare the LiDAR data against the satellite imagery of the dock and its surroundings. The alignment between the point cloud and the imagery is striking –the features captured by the LiDAR scan precisely map to the corresponding objects in the satellite imagery. Even a small strip of unclassified points, rendered in red, is nearly perfectly aligned with a large, long object in the water at the same location and orientation. This level of alignment and detail highlights the incredible accuracy and resolution of the LiDAR scan visualized using 3D Tiles.

In addition to point clouds, the EO-DT includes the ability to render 3D Building Tilesets, provided by Cesium, Google and other providers. These tilesets represent detailed architectural models, allowing users to explore and analyze urban environments in a highly realistic manner. By integrating these pre-built 3D Tileset buildings, the platform offers an enhanced level of detail and realism for users focusing on urban environments and navigation. Both the LiDAR and point cloud

3D Tilesets are meant to demonstrate the versatility of 3D Tiles and how we can use 3D Tiles in more ways than just global volume rendering, which we will discuss in detail in the following section.

Figure 50. Downtown San Fransisco with 3D Tiles Buildings Dataset.

4.2 Data Visualization Format

During the process of prototyping our digital twin, we investigated multiple formats for visualization, both for storing scientific data and for the visualization data itself. Ultimately, we decided to use the Open Geospatial Consortium (OGC) 3D Tiles open-specification format to store data for rendering in the visualization engine, which we generated from scientific data files that would remain in their original CSV or HDF (netCDF/GRIB2) format. 3D Tiles takes advantage of binary representations of the data and provides the ability to attach extensive metadata and to style the data based on properties and metadata. The OGC 3D Tiles format uses embedded Khronos Group glTF 2.0 graphics format which has broad support in the graphics industry and is defined through an open and commonly supported specification. This is important because it opens the number of available tools for manipulation and editing and allows for the work of other commercial and private vendors and projects to extend the capabilities over time.

To assist in efficient processing of datasets based on the data's geospatial location, we used the Google S2 grid for 2D datasets and developed our own 3D geospatial grid for volumetric datasets. For the visualization engine, we used the previously discussed open-source, web-based CesiumJS, which can display the data for 2D/3D/4D visualizations at high resolution. These formats are our recommended formats for storing and visualizing data in an Earth Observations Digital Twin.

4.2.1 Hierarchical/Variable Resolution Gridding

Because of the wide variability in physical properties within the Earth environment, a single resolution grid cannot efficiently capture the behavior consistently, so a variable resolution grid is highly preferred. In order to capture a highly varying feature on a single resolution grid, a high resolution is required, but this wastes disk storage space by capturing excessive data in areas of low variability. Similarly, if one chooses a low-resolution grid to be more space efficient, it can miss features that are too small to be captured in the coarse grid. The solution is to vary the resolution by using a grid which can subdivide a finer resolution in cells and regions of interest

and use a lower resolution in areas of slower variability – known as hierarchical gridding. There are several ways to subdivide groups of data to match the resolution. A common nesting solution in Cartesian coordinates is to use a quadtree (in 2D) or octree (in 3D) subdivisions to selectively split the resolution of cells evenly by powers of 2 (see Figure 51). Every cell of a nested grid does not need to be divided – it is only necessary when the rapid variability needs to be captured within the cell. Thus, one can adjust the grid on a cell-by-cell basis to capture the detail required for the region being stored. There are many gridding systems available which can nest higher and lower resolutions and still be visualized without gaps or artifacts. These types of grids scale in a consistent and predictable manner so that any level cell can be visualized, accessed, or stored no matter its depth within the hierarchy. Structuring and accessing the hierarchical cells in this manner is known as implicit tiling - since a cell ID is all that is needed to locate the cell's volume within the gridded space, regardless of resolution. The identification of its parent or adjacent cells is not required, and the cell can be implicitly referenced. In addition, a table of which cells are defined and/or contain data can be produced to dramatically speed the loading (or not) of cells within a viewing region.

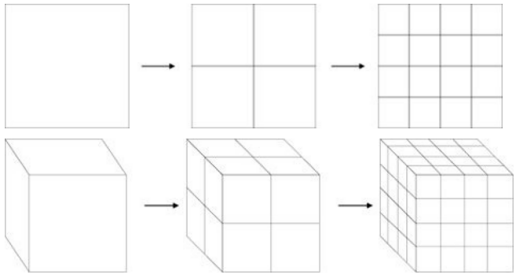

Figure 51. Quadtree and Octree subdivision examples.

Storing the variable resolution grid information can be challenging in databases or array storage formats if the grid array is not consistently sized in all dimensions. To solve this, these grids are generally stored in hierarchical file structures that mirror the parent/child structure of the grid itself. This allows for some folders to contain child folders/cells and others to not depending on whether data exists at the levels resolution and location – effectively producing a queryable nested grid that the visualization engine can use to efficiently load data based on zoom level, current screen resolution, screen location in the scene, and matched to the data which is available. Further, the data may contain information for only certain regions, and none in others. This type of storage is known as sparse data storage and allows for mixed resolution grids to be stored based on the complexity of the properties being represented and presence or lack of data in regions. In Figure 52, you can see a point cloud containing a mixture of highly density and low density information. The dense cluster of points in the central region could represent rapidly varying properties and therefore has a high cell density, while the empty regions in the corners have low variability and thus low cell densities. This produces an efficient storage method that can accurately store the scene complexity with much less disk space. Further, only data for an active viewing region needs to be delivered to the client for rendering - allowing massive data sets to be viewable in modest clients by streaming only the elements of the hierarchy which are needed (and minimizing the read/stream resource needed by the server).

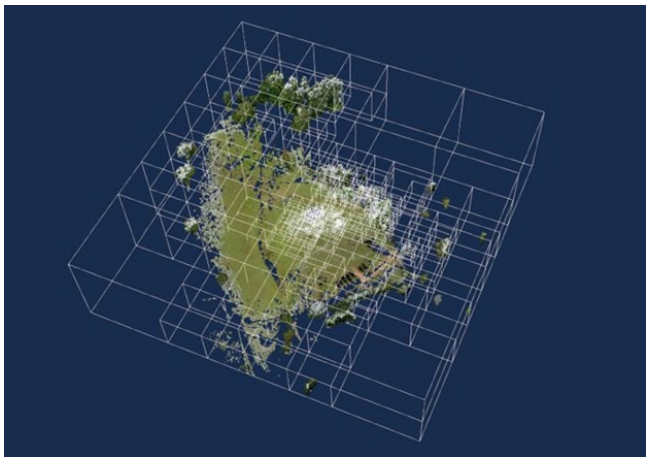

Figure 52. A point cloud organized into a sparse octree (data source: OGC 3D Tiles specification/Trimble)

4.2.2 Global Gridding Challenges

The variety of grids that are used for the scientific data presents challenges to fusing information effectively – both for scientific modeling and simulation and for visualization. Our initial plan was to convert all incoming data to single sparse hierarchical grid so that analysis in the visualization tool could occur and that differences between data sets could be obtained. This plan was quickly scuttled due to the complexity of the data grids and formats as well as the scientific rigor and compute resource that would be required to properly re-grid the datasets between grid formats without significantly altering the accuracy of the data. In addition, many gridding systems are not structured to effectively represent planetary scale data without significant grid imprinting or discontinuities at the poles or other locations when converted to 3D – so there was no "one size fits all" grid to use. Next, we describe the gridding systems that were explored and some of the shortcomings with them with respect to their use in a digital twin.

4.2.2.1 Oblate Spheroid Representations

Critical to the fusion of multiple data sources is a common coordinate reference system (CRS), which places items from their respective measurement platform in the right locations in the representation. The prevalence and accuracy of GPS has led to a de facto standard within most geospatial tools using WGS84 as the oblate spheroid representation of choice. This CRS is reasonably accurate for the work being done for this program but does leave some CRS-to-CRS transformation work if the originating data source is not in WGS84. Where needed, we would rely on the Proj library (https://proj.org) to perform any conversions. This library has been well vetted and contains mappings between the most commonly used CRSs. But care should be taken with any data source to harmonize the data on a common CRS and confirm any conversions align through reference testing.

4.2.2.2 Mercator Projection Gridding

Many standard GIS data products use a Mercator projection with a fixed latitude/longitude grid, which could be extended to 4D by adding height and time, but which present numerous challenges for scientific volumetric data processing. The primary issue using latitude/longitude grid is the variable size of the grid elements as you move from the equator to the poles. Models which

calculate mass or energy transfer from cell-to-cell are complicated because of the broad difference in both volume and width as you move around the grid – necessitating varying time or distance steps when used for models. These size differences also complicate storage of polar observation data, such as from an orbiting satellite observation, since even a consistent observation swath size would represent a very dynamic grid/data point count which varies based on latitude. These inconsistencies also result in severe grid imprinting near the mapping discontinuity at the poles which is visualized as 'puckering' of the grid.

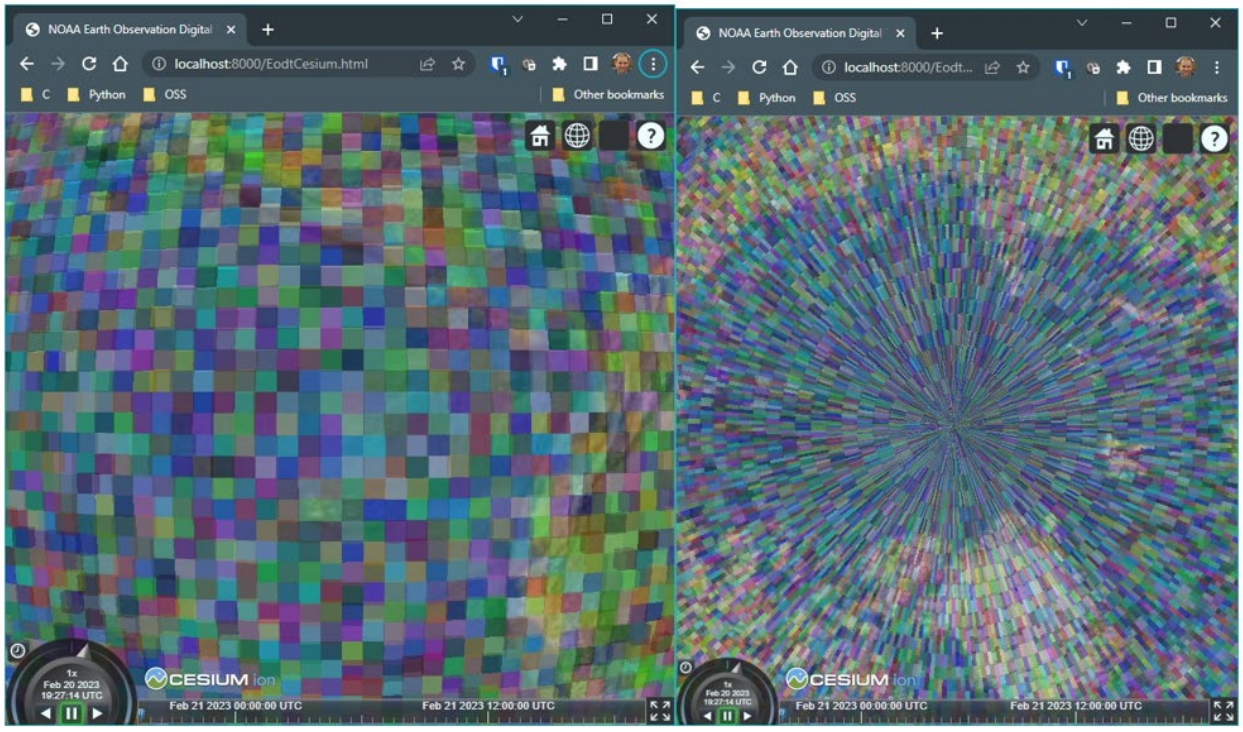

Figure 53. Random colored voxels on latitude/longitude grid near equator (left) and the pole (right) showing the vast cell size differences and the severe grid imprinting near the pole (1° x 1° grid)

The standard Mercator grid also introduces challenges when extending to 3D volumes that extend significantly from the Earth's surface. As one increases height above the surface, the volume of each cell increases due to the widening and non-parallel nature of the vertical edges. This challenges models which may rely on consistent volumetric sizes with increasing altitude.

4.2.2.3 Alternate Projection Grids

Gridding systems which use alternate projections to resolve the Mercator projection discontinuity near the pole and can resolve the large cell size difference have been explored and are better suited for scientific data analysis. Historically, there have been many different equal area or near-equal area projections developed and initial scientific reviews used these to attempt to solve the problem. However, most suffer from complex cell geometry which would make them difficult to use in dense cell visualization and the computation space in general. That changed with the broad adoption of GIS mapping systems which led to the creation of several computationally efficient designs as well as several open-source libraries to perform more consistent size cells. We reviewed the viability of several projections for use the in this project including Uber H3, Jet Propulsion

Laboratory's Hierarchical Equal Area isoLatitude Pixelation (HEALPix), Google S2, and NOAA Geophysical Fluid Dynamics Library Finite Volume cubed-sphere (FV3).

4.2.2.3.1 Uber H3 Icosahedral Projection

Uber H3 Hierarchical Geospatial Indexing System (Inc., 2018) uses a gnomonic projection centered on icosahedron faces which is a projection of the Earth on to a twenty sided platonic solid, then constructs 122 similarly sized hexagonal and pentagon base cells over those projections to form the base level of the grid base grid. This allows for fast indexing and a cell which can be subdivided to any resolution. Because a sphere cannot be covered with hexagons alone, 12 pentagons at each resolution must be used to complete the grid. In addition, the mapping produces slight variations in the size and shape of the cells based on their position relative to the icosahedron vertices. This produces a near consistent cell area with a maximum cell area variation of 1.993 (max to min area ratio) over its full resolution. The grid uses a combination of hexagons and pentagons whose rendering would be slower than a rectangular prism due to the increased number of vertices (12 vs 8).

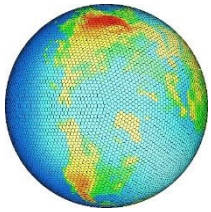

Figure 54. Uber H3 global tiling

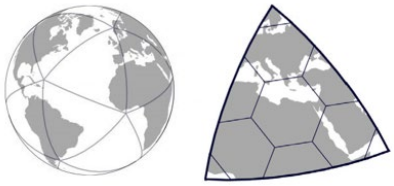

Figure 55. Uber H3 base icosahedron and tiles

However, complexity in this grid arises with the need to support both hexagon and pentagon primitive cell shapes, and with the difficulty extending this grid to 3 dimensions. Additionally, the cell parent/child relationship for the H3 grid was complex and did not nest completely – resulting in the inability to use the grid for scientific models requiring grid to grid transport between cells.

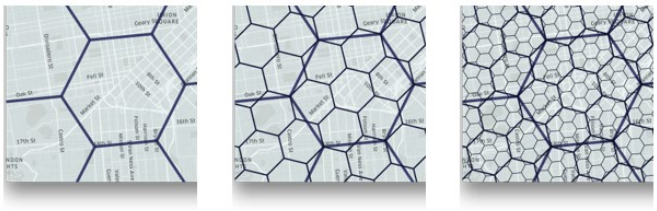

Figure 56. Uber H3 child cell decomposition.

4.2.2.3.2 JPL HEALPix 2-Sphere Projection

The Jet Propulsion Laboratory uses a Hierarchical Equal Area isoLatitude Pixelation (HEALPix) grid (Górski, et al., April 2005) for the processing of cosmic microwave background information and contains truly equal area cells. This projection is derived from mapping of the 2-sphere to the Euclidean plane and can be thought of as twelve diamond shaped facets which can be

subdivided. While this grid does produce equal area cells, the complex nature of the cell shape would make visualization rendering extremely slow and complex. Likewise, regridding operations could be challenged if second and third order cell-to-cell transport were required.

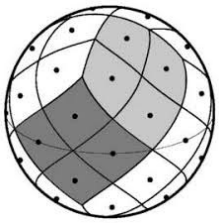

Figure 57. HEALPix global tiling

4.2.2.3.3 NOAA FV3 Cubed-Sphere

The Finite Volume Cubed Sphere (Harris, 2021) grid is based on cube projection onto a spherical surface. This produces 6 curved sides that form a grid with good uniformity over the full sphere and only minor grid imprinting issues near the face edges and vertices. FV3 also supports 3D gridding and has significant scientific backing given its use in the numerical weather prediction system. The cells also can render quickly given their relatively rectangular shape. However, the FV3 code is distributed only in Fortran90 code and has limited external library support and limited wrappers in other languages, requiring significant additional development for web visualization.

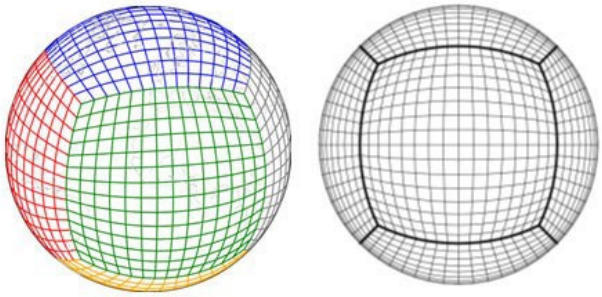

Figure 58. Cubed Sphere with its six base faces

4.2.2.3.4 Google S2 Cubed-Sphere

Google developed the S2 library (Google, 2015) based on the Cubed Sphere projection, so it is very similar to FV3 at the surface, with six base levels cells which are subdivided to provide higher resolution. It is efficiently indexed and has major software library support. The projection provides good grid uniformity over the globe with a maximum/minimum ratio not exceeding 2.12 over its entire resolution. S2 is based on spherical Earth and side steps direct oblate spheroid calculations, which could lead to some discrepancies between very detailed data. S2 also only supports 2D data, so it would need to be extended to 4D with height and time additions and would create the need to extend their more complex Hilbert space filling curve to three dimensions.

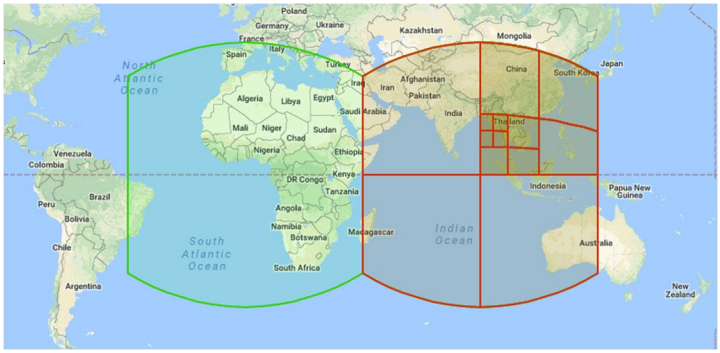

Figure 59. Google S2 Hierarchical Gridding showing 5 levels of detail and the variation in cell size.

The use of Google S2 cubed-sphere grid was chosen for the 2D visualizations to provide a consistent and computationally efficient geospatial grid, and it is already supported within the CesiumJS 3D Tiles format being used. The S2 grid is used within Google Earth and is a hierarchical grid using quadtree subdivisions between levels as described previously. Every cell within S2 is stored as a 64bit value to maintain consistent data size. S2 grid cell sizes range from ~9.2M km on edge (level 0) down to ~0.86cm on edge (level 30) and are approximately similar sized at each level across the entire globe - having a maximum to minimum area ratio not exceeding 2.12 at any level of resolution. The S2 grid is also ellipsoid agnostic – meaning it can readily express latitude and longitudes on the WGS84 ellipsoid. Each S2 level fully indexes the entire surface of the Earth and can rapidly convert between latitude/longitude and S2 cell index, center, and vertices. Further, the library uses very efficient binary operations to perform geospatial operations making it significantly faster than traditional geospatial calculation algorithms which rely on trigonometric functions. The S2 grid cells are implicitly indexed along a space filling curve using a single 64-bit index to uniquely identify, locate, and position the S2 cell globally at any level of resolution down to sub-centimeter scale. This implicit indexing capability is critical as the resolution of observations increases since the number of pixels quickly becomes untenable to directly load dataset at very high resolutions. So, each observation point can both be located and sized with a single S2 index value without the need for a full grid.

Grid Name	**Supported Language Libraries**	**Dim.**	**Projection**	**Gridding System Challenges**
HEALPix	C, C++, Fortran90, IDL, Python	2D	Spherical	Grid shape is very complex and would be very slow to visualize. Specific to 2D.
FV3	Fortran90 (some unsupported other languages	3D	Cubed Sphere	Minimal library support outside of Fortran, would be difficult to visualize
Uber H3	28+ languages, including C++, Python, Java, C#	2D	Gnomonic on Icosahedron faces	Hierarchy is not consistent between levels (children do not stay within parent). Specific to 2D.
Google S2	C++, Java, Go, Python	2D	Cubed Sphere	Based on spherical rather than oblate spheroid geometry. Specific to 2D.

Table 10. Software Summary of Hierarchical Gridding Systems Explored

4.2.3 Volumetric Data Visualization

High-resolution Earth observation datasets can have millions or even billions of data points with rich metadata. To efficiently render and manipulate datasets of this size, reduced resolution versions of the data must be used. A technique known as Hierarchical Level of Detail (HLOD) is used to store progressively reduced resolutions of the data to be visualized – only loading the resolution that is necessary for the client when translating and zooming the view. For data integration and comparison, a digital twin also needs to support the ability to store the data in a dynamically scalable grid that can support nested resolutions of data that can be quickly mapped to existing coordinate systems. OSS began our DT prototype with the publicly available Google S2 grid to locate and process data for 2D visualizations; however, it would have been complicated to extend S2 in the vertical direction. To effectively move to volumetric data visualization, we created a custom 3D geospatial grid to locate and process 3D data. Similarly, we explored how to store and represent volumetric data while maintaining the HLOD and dynamic gridding that is needed to capture rapidly changing features.

To view the volumetric data, we did not want to attempt to directly render the scientific data because of the varied formats and gridding systems. So, we instead explored two alternative approaches: 1) leaving scientific data in its original grid and format, but re-gridding select data in a volumetric visualization format, 2) leaving scientific data in its original grid and format but storing it as point data in a scalable visualization container format for efficient display.

4.2.3.1 Volumetric Gridding

As we extended our visualization capability to volumetric data, we required a grid similar to S2 but with an additional height dimension, so an indexing scheme like the one S2 uses was an extremely important part of our architecture. Volumetric data gridding would require hierarchical indexing of cells within the grid for fast and efficient lookup and mapping. We chose to use the Morton space filling curve for indexing in 3D, which allows for a simple interleaving of the bits from a cell's position in the x, y, and z directions to quickly give its Morton index or position along the entire curve but sacrifices spatial locality along the space filling curve (meaning not all cells adject in the curve are adjacent in space). Thus, one efficient computation using bit operations can give an index that uniquely identifies the cell in the entire volumetric grid space. The Morton curve was chosen over the Hilbert space-filling curve that is used in the S2 grid for computational simplicity. With the Hilbert curve, the direction in which the index moves changes due to the way that the curve snakes around the grid, and thus swap tables are necessary when preparing data for indexing. The Morton curve, on the other hand, is the simple "Z-order" curve, which always moves in a Z pattern through each section of cells. We also decided to use a latitude, longitude, altitude gridding system for our volumetric grid, though conversion to Cartesian coordinates is also built in for visualization purposes as well as possible ingestion of Cartesian data. We chose to use 32 subdivision levels with the octree sub-divisioning format, with the altitude dimension extending from the center of the Earth to seven Earth radii (~38,000 km above the surface). The grid is based on the WGS84 ellipsoid to maximize compatibility with common geospatial coordinate systems. The maximum radial distance of seven Earth radii was chosen because it results in cells at the smallest subdivision level of about one cubic centimeter at the equator on Earth's surface. This gives up to centimeter-level resolution in each dimension. The maximum

distance from Earth's surface is also slightly higher than geostationary orbit (35,786 km altitude), so the volumetric grid can represent data all the way out to GEO locations.

We have explored the possibility of altering our grid to not be limited either in increasing or decreasing extent and altering from oblate spheroid to cartesian but have not completed the work due to time constraints. This is desired since there are significant scientific data sets beyond geosynchronous orbit, such as heliophysics, lunar/interplanetary physics and orbits – and similarly we would like to have the ability to render smaller objects like vehicles and structures with high resolution. Also, moving from a geo-centric oblate spheroid coordinate system to an inertial cartesian grid would simplify orbital mechanics calculations as well as solar system body-to-body interactions, missions, and simulations.

4.2.3.2 Hierarchical Levels of Detail

We also determined that it was necessary to add hierarchical levels of detail (HLOD) to our visualization data format. This allows for faster rendering of high-resolution datasets by progressively refining the visible data as the user zooms in. Thus, while a smaller, lower-resolution version of the data is loaded at first when the entire globe is in view - reducing latency and data transmission payload size within the viewer, more detail can be rendered as the user moves closer to regions of interest in the dataset. This was a necessary feature to implement, as the rendering speed of large volumetric datasets such as GFS in our CesiumJS viewer dropped below 10 FPS in some cases without HLOD. HLOD is supported in the S2 grid due to its quadtree sub-divisioning scheme, and we intentionally designed our 3D grid to support HLOD the same way that the S2 grid could. Using the octree structure, data points in the same "parent" cell – which encompasses eight equally-sized "children" as shown previously in the bottom half of Figure 51 – can be represented as one point at a lower resolution. This one averaged point can be shown when the user is farther away from the area of interest in the viewer, being replaced by its children when the user zooms in enough. This concept can be applied iteratively, walking as far "up" the octree structure as needed to achieve a suitable balance between performance and resolution for the optimal viewing experience.

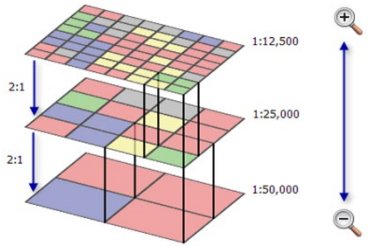

Figure 60. Hierarchical levels of detail. Credit: ESRI/ArcGIS

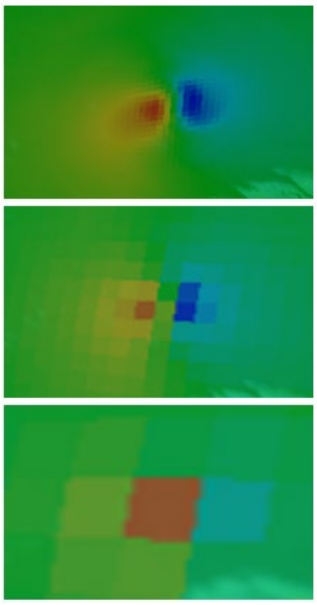

Figure 61. Feature in 2D GFS wind speed data at three resolutions, using hierarchical levels of detail (HLOD).

When implementing HLOD, we noticed that some features in the original data that had high variation over a small area were disappearing as we averaged up multiple levels. Because of this, we modified our formula to weight child values that were farther from the mean more significantly. This allowed rapidly varying features (such as the one in Figure 61) to be noticeable even in the coarser resolution data. Importantly, the original data is not affected by this averaging algorithm, only higher level representations.

Examples of HLOD implemented in GFS tilesets is shown in Figure 60. *HLOD made a significant difference in performance for high-resolution datasets, so we recommend implementation in a high-fidelity Earth observations digital twin data format.*

4.2.3.3 Boxes vs Point Clouds Rendering

When transitioning from 2D to 3D visualization, we expanded S2 cells to large 3D voxels by drawing fixed color boxes around each grid point that had data. However, as we continued to generate 3D datasets, we realized that the large data voxels being rendered for volumetric datasets are difficult to make look smooth, especially with multiple altitude layers of data. The voxel edges are especially noticeable because they double up between adjacent cells, making ridges that result in the dataset looking blocky. In addition, it is difficult to distinguish data at multiple altitude levels – depending on the opacity, a dataset would either be nearly transparent from the top or only the highest altitude layer would be visible.

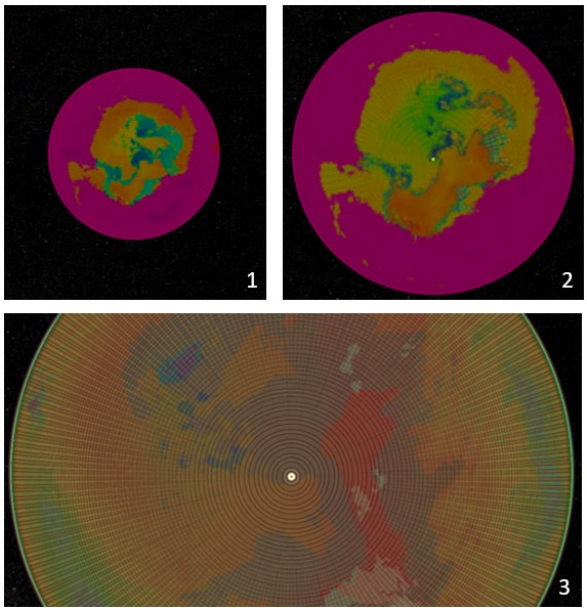

Figure 62. 3D GFS temperature data at the South Pole, visible at three levels of resolution (increasing from 1 to 3).

This blocky appearance is due to one of the major drawbacks of rendering data as a block is the fact that we are unable to effectively interpolate color across a single box via trilinear interpolation without complex redundant data storage or complex spatial queries. This is because when parallelized on a GPU, each box primitive has no knowledge of its neighbors and therefore is unable to interpolate with neighboring values within the camera's field of view. A common solution to this problem is raycasting, which aims to more accurately represent color and transparency at varying depths using virtual rays of light passing through the data boxes but is computationally very expensive.

For this reason, the team decided to look at other ways of rendering volumetric data. Direct volume rendering (bypassing the tileset format) and directly calculating voxel values was considered, but the most straightforward approach given the work thus far was to shift the method of storing data within tilesets from boxes to a "point cloud". This approach was determined to affect the greatest improvement in volumetric rendering without a complete overhaul of our entire data processing and visualization workflow.

For data processing, this meant that the tilesets would be modified from storing data as glTF triangle primitives forming boxes to glTF point primitives storing just the data points. Within the "tiling" process (generating a tileset from an existing dataset), the 3D Tiles content files became collections of point primitives rather than a set of triangle primitives. This reduced the size of a tileset for the same original dataset, since drawing voxels required 12 triangles (two for each of the six faces of the voxel), each made up of three points in space for a total of 36 points stored per data point, whereas the point primitives only require one point at the location of the data. Removing the loop over box corners and accessing only the actual point location also speeds tileset generation.

Using points instead of boxes has the additional advantage of no longer requiring re-gridding of datasets onto our internal grid. This step was required because to make the boxes regular, the data would need to be spread evenly through the box grid. However, when showing a data point as a color, there is a natural decay away from the measured point to adjacent points where the color should be interpolated between actual data points. By leaving the points in their original location, we could scale the size of the translucent point primitive to simulate this interpolation without performing full calculations. To further increase the blending, we used a diffuse shading of the data point with distance from the actual data location. This simulates the increasing uncertainty as you move away from the measured/calculated value and naturally blends the color the adjacent values. There are numerous distribution functions which can be used for this fading effect, and we explored the use of several to determine a visually appealing result, including square/box (no fading), Gaussian, Beta, Planck-Taper, and Tukey. Figure 63 provides a view of the methods we explored, with the top right as the base image.

Our point cloud approach uses a diffuse shading mechanism to achieve a smooth blended look between adjacent points, which removes the need to access data other than the current point. It also allows us to store point locations directly from the original dataset when generating a tileset - simplifying the process for dealing with sparse and multi-resolution datasets.

Figure 63. Various volumetric rendering examples using several distribution curves mapped transparencies based on distance from the data point's central location. We also varied the settings of the distribution curves to increase or decrease the blend to adjacen

4.2.4 Data Visualization Format Performance Testing

To optimize the transmission of 3D tiles, we explored two main types of compression. Our goal is to compress information like vertex positions, normal, texture coordinates, and any other generic vertex attributes, improving the efficiency and speed of transmitting 3D content over the web. The first compression method that we explored is known as Draco – a glTF extension for mesh compression developed by Google to compress and decompress 3D meshes, significantly reducing the size of 3D content. The issue with Draco compression is that this type of compression is lossy, which led to loss of precision for each tile. This resulted in tile geometry that was slightly smaller than expected, causing gaps in the tessellation of the 3D tileset as we can see in Figure 64 below.

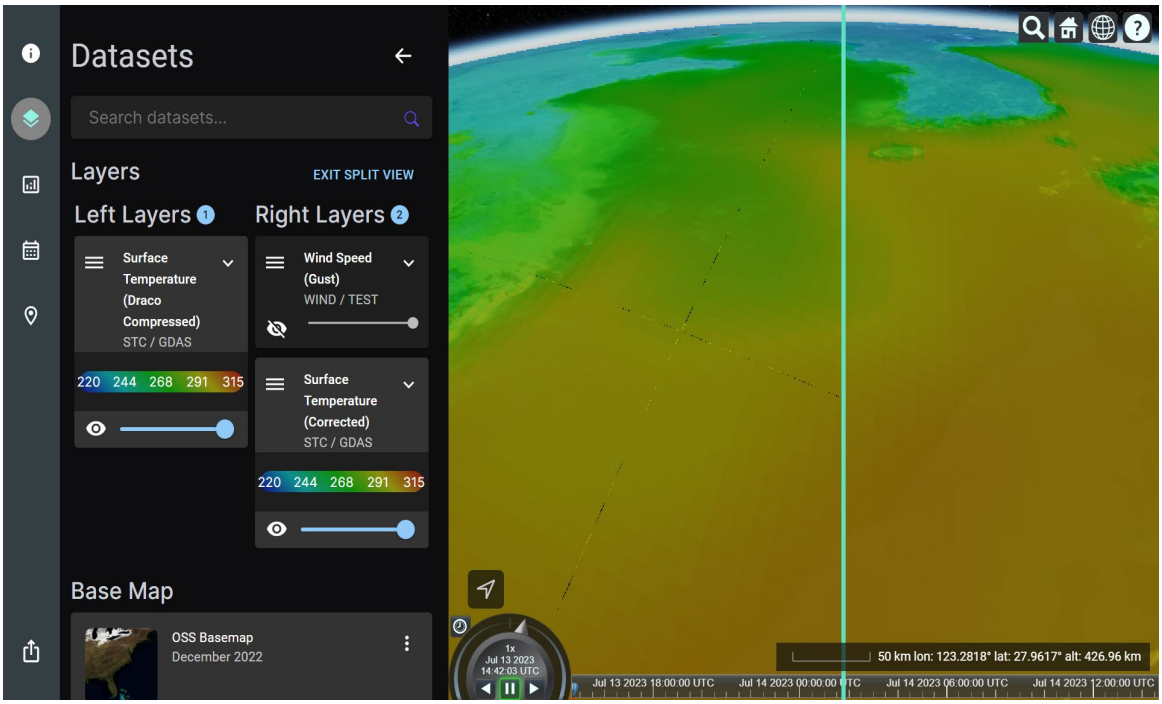

Figure 64. Split scene using two compression methods. Left: Lossy draco compression led to gaps at the edges of tiles. Right: Lossless gzip+glb compression.

Although one might argue that the amount of data lost during draco compression is negligible, the visual discrepancy created by the lossy compression led us to use gzip compression alone because gzip is lossless. During the tiling process of generating tilesets, we use the compressed glTF format known as ".glb" which significantly reduces the size of the 3D mesh before we further compress the content with gzip. We can see on the right hand side of Figure 64, that the gzipped .glb.gz content in the 3D Tileset no longer gets clipped at tile edges due to lost information. We've implemented gzipping at compression level 9 within the tiling service, so each tileset is generated with both gzipped_content and content directories. The first directory contains 3D tileset contents in the format content.glb.gz which is decompressed by our tileset service before being served to the client. The second directory is used with clients that do not natively support gzip compression. Initial benchmarking showed nearly a 50% load time for the compressed tilesets.

Tileset contents have been gzipped with a compression level 9. The following table shows the compression ratio and the time it took to load the tileset in Cesium. Values have been **averaged over 10 runs.**

Tileset	Size	Size (gzipped)	Load Time	Load Time (gzipped)	Compression Ratio
Suface Temp	163MB	47.6MB	27.291 seconds	12.453 seconds	3.42
Cloud Coverage	163MB	47.8MB	29.978 seconds	13.408 seconds	3.41
Humidity	165MB	48.5MB	33.633 seconds	14.966 seconds	3.40

Figure 65. Initial benchmarks saw <50% load time for compressed 3D tilesets.

The time to ingest and process raw data into 3D Tilesets is another consideration when designing a platform like EO-DT. In the Figure 66 below, we can see how the resolution of the input data affects the time to build the corresponding 3D Tileset for a single thread. In addition to this consideration, we also spent some time implementing pipeline concurrency in the actual tiler microservice API. This optimization may vary depending on the use case, but our process of generating 3D Tilesets is both an I/O and CPU intensive tasks which can be balanced in several ways. For our implementation, a thread pool was used to implement concurrency in our tiling service and optimize the backend, while glb and gzip compression were both used to optimize transmission and time to render on the client side.

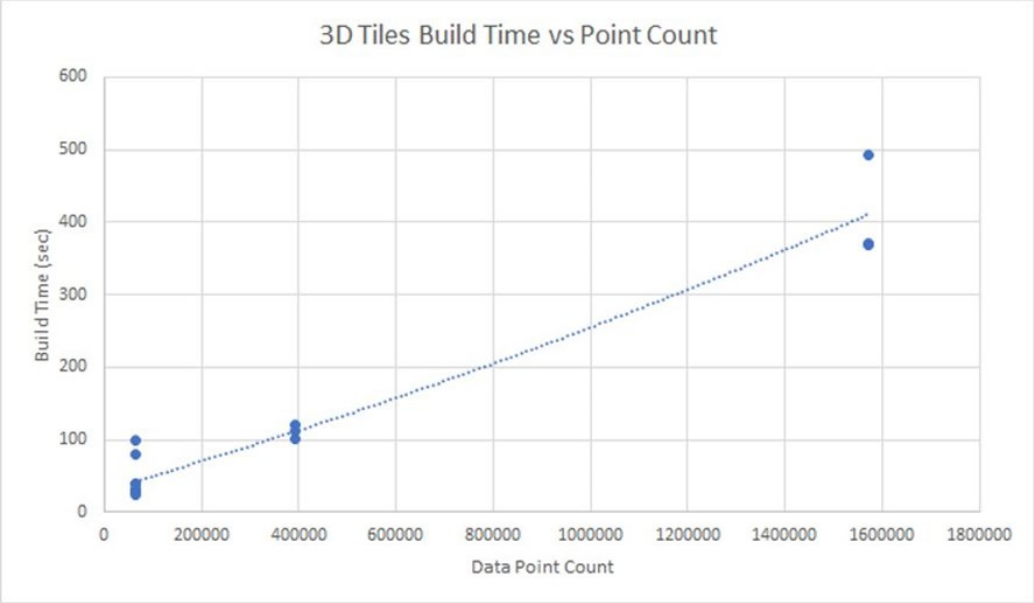

Figure 66. Initial 3D Tiles build time testing (single-thread, unoptimized code).

4.2.5 UI Platform Design and Architecture

The EO-DT platform was designed using a traditional microservices architecture where services are decoupled from the client. For our EO-DT platform, this approach made the most sense due to the broad nature of the services we wanted to implement, ranging from 3D Tileset services, to orbit propagation, to LLM chatbots, to HPC model workflows, and much more. Services are deployed independently, which can make it easier to manage bug fixes and future releases, improving agility. This decoupling of services is also crucial for improved fault-isolation. If an individual microservice becomes unavailable, it will not disrupt the entire platform. In addition to independent deployments, services can also be scaled independently of each other. Subsystems can be scaled up if they require more resources, without upscaling the entire platform.

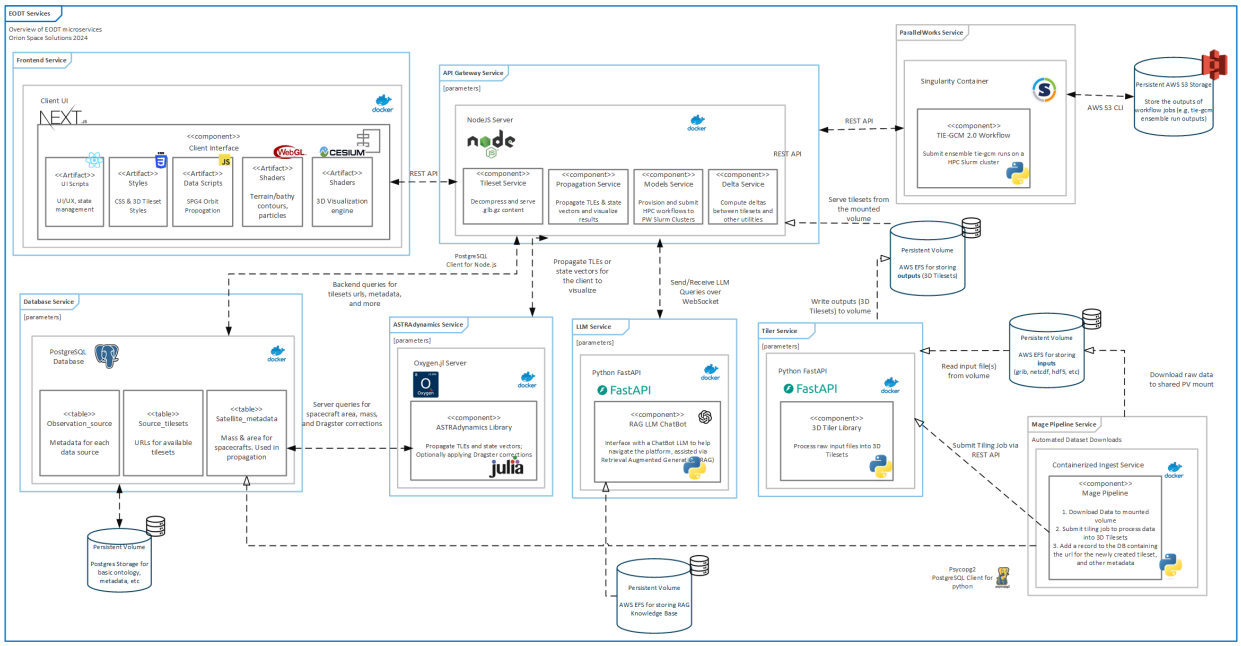

Figure 67: EODT EO-DT Microservices Overview (see appendix for larger image).

In this architecture, services are responsible for persisting their own states. This differs from the traditional mode, where a separate data layer handles data persistence. Our backend services for EO-DT were designed to be stateless. The frontend client application maintains any needed state for the user. Initially this was achieved using the React Context API for orchestrating a global state; however, we eventually migrated most of our core logic to use Redux for improved traceability and agility when developing state management for the application.

Another common component to microservice architectures is an API gateway which is the single-entry point for clients. Instead of calling many services directly, client requests are routed through the API gateway which can dynamically forward the calls to the appropriate backend service. API gateways are used for managing the various policies for the application such as authorization, authentication, networking, billing, security policies, and more.

4.2.6 UI Platform Services & Features

The EO-DT platform is comprised of several services and features that empower users with the tools for better understanding their data.

4.2.6.1 Datasets Service

The datasets service is the primary service in the EO-DT platform, allowing users to visualize high-precision, time-dynamic, volumetric data using 3D Tiles v1.1 data sources. Version 1.1 of 3D Tiles was used due to the addition of the EXT_mesh_features and EXT_structural_metadata glTF extensions. These are extensions to the 3D Tiles format which allow for the storage of custom metadata to all levels of the stored data and the ability to filter and style the data based on the information at runtime. We leverage these metadata extensions for both feature styling as well as data selection and filtering.

Data picking is the highlighting and showing of additional information about a data point and is a crucial feature for interacting and analyzing volumetric data in the EO-DT platform. To perform data picking, we attach event handers to the onClick method of the viewer. Whenever the user performs a left-click, we evaluate whether a 3D Tile feature has been selected. If it has, then we can extract each key/value pair from its feature table to efficiently retrieve metadata such as a point's longitude, latitude, and dataValue at that point to then be displayed in a UI legend component. In addition to displaying the selected value onClick, we also created a basic schema for unit conversions. This required storing the unit type in the tileset metadata and including client-side logic for defining the available conversions for each unit type.

Metadata is used for efficient retrieval of legend values in our data picking feature. We also leverage metadata to style our 3D Tilesets. The CesiumJS 3D Tiles Styles specification provides a concise schema for declarative styling of 3D tileset features. A style defines expressions that evaluate the representation of a 3D Tiles feature, for example color and show properties.

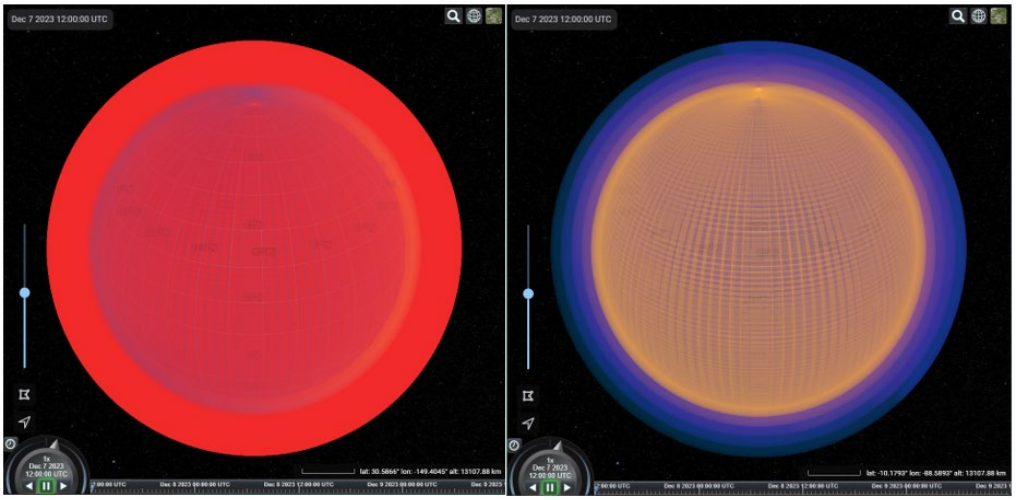

Figure 68. MSIS Volumetric 3D Tileset with same colormap – left is using linear breakpoints with constant alpha; right is using logarithmic breakpoints with alpha values increasing with altitude.

As we transitioned from generating 2D to 3D datasets, it became more apparent that we would need to improve the color blending between points and voxels within the tileset. The two main reasons for the drastic difference in the appearance from our 2D to 3D tilesets came from the added vertical dimension, which introduced a new layer of complexity to our challenge of accurate color blending. The second issue came from the distribution of the data since the data sources we used for our initial volumetric tilesets were heavily skewed to certain values. This meant that we needed to implement a logarithmic scale for calculating our breakpoints for each colormap to help keep colors from oversaturating volumetric regions. The second change we made was implementing better transparency leveling of the colors, which is needed to make the boxes closer to Earth's surface opaquer than the boxes that are located at higher altitudes otherwise the lower values would be obscured.

4.2.6.2 Data Slicing Feature

Data slicing is another key feature to our EO-DT platform which enables users to control the visible portions of a tileset by adjusting UI controls. We implemented data slicing in two ways. The first

way leverages the ClippingPlane class from CesiumJS to create a collection of clipping planes which form a clipped volume region with respect to some tileset. This type of a clipper component is useful if there are several 3D Tilesets that you would like to clip, located in the same position. Instead of setting the visible data extents for each tileset as we will see next, a user can simply clip whatever data happens to exist in the defined region. The code to implement the slicer component using clippingPlanes is far more complex than "clipping" via 3DTileStyles, especially writing the code for interacting with the volume and changing its size, which involves many transformation and translation calculations.

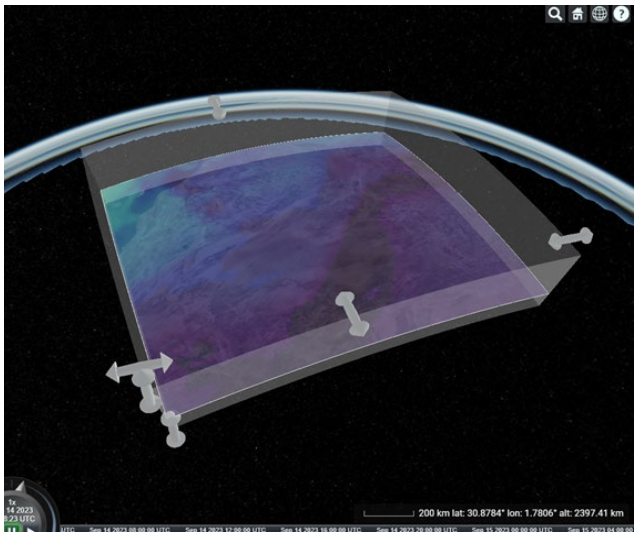

Figure 69. Volumetric Slicer Component.

Our second method of data slicing uses the 3D Tile Style specification to define a visible latitude and longitude extent for a single tileset, and then any point whose location is outside those bounds is hidden. This method achieves slicing at the tileset level, meaning no other data will be sliced but the tileset whose settings are being changed. While this method only slices relative to a single tileset, it is much more efficient as the tileset styling is handled in parallel by the GPU. In Figure 70 below is an example of a tileset whose visibility is defined using the extent sliders. Visible data can be controlled by adjusting the extents of the longitude, latitude, and altitude. One implementation note when creating this feature is that we needed to add metadata fields for the minimum and maximum height of the tileset, so that we are able to construct the bidirectional spatial extent sliders appropriately.

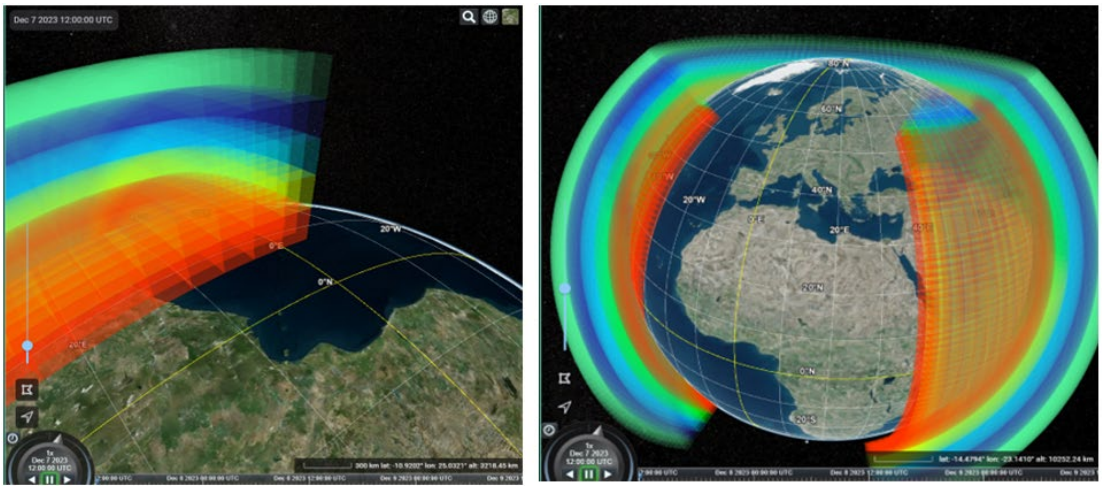

Figure 70. Volumetric Slicer via 3D Tiles Style.

4.2.6.3 Data Iso-surface Selection Feature

We also implemented the ability to define visible data based on a range of values (bounded by the data's min and max). This allows the user to filter the data visible to subsets in a specific range of interest. This is a key method for highlighting subsets of data or identifying areas with similar values. One special case of data filtering is when the user confines the data filtering to a single, scalar value. Computationally, the result is similar to an isosurface (like an isoline in 2D) – a continuous function that represents a surface of points of a constant value. Altering the data in the 3D Tiles schema to support actual isosurfaces using techniques as marching cubes is outside the scope of this project; however, in Figure 71 below you can see that we can still achieve analogous functionality by setting the range extents of visible data.

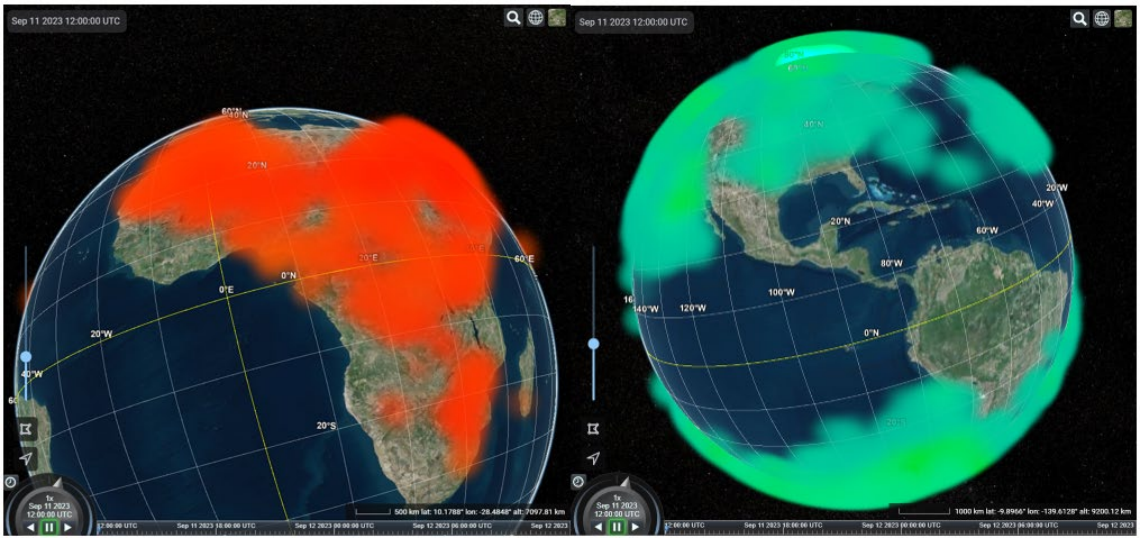

Figure 71. Setting visible data extents by value (GFS Temperature) – left shows subrange on max values (hot temperatures); right shows subrange on min values (cold temperatures).

In a previous section, we introduced our transition from boxes to points when rendering our volumetric 3D Tilesets. The main motivation behind this transition was to improve color blending in each dimension, especially depth. Other popular solutions for achieving volume rendering often

requires redundant auxiliary data storage or highly customized WebGL programs (which can differ from OpenGL specifications in disappointing ways) which implement a structure that is entirely different than our method of abstraction using 3D Tiles. Thus, we implemented a hybrid styling approach to improve color blending without entirely abandoning 3D Tiles.

4.2.6.4 Point Rendering Techniques

Our first task to improving point cloud 3D Tilesets was to render the points as spheres to reduce the visual space between points while also adjusting the transparency color and sphere size of each data point based on camera location. However, on a curved globe in 3D, the distance between two data points varies based on distance to the camera position and relative view of the camera as it moves around the scene. We implemented a custom GPU vertex shader to dynamically change size, color, and alpha value for all viewable points in parallel based on camera location. It is important to note that without using GPU shaders, it would be too slow to dynamically adjust the sizes, colors, and alphas for all points simultaneously. Figure 72 below shows the initial result of rendering the points as transparent spheres whose size is based on the screen resolution and distance from camera. In combination with our previously implemented 3D color mapping, this not only reduced the grid structure of the tileset, but also made it easier to distinguish structures within the volume itself. Depth perception is increased as users are now able to more easily spot structures and patterns, especially at lower levels.

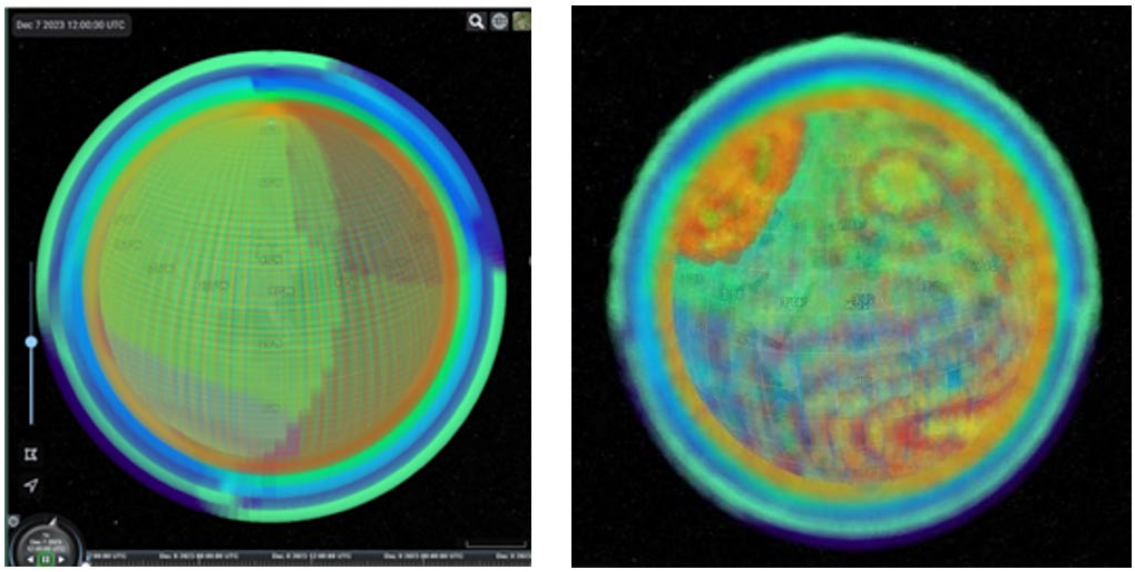

Figure 72. [Left] Volumetric tileset rendering as voxels. [Right] Volumetric tileset rendering as semitransparent points using a GLSL shader program to dynamically rescale points based on a pixel size related to distance to the

We encountered a problem at this stage linking CesiumJS styling features with custom shader programs using 3DTiles Tilesets (it is still an experimental feature in CesiumJS), which can lead to rendering issues as shown in Figure 73. This made it somewhat difficult to troubleshoot during development, so we advise not using custom shaders in conjunction with Cesium3DTileStyle (at least until customShaders are officially supported).

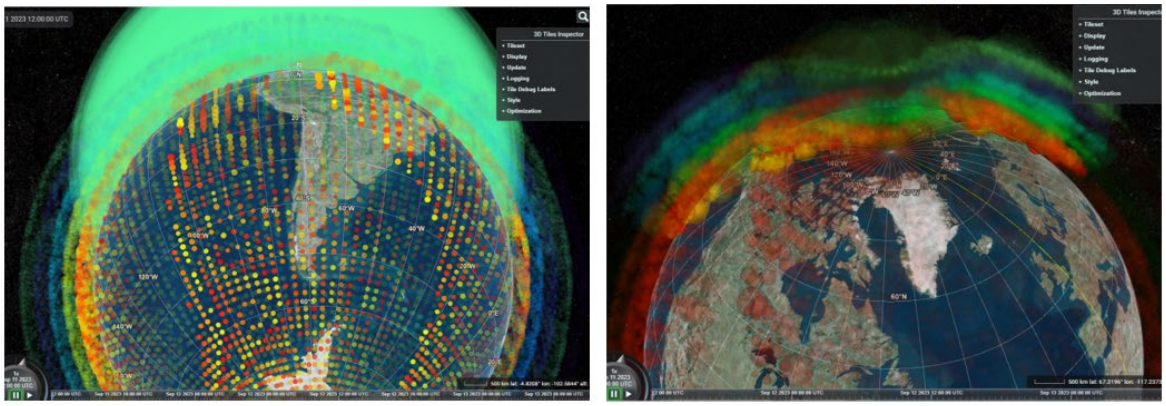

Figure 73. Examples of rendering issues when using custom GLSL shaders with Cesium3DTileStyle.

The next visualization improvement task we worked on for point cloud tilesets is to prevent too much or too little overlap between sphere rendered points which would over- or under-extend the influence of a data point to its neighbors. We decided to increase the transparency of the data points color the further from its central location and allow the natural transparency blending of the scene to blend adjacent data points. A fragment shader is needed to perform fading of pixels during rendering of a primitive. While a vertex shader determines the size and location of a single point primitive, a fragment shader controls the display of pixels which make up the sphere on the screen. Both the vertex shader and fragment shader are parallelized on the GPU to perform all rendering of the scenes data efficiently. The methods that we explored for diffusing overlaps between points are described above in Section 4.2.2.3 and Figure 63.

In Figure 74 below, we can see how applying different alpha blending functions at the overlaps between points can produce varying results and/or visual artifacts. Note the difference in the appearance at the globe's horizon on the left-hand side of each image.

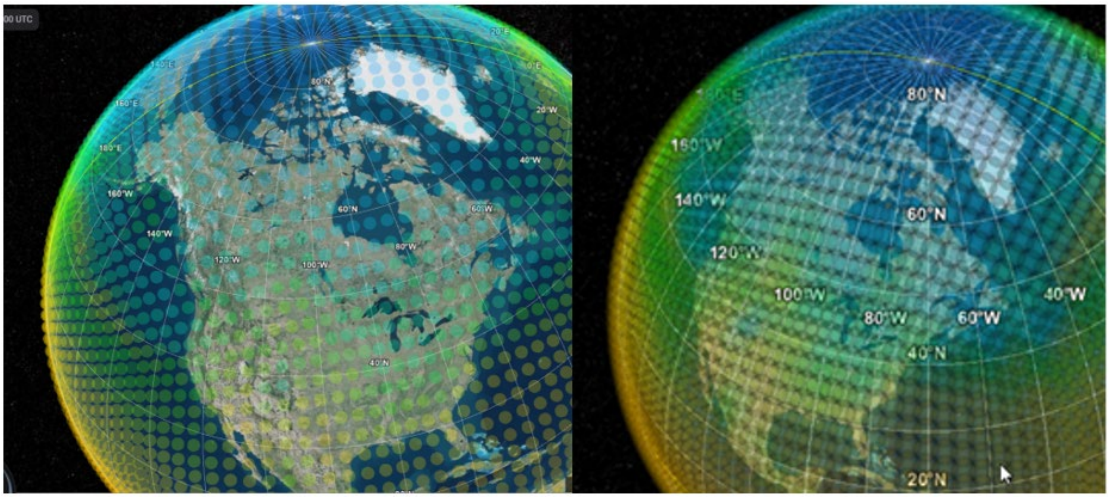

Figure 74. [Left] Smoothstep alpha blending on overlaps. [Right] Gaussian alpha blending on overlaps.

Since points for low-resolution datasets could appear far apart, we are added the ability to render super-sampled points for smoother visualization. This improves the user experience interacting with low-resolution datasets, and interpolating between data points is a common feature of other visualization applications. However, we recognize the importance of knowing what data is really coming from the model or measurement and what is not, so any super-sampled points shown in

the UI can be toggled on or off with a toggle button. An important consideration is that adding super-sampled points requires additional points in the tileset as well as a metadata flag to indicate that the data is interpolated for the toggle feature, although the additional points should only be necessary for datasets that were low-resolution (and therefore small) in the first place.

Our final implementation uses a GLSL smoothstep interpolation function to smooth the alpha values of overlaps between points. The smoothstep function is defined as a Hermite interpolation between two values. The Hermite interpolation is a generalized method of Lagrange interpolation. This solution achieved the diffuse blending of overlapping points while minimizing the distance between points. It also pushed us even closer to our final goal of eliminating the visible grid structure of voxels while improving the cloudy, diffuse appearance via improved color blending. Figure 75 to the right shows a point cloud 3D Tileset for the GFS temperature product, using both custom shaders we just covered.

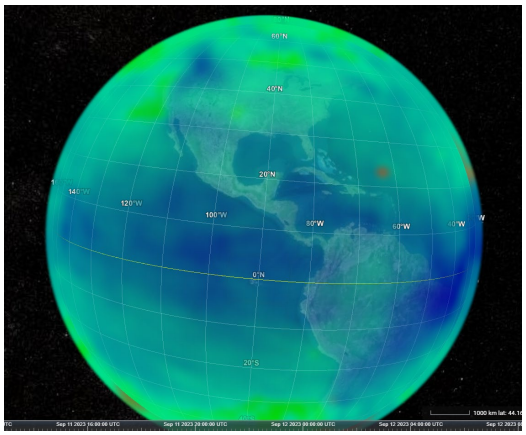

Figure 75. Achieve diffuse appearance by dynamically resizing points and blending overlapping alphas (5 pressure levels of GFS Temperature at 0.25° horizontal resolution).

To provide more flexibility in the rendering, we implemented the ability for users to define the base point size for a tileset. This feature is designed to better control the rendering based on the resolution of the volumetric data. The base size for each point is set using a uniform, which is passed into the vertex shader when it executes each frame. Executing the shader program on the GPU also means that this is efficient enough for a user to adjust base point sizing at run time. In Figure 76, we can see how the appearance of structures within the point cloud tileset change as the user changes the base point size. While deciding what size to set largely comes down to preference, this feature also helps accommodate the slight differences in point scaling for datasets with different resolutions.

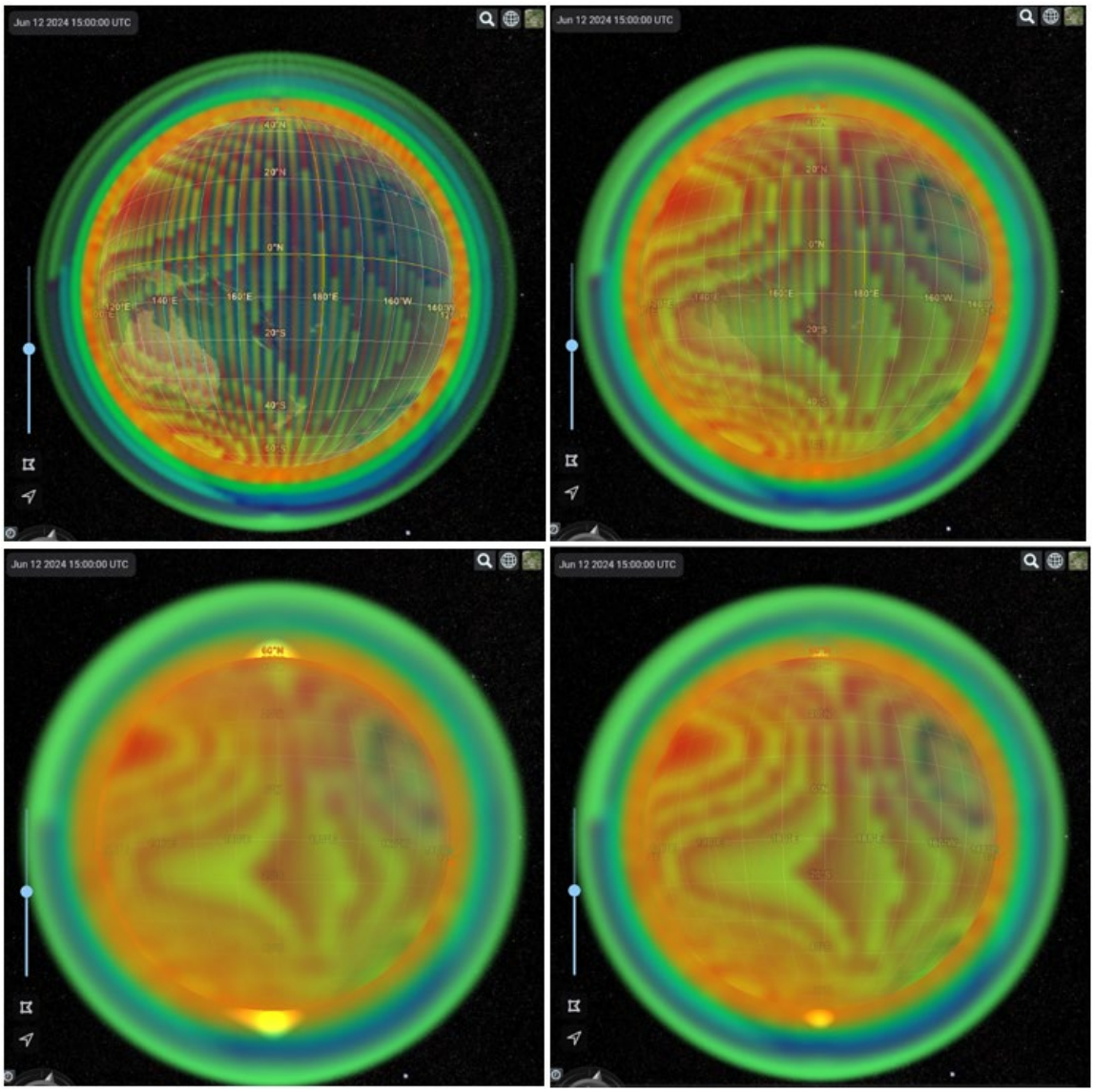

Figure 76. [from top-left, clockwise] Adjusting the base point size at runtime from 5px to 16px.

4.2.6.5 Models Service

One unifying vision that we at Orion have for digital twins is the ability to answer the questions of "what now?", "what next?", and "what if?". The question of "what now?" is what we try to represent with the current best estimate of the state of the system, based off observations or models. This empowers users with a high-fidelity visualization of the current simulated environment as we have seen. We are able to answer the next question of "what next?" by tiling and visualizing forecasts of the system. This may include forecasts obtained from systems such as GFS or customized forecasts depending on the data product. Lastly, to address the question of "what if?", we've implemented interactive, exploratory scenario planning in the form of high-performance compute (HPC) model submissions.

To implement the computational infrastructure and orchestration required to support HPC cluster workflows, we partnered with a company called ParallelWorks (PW) who provides a platform for

running workflows in a cloud environment of your choice. PW allows users to define HPC clusters configured to their needs using a JSON template. Users are then able to define workflows to run on their clusters via a GitHub repo or a bash script. Additionally, workflows and other PW methods like starting clusters can be executed using the PW REST API with a valid API key, which is what we rely on for managing workflows from the UI.

Our first end-to-end implementation of the models service configures the TIE-GCM 2.0 model to be run as an ensemble of N members of a SLURM cluster. The user begins by selecting TIE-GCM 2.0 from the models catalogue and then filling out the required inputs for the model. The model submission form contains all the required fields for submitting a remote workflow to a ParallelWorks cluster ranging from parameters such as temporal settings to cluster settings. In Figure 77, we can see the workflow submission form for TIE-GCM 2.0. We leveraged TypeScript to create interface schemas for available model workflows. This way, we are able to dynamically generate workflow submission forms for a wide variety of input parameters, as long as they match the submission form schema.

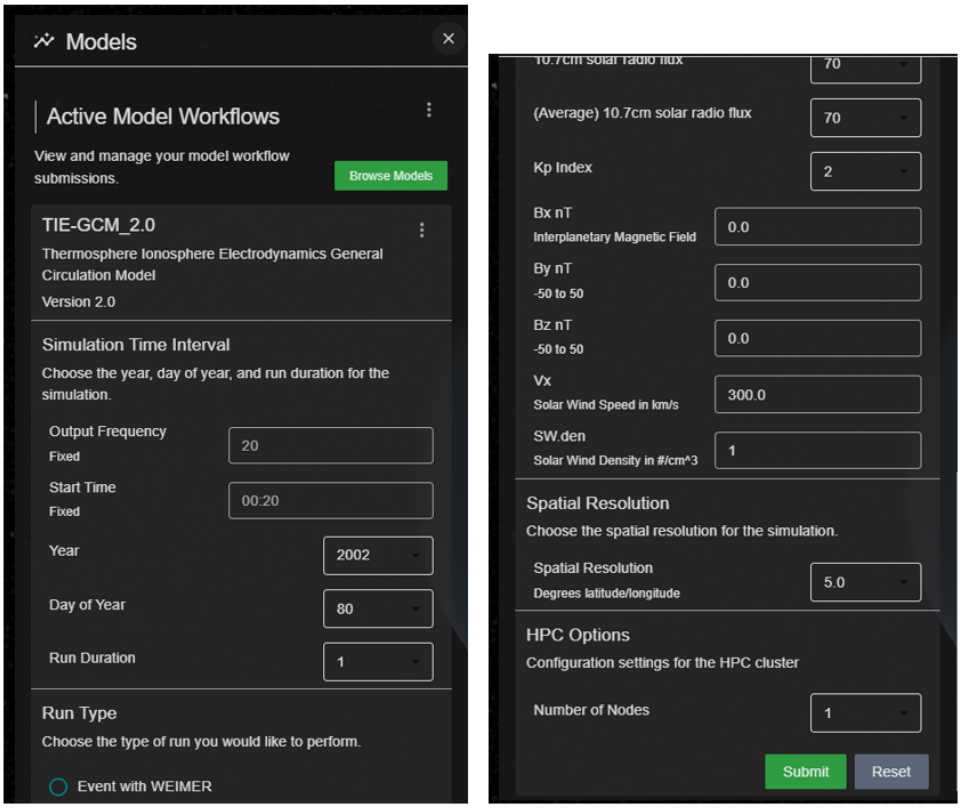

Figure 77. Expanded view of the TIE-GCM 2.0 HPC workflow submission form.

The workflow submission form also includes cluster specific parameters such as the number of nodes in the ensemble. Upon submission, the workflow inputs are sent to the PW cluster via HTTPS and the workflow begins to run. We implement a basic polling system on the frontend which uses the workflow/{id}/status endpoint to watch for the complete workflow completion. For each workflow submission, we keep track of the job id for the submission as well as the input parameters for that job. By saving the job id and the workflow name to localStorage, we were also able to implement the ability for the browser to persist polling for a workflow submission through

a page refresh. This is accomplished by checking for any existing job ids in localStorage upon mounting the ReactJS component and deleting these keys upon job completion. In a production system, we would recommend that job tracing be managed by a database instead of the client's localStorage; however, this works as expected for prototyping.

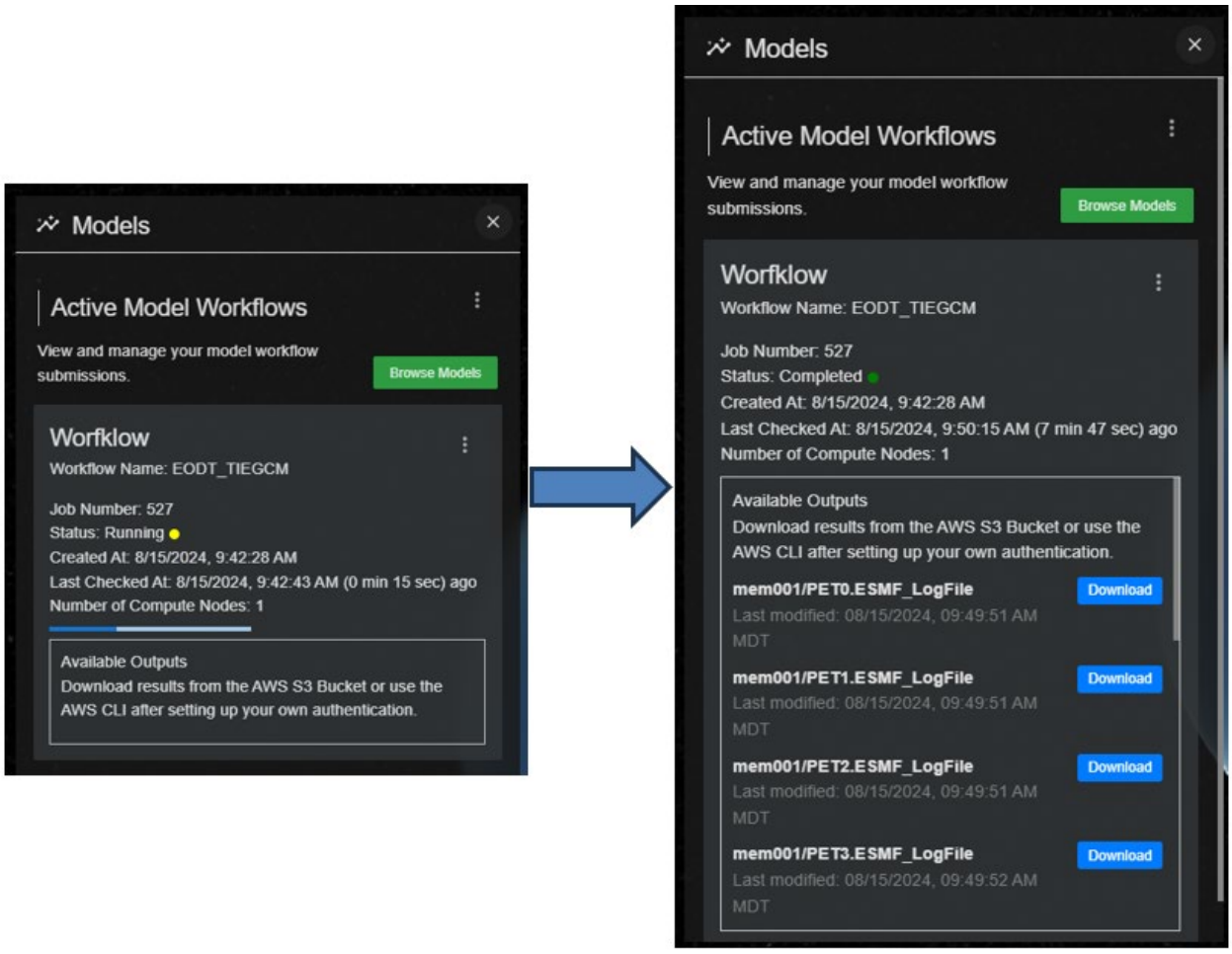

Figure 78. (Left) UI for an in-progress HPC workflow tracked by the client-side polling system. (Right) UI for a completed HPC workflow for TIE-GCM 2.0.

Upon total completion, the polling system on the frontend terminates and we use our AWS S3 credentials to display links to the output files from the workflow. We use our bucket credentials to pre-sign these URLs so that the files can be downloaded from the browser with one click. The files that get returned will vary depending on the model.

The capability to submit user-driven HPC jobs to run on-demand in the cloud is a major component of enabling "what if" scenarios. The digital twin system can create HPC workflows combining these jobs together to string together processing chains based on varying inputs and outputs. The semantic meaning of these workflows must be tied together, which can be done through fixed "hard-coded" use cases (such as jobs with known parameters to be specified by the user), or potentially, with additional work, dynamically driven by natural language processing (NLP) and knowledge meshes (KM). In this case, the large language model could decompose the user query into a series of agentic requests where each agent could handle one aspect of the request. The

effort required for this work is described in the KM/NLP BAA. Either way, the capability for digital twins to manage cloud-based HPC jobs is a major contributor to "what if" hypothetical scenario exploration.

4.2.6.6 RAG LLM Chatbot Service

AI/ML assisted services empower operators with advanced tools and capabilities for extracting actionable insights from the EO-DT platform. Applications of LLM chatbots have only seemed to increase with the growth in generative AI over the past few years. At their core, an LLM is essentially a program that is very good at predicting the next word in a sentence. However, LLM hallucinations, when the system produces outputs that are coherent and grammatically correct but factually incorrect or nonsensical, are a serious problem for these systems.

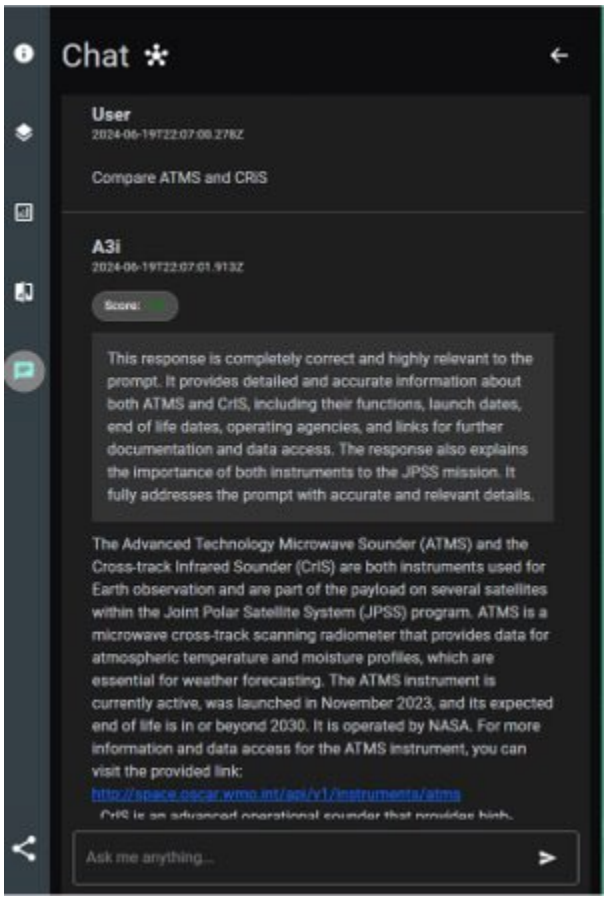

Figure 79. LLM response with evaluation metric.

The EO-DT RAG LLM service provides an interface for users to gain new insights into their data by asking questions like "Tell me more about X…" or "Compare X and Y…". Often times, the names of datasets and their sources are abbreviated which is one issue for LLMs when trying to interpret the semantic meaning of a user's question. This is where RAG comes in to help – by providing the system with a document store of contextually relevant documents, the LLM is able to learn things like abbreviations or URL formatting as we can see in Figure 79.

We also implemented a Faithfulness and Hallucination LLM evaluation metrics to provide a measure of LLM response trustworthiness to the user (Figure 79). New research indicates that using rubric evaluation provides more nuanced metrics of response trustworthiness by using a second LLM to evaluate the first LLM's response. We define a rubric to assess response completeness and accuracy with both a numeric score (0-5) and a written justification for the score. These are all measures taken to enhance the trustworthiness and explainability of the LLM.

Another benefit of implementing RAG in our LLM is the ability for users to ask to see certain data from a specific date and time (in plain English). We provided the LLM with references to the proper URL formatting given a data source, id, and datetime. This enabled the system to replicate this format and provide users with URLs that matched the required schema of our tileset server. The figure below shows an example of the user asking to see relative humidity data from February 14, 2024 at 4:50am UTC. The LLM provides some basic information about the data as well as an interactive link which the user can click to open the data in a new window.

One of the challenges for URL formatting comes from the variety of ways that dates and times can be formatted, as well as any timezone conversions needed. Another piece of implicit logic that the model needed to replicate is the ability to generate correct timestamps for arbitrary times. In other words, each data source is produced at different intervals – GFS tilesets every 6 hours, our MSIS tilesets every 3 hours, etc. Therefore, when the user provides a time with their query, the system must also generate the correct time for the URL, with respect to the dataset they are querying.

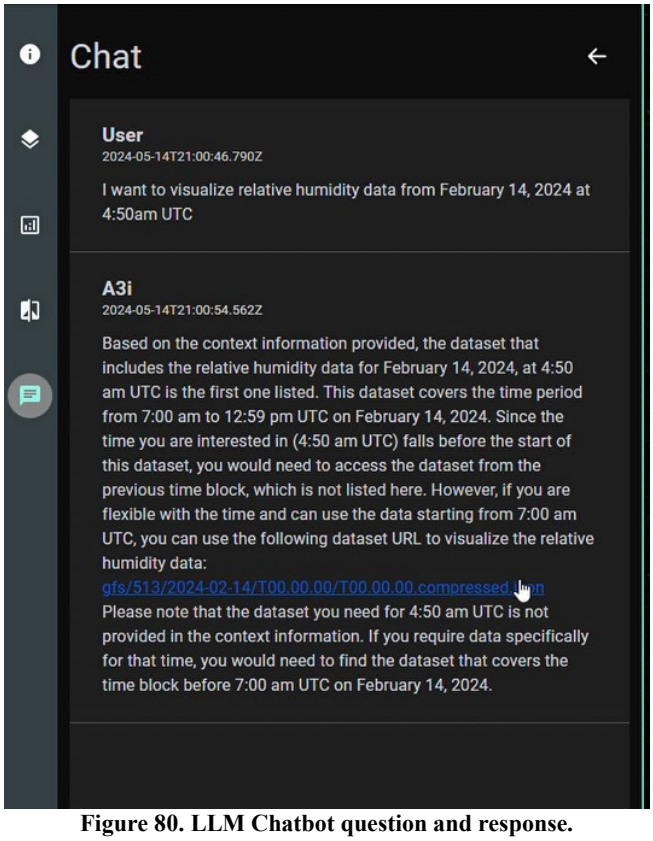

Figure 80. LLM Chatbot question and response.

Because this is a prototype system, we designed LLM generated URLs to be opened in a separate window in a sidecar fashion. This was primarily due to the inconsistency of the URL formatting

from the model in the beginning, which occasionally produced incorrect URLs. However, we can see a full example from the query in the figure above which results in the expected dataset in Figure 80. RAG LLM chatbots are crucial tools for helping users derive actionable insights from the platform.

4.2.6.7 Orbits Service

An Earth DT can benefit from visualizations of not only satellite observation data but also the orbits of space objects themselves. The near-Earth space environment can greatly impact the trajectories and operations of spacecraft, so combining the ability to visualize Earth observation and model data with their orbits is very useful. To visualize orbits, we implemented a three-step process: pulled state information (in this case Two-Line Element sets, or TLEs) from publicly available sources such as space-track.org, connected the UI to propagators via API requests that propagated this state information over a time interval, and then plotted the resulting spacecraft locations as a polyline in CesiumJS.

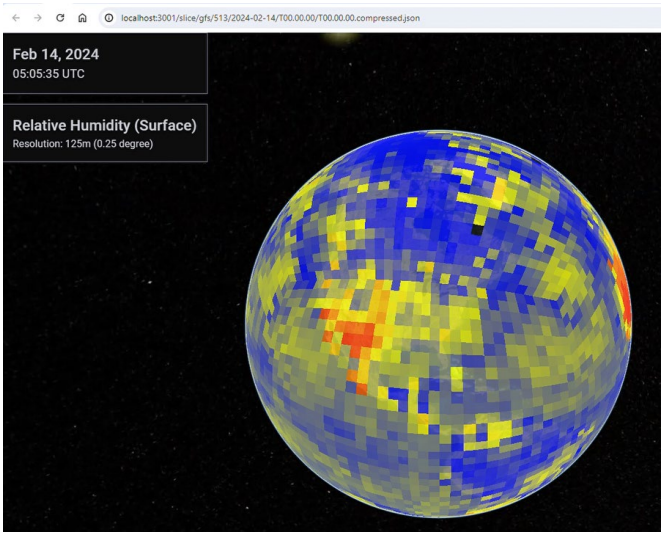

Figure 81. Resulting tileset from LLM query visible in the UI.

The two propagators that we used were a publicly available SGP4 (simplified general perturbation model) library and OSS' own custom propagator, called Astrolabe. SGP4 is a standard propagator that has freely downloadable libraries in multiple languages from places such as space-track.org. It is the U.S. Space Force (USSF) standard and widely used, and it is meant for propagating TLEs with essentially a hard-coded, simple drag model. Astrolabe has the capability to use different density models as drag backgrounds for propagation, allowing for comparison and validation of thermospheric density models that pick up on changing space environmental conditions more accurately than a simplified model such as SGP4. We wrapped Astrolabe in an API that allows a user to pass the desired density model with the propagation request and set up the UI to make this request with both the NRL Mass Spectrometer Incoherent Scatter radar model (MSIS) and Dragster, OSS' assimilative thermospheric density model. We requested TLEs from space-track.org for dozens of satellites and propagated them using Astrolabe with the aforementioned density models as well as with SGP4, as shown in Figure 83. Other commercial propagators such as those in the orbit determination / satellite toolkit (ODTK/STK) from AGI should be able to use multiple background density models as well for these types of comparisons (Orbit Propagators for

Satellites, 2024). If a particular density model offers uncertainty, the spacecraft could be propagated with this uncertainty in density to get an uncertainty in position after some amount of time. We also implemented the ability to visualize such an uncertainty, appearing as an outlined ellipsoidal "error bubble" around the spacecraft's position (Figure 82). This uncertainty is not available for any density model that OSS has access to at the moment, so the bubble we have rendered so far is only a template, but it is an area of focus in thermospheric models right now and planned to be included in Dragster.

In addition to propagating with multiple density models, the user can combine the propagated spacecraft visualization with Earth observation datasets to show the spacecraft flying above or through this data. An example is flying a spacecraft through a thermospheric density model to see where drag on the spacecraft might be higher or lower. With the split-screen feature (described below), two different density models could even be compared, allowing a user to see the difference between the densities at each location in space as well as with the effects of propagating the spacecraft with the different models. An example of this feature is shown in Figure 84.

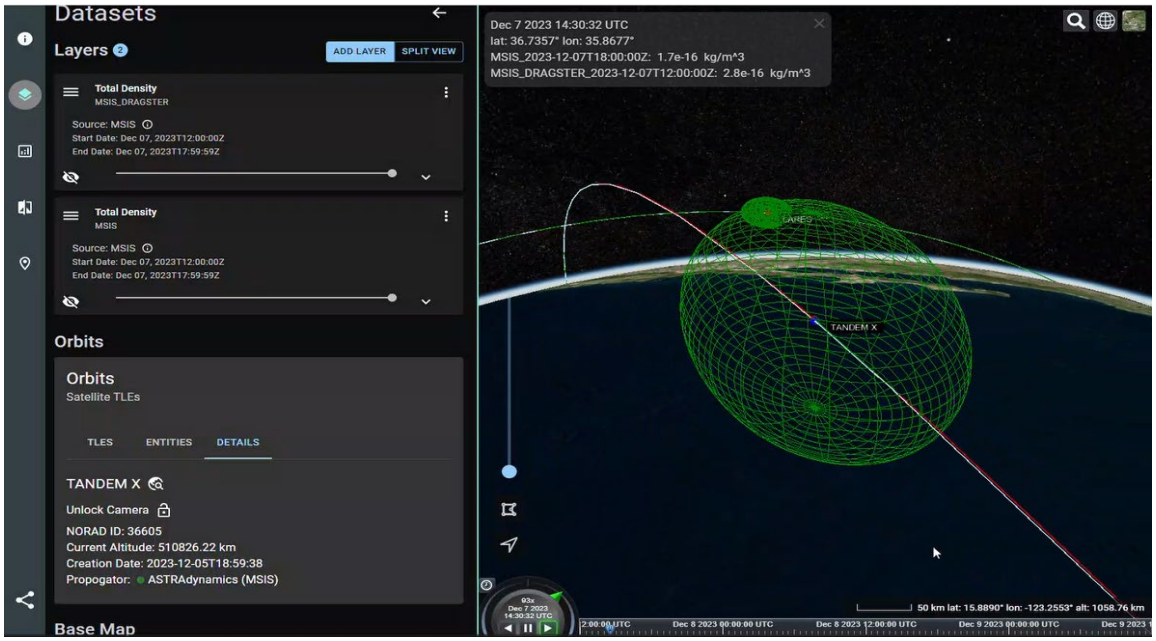

Figure 82. Orbit of TANDEM-X satellite with sample positional uncertainty bubble.

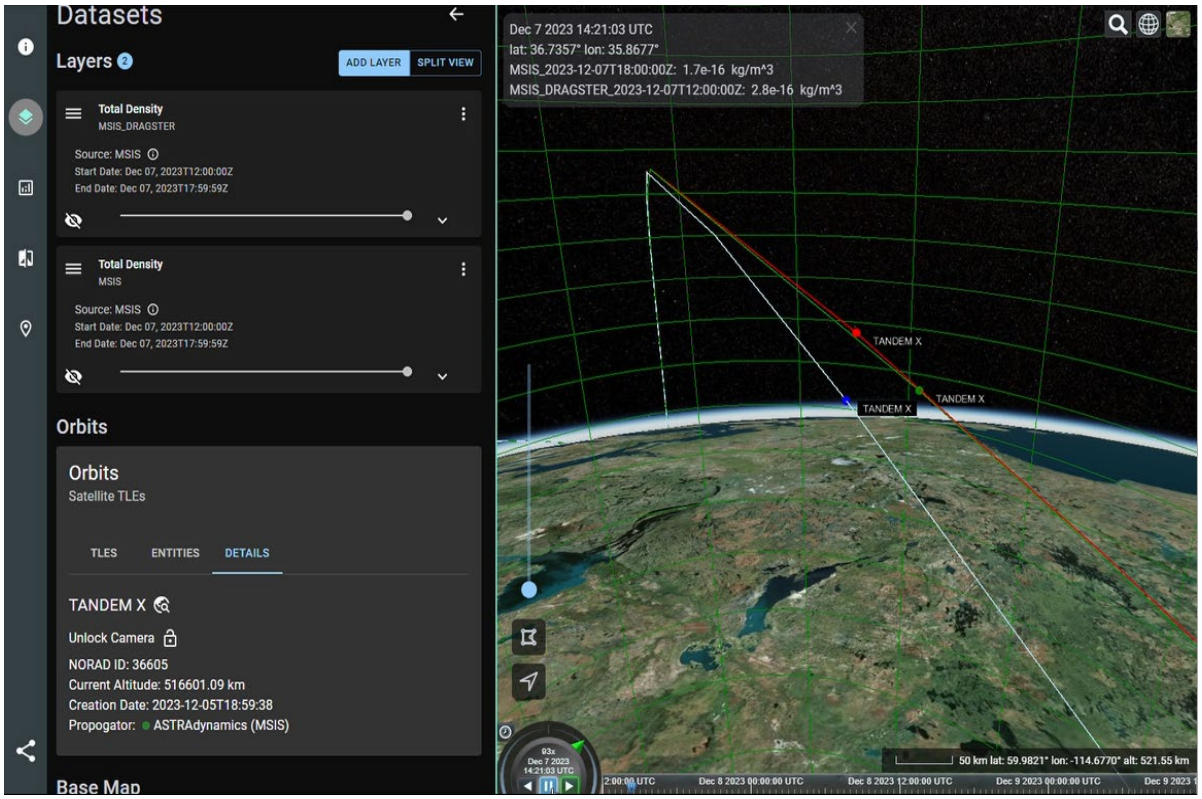

Figure 83. Orbit propagation of the TANDEM-X satellite with three different propagators, SGP4 (white line / blue dot), Astrolabe with MSIS background (green line/dot), and Astrolabe with Dragster background (red line/dot).

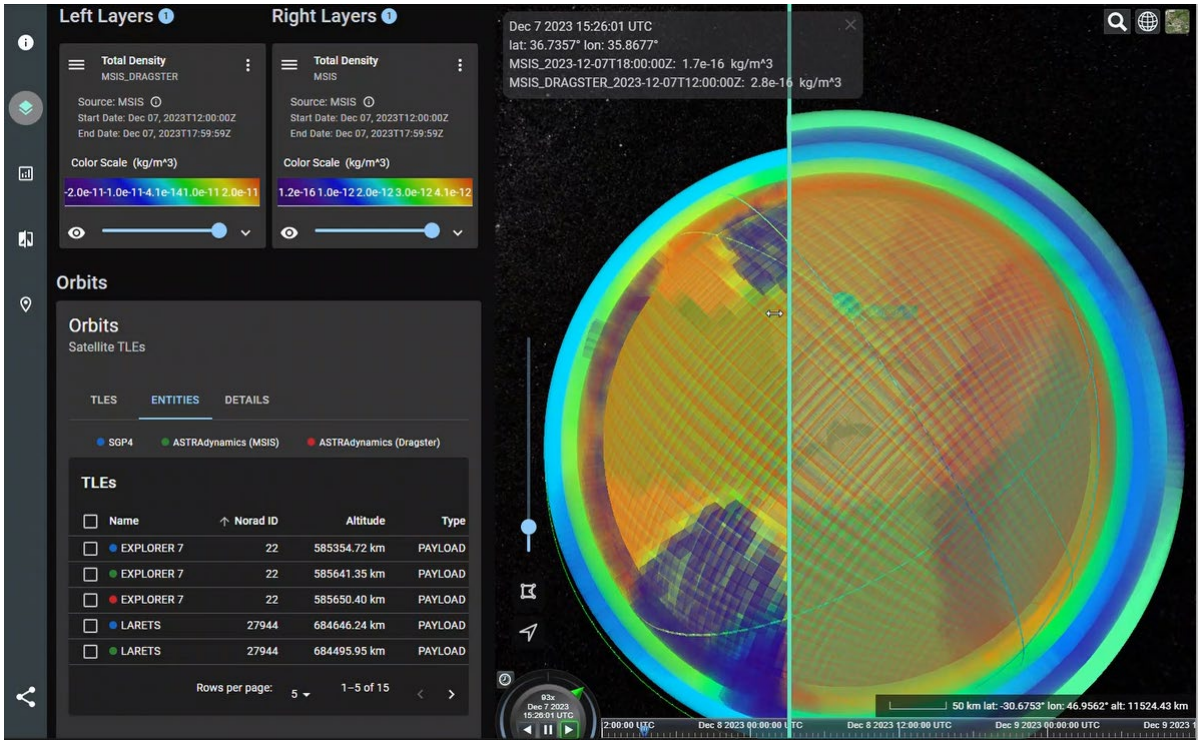

Figure 84. Visualization of TANDEM-X orbit with two density models in split-screen; Dragster (left) and MSIS (right).

4.2.6.8 Tileset Deltas Service

Another feature that we added was the ability to visualize the "delta", or difference, between two datasets. This is useful, for example, if a user is interested in the difference in GFS-reported surface temperature between a time of interest and two weeks prior, or if they are interested in the difference in TIE-GCM model output when the solar index inputs are adjusted by ten percent. This involves taking the difference between the datasets at each point on the grid and rendering a tileset that contains just those differences. A major difficulty with this feature is the fact that not all datasets are on the same grid. Another is that generating an entirely new tileset with delta values takes a few minutes, making it too slow to do upon the user's request in the UI. We have come up with two potential solutions to these concerns and implemented them in our prototype. Delta visualizations are shown in Figure 85 and Figure 86.

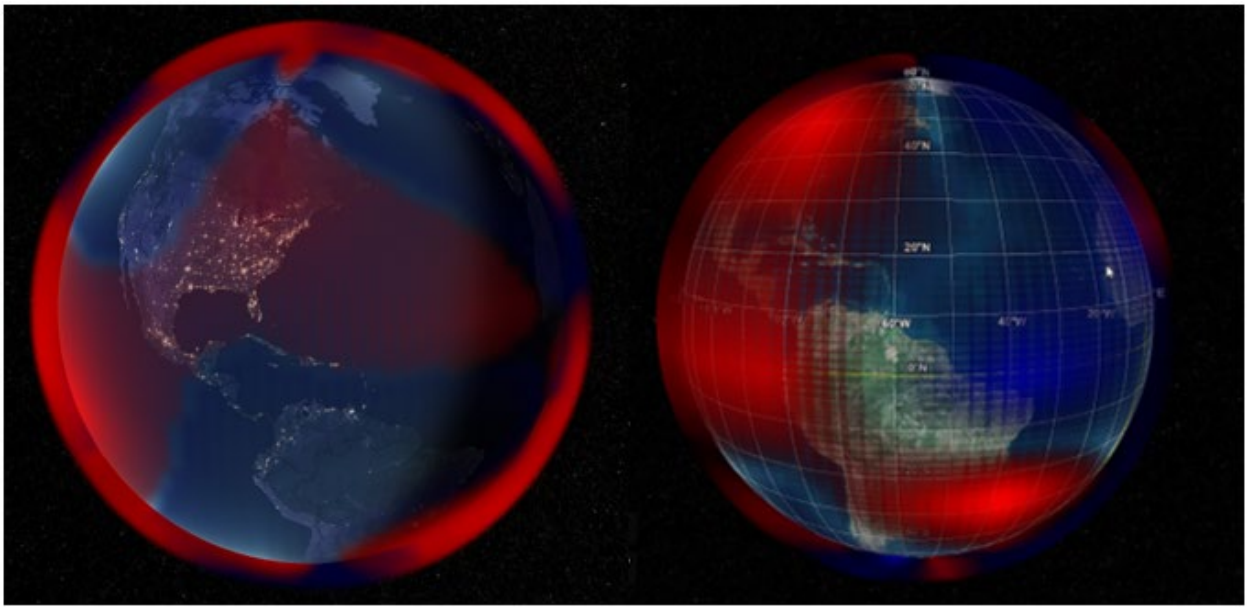

Figure 85. Additional "delta" tileset visualizations, with both point cloud (left) and voxel (right) rendering methods.

The first possible way to display deltas is by performing temporary changes to an existing tileset. We do this by making a pseudo-tileset; that is, changing the data values of an existing tileset and re-implementing the color shading. During the generation of each tileset in our pipeline, we generate a file containing each data point on the visualization grid with its value and location, storing it alongside the 3D-Tiles-formatted tilesets. We used a csv file for this, but greater storage and computational efficiency could be achieved with other data formats, such as netCDF files which are stored in binary format and can be operated on extremely quickly with the netCDF operators (NCO) tool. The UI can access this file and perform a difference at each point, temporarily replacing the data for one of the original datasets with the new delta values. This is only done in the UI and does not rewrite the tileset, so it does not remove any existing data, but it also must be performed anew each time the user would like to see the delta between these datasets, and the page must be refreshed to return to the original dataset. This cannot solve the grid mismatch problem, but it is the quickest and simplest solution to visualizing deltas between two similar datasets "on the fly". It would work for the use cases described above, i.e. two different dates of the same satellite data product or two different runs of the same model.

Another way to display deltas would be to separately generate “delta” tilesets for dataset pairs so that they are available to the user alongside the original datasets themselves. This would remove the burden of difference calculations from the UI and instead bake it into the data processing pipeline as tilesets are generated from incoming data. In this way, processing scripts could be generated to do necessary the interpolations for datasets that do not have the same native grid or resolution. In addition to not being on the fly, this option has limitations, as generating a delta tileset for every possible pair of datasets available in the DT would quickly become untenable. However, it could be done automatically for pairs of interest that have a high likelihood of being requested. We did this for a few days’ worth of data for two different thermospheric density models (MSIS and Dragster, see section 4.2.6.7 for more on these models) in our prototype.

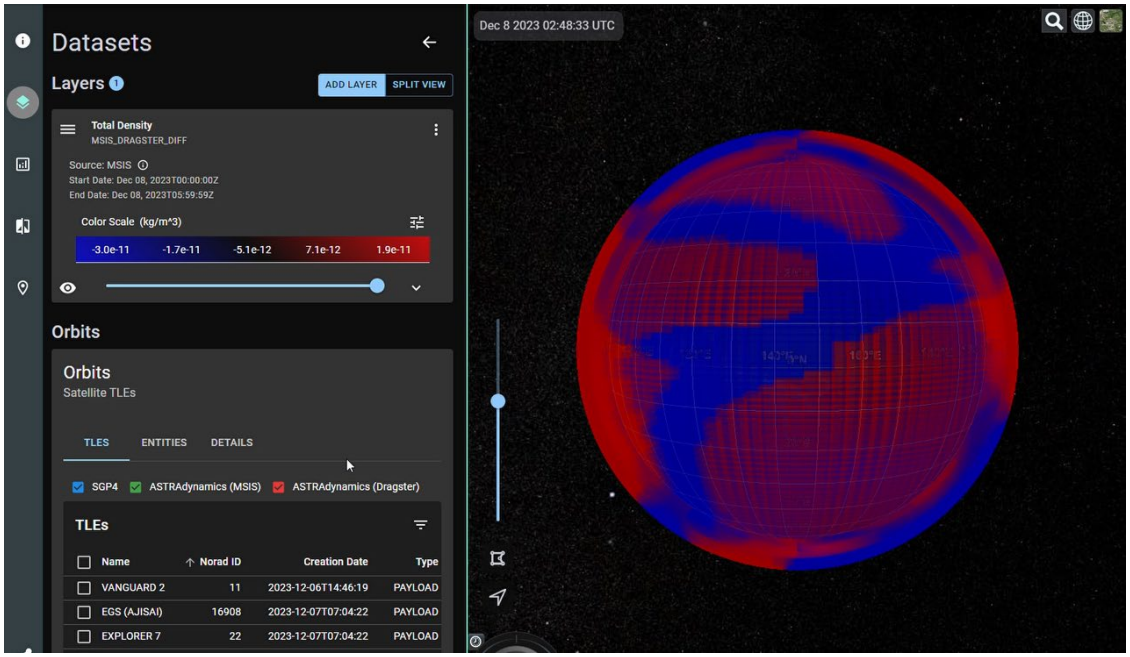

Figure 86. Example of a "delta" tileset showing the difference between Dragster and MSIS densities on Dec. 8, 2023, with the color bar for the difference values to the left.

An ideal “deltas” capability would be on the fly, available for any pair of datasets, and not taking more than a few seconds to run. If possible, implementing our first solution with an efficient storage format for the data values and interpolation capabilities in the UI to handle mismatched grids would be our recommended solution. However, a combination of the two prototyped solutions may be the most reasonable course of action for an operational digital twin. Separate tilesets could be generated with the differences between datasets that are anticipated to be commonly requested, and other requests for datasets on the same grid could be done on the fly.

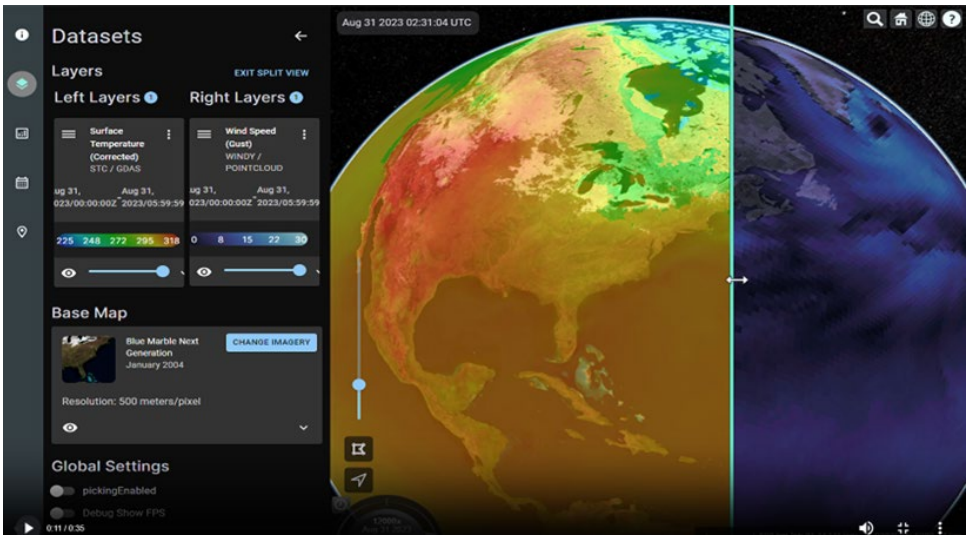

Figure 87. Split View within CesiumJS.

4.2.7 Opportunities for Improvement

There are several outstanding issues with the software that we did not have time to resolve. These were minimized from occurring and will continue to be worked on until code submission. Each of these problems are fairly esoteric and have paths for resolution.

4.2.7.1 Geometric Error and Screen Error Issues

When display data stored using hierarchical level of detail, you must specify the resolution of the data at each level within the dataset. This resolution is known as geometric error and is used by the client to determine when to load the next level-based on the screen resolution and desired accuracy. Geometric error is typically set using uncertainty of the underlying data at each level, but because of the curved nature of many scenes and prefetching by the client – these configurations can be a challenge to set. Further work to optimize the geometric error settings is needed to properly load the data with resolution just exceeding the screen resolution of the current view (adjusted after any movement within the scene). Incorrectly loading a high-resolution level too early can result in large slow down and corresponding drop in framerate.

Similarly, the screen space error must be defined for CZML, time dynamic data sources which seems to have issues within CesiumJS. When zooming to moving objects within high resolution point clouds, we occasionally see objects or datasets disappear and then fail to rerender upon unzoom. We will actively engage with Cesium if we determine a simple reproduce case for duplicating this bug.

4.2.7.2 Custom Shader to Engine Shader Interactions

There are several rendering issues that we believe are due to interactions between the custom shaders written to draw volumetric data and the build in shaders used by CesiumJS. These issues appear as odd rendering artifacts which will sometime hide or show datasets. When moving underneath a dataset (lower in altitude than the data), sometimes the dataset will disappear. Likewise, when on split screen view, the datasets occasionally disappear or miscolor. We have

minimized their occurrence, but work should be done to consolidate all shader work into a single source rather than splitting between data shaders and Cesium scene shaders.

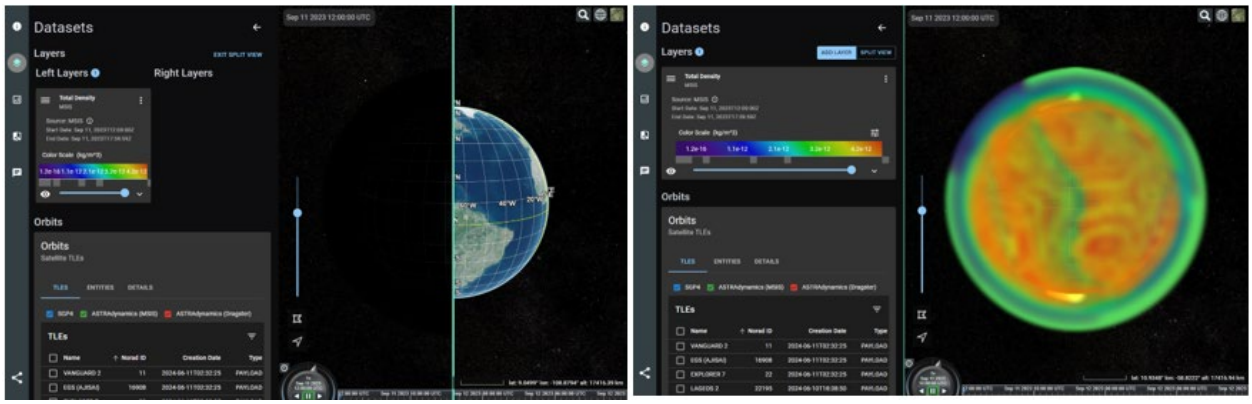

Figure 88. Shader issues. Left: missing rendering on split view. Right: miscoloring of data.

4.2.7.3 Generalized Processing

The processing that we have implemented in our DT is restricted; that is, we are only able to generate tilesets for datasets that we have specifically introduced. This is because each dataset has a different format and needs a custom script to open it and extract the desired variables. While it would be useful to have a fully general processor for Earth observation data, this turned out to not be possible. However, some generality could be added by tweaking some parts of the processing script (for example, passing in the desired variable name in the request to tile a dataset in netCDF format, allowing the processing script itself to handle any netCDF file with the same dimensions).

4.2.8 Future Development

There are numerous improvements that can enhance the usefulness and reusability of the digital twin in the future. Some opportunities for future development that the OSS team has identified are listed here.

4.2.8.1 OpenEO

OpenEO is a set of standards for APIs which support earth observation information as tiles. This standard includes folder structuring similar to the 3D Tiles format, but also addresses the metadata representation of the datasets for the client to query, so can be used to standardize the client interaction with the server for querying data sources. Through observations of and discussions with the digital twin community, the OSS team determined that openEO was a frontrunner for Earth observation standards and worth implementing, but full implementation of the openEO standard was not possible within the period of performance of this contract.

4.2.8.2 OGC API

The Open Geospatial Consortium (OGC) API is another open-source specification similar to openEO which aims to standardize the querying of spatial data on the web. According to the documentation, the API provides the building blocks to create, modify, and query features on the web. Features in this context is defined as real world phenomena that is of intereset. The OGC

API standard is especially interesting because OGC is the same group that maintains the 3D Tiles specification, which would ideally compliment the OGC API. Open standards such as the OGC API and OpenEO are important not only for collaboration and disseminating information, but are also highly recommended for operationalizing any EO-DT platform.

5 Recommendations Summary

We have developed a series of recommendations to follow if developing a digital twin for Earth Observations. These were learned through our development of our digital twin, and most are concerned with maximizing the reuse of code and keeping the formats open to get the most community involvement.

5.1 Specific digital twin recommendations:

- For all software tools and processes, we recommend using open-source software tools and engines. The geospatial community has many open-source libraries that meet the needs for most digital twin development and is actively developing standards which align with digital twin design. The Open Geospatial Consortium seems to have significant weight across a broad range of organizations (including NOAA, NASA, USGS, NGA, Maxar, etc.) and is actively facilitating relevant geospatial standards without a paywall.
- We also recommend using open-source formats for data storage and streaming – and particularly, the Open Geospatial Consortium 3D Tiles format. This format can store nearly every data set encountered and maintains continual growth and customization capability for future needs. Using 3D Tiles format makes it possible to stream data from very high-resolution data sets encountered in scientific visualization (even data sets larger than could fit on a computer, like the Google Earth data set which is >3PB). Similarly, it has significant backing by major visualization teams, including graphics companies like Nvidia (Omniverse product), Unreal, Unity. The larger the set of active users in the format, the more full-featured the tool suites and the longer they will be actively developed.
- For the client software, we recommend using web-based tools for viewing data so that no client software needs to be loaded for use. The CesiumJS platform is an excellent open-source platform that is well suited to deliver geospatial based digital twins. It is architected to provide dynamic layers to increase and decrease the data being loaded in a componentized way. Further, it can leverage computational resource on the GPU to speed the manipulation and filtering of data in parallelized fashion for high performance on complex datasets.
- We recommend the automation of the data processing pipeline to continually deliver information as it becomes available using an automated ETL pipeline. Likewise, deploying the processing pipeline using continuous integration and continuous delivery techniques produces the highest level of uptime and maintainability. These tools minimize the manual processing which can create bottlenecks to delivering scientific data to the community.
- We have found that leaving the source scientific data in its original format and producing a second visualization format was most appropriate to maintain visual capability as well as the ability to use existing scientific tools and flows rather than trying to harmonize all data sets for delivery.
- The visualization data format was best stored as specifically measured/modeled data points as point primitives embedded in a hierarchical grid format with multiple hierarchical levels of detail provided the most efficient storage and streaming combination to maintain the necessary fidelity of physical properties. With this implementation, the hierarchical grid is used to partition the data into chunks that can be easily streamed and stored, and the scientific data can be kept in its original grid with copious metadata – but in a container format that can be easily manipulated by the client.

5.2 ML Recommendations

- Provide and maintain a toolkit of ML based data processing algorithms for users. Algorithms that allow for common ML based data processing drastically speed up the process to development and encourage community interaction. The toolkit should enable the user to download and process data

over specified timespans to obtain an ML ready dataset. It should have the same interface for all filetypes and data formats, although it will necessarily yield dataset dependent data structures.

- Consider implementing custom encoding algorithms for individual datasets as part of the data processing toolkit, or as part of the ML data processing chain. Certain data sources, such as the CrIS L1b data, are particularly amenable to accurate encoding techniques. Classical algorithms, such as the PCA or DCT can be readily implement by users in their own data processing chains; however, by providing algorithms for users, they can be more easily implemented, can be standardized for computational efficiency and can be implemented with optimized or stored parameters for each dataset. Machine learning based algorithms, such as a VAE, can be developed by NOAA developers, potentially perform better than classical algorithms and provided as operational models for users to leverage in their training sessions. We found the volume of data that is generated by individual datasets and instruments can make training machine learning models a very computationally expensive task. Providing users a method to reduce the dimensionality of the data—without sacrificing the quality of the data—can decrease training times and potentially improve generalizability of models by allowing them to be trained on larger datasets.
- Utilize Explainable AI methods for understanding machine learning models. Specifically, we recommend the exploration of KANs [reference section] for the development of models with few input and output parameters. KANs are a novel tool for the extraction of closed form expressions of relationships learned from data. Their architecture allows for the model to extract a complex composition of learned univariate functions. By extracting an explicit learned formula, with an elementary form, it is easier to understand and analyze the nature of the learned relationship. However, we found it difficult to utilize the KAN architecture on datasets with many input or output parameters.
- Experiment with model architectures. There is a plethora of model architectures that can be used for each task or use case. While there are common themes, such as using convolutional networks for image based or special datasets, there is no prescribed methodology for choosing a model architecture. We recommend that when developing models to be deployed and accessible for users, that simple architectures are tested first. As the need for accuracy or efficiency becomes apparent, iterating on model architectures with features such as residual or densely connected layers can be beneficial. When necessary and relevant, cutting-edge architectures should be implemented, but may require more effort to tune and develop.
- Implement ML pipelines for automated training, evaluation and deployment of models. Each of the steps within the model development lifecycle can take a significant amount of time. Automating these processes will enable more efficient development of Machine Learning models. Core to this automation is the standardization of model evaluation and reporting techniques, including the development of qualitative and quantitative evaluation techniques and metrics which can help to generate trust deployed ML models. Our experiments did not focus on this end-to-end implementation of Machine Learning infrastructure; however, we are confident that establishing this infrastructure will lead to dramatic long-term increases in efficiency.
- Establish and utilize an MLOps framework that enables streamlined development of models, and model deployment pipelines.We believe that a robust MLOps platform should maximize the user experience of the developer through a powerful infrastructure. Through this project we explored necessary tools to enable the training and inference of ML models and we recommend the following:
    - Select a trusted third-party tool that streamlines the ingestion of data, data cleaning efforts, and data preprocessing.
        - We looked at KubeFlow, but Azure Synapse, Airflow, and Databricks are all useful data tools.
    - Select a trusted third-party tool that enables model development, training, testing, evaluation, and deployment.

        - Again, we looked at KubeFlow, but Azure Machine Learning and SageMaker are great options as well.
    - Prioritize necessary compute and prioritize hybridization
        - We selected ParallelWorks for our high compute jobs such as model training and we recommend the same. Their platform is highly competitive, and their multi-cloud support is a differentiator.
- Collaborate with data experts. A core principle we recommend is a deep synchronicity between the data experts and the model developers. The data experts should guarantee the model developers have a clear understanding of the data being used and the proposed goal of the model while the model developer should ensure the data expert has a clear understanding of the techniques used for processing and model design.

6 Program BAA Fulfillment

- A flexible digital earth twin system, that allows the user (or public in general) to easily access current and past environmental data, as described in section E. ✓
    - We used data from Atmosphere, Ocean, and Space Weather components described in section E. Our automatic pipelines allow for visualization of current data as it becomes available, and any past datasets can also be processed and visualized.
- A modern visualization toolset to allow the display of the DT data as well as the environmental time series/trends. ✓
    - Cesium!
- The EO-DT system should be able to ingest several input files as described in section F. ✓
    - See data sources section
- The gridding of the data should be flexible (uniform and non-uniform gridding) ✓
    - Point clouds take points as is; we also did regridding work for voxels
- The grid value computation should be based on a mechanism of fusion of data (using ML and/or CV or other efficient techniques) from multiple sources. ✓
    - Sea Ice Concentration estimates from ATMS and CrIS
- Effort should be made to standardize the EO-DT with other DT efforts (such as those in NASA) to move toward interoperability, common interfaces, etc. ✓
    - Include openEO research/work
- The EO-DT output should be at least compatible with expected format for major NOAA systems. In particular GSI and UFS. ✓
    - UFS UPP (Unified Post Processor) outputs GRIB2 files, which we can tile
    - GSI seems to be able to use/output binary or netCDF files
    - NOAA seems to care more that our ML outputs can go into these
- The digital earth twin information should meet the following parameters
    - Geographic coverage: Variable, by default Global ✓
        - We have tiled global (most) and sparse (CrIS, sea ice) data
    - Resolution: Variable, by default: 10 kms ✓
        - Hierarchical resolutions
        - STC claimed GFS was 10 km in their final report
    - Refresh Rate: Variable, sub-hourly ✓
        - Pipeline isn't running this often but we can do it
    - Precision/uncertainty: Should be roughly equivalent to known precision/uncertainties of variables covered by the EO-DT system ✓
        - We aren't changing the values at all

    - Validation methodology: Up to Offeror. Suggested: Simple comparisons to NWP fields and other models. ✓
        - Validation is mostly relevant for AI part as other processing retains original data values (potentially super-sampling?)
    - Duration: Suggested: Moving window of 2 weeks for DT ✓
        - Original data stored between 1-30 days, tilesets planned for 2 weeks
- A computer processing time: Less than 10 mins to process one full global DT ✓
    - Need to test current tiling process, expect we are just under for 5 levels of GFS, though code is not optimized
- The digital earth twin code should be flexible, documented and easily expandable later to add additional parameters ✓
    - As processing for additional datasets is added, it can be indicated in API req
- Software location: run either cloud based or on-premise ✓
    - Containers can be run anywhere
    - Have both components (pipeline/tiler on prem, PW for model runs)
- Code language: Up to offeror: Suggested approach is to seek compatibility as much as possible, with Government systems ✓
    - Data processing is in python
- Others: This project should include a report documenting the lessons learned from this project and recommendations on how to proceed with increasing the parameters included in the digital twin and moving a digital twin (this or future) to operations. ✓

7 Program Management and Execution

7.1 NOAA/OSS Engagement

OSS proposed to the opportunity on Aug 2, 2022 and was awarded the contract on Sept 23, 2022. OSS engaged in numerous meetings throughout the program, beginning with a kickoff presentation Oct 11, 2022, and concluding with a Final presentation on Sept 11, 2024. Monthly progress meetings were made to varying sized teams throughout the program. Quarterly progress reports were delivered every 3 months electronically.

Our previous mid-year progress meeting was held on Thursday March 9th, 2023. Monthly progress demonstrations were held on April 13th, May 11th, June 8th, and July 18th. Internal Project meetings have been held every Monday, and Technical Meetings have been held the second Tuesday of every month. These meetings were held to communicate strategy, discuss technical approach, and evaluate risks and mitigations.

8 References

About OpenEO. (2024, June 17). Retrieved from openEO: https://openeo.org/about.html

API Reference. (2024, June 21). Retrieved from Scikit Learn: https://scikit-learn.org/stable/api/

Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein GAN. *arXiv*. Retrieved from https://arxiv.org/abs/1701.07875

Bodner, A. D., Tepsich, A. S., Spolski, J. N., & Pourteau, S. (2024). Convolutional Kolmogorov-Arnold Networks. *arXiv*. Retrieved from https://arxiv.org/abs/2406.13155

Brovelli, M. H. (2016). NASA WEBWORLDWIND: Multidimensional Virtual Globe for Geo Big Data Visualization. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume XLI-B2*, 563-566.

Cervenec, J. e. (2018). *Fluid Earth, an Educational Tool for Visualizing Earth's Atmosphere, Oceans, and Permafrost.* the Education and Outreach Group at the Byrd Polar and Climate Research Center (BPCRC), The Ohio State University (OSU).

Cesium. (2024). *Cesium Bathymetry*. Retrieved from Cesium.com: https://cesium.com/platform/cesium-ion/content/cesium-world-bathymetry/

Chase, H. (2022). LangChain. Retrieved from https://github.com/langchain-ai/langchain

Chollet, F. a. (2015). Keras. Retrieved June 21, 2024, from KERAS: https://keras.io/

Cozzi, P. (2016). *WebGL Insights.* Boca Raton, FL: CRC Press.

Devarakonda, A., Naumov, M., & Garland, M. (2018). AdaBatch: Adaptive Batch Sizes for Training Deep Neural Networks. *arXiv*. Retrieved from https://arxiv.org/abs/1712.02029

GEBCO. (2024). *GEBCO Bathymetry Data*. Retrieved from Gridded Bathymetry Data: https://www.gebco.net/data_and_products/gridded_bathymetry_data/

GeoData. (2012). *cesium-terrain-builder github*. Retrieved from github.com: https://github.com/geo-data/cesium-terrain-builder

Google, I. (2015). *S2 Geometry.* Google, Inc. - https://s2geometry.io, https://github.com/google/s2geometry.

Górski, K. M., Hivon, E. s., Banday, A. J., Wandelt, B. D., Hansen, F. K., Reinecke, M., & Bartelmann, M. (April 2005). HEALPix: A Framework fro High-Resolution Discretization and Fast Analysis of Data Distributed on a Sphere. *The Astrophysical Journal, Volume 622, Issue 2*, 759-771.

Goyal, M., Tatwawadi, K., Chandak, S., & Ochoa, I. (2018). DeepZip: Lossless Data Compression Using Recurrent Neural Networks. *2019 Data Compression Conference (DCC)*, 575-575. Retrieved from https://api.semanticscholar.org/CorpusID:53775327

Harris, L. C. (2021). *A Scientific Descrption of the GFDL Finite-Volume Cubed Sphere Dynamical Core.* Geophysical Fluid Dynamics Laboratory, NOAA.

Huag, G. e. (2017). Densely Connected Convolutional Networks. *IEE Conference on Computer vision and Pattern Recognition (CPVR)*, 2261-2269.

Inc., U. T. (2018). *H3: A Hexagonal Hierarchical Geospatial Indexing System.* https://github.com/uber/h3.

Kubeflow. (n.d.). Kubeflow. Retrieved from kubeflow.org

Kubernetes. (2023). *Overview*. Retrieved June 21, 2024, from https://kubernetes.io/docs/concepts/overview/

Liu, J. (2022). LlamaIndex. doi:10.5281/zenodo.1234

Liu, Z., Wang, Y., Vaidya, S., Ruehle, F., Halverson, J., Soljačić, M., . . . Tegmark, M. (2024). KAN: Kolmogorov-Arnold Networks. *arXiv*. Retrieved from https://arxiv.org/abs/2404.19756

Meier, W. N., Fetterer, F., Savoie, M., Mallory, S., Duerr, R., & Stroeve, J. (2013). NOAA/NSIDC Climate Data Record of Passive Microwave Sea Ice Concentration, Version 2. National Snow and Ice Data Center. doi:10.7265/N55M63M1

Meier, W. N., Fetterer, F., Windnagel, A. K., & Stewart, J. S. (2021). NOAA/NSIDC Climate Data Record of Passive Microwave Sea Ice Concentration, Version 4. National Snow and Ice Data Center. doi:10.7265/efmz-2t65

NOAA. (2019, Sept). *NOAA Open Data Dissemination (NODD) Program*. Retrieved from Registry of Open Data on AWS: https://registry.opendata.aws/collab/noaa/

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . Perrot, M. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research, 12*, 2825-2830.

Project, O. (2006). *GDAL*. Retrieved from gdal.org: https://gdal.org/programs/gdalwarp.html

Schlegl, T., Seeböck, P., Waldstein, S. M., Langs, G., & Schmidt-Erfurt, U. (2019). f-AnoGAN: Fast unsupervised anomaly detection with generative adversarial networks. *Medical Image Analysis*, 30-44. doi:https://doi.org/10.1016/j.media.2019.01.010.

Schlegl, T., Seeböck, P., Waldstein, S. M., Schmidt-Erfurth, U., & Langs, G. (2017). Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery. *arXiv*. Retrieved from https://arxiv.org/abs/1703.05921

Wang, L., Tremblay, D., Zhang, B., & Han, Y. (2016). Fast and Accurate Collocation of the Visible Infrared Imaging Radiometer Suite Measurements with Cross-Track Infrared Sounder. *Remote Sensing*. Retrieved from https://doi.org/10.3390/rs8010076

Wenwen, L. W. (Mar 2017). PolarGlobe: A web-wide virtual globe system for visualizing multidimensional, time-varying, big climate data. *International Journal of Geographical Information Science*, 1562-1582.

World Meteorological Organization. (n.d.). *Observing Systems Capability Analysis and Review Tool (OSCAR)*. Retrieved June 19, 2024, from https://www.wmo-sat.info/oscar/

9 Appendix

9.1 Glossary of Terms

Term	Definition
GPGPU	General Purpose GPU. Using the GPU to perform non-graphics calculations in parallel.
Hierarchical Grid	A grid which subdivides cells in a consistent manner (often by powers of 2).
Hierarchical Level of Detail	Storing multiple resolutions of the same data so that visualization can more efficiently load the resolution that matches the display resolution of the client.
Implicit Tiling	Storing and indexing of a hierarchical grid to allow for random access of the contained cells to speed spatial queries and to allow for efficient traversal of the data being stored.
Retrieval Augmented Generation (RAG)	The process of optimizing the output of a large language model so that it references an authoritative knowledge base outside the training data set.
Sparse data storage	Storing of a hierarchical grid to more closely match the resolution of the variability or density of the property being stored.

9.2 Gridding/Tile Statistics Tables

We describe some of the statistics for the grids and formats used in the digital twin.

9.2.1 Google S2 Cell Table

Level	Number of Cells	Min Area	Max Area	Units
0	6	85011012.19	85011012.19	km^2
1	24	21252753.05	21252753.05	km^2
2	96	4919708.23	6026521.16	km^2
3	384	1055377.48	1646455.5	km^2
4	1536	231564.06	413918.15	km^2
5	6k	53798.67	104297.91	km^2
6	24k	12948.81	26113.3	km^2
7	98K	3175.44	6529.09	km^2
8	393K	786.2	1632.45	km^2
9	1573K	195.59	408.12	km^2
10	6M	48.78	102.03	km^2
11	25M	12.18	25.51	km^2
12	100M	3.04	6.38	km^2
13	402M	0.76	1.59	km^2
14	1610M	0.19	0.4	km^2
15	6B	47520.3	99638.93	m^2

Level	Number of Cells	Min Area	Max Area	Units
16	25B	11880.08	24909.73	m^2
17	103B	2970.02	6227.43	m^2
18	412B	742.5	1556.86	m^2
19	1649B	185.63	389.21	m^2
20	7T	46.41	97.3	m^2
21	26T	11.6	24.33	m^2
22	105T	2.9	6.08	m^2
23	422T	0.73	1.52	m^2
24	1689T	0.18	0.38	m^2
25	7.00E+15	453.19	950.23	cm^2
26	2.70E+16	113.3	237.56	cm^2
27	1.08E+17	28.32	59.39	cm^2
28	4.32E+17	7.08	14.85	cm^2
29	1.73E+18	1.77	3.71	cm^2
30	7.00E+18	0.44	0.93	cm^2

9.2.2 Standard Latitude Longitude Grid Table

Grid Size	Number of Cells	Area (at equator, approx.)	Units
5.0000000000	2,592	309,800.73062	km²
2.5000000000	10,368	77,450.18266	km²
1.0000000000	64,800	12,392.02922	km²
0.5000000000	259,200	3,098.00731	km²
0.2500000000	1,036,800	774.50183	km²
0.1250000000	4,147,200	193.62546	km²
0.0625000000	16,588,800	48.40636	km²
0.0312500000	66,355,200	12.10159	km²
0.0156250000	265,420,800	3.02540	km²
0.0078125000	1,061,683,200	0.75635	km²
0.0039062500	4,246,732,800	0.18909	km²
0.0019531250	16,986,931,200	47,271.84000	m²
0.0009765625	67,947,724,800	11,817.96000	m²
0.0004882813	271,790,899,200	2,954.49000	m²
0.0002441406	1,087,163,596,800	738.62250	m²
0.0001220703	4,348,654,387,200	184.65562	m²
0.0000610352	17,394,617,548,800	46.16391	m²
0.0000305176	69,578,470,195,200	11.54098	m²
0.0000152588	278,313,880,780,800	2.88524	m²
0.0000076294	1,113,255,523,123,200	0.72131	m²
0.0000038147	4,453,022,092,492,800	0.18033	m²
0.0000019073	17,812,088,369,971,200	0.04508	m²
0.0000009537	71,248,353,479,884,800	112.70485	cm²
0.0000004768	284,993,413,919,539,000	28.17621	cm²

Grid Size	Number of Cells	Area (at equator, approx.)	Units
0.0000002384	1,139,973,655,678,160,000	7.04405	cm^2
0.0000001192	4,559,894,622,712,630,000	1.76101	cm^2
0.0000000596	18,239,578,490,850,500,000	0.44025	cm^2
0.0000000298	72,958,313,963,402,000,000	0.11006	cm^2
0.0000000149	291,833,255,853,608,000,000	0.02752	cm^2
0.0000000075	1,167,333,023,414,430,000,000	0.00688	cm^2
0.0000000037	4,669,332,093,657,730,000,000	0.00172	cm^2

9.2.3 Quantized Mesh Terrain Tiles (ALOS ~30m)

Level	Approximate Number of Tiles	Disk Usage (bytes)
0	2	26,094
1	8	49,309
2	32	115,036
3	128	318,349
4	512	914,118
5	2,048	2,835,989
6	8,192	10,132,376
7	32,768	8,579,292
8	131,072	49,027,528
9	524,288	245,579,446
10	2,097,152	1,126,303,948
11	8,388,608	4,717,496,382
12	33,554,432	18,850,318,655
13	134,217,728	77,582,223,368
14	536,870,912	
15	2,147,483,648	
16	8,589,934,592	
17	34,359,738,368	
18	137,438,953,472	
19	549,755,813,888	
20	2,199,023,255,552	
21	8,796,093,022,208	
22	35,184,372,088,832	
23	140,737,488,355,328	
24	562,949,953,421,312	
25	2,251,799,813,685,250	
26	9,007,199,254,740,990	
27	36,028,797,018,964,000	

9.3 Microservice Software Architecture Diagram

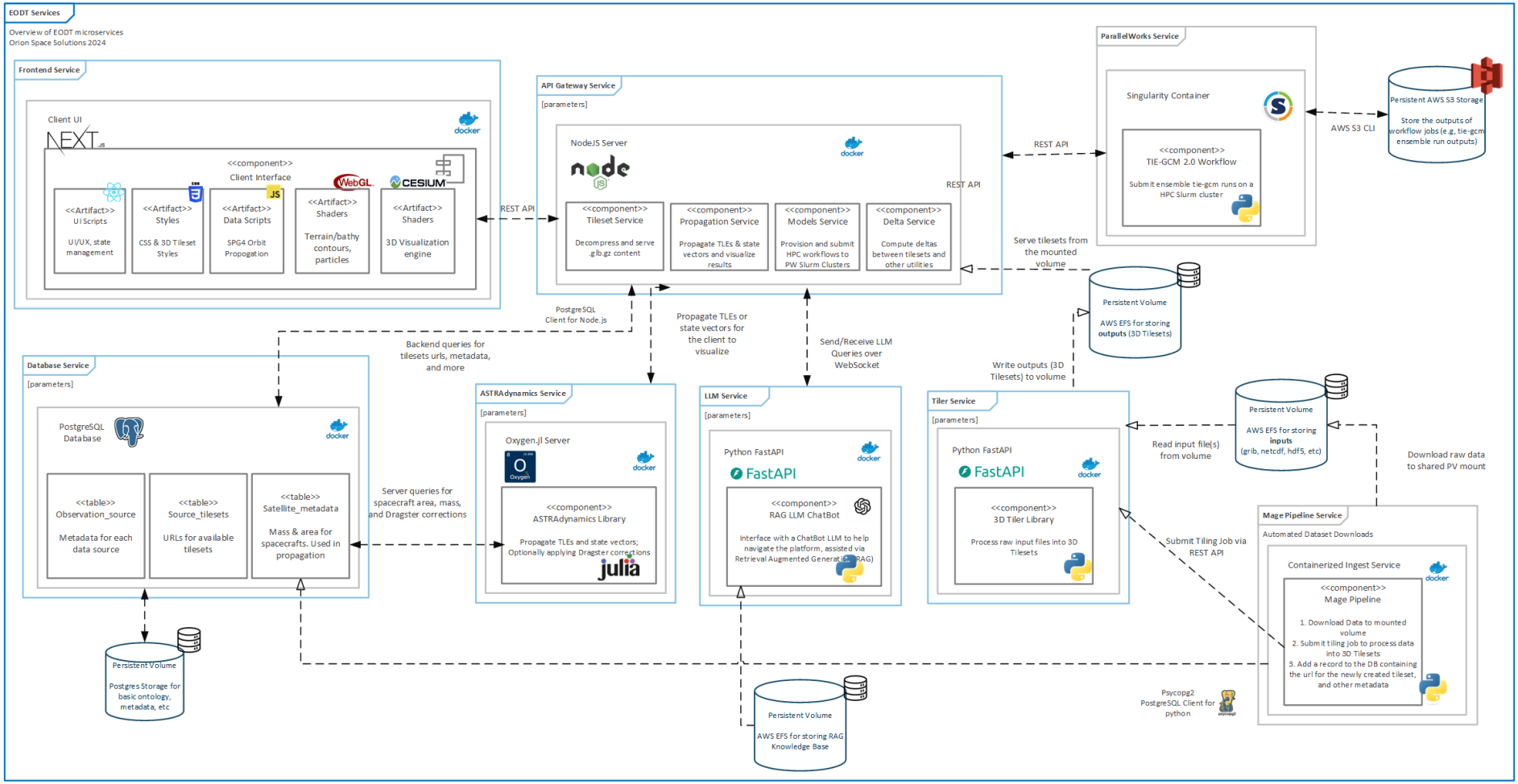

Figure 89. Architecture Diagram for the EO-DT Microservices