Technical Requirements and Dependencies for Satellite Imaging Disaster Monitoring Project

Cloud-Based Platforms (Essential)

Google Earth Engine (GEE)

- **Purpose**: Primary platform for accessing and processing satellite imagery without local storage
- **Setup Requirements**:
- Google account
- Earth Engine account approval (apply at https://signup.earthengine.google.com/)
- No local installation needed (browser-based)
- **Advantages**: Handles petabyte-scale data processing in the cloud, eliminating local hardware constraints

Google Colab

- **Purpose**: Cloud-based Python notebook environment for data analysis and visualization
- **Setup Requirements**:
- Google account
- No local installation needed
- **Advantages**: Free GPU/TPU access, pre-installed libraries, shareable notebooks

Python Environment (Local Development)

Core Libraries

- **NumPy**: Numerical computing (arrays, mathematical functions)
- **Pandas**: Data manipulation and analysis
- **Matplotlib/Seaborn**: Data visualization
- **Jupyter**: Interactive notebook environment (if working locally)

Geospatial Libraries

- **Rasterio**: Reading and writing geospatial raster data
- **GeoPandas**: Working with geospatial vector data
- **Folium**: Interactive map visualization
- **EarthPy**: Tools for working with spatial data
- **Xarray**: Working with multi-dimensional arrays (especially useful for time-series satellite data)

Earth Engine Python API

- **Purpose**: Interact with Google Earth Engine from Python
- **Installation**: `pip install earthengine-api`
- **Authentication**: Requires OAuth2 setup

Cloud-Optimized Libraries

- **Dask**: Parallel computing library for handling larger-than-memory datasets
- **fsspec**: Filesystem interfaces for cloud storage

MacBook Air M1 Considerations

Python Environment Setup

- Use Miniforge (Conda) for M1-optimized packages: https://github.com/conda-forge/miniforge
- Install Python packages with M1 support: `bash conda install -c conda-forge numpy pandas matplotlib jupyter rasterio geopandas folium xarray dask`

Memory Management Strategies

- Use streaming approaches for data processing
- Implement chunking for large datasets
- Leverage cloud processing whenever possible
- Avoid loading entire datasets into memory

Storage Management

- Use cloud storage solutions (Google Drive, AWS S3) for intermediate results
- Implement data filtering at source (in GEE) before downloading
- Consider external SSD for local development if needed

Development Tools

Version Control

- **Git**: Track code changes
- **GitHub**: Host repository and collaborate

Documentation

- **Markdown**: Document project progress and findings
- **Sphinx/MkDocs**: Generate comprehensive documentation (optional)

Specialized Tools for Disaster Monitoring

Fire Monitoring

- **FIRMS Tools**: Fire Information for Resource Management System
- **Python Libraries**:
- `satpy`: Reading and processing satellite data
- `pyhdf`: Working with HDF format (common for MODIS data)

Flood Monitoring

- **SAR Processing Tools**:
- `snappy`: ESA SNAP Toolbox Python interface (for Sentinel-1 SAR data)
- `sarpy`: Tools for reading, processing SAR data

Hurricane/Tornado Monitoring

- **Weather Data Processing**:
- `metpy`: Meteorological data analysis
- `netCDF4`: Working with NetCDF files (common format for weather data)
- `wrf-python`: Working with Weather Research and Forecasting model data

API Access Requirements

NASA Earthdata

- **Account Setup**: Required for accessing NASA datasets
- **Authentication**: API key or token-based

- **Python Library**: `earthdata` package

Copernicus Open Access Hub

- **Account Setup**: Required for accessing Sentinel data
- **Authentication**: Username/password
- **Python Library**: `sentinelsat`

NOAA Data Access

- **Account Setup**: Some datasets require registration
- **Authentication**: API key for some services
- **Python Library**: Various depending on specific data product

Cloud Storage Options

Google Drive

- **Integration**: Native with Google Colab
- **Python Access**: `google.colab` module, `pydrive`

AWS S3 (Optional)

- **Setup**: AWS account (free tier available)
- **Python Access**: `boto3` library
- **Cost**: Pay-as-you-go pricing

Installation Instructions

Google Earth Engine Setup

1. Sign up for Earth Engine at https://signup.earthengine.google.com/
2. Wait for approval (typically 1-2 business days)
3. Access the Earth Engine Code Editor at https://code.earthengine.google.com/

Python Environment Setup (Local - Optional)

```
# Install Miniforge for M1 Mac
curl -fsSL https://github.com/conda-forge/miniforge/releases/latest/download/Miniforge3-MacOSX-arm64.sh -o Miniforge3.sh
bash Miniforge3.sh -b -p $HOME/miniforge3
```

```
# Create environment
conda create -n satellite-monitoring python=3.9
conda activate satellite-monitoring

# Install core packages
conda install -c conda-forge numpy pandas matplotlib jupyter
conda install -c conda-forge rasterio geopandas folium xarray dask

# Install Earth Engine API
pip install earthengine-api

# Install specialized packages
pip install sentinelsat pyhdf netCDF4 metpy
```

Google Colab Setup

1. Go to https://colab.research.google.com/
2. Create a new notebook
3. Install Earth Engine API: `python !pip install earthengine-api`
4. Authenticate: `python import ee ee.Authenticate() ee.Initialize()`

Performance Optimization Strategies

Data Filtering

- Filter data by region of interest before processing
- Use temporal filtering to reduce dataset size
- Select only necessary bands/variables

Computation Strategies

- Use Earth Engine's server-side processing
- Implement progressive loading for visualization
- Cache intermediate results

Visualization Optimization

- Use decimation for large datasets
- Generate thumbnails for quick previews
- Use vector formats (GeoJSON) for lightweight display

Troubleshooting Common Issues

Memory Errors

- Reduce chunk size in processing
- Use generator patterns instead of loading full datasets
- Move processing to cloud platforms

API Rate Limiting

- Implement exponential backoff for retries
- Cache results to minimize redundant API calls
- Use bulk download options when available

Data Format Compatibility

- Convert between formats using appropriate libraries
- Use standardized formats (GeoTIFF, NetCDF) when possible
- Document data structures for consistency