Satellite Image ChatBot

Overview

A stateful agentic pipeline for working with satellite imagery (Sentinel-1 SAR and Sentinel-2-like RGB). It combines a deep-learning segmentation models (oil-spill detector, cloud-segmentation), GAN model (cloud removal), preprocessing/postprocessing utilities, file download and metadata-extraction tools, an LLM bound to those tools, a LangGraph agent for streaming tool calls, whisper tool for chatting with audio and a Gradio UI for chat + image preview + file download.

Table of contents

Quick summary

- **What it does:** Accepts user prompts (chat), downloads/reads satellite images, runs preprocessing (equalization, denoising), runs an oil-spill segmentation model, returns images/metadata, and explains results with an LLM.
- **Core tech:** Keras/TensorFlow model, rasterio , opencv-python , langchain + langchain-google-genai , langgraph, and gradio UI.

Security & deployment notes

- **Remove the hard-coded API key** (present in the working script). Use environment variables or secret managers:

```
export GOOGLE_API_KEY="your-real-key"
```

- **Model path** (OIL_MODEL_PATH) points to a Kaggle dataset path. Replace or make configurable for local deployments.
- **Workdir:** the code uses /kaggle/working by default. Prefer a configurable WORKDIR (./data , ./outputs) to avoid accidental writes to root.
- **URL validation**: validate user-supplied download links before invoking downloads.

Requirements

Put these in requirements.txt or install via pip .

- Python 3.9+
- tensorflow / keras (matching model training version)
- rasterio
- numpy
- opencv-python
- pillow
- scipy
- matplotlib
- gradio
- langchain, langchain-core, langchain-google-genai
- langgraph
- gdown (optional — for Google Drive downloads)

Example:

```
pip install rasterio numpy opencv-python pillow scipy matplotlib gradio gdown keras tensorflow langchain langchain-google-genai langgraph
```

Configuration / environment variables

- **GOOGLE_API_KEY** required for Google GenAI LLM access.
- **OIL_MODEL_PATH** path to the Keras model file used for segmentation.
- **IMAGE_PATH** optional default image for tests.
- **WORKDIR** /kaggle/working .

Architecture & components

1 - Oil Spill Detection Model

Purpose:
Detect oil spills in SAR (Synthetic Aperture Radar) satellite images (VV band).

Core class:

Encapsulates loading, preprocessing, prediction, and visualization.

- **__init__(model_path, model_input_shape)**
  - Loads a trained Keras .h5 oil spill model.
  - model_input_shape: (height, width) of model input.
  - Prints confirmation: "Oil spill model loaded."
- **load_image(path)**
  - Reads **VV band (band 1)** from raster file.
  - Returns a float32 numpy array of shape (H, W).
- **preprocess_image(vv)**
  - Resizes VV band to model input shape.
  - Normalizes to [0, 1].
  - Expands to **4D tensor** (1, H, W, 3) by stacking 3 channels.
  - Also returns a **histogram-equalized uint8 image** for visualization.
  - Output: (input_tensor, equalized_preview).
- **predict_oil_spill(input_tensor,water_mask=None,enable_water_mask=False)**
  - Runs model prediction → outputs class map.
  - Handles **binary or multi-class** outputs:
    - 0 → background
    - 1 → oil spill (yellow–cyan color in colormap)
    - 2 → look-alike (red color, can merge with oil if region overlaps)
  - Optional **water mask**: if enabled, non-water areas are masked out.
  - Returns: **RGB-colored segmentation mask**.
- **visualize_result(image, title="Result")**
  - Displays grayscale (VV) or RGB mask with matplotlib.
- **run_pipeline(image_path, water_mask=None, enable_water_mask=False)**
  - Full inference:
    1. Load VV band
    2. Preprocess
    3. Predict mask
  - Returns: **final RGB mask**.

Input → Output Summary

- **Input:**
  - SAR raster with VV band (band 1).
  - Shape: (H, W) float32.
- **Output:**
  - RGB mask (H, W, 3) with color-coded classes:
    - **Background** → black [0,0,0]
    - **Oil spill** → cyan [0,255,255]
    - **Look-alike** → red [255,0,0]
  - Optional original preview: (H, W, 3) stacked from first 3 bands.
  - Saved output mask as <filename>_prediction.png.

2 - Cloud Segmentation Model (UNET)

Purpose:
Detect clouds in multispectral satellite images (RGB + NIR) and generate a binary cloud mask.

Components:

- **Dataset class (CloudDataset)**
  - Loads aligned spectral bands (Red, Green, Blue, NIR) and ground-truth cloud masks.
  - Normalizes pixel values to [0,1].
  - Returns (image_tensor, mask_tensor) for training/evaluation.
  - Supports PyTorch tensors (CHW format) or NumPy-style (HWC format).
- **UNet / UNET architectures**
  - UNet: classic encoder–decoder with skip connections, configurable input channels (default 4: R,G,B,NIR) and output classes (default 2: cloud vs. clear).
  - UNET: ResNet34-based encoder backbone with learned upsampling (ConvTranspose2d) in the decoder.

Input → Output summary:

- **Input:**
  - A multi-band TIFF with **at least 4 channels** (Red, Green, Blue, NIR).
  - Shape: (H, W, 4), normalized to [0,1].
- **Output:**
  - predicted_mask: (H, W) binary array (0 = clear, 1 = cloud).
  - original_image: (H, W, 3) RGB array for visualization.
  - Optional: saved mask as <basename>_cloudmask.png.

3 - Cloud Removal Model (Sentinel2GAN / Pix2Pix)

Purpose:
Remove clouds from Sentinel-2 multispectral imagery using a **Pix2Pix GAN** generator. Produces a cloud-free image in both **GeoTIFF** and **PNG preview** formats.

Core wrapper:

Encapsulates preprocessing, GAN inference, and postprocessing.

- **load_image(image_path)**
  - Loads the input GeoTIFF and its raster profile (metadata).
  - Returns (raw_img, profile).
- **preprocess(raw_img)**
  - Normalizes image, scales to GAN input range (e.g. [-1, 1]).
  - Converts to input tensor.

    - Returns (tensor, arr_tanh).
- **predict(tensor)**
    - Runs Pix2Pix GAN forward pass.
    - Returns raw network output.
- **postprocess(raw_out)**
    - Converts GAN output back to [0,1].
    - Generates RGB visualization image.
    - Returns (out_01, rgb_01) where:
        - out_01: float32 array (H, W, C) normalized
        - rgb_01: preview RGB (H, W, 3)

Input → Output Summary

- **Input:**
    - Cloudy Sentinel-2 TIFF image (multispectral).
    - Shape: (H, W, C) where C = number of spectral bands.
- **Output:**
    - output_image: RGB preview PNG (uint8).
    - output_path: cloud-free GeoTIFF (uint16, same bands as input).
    - original_image: raw loaded array for reference.

Notes: - Post-processing merges look-alike classes into oil when connected to genuine oil detections. Model output handling supports both multi-class softmax and single-channel sigmoid.

Tools (registered via @tool and bound to the LLM)

All tools are included in the tools list and bound to the LLM with. bind_tools(tools) so the agent can call them during streaming.

- **store_satellite_metadata** (image_path: str) -> str — returns rasterio metadata and tags as JSON.
- **plot_sentinel1_image** (image_path: str) -> str — create pseudo-RGB from VV and VH and save a PNG preview.
- **plot_rgb_image** (image_path: str) -> str — read first 3 bands from a TIFF and save a PNG preview.
- **download_image_from_url** (file_url: str, outputname: str = "downloaded_file") -> str — supports HTTP(S) and Google Drive (via gdown). Saves under /kaggle/working by default.
- **oil_spill_segmentation** (image_path: str) -> str — runs OilSpillDetector.run_pipeline and saves <image>_prediction.png .
- **hist_equalize_sentinel1** (image_path: str) — equalizes VV & VH, writes <image>_equalized.tif and returns the path.

- **noise_filtering** (image_path: str, filter_type="median", kind="sentinel1", kernel_size=3, sigma=1.0) — median or gaussian filter on all bands; writes <image>_{filter}_filtered.tif .
- **get_segmented_metadata** (segmented_path: str) -> str — returns JSON metadata about the output image (format, size, file size).
- **equalization_Rgb** (img_path: str) -> str — per-channel histogram equalization for RGB images, saves <image>_equalized.tif .

The code defines a State TypedDict to carry messages.

- LLM is created via ChatGoogleGenerativeAI(...) and tools are bound to it. The LLM can call tools during invoke streaming.
- model_call(state) crafts a SystemMessage and invokes the LLM with the system prompt + messages.
- should_continue(state) checks the last streamed message for tool_calls to decide whether to route to the ToolNode or finish.
- Graph:
  - Entry node: our_agent (runs model_call)
  - Conditional branch: should_continue → tools (ToolNode) or END
  - tools executes requested tools and loops back to our_agent for follow-up LLM reasoning.
  - Streaming is done with app.stream(state, config=..., stream_mode="values") to capture intermediate tool and LLM messages.

Gradio UI

- Gradio Blocks defines a chat UI with:
- gr.Chatbot for conversation streaming.
- gr.Textbox for user input (Enter triggers submit).
- gr.Image preview for processed images.
- Hidden gr.Textbox to carry last download path for gr.File download.
- chat_with_agent_general runs the agent streaming loop, extracts file paths from tool outputs using regex, opens images for preview, and returns (history, state, img, download_path) to the UI.
- chat_and_store wraps the chat call and clears input afterwards.
- A small JS snippet toggles a triangular download button state based on hidden path value.

How to run

Kaggle (recommended when model and data are on Kaggle)

1. Place deeplabv3_model.keras in a Kaggle dataset and set OIL_MODEL_PATH accordingly.
2. Place input TIFF(s) into the working directory or dataset.
3. Run the notebook and launch the Gradio UI cell. The app will serve locally in the Kaggle session.

Local quickstart

1. Export API credentials:

   export GOOGLE_API_KEY="your-key"

1. Install dependencies (see Requirements).
2. Update OIL_MODEL_PATH to a local model file and WORKDIR if needed.
3. Run the script or notebook and open the Gradio UI.

Programmatic tool usage

You can directly call tools in a Python REPL or notebook:

```
print(store_satellite_metadata(img_path))

plot_path = plot_sentinel1_image(img_path)

seg_path = oil_spill_segmentation(img_path)
```

Usage examples (prompts)

- "/kaggle/working/sentinel1_image.tif display the image information"
- "/kaggle/working/sentinel1_image.tif segment the image and display the image information of the original image"
- "Download this file: https://.../sentinel.tif and then run segmentation."
- "Apply median noise filter with kernel_size=5 on this image and show me a preview."
- "Explain segmentation results (area, bbox, pixels) in plain English."

Output conventions & file naming

- **Equalized file:** <original>_equalized.tif
- **Filtered file:** <original>_<filter>_filtered.tif
- **Preview PNG:** <original>_plot.png
- **Prediction mask**: <original>_prediction.png
- **Downloaded files:** saved under /kaggle/working (or configured WORKDIR).

Troubleshooting & common pitfalls

- **Model load errors:** verify Keras/TensorFlow versions and provide custom_objects if model used custom layers.
- **Missing bands**: Sentinel-1 may be provided with only VV; the code falls back gracefully but check assumptions.
- **Large files:** TIFFs can be very large; consider downsampling for previews or reading windows via rasterio.windows .
- **gdown not installed:** download_image_from_url returns an error message; install gdown (pip install gdown).
- **Regex fallback:** The UI parsing of tool outputs relies on heuristics/regex; more robust integration is achievable by returning structured dicts from tools.

Suggestions & improvements

- Add a YAML/JSON config for model paths, workdir, LLM settings, and ports.
- Return structured outputs from tools (e.g. {"path": "...", "type": "image"}) to avoid fragile regex parsing.
- Add authentication or restrict Gradio to internal use when deployed publicly.
- Add unit tests (pytest) mocking rasterio and the model.
- Add georeferencing outputs (convert pixel bbox to lat/lon using rasterio.transform) and provide GeoJSON responses.
- Consider streaming large downloads with progress reporting.

TODO & refactors

- Remove hard-coded GOOGLE_API_KEY and OIL_MODEL_PATH — move to config.
- Consolidate duplicate helper functions (e.g. normalize_to_uint8 appears multiple times).
- Improve file-path extraction/structured tool returns.
- Expand chat_with_agent_general unit tests and error handling.
- Add TTA / uncertainty estimates for segmentation.

Appendix — function / file index

- OilSpillDetector (class): model load, preprocess, predict, visualize, run_pipeline
- Tools:
  - store_satellite_metadata
  - plot_sentinel1_image
  - plot_rgb_image

    - download_image_from_url
    - oil_spill_segmentation
    - cloud segmentation
    - cloud removing
    - hist_equalize_sentinel1
    - noise_filtering
    - get_segmented_metadata
    - equalization_Rgb
- UI:
    - chat_with_agent_general — streaming loop and parsing
    - Gradio Blocks — chat UI, preview, download