Matthew Lewis

# Satellite Intelligence Processor Breakdown

## **Project Introduction:**

In the domain of geospatial intelligence, where timely insights from satellite imagery can inform critical decisions in defense, disaster response, or environmental monitoring, this project was my attempt to create an automated processor for fetching, analyzing, and visualizing satellite data. Drawing from real-world needs like ISR (Intelligence, Surveillance, and Reconnaissance), I aimed to build a system that could pull historical and current imagery for a given location, detect changes or objects, and generate reports—all in a self-contained app. As a learning exercise, it started with high ambitions, but I quickly ran into roadblocks with low-resolution free satellite data and inconsistent historical archives, which made accurate comparisons unreliable. This turned it into more of an exploratory prototype than a polished tool, teaching me valuable lessons about data dependencies in ML-driven projects.

## **Solution and Core Concept:**

To tackle geospatial analysis, the project planned a pipeline that integrated API fetches from open sources like Sentinel Hub or USGS for satellite imagery, followed by ML-based processing for object detection and change mapping. Users could input coordinates or upload local image folders, triggering automated downloads of multi-temporal data (e.g., before/after images) for anomaly spotting via techniques like pixel differencing or pre-trained models. The core idea was modularity: separate modules for ingestion, preprocessing (e.g., resampling low-res images), analysis, and output via a web dashboard. However, as I iterated, I found that free-tier data often lacked the resolution needed for fine-grained detection, and historical fetches were spotty, leading to incomplete datasets and false positives. This centralized approach was meant to preserve causal links in intelligence workflows, like linking image changes to potential threats, but it highlighted the need for better data strategies in future attempts.

## **Project Goals:**

- Create an automated fetcher for satellite imagery from public APIs
- Implement ML models for object detection and change analysis
- Build a web interface for inputting locations and viewing results
- Generate intelligence reports with visualizations and anomaly highlights
- Allow local folder processing as a fallback for custom datasets
- Enable iteration through configurable parameters like date ranges or detection thresholds

## **Scope Clarification and Non-Goals:**

This project intentionally focused on rapid prototyping to explore geospatial ML, abstracting away production-scale complexities like real-time streaming or high-security encryption. It did not aim to handle classified data, integrate with enterprise GIS systems, or achieve sub-meter accuracy, as those would require premium APIs beyond my budget. These limits were set to keep it educational, but they also exposed gaps—like the unreliability of free historical data for meaningful comparisons—which became clear during development and contributed to pausing the project.

**Tech Stack:**

- Python (core language for scripting and processing)
- FastAPI and Uvicorn (backend API and web server)
- Geopandas, Rasterio, and Fiona (geospatial data handling)
- Scikit-learn and OpenCV (ML for detection and image processing)
- Folium or Leaflet (interactive mapping in the dashboard)
- Jinja2 (HTML templating for reports)
- Docker (planned for containerization, though not fully implemented)

**Implementation:**

The planned structure broke down into a modular pipeline, with directories for app (backend), models (ML scripts), and data (sample imagery):

1. Data Ingestion (app/ingest.py): Fetch imagery via APIs based on user coordinates and date ranges, or load from local folders; handle resampling for low-quality inputs.
2. Preprocessing (app/preprocess.py): Align images temporally, normalize resolutions, and prepare for analysis—where I first noticed alignment issues with inconsistent historical data.
3. Analysis (app/analyze.py): Apply change detection (e.g., differencing) and object models (e.g., YOLO variants) to flag anomalies like new structures or vegetation loss.
4. Reporting and Dashboard (app/main.py, templates/index.html): Serve APIs for results and render a UI with maps, highlighted changes, and exportable reports.

Usage was envisioned as CLI for quick tests or dashboard for interactive sessions, with Docker for easy setup. In practice, fetches worked sporadically, but low-res images led to poor model performance.

**Lessons Learned and Attempted Implementation:**

This was a humbling dive into geospatial ML, where my initial optimism clashed with real-world data limitations. I discovered midway that free historical archives often missed key dates or had artifacts, making comparisons ineffective. While the core fetch and basic detection ran locally, advanced features like dynamic map-click fetching and multi-image folder analysis failed due to API quotas and resolution issues. It remains paused, but the attempt reinforced the importance of robust data pipelines and inspired simpler projects like my telemetry simulators.