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 model (oil-spill detector), preprocessing/postprocessing utilities, file download and metadata-extraction tools, an LLM (Google Gemini via `langchain-google-genai`) bound to those tools, a LangGraph `StateGraph` agent for streaming tool calls, 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` (`StateGraph`)

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 (do not commit this key).
- `OIL_MODEL_PATH` — path to the Keras model file used for segmentation.
- `IMAGE_PATH` — optional default image for tests.
- `WORKDIR` — optional directory for downloads and outputs. Default in code: `/kaggle/working`.

Architecture & components

`OilSpillDetector` class

Responsibilities:

- `__init__(model_path, model_input_shape)` — loads the segmentation model.
- `load_image(path)` — reads the VV band (band 1) using `rasterio`.

- `preprocess_image(vv)` — resizes to model input shape, normalizes, and builds a 4D model tensor. Also returns an equalized uint8 image for visualization.
- `predict_oil_spill(input_tensor, water_mask=None, enable_water_mask=False)` — model inference, class conversion (softmax or sigmoid handling), post-processing (connected components / merging look-alikes), and color-coded RGB mask output.
- `visualize_result(image, title)` — uses `matplotlib` for inline display (returns image for saving).
- `run_pipeline(image_path, water_mask=None, enable_water_mask=False)` — full pipeline.

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`.

Agent & Graph (LangGraph)

- 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.

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.

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`).

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`
- `hist_equalize_sentinel1`
- `noise_filtering`
- `get_segmented_metadata`
- `equalization_Rgb`
