Oil Spill Identification from Satellite Images Using Deep Neural Networks.

Submitted in partial fulfilment of the requirements of the degree of

BACHELOR OF ENGINEERING

by

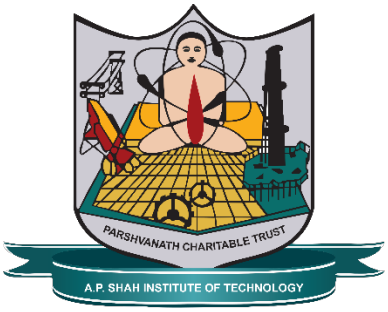
Prathamesh Mhatre (22106026)

Raj Nikam (22106027)

Tejashri Maske (22106051)

Guide:

Prof. Dr. Jaya Gupta



Department of Computer Science & Engineering

(Artificial Intelligence & Machine Learning)

A. P. SHAH INSTITUTE OF TECHNOLOGY, THANE

(2025-2026)

A. P. SHAH INSTITUTE OF TECHNOLOGY, THANE

CERTIFICATE

This is to certify that the project entitled **"Oil Spill Identification from Satellite Images Using Deep Neural Networks."** is a bonafide work of **Prathamesh Mhatre (22106026), Raj Nikam (22106027), Tejashri Maske (22106051)** submitted to the University of Mumbai in partial fulfilment of the requirement for the award of the degree of **Bachelor of Engineering** in **Computer Science & Engineering (Artificial Intelligence & Machine Learning).**

Prof. Dr. Jaya Gupta	Prof. Sayali P. Badhan
Guide	Project Coordinator
Prof. Dr. Jaya Gupta	Dr. Uttam D. Kolekar
Head of Department	Principal

A.P. SHAH INSTITUTE OF TECHNOLOGY, THANE

Project Report Approval for B.E.

This project report entitled **"Oil Spill Identification from Satellite Images Using Deep Neural Networks."** by **Prathamesh Mhatre (22106026), Raj Nikam (22106027), Tejashri Maske (22106051)** is approved for the degree of ***Bachelor of Engineering*** in ***Computer Science & Engineering (Artificial Intelligence & Machine Learning), 2025-26***.

Examiner Name	Signature
1._____	_____
2._____	_____

Date:

Place:

Declaration

We declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

(Prathamesh Mhatre : 22106026)

(Raj Nikam : 22106027)

(Tejashri Maske : 22106051)

Date:

Abstract

Oil spills pose a serious threat to marine ecosystems, making timely and accurate detection a critical environmental task. Traditional remote sensing methods often suffer from false detections and limited generalization across different conditions. To address these challenges, this project investigates **deep learning-based oil spill detection** using two complementary datasets: the **LADOS aerial imagery dataset** and a **satellite-based SAR dataset** from Kaggle. The LADOS dataset provides pixel-level annotations for multiple oil-related classes such as oil, emulsion, and sheen, while the SAR dataset captures large-scale oceanic conditions, enabling cross-domain evaluation. Two state-of-the-art semantic segmentation models, **U-Net** and **DeepLabv3+**, are implemented and trained to classify and localize oil spills. Their performance is evaluated using standard segmentation metrics including Intersection over Union (IoU), precision, recall, and F1-score. By comparing results across both aerial and satellite data, this study highlights the strengths and limitations of each model and dataset, and explores the potential of hybrid approaches for improving robustness. The findings contribute toward developing scalable, automated solutions for **real-time marine pollution monitoring**, aligning with **UN Sustainable Development Goal 14** (Life Below Water).

Keywords: Oil Spill Detection, Deep Learning, U-Net, DeepLabv3+, Synthetic Aperture Radar (SAR), LADOS Dataset, Marine Pollution Monitoring.

Contents

List of Figures

ABBREVIATIONS

SAR	Synthetic Aperture Radar
UAV	Unmanned Aerial Vehicle
LADOS	Aerial Imagery Dataset for Oil Spill Detection, Classification, and Localization Using Semantic Segmentation
IoU	Intersection over Union
CNN	Convolutional Neural Network
RGB	Red, Green, Blue
ASPP	Atrous Spatial Pyramid Pooling
PCA	Principal Component Analysis
API	Application Programming Interface
UI	User Interface
GPU	Graphics Processing Unit
CPU	Central Processing Unit
RAM	Random Access Memory
SSD	Solid State Drive
HDD	Hard Disk Drive
PSU	Power Supply Unit
IDE	Integrated Development Environment
OS	Operating System
AWS	Amazon Web Services
EC2	Elastic Compute Cloud
S3	Simple Storage Service

Chapter 1
Introduction

Oil spills are among the most damaging forms of marine pollution, causing long-lasting ecological, economic, and social consequences. Rapid and accurate detection of oil spills is essential for enabling timely response and minimizing damage to ocean ecosystems and coastal communities. Traditional approaches to oil spill monitoring, such as manual observation and threshold-based image analysis of remote sensing data, are often limited by false detections, high dependency on expert interpretation, and poor adaptability to diverse oceanic conditions.

Recent advances in remote sensing and deep learning have opened new opportunities for automated oil spill detection. Remote sensing data from different sources such as Synthetic Aperture Radar (SAR) and aerial imagery provide valuable insights into spill characteristics. SAR imagery, in particular, is widely used because of its ability to capture large-scale ocean surfaces under all weather and lighting conditions. However, there were challenges in existing oil spill detection research, including:

High False Positives in SAR Imagery:

- Oil spills appear similar to look-alikes (low wind zones, algae, natural slicks).
- Traditional threshold-based methods often misclassify.

Limited Datasets

- Early works relied on small SAR datasets, making deep learning models hard to generalize.
- No standardized benchmark for fair comparison.

Binary Classification Only

- Many studies reduced the problem to “oil vs background.”
- Real-world scenarios include multiple categories (oil, sheen, emulsion, ship, platform).

Resolution vs Coverage Trade-Off

- Aerial imagery (like LADOS) gives detail but is geographically limited.
- Satellite imagery covers vast areas but at lower resolution.

To address these challenges, this project investigates the application of **deep learning–based semantic segmentation** models for oil spill detection across both aerial and satellite imagery. Specifically, we implement and evaluate **U-Net** and **DeepLabv3+**, two state-of-the-art architectures for pixel-level image segmentation. Using the **LADOS aerial imagery dataset** and a publicly available **SAR dataset from Kaggle**, we compare model performance in terms of **Intersection over Union (IoU), precision, recall, and F1-score**. The study further explores the potential of combining aerial and satellite data to enhance robustness and generalization.

By analyzing model performance across two complementary datasets, this work provides insights into how data modality influences detection accuracy. The results will highlight the strengths of high-resolution aerial imagery in detailed classification and the scalability of SAR imagery in large-area monitoring, while also identifying limitations that persist in each.
Ultimately, the project aims to contribute to the development of a more reliable and scalable oil spill detection framework. Such a system can support early response strategies, reduce environmental risks, and promote sustainable ocean management in alignment with United Nations Sustainable Development Goal 14: Life Below Water.

Chapter 2

Literature Survey

Year	Title	Domain	Algorithm	Dataset	Gap
2019	*Oil Spill Identification from Satellite Images Using Deep Neural Networks*	SAR satellite; semantic segmentation	DeepLabv3+, DeepLabv2, U-Net, LinkNet, PSPNet (benchmarked)	Public SAR dataset introduced by authors	Look-alike confusion in SAR; many works non-comparable due to custom data;
2020	*Large-scale detection and categorization of oil spills from SAR images with deep learning* (arXiv:2006.13575)	SAR / segmentation & categorizatio n	Deep learning segmentation + categorization pipeline (author's DL models)	Large SAR collection assembled by authors	Focus on SAR; categorization improves analysis but still vulnerable to look-alikes and limited multi-modal validation.
2024	*Utilizing deep learning algorithms for automated oil spill detection in medium-resolution optical imagery* (PubMed:3908391)	Optical (Sentinel-2 / Landsat)	U-Net variants with attention (CBAM etc.)	Medium-resolution optical imagery (Sentinel-2 / Landsat experimental sets)	needs complementary sensors; Optical data suffers from clouds and illumination variability.
2024	*Detection of Marine Oil Spill from PlanetScope Images Using CNN and Transformer Models* (MDPI Mar. 2024)	High-res optical (PlanetScope)	CNN + Transformer hybrid models	UPlanetScop e high-resolution imagery	High-res optical shows promise but limited by cloud cover and data cost; transformer models add compute

					requirements.
2025	*A novel YOLOv11-Driven deep learning algorithm for UAV multispectral oil spill detection in Inland lakes* (Springer, 2025)	UAV multispectral; inland waters	YOLOv11 variant (ADHF + SimAM attention)	UAV multispectral imagery (inland datasets)	Focused on inland/multispectral ; good for UAVs but limited scalability for ocean-wide monitoring;
2025	*Improved Lightweight Marine Oil Spill Detection Algorithm of YOLOv8 (LSFE-YOLO)* (SSRN / preprint, 2025).	SAR; real-time detection	LSFE-YOLO (YOLOv8s + FasterNet backbone, GN-LSC head, SE attention)	SAR detection dataset assembled by authors (bounding-box labels)	Lightweight for detection but not mask-accurate segmentation; limited multi-class liquid taxonomy.
2025	*Deep Learning-Based Hyperspectral Oil Spill Detection for Marine Pollution Monitoring in the Gulf of Mexico* (SSRN / preprint, 2025).	Hyperspectral airborne; semantic segmentation	U-Net, DeepLabv3 on PCA-reduced hyperspectral patches	HOSD / hyperspectral dataset (Gulf of Mexico scenes)	Hyperspectral offers strong spectral separability but is compute-intensive, scarce, and complex to preprocess (PCA, patching); moderate IoU

					indicates more work needed.
2025	*LADOS: Aerial Imagery Dataset for Oil Spill Detection, Classification, and Localization Using Semantic Segmentation* (Data, 2025).	Aerial / UAV RGB; semantic & instance segmentation	Benchmarking: DeepLabV3+, SegFormer, Mask2Former, instance/semantic baselines	**LADOS** — 3,388 RGB images, pixel masks across 6 classes (Oil, Emulsion, Sheen, Ship, Oil-platform, Background)	Excellent multi-class aerial benchmark but limited geographic/altitude coverage vs satellites; class imbalance (certain classes underrepresented).
2020	Oil spill detection using machine learning and infrared images.	Remote sensing, Oil spill detection	Convolutional Neural Networks (CNN) – ResNet-based models for segmentation and classification	Sentinel-1 SAR imagery, North Sea dataset	Lack of generalization to different sea conditions and regions; challenges in distinguishing look-alikes (e.g., low wind areas)
2019	Oil Spill Identification from Satellite Images Using Deep Neural Networks	Remote sensing, Deep learning	CNN (AlexNet, ResNet, GoogLeNet variants) for classification	European Maritime Safety Agency (EMSA) dataset from Sentinel-1	Limited labeled data; models require improvement in robustness to noise and diverse ocean conditions
2000	Neural networks for oil spill detection using ERS-SAR data	SAR image analysis, Environmental monitoring	Feedforward Neural Network (multilayer perceptron)	ERS-1/2 SAR images (European Space Agency)	Early-stage model — limited computational power and data diversity; lacks transferability to newer sensors
2020	Sensors, Features,	Review,	Comparative	Compilation	Identifies need for

	and Machine Learning for Oil Spill Detection and Monitoring: A Review	Remote sensing, ML	analysis (not one specific model) of ML techniques (SVM, ANN, CNN, Random Forest)	of datasets from SAR, optical, and hyperspectral sensors	standardized datasets and multimodal data fusion approaches for real-time detection
2022	A review: Data pre-processing and data augmentation techniques	processing and data augmentation techniquesData science, Machine learning preprocessing	Review of augmentation methods (rotation, scaling, flipping, GANs, SMOTE, etc.)	Applied across generic ML datasets	Lack of domain-specific augmentation for remote sensing and environmental applications
2003	Studies of the formation process of water-in-oil emulsions	Marine pollution, Environmental chemistry	Experimental study (no ML algorithm)	Laboratory-based measurements of oil-water emulsions	No integration with remote sensing or ML; focuses on physicochemical aspects, not detection automation
2020	*Oil spill detection using machine learning and infrared images*	Infrared imaging, Environmental monitoring	Machine learning classifiers (SVM, Random Forest, k-NN, Decision Tree)	Infrared (IR) camera images under controlled lab and field settings	Limited scalability to real-world marine conditions; small dataset size; need for integration with SAR/optical data

Chapter 3

Limitations of Existing Systems

Current oil spill detection systems face several limitations that hinder their accuracy, scalability, and real-world applicability:

- **High False Positives**
  Traditional SAR-based methods often misclassify look-alike phenomena such as low wind zones, algae blooms, or natural slicks as oil spills. This leads to unreliable detection and unnecessary follow-up operations.

- **Dependence on Manual Interpretation**
  Conventional approaches still rely heavily on expert analysis of imagery. This process is time-consuming, inconsistent, and not feasible for continuous large-scale monitoring.

- **Limited Datasets**
  Earlier studies were constrained by small or custom datasets, reducing the generalizability of models. The lack of standardized benchmarks makes fair comparison between methods difficult.

- **Binary Classification Focus**
  Many systems classify images simply as “oil” or “no oil,” ignoring the complexity of real scenarios where multiple categories (oil, sheen, emulsion, ships, platforms) must be distinguished.

- **Resolution vs. Coverage Trade-Off**
  Aerial imagery provides high resolution and fine details but is geographically limited. In contrast, satellite SAR covers vast ocean areas but at lower resolution, reducing the precision of detection.

- **Model Complexity vs. Real-Time Performance**
  Deep learning models like DeepLabv3+ achieve high accuracy but are computationally heavy, making them unsuitable for real-time deployment. Lightweight models such as

YOLO variants are faster but compromise on pixel-level precision.

- **Spectral Limitations**
  RGB and SAR imagery capture limited information and often struggle to differentiate oil chemically from water. While hyperspectral data offers richer detail, it is expensive, rare, and computationally intensive.

These limitations collectively hinder the development of robust, scalable, and real-time oil spill detection systems. Addressing them requires leveraging **multi-modal data (aerial, satellite, hyperspectral)** along with **efficient deep learning models** that balance accuracy, speed, and adaptability.

Chapter 4

Problem Statement, Objectives and Scope

Problem Statement

Oil spills cause severe environmental and economic damage, yet current detection methods remain limited. Traditional approaches often give false positives and require manual interpretation, while deep learning studies mostly use **single-modality datasets**, reducing generalization. Many also simplify detection to binary classification, overlooking real-world complexity. This highlights the need for a **multimodal study** using aerial and satellite imagery with advanced deep learning models.

Objectives

1. To implement and evaluate **U-Net** and **DeepLabv3+** architectures for pixel-level oil spill detection.
2. To utilize and analyze two complementary datasets: the **LADOS aerial imagery dataset** and a **satellite-based SAR dataset from Kaggle**.
3. To compare model performance using standard evaluation metrics such as **Intersection over Union (IoU), precision, recall, and F1-score**.
4. To study the effect of data modality (aerial vs satellite) on detection accuracy and generalization.
5. To explore the feasibility of a **hybrid detection framework** that combines the strengths of aerial and satellite data.

Scope

- The study focuses on **remote sensing–based oil spill detection** using deep learning.
- The study is restricted to **two datasets**: LADOS (aerial RGB imagery with pixel-level annotations) and a publicly available Kaggle SAR dataset (satellite-based).
- Only two deep learning architectures, **U-Net and DeepLabv3+**, are considered due to their proven effectiveness in semantic segmentation tasks.

- The evaluation is limited to **offline experiments**, with no real-time deployment, but results will serve as a foundation for future operational systems.
- The project emphasizes **comparative analysis and cross-dataset evaluation**, providing insights into how different modalities influence detection performance.

Chapter 5

Proposed System

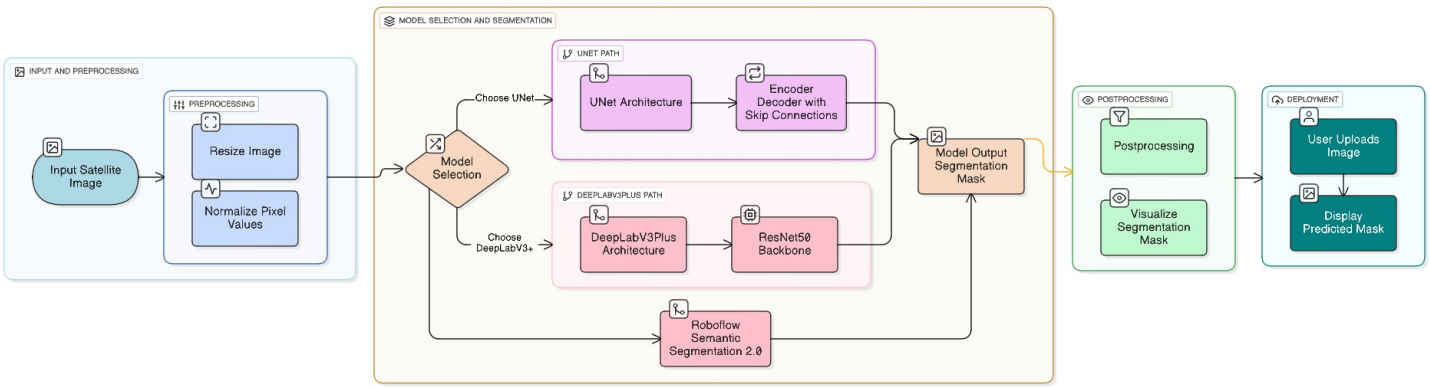


Figure 5.1 System Design for Oil Spill Detection

This flowchart illustrates a pipeline for processing and segmenting satellite images using various deep learning models for semantic segmentation. The diagram is divided into several key sections: **Input and Preprocessing**, **Model Selection**, multiple model-specific paths (U-Net Path, DeepLabV3Plus Path, and Roboflow Path), **Model Output Segmentation Mask**, **Postprocessing**, and **Output/Deployment**. It represents a system where a user uploads a satellite image, which is preprocessed, segmented using one of several selectable models, postprocessed, visualized, and finally displayed as a predicted mask.

The flowchart uses a left-to-right structure, starting from the input on the left and ending with the output on the right. Boxes represent processes or components, arrows indicate data flow, and colors group related elements (e.g., blue for input/preprocessing, orange for model selection, purple/pink for model architectures, green for postprocessing/output). There are three parallel paths under model selection, converging back into a single output stream.

5.1.1. Input and Preprocessing

This is the starting point. It prepares the raw satellite image for model input, ensuring consistency in size and pixel values, which is crucial for deep learning models to perform accurately.

Input Satellite Image:

- This is the entry point: a raw satellite image (e.g., from sources like Google Earth, Landsat, or Sentinel satellites). Satellite images are typically multi-band (e.g., RGB + infrared) and high-resolution, but they can vary in size, lighting, and noise.
- **Purpose:** Serves as the raw data to be segmented. Segmentation here likely means dividing the image into meaningful regions (e.g., semantic segmentation where each pixel is classified into categories like "urban", "forest", "water").

Resize Image:

- **Purpose:** Resizes the image to a standard dimension expected by the models (e.g., 512x512 pixels). This is essential because models like U-Net or DeepLab are trained on fixed-size inputs, and resizing prevents issues with variable resolutions. Techniques might include bilinear interpolation or cropping to maintain aspect ratio.
- **Output:** A resized version of the input image.

Normalize Pixel Values:

- **Purpose:** Scales pixel intensities to a standard range, often [0, 1] or [-1, 1], by dividing by 255 (for 8-bit images) or using mean/std normalization based on dataset statistics. This helps models converge faster during inference and handles variations in brightness/contrast from different satellite sensors.
- **Output:** Normalized image ready for model input.

5.1.2. Model Selection

- **Purpose:** Acts as a decision point where the system chooses one of three segmentation models based on criteria like user preference, task requirements, or performance metrics. This allows flexibility—e.g., U-Net for simpler tasks, DeepLab for more complex feature extraction.
- **Inputs:** Preprocessed image from the left.
- **Outputs:** Branches into three parallel paths:
  - Top: "Choose UNet" arrow to the U-Net Path.
  - Middle: "Choose DeepLabV3+" arrow to the DeepLabV3Plus Path .
  - Bottom: No explicit label, but directly to the Roboflow Path.
- Selection is automated (e.g., based on image type) but can also be manually overridden..

5.1.3. Model-Specific Paths

These are the core segmentation engines. Each path processes the preprocessed image independently and produces a segmentation mask (a pixel-wise label map where each pixel is assigned a class).

U-Net Path

This path uses the U-Net architecture, a popular encoder-decoder model for biomedical and satellite image segmentation. It's known for efficiency with limited data due to skip connections.

- **UNet Architecture**:
  - **Purpose:** The overall structure of U-Net, which consists of a contracting path (encoder) for feature extraction and an expanding path (decoder) for precise localization.

- **Encoder**:
  - **Purpose:** Downsamples the image through convolutional layers and max-pooling to capture high-level features (e.g., shapes of buildings or roads in satellite images). Typically involves multiple blocks with increasing channels (e.g., 64 → 128 → 256).

- **Decoder with Skip Connections**:

- **Purpose:** Upsamples the features back to the original resolution using transposed convolutions or upsampling. Skip connections concatenate features from corresponding encoder layers to the decoder, preserving fine details lost during downsampling. This is key for accurate boundaries in segmentation masks.

- **Flow:** Preprocessed image → UNet Architecture → Encoder → Decoder → Output to "Model Output Segmentation Mask".

DeepLabV3Plus Path

- This path employs DeepLabv3+, an advanced model from Google for semantic segmentation, excelling in capturing multi-scale context (useful for satellite images with varying object sizes).
- **DeepLabV3Plus Architecture**:
  - **Purpose:** Combines Atrous Spatial Pyramid Pooling (ASPP) for multi-scale features with an encoder-decoder structure.
- **ResNet50 Backbone**:
  - **Purpose:** Uses a pre-trained ResNet-50 as the feature extractor (encoder). ResNet-50 is a deep residual network with 50 layers, handling vanishing gradients via skip connections. It's modified with atrous convolutions (dilated filters) to maintain resolution while increasing receptive field—ideal for dense predictions in large satellite scenes.
- **Flow:** Preprocessed image → DeepLabV3Plus Architecture → ResNet50 Backbone → Output to "Model Output Segmentation Mask".

Roboflow Path (Bottom Path)

- This refers to a pre-trained model or API from Roboflow, a platform for computer vision datasets and models. "Roboflow Semantic Segmentation 2.0" is a specific version or model hosted on Roboflow.
- **Roboflow Semantic Segmentation 2.0**:

- **Purpose:** A ready-to-use semantic segmentation model, possibly based on architectures like YOLO or custom variants, fine-tuned on datasets via Roboflow's tools. It's designed for quick deployment, especially for users without custom training.
- This path is simpler in the diagram, implies a API call rather than detailed internal structure.

- **Flow:** Preprocessed image → Roboflow Semantic Segmentation 2.0 → Output to "Model Output Segmentation Mask".

5.1.4. Model Output Segmentation Mask

- **Purpose:** The raw output from the selected model—a segmentation mask. This is a tensor where each pixel has a class probability or label.

5.1.5. Postprocessing.

Refines the raw mask, e.g., applying thresholding, morphological operations (erosion/dilation) to remove noise, or conditional random fields (CRF) for smoother boundaries. This step improves visual and quantitative accuracy.

- **Visualize Segmentation Mask**: Converts the mask into a human-readable visualization, e.g., overlaying colored labels on the original image (red for buildings, green for vegetation).

5.1.6. Output and Deployment

- This is the final stage.
- **User Uploads Image**:
  - Indicates the user interface where the original image is uploaded.
- **Display Predicted Mask**:
  - Purpose: The end output—displays the visualized segmentation mask to the user, e.g., via a web app . This includes side-by-side comparisons of input and output.

Use Case Diagram:

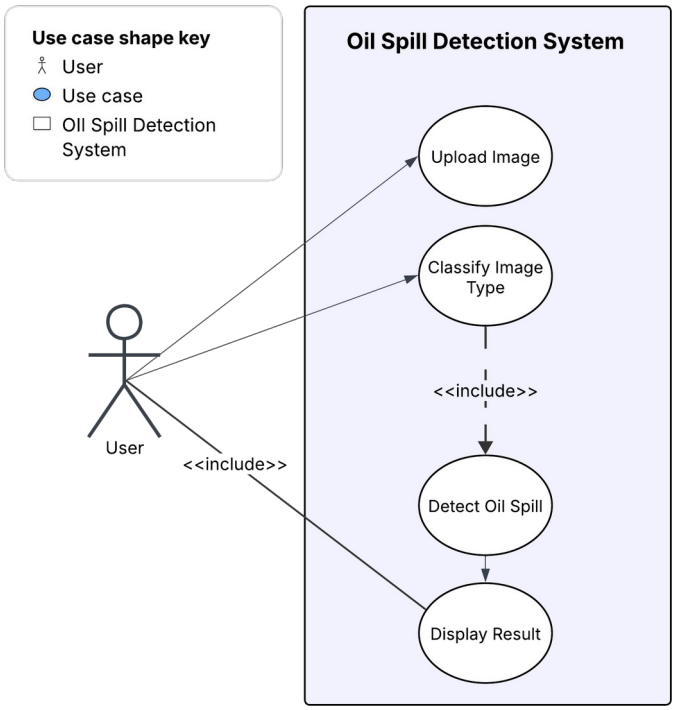


Figure 5.2 Use case diagram for Oil Spill Detection

Actors:

- **User:** The primary actor who interacts with the system to upload an image and view detection results.

Use Cases:

1. **Upload Image:**
   The user initiates the process by uploading an image that potentially contains an oil spill. This serves as the input for further processing.

2. **Classify Image Type:**
   Once the image is uploaded, the system classifies the image based on its characteristics (e.g., satellite image, drone image, etc.).

- This use case **includes** the *Detect Oil Spill* use case, as classification is a prerequisite for oil spill detection.

3. **Detect Oil Spill:**
   The system processes the classified image using detection algorithms or models to identify the presence of an oil spill.

4. **Display Result:**
   After analysis, the system displays the detection result to the user, indicating whether an oil spill was detected along with any relevant details.

Relationships:

- The *User* interacts directly with all four use cases.

- The *Classify Image Type* use case includes the *Detect Oil Spill* use case, representing a dependency where classification triggers detection.

Chapter 6

Experimental Setup

6.1. Software Requirements

6.1.1. FastAPI [standard]:

- **Purpose**: A high-performance, asynchronous Python web framework for building RESTful APIs.
- **Use Case**: Serves as the backend framework to handle API requests, process file uploads (via python-multipart), perform image processing (using Pillow and opencv-python), execute machine learning model inference (via tensorflow and inference-sdk), and deliver data visualizations (using matplotlib). Supports endpoints for the React frontend to interact with.
- **Version**: Latest stable version (e.g., 0.115.0 as of October 2025, subject to updates), with the [standard] extra for additional dependencies like uvicorn for running the server.
- **Dependencies**: Requires python-multipart for file uploads and uvicorn for serving the application.

6.1.2. React:

- **Purpose**: A JavaScript library for building a dynamic, component-based frontend user interface.
- **Use Case**: Creates an interactive UI for users to upload images or files, display processed outputs (e.g., image analysis, ML predictions), and render visualizations (e.g., charts generated by matplotlib and served via FastAPI).
- **Version**: Latest stable version (e.g., 18.x).
- **Dependencies**: Integrated with Vite for development and build, and Tailwind CSS for styling.

6.1.3. Tailwind CSS:

- **Purpose**: A utility-first CSS framework for styling the React frontend.
- **Use Case**: Provides responsive, customizable designs for the UI, enabling a modern interface for file upload forms, result displays, and visualizations.
- **Version**: Latest stable version (e.g., 3.x).
- **Integration**: Configured with Vite for optimized CSS processing in development and production.

6.1.4. Vite:

- **Purpose**: A fast build tool and development server for modern web applications.
- **Use Case**: Powers the React frontend development with hot module replacement (HMR) for rapid iteration and creates optimized production builds for deployment.
- **Version**: Latest stable version (e.g., 5.x).

6.1.5 Python Libraries:

python-multipart:

- **Purpose**: Enables FastAPI to process multipart form data.
- **Use Case**: Handles file uploads (e.g., images or datasets) sent from the React frontend to the FastAPI backend for processing with Pillow, opencv-python, or tensorflow.
- **Version**: Latest stable (e.g., 0.0.9).

Pillow:

- **Purpose**: A Python imaging library for image manipulation.
- **Use Case**: Preprocesses images (e.g., resizing, cropping, format conversion) before feeding them into ML models or for rendering processed images in the frontend.
- **Version**: Latest stable (e.g., 10.x).

NumPy:

- **Purpose**: A library for numerical computations and array operations.

- **Use Case**: Supports data preprocessing for ML models, image processing with opencv-python, and generating data for matplotlib visualizations.
- **Version**: Latest stable (e.g., 2.x).

Matplotlib:

- **Purpose**: A plotting library for creating static, interactive, or web-compatible visualizations.
- **Use Case**: Generates charts (e.g., bar, line, scatter) to visualize ML model outputs, statistical analysis, or image processing results, served via FastAPI to the React frontend.
- **Version**: Latest stable (e.g., 3.x).

TensorFlow (2.12.0):

- **Purpose**: A machine learning framework for building and running ML models.
- **Use Case**: Executes pre-trained or custom models for tasks like image classification, object detection, or regression, integrated with inference-sdk for streamlined deployment.
- **Version**: Specifically 2.12.0 as requested.

Opencv-python:

- **Purpose**: A computer vision library for advanced image and video processing.
- **Use Case**: Performs tasks like edge detection, filtering, or feature extraction on images before feeding them into tensorflow models or for post-processing results.
- **Version**: Latest stable (e.g., 4.x).

6.1.6. Visual Studio Code (VSCode):

1. **Purpose**: The primary Integrated Development Environment (IDE) for coding and debugging.
2. **Use Case**: Supports development of FastAPI backend, React frontend, and Python scripts, with extensions for Python (e.g., Pylance), JavaScript, and Git integration.
3. **Version**: Latest stable release.

6.1.7. Version Control - Git:

- **Purpose**: Manages code versioning and collaboration.
- **Use Case**: Tracks changes in the FastAPI backend, React frontend, and Python scripts, enabling team collaboration via platforms like GitHub or GitLab.
- **Version**: Latest stable.

6.1.8. Operating System:

- Windows 11, Linux (Ubuntu 22.04 LTS recommended for TensorFlow compatibility), or macOS (with limitations on GPU support). Linux is preferred for stability in DL environments.

6.2. Hardware Requirements

6.2.1 Processor (CPU):

- **Minimum**: Intel Core i5 (or AMD Ryzen 5 equivalent) with at least 6 cores and 12 threads, clock speed of 3.0 GHz or higher. Handles general scripting, data preprocessing (e.g., patching SAR images with libraries like NumPy or OpenCV), and lightweight inference. Multi-core support is crucial for parallel data loading during training.

- **Recommended for Training/Development**: Intel Core i7/i9 (or AMD Ryzen 7/9) with 8-16 cores and 16-32 threads, clock speed of 4.0 GHz+ (e.g., Intel Core i9-13900K or AMD Ryzen 9 7950X). Accelerates CPU-bound tasks like data augmentation, file I/O for large SAR datasets, and hyperparameter tuning. In distributed training setups (e.g., multiple GPUs), a strong CPU manages orchestration.

- **Rationale**: SAR processing involves heavy numerical computations; users report using high-core CPUs to reduce bottlenecks in data pipelines. For example, in MATLAB SAR target classification examples, a CPU is fallback if no GPU is available, but training times increase significantly.

- **For Production/Deployment**: AWS EC2 instances like c5.xlarge (4 vCPUs) or higher for cloud-based inference, or embedded systems like NVIDIA Jetson for edge deployment (e.g., on drones for real-time oil spill monitoring).

6.2.2. RAM (System Memory):

- **Minimum**: 16 GB DDR4 (or DDR5) at 3200 MHz. Sufficient for basic development, loading small SAR image batches, and running inference on single images.
- **Recommended for Training/Development**: 32-64 GB DDR4/DDR5 (e.g., 2x16 GB or 4x16 GB modules for dual-channel/quad-channel performance). Essential for handling large SAR datasets in memory (e.g., loading patched images during U-Net training epochs). U-Net training on remote sensing images often requires large batch sizes (e.g., 16-32) to converge effectively, and insufficient RAM can cause swapping to disk, slowing down processes by 5-10x.
- **Rationale**: Reports from NVIDIA forums indicate 32-64 GB is common for satellite imagery segmentation (e.g., DSTL dataset with U-Net), as SAR images can consume 10-20 GB during training with augmentations. In Google Colab (often used for prototyping), the free tier offers ~12 GB, but pro versions provide up to 52 GB for faster training.
- **For Production/Deployment**: 16-32 GB, as inference typically processes one image at a time. In cloud setups like AWS EC2, use instances with scalable RAM (e.g., m5.2xlarge with 32 GB).

6.2.3. Storage:

- **Minimum**: 256 GB SSD (NVMe preferred for faster read/write speeds). Stores code, small SAR datasets, and model checkpoints.
- **Recommended for Training/Development**: 1 TB+ NVMe SSD (e.g., Samsung 990 Pro or WD Black SN850X). SAR datasets like Sentinel-1 oil spill collections can be 100 GB+ (e.g., thousands of high-resolution GeoTIFF files). Fast storage is critical for quick data

loading during training epochs, reducing I/O bottlenecks. Include additional HDD (2-4 TB) for archiving raw data.

- **Rationale**: Deep learning on remote sensing images involves frequent reads of large files; SSDs provide 5-10x faster access than HDDs. In examples like PyImageSearch U-Net tutorials, datasets are loaded from disk, and slow storage can double training time.

- **For Production/Deployment**: 512 GB SSD, with cloud storage like Amazon S3 for scalable dataset hosting (e.g., store SAR images remotely and fetch on-demand).

6.2.4. Graphics Processing Unit (GPU):

- **Minimum**: NVIDIA GPU with at least 8 GB VRAM, CUDA compute capability 3.5+ (e.g., GTX 1660 Ti or RTX 2060). TensorFlow 2.12.0 requires CUDA 11.8 and cuDNN 8.6. Basic training on small SAR patches or inference on low-resolution images.

- **Recommended for Training/Development**: NVIDIA RTX 30/40 series with 12-24 GB VRAM (e.g., RTX 3080 Ti 12 GB, RTX 4090 24 GB, or A100 40 GB for professional setups). For multi-GPU, 2-4 cards with NVLink support. Accelerates U-Net training by 10-100x over CPU, handling convolutional operations on SAR images. SAR oil spill detection often uses patched images (e.g., 512x512), but large batches or high-res models require more VRAM to avoid out-of-memory errors.

- **Rationale**: From NVIDIA forums, 4x Tesla V100 (32 GB each) were used for U-Net on satellite imagery like DSTL dataset. Papers on land cover segmentation (similar to oil spill tasks) report using RTX 3080 Ti (16 GB VRAM). Reddit discussions note that while U-Net can train without a GPU, it's impractical for remote sensing datasets (e.g., days vs. hours). For SAR-specific, MATLAB examples require CUDA 3.0+ GPUs; modern ones like A100 offer better utilization (up to 37% for similar models).

- **For Production/Deployment**: NVIDIA T4 or A10 (16 GB VRAM) in cloud (e.g., AWS g4dn.xlarge), or edge devices like Jetson AGX Xavier (32 TOPS AI performance) for real-time detection on drones/ships. Lightweight U-Net variants (e.g., MobileUNet) can run on CPUs for low-power setups.

6.2.5. Power Supply and Cooling:

- **Minimum**: 650W 80+ Bronze rated PSU. Supports basic GPU setups.
- **Recommended**: 850W+ 80+ Gold/Platinum PSU (e.g., Corsair RM850x) with adequate cooling (e.g., liquid cooling or multiple case fans). High-end GPUs like RTX 4090 draw 450W+ under load during training, and SAR processing can run for hours/days, generating heat.
- **Rationale**: Prolonged training sessions risk thermal throttling; users report stable PSUs prevent crashes in deep learning workflows.

6.2.6. Networking (Optional for Cloud/Distributed Training):

- **Minimum**: Gigabit Ethernet.
- **Recommended**: 10 Gbps Ethernet or InfiniBand for multi-node setups. Transferring large SAR datasets from cloud storage (e.g., Copernicus Open Access Hub) or distributed training across machines.
- **Rationale: SAR datasets are massive; fast networking reduces download/setup time.**

Chapter 7

Project Planning

Gantt Chart:

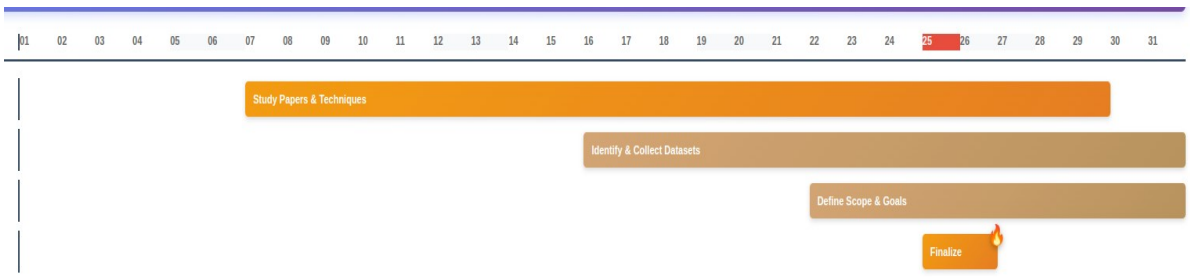


Figure 7.1 July (Gantt Chart)

Figure 7.2 August (Gantt Chart)

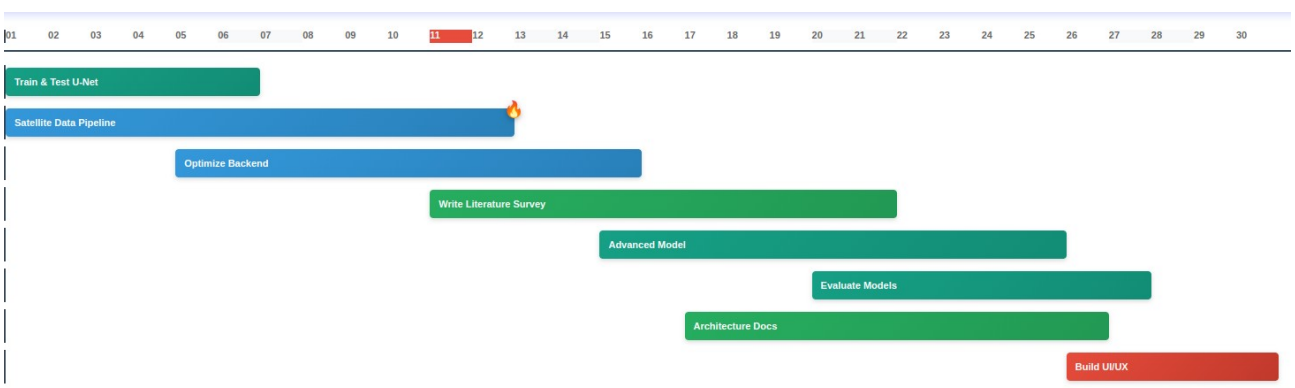


Figure 7.3 September (Gantt Chart)

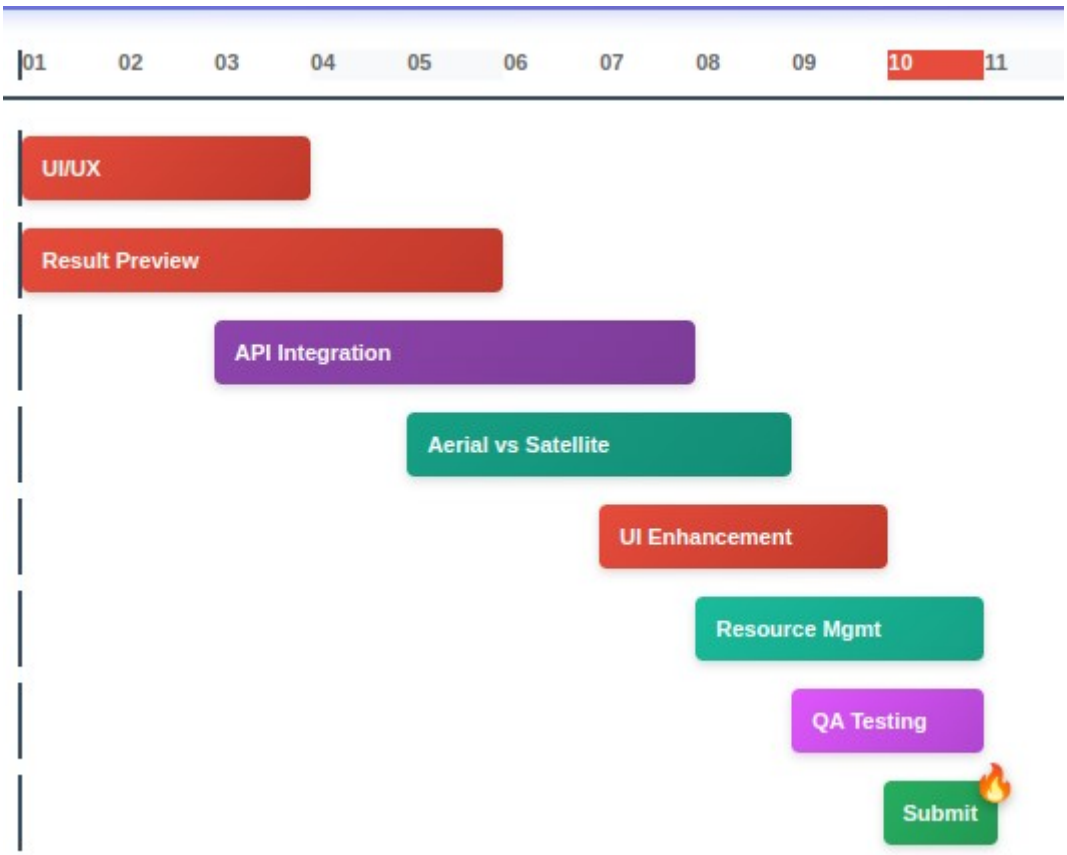


Figure 7.4 October (Gantt Chart)

Chapter 8

Expected Outcome

Upload page:

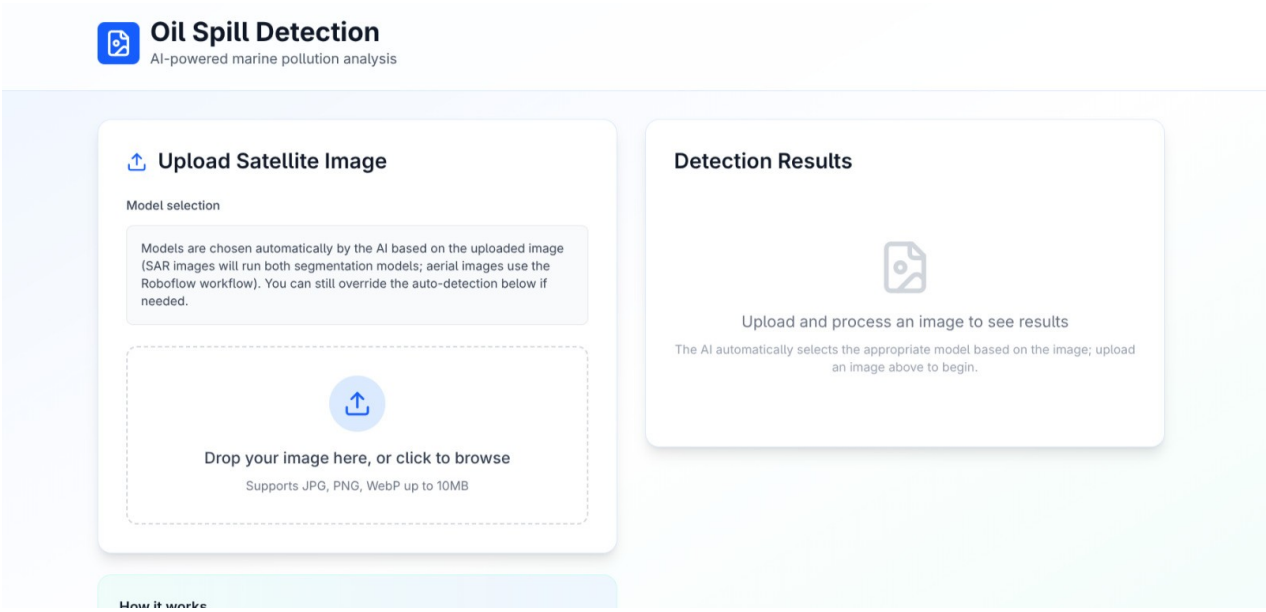


Figure 8.1 Upload Page

The image depicts the upload page screen of the app. Here user can upload images where they can check for oilspills using SAR/UAV images.

Image Upload Classification:

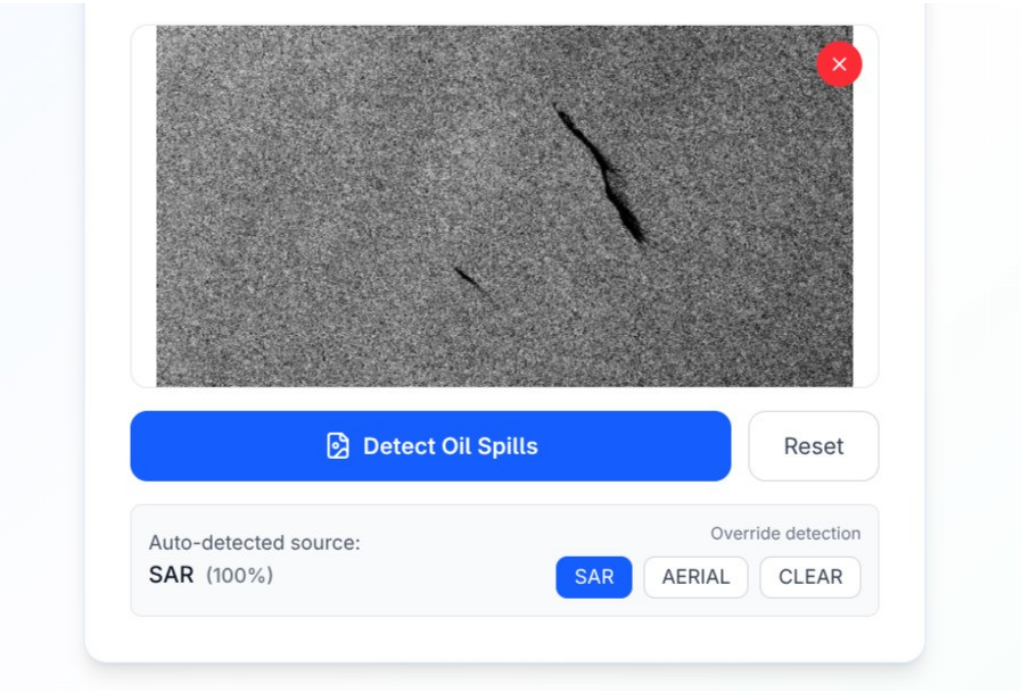


Figure 8.2 Uploaded SAR image identified

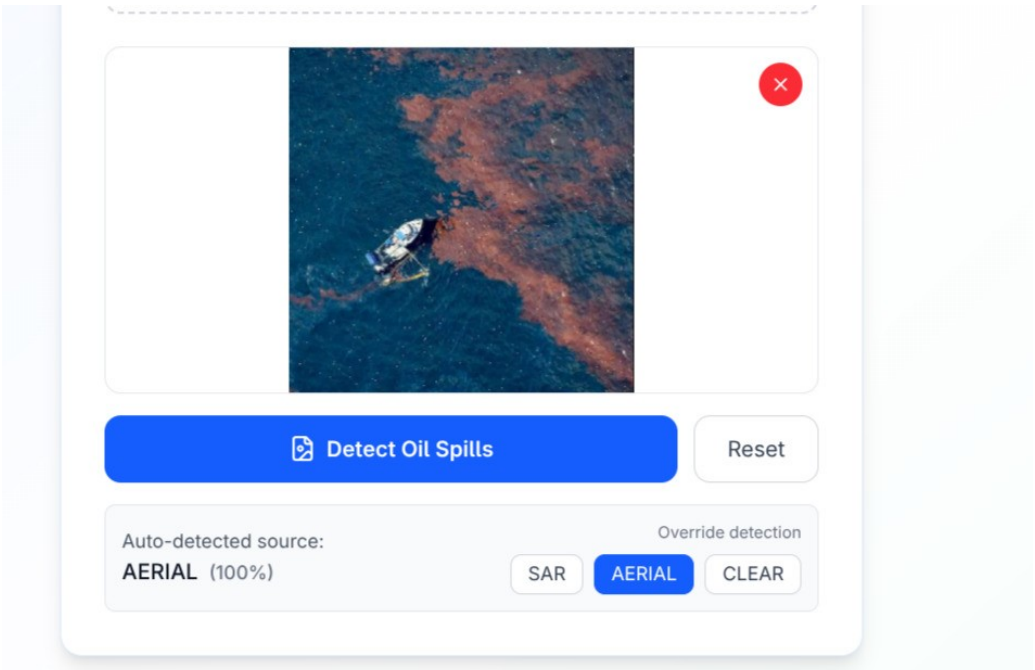


Figure 8.3 Uploaded Aerial Image identified.

Results: SAR Image Mask:

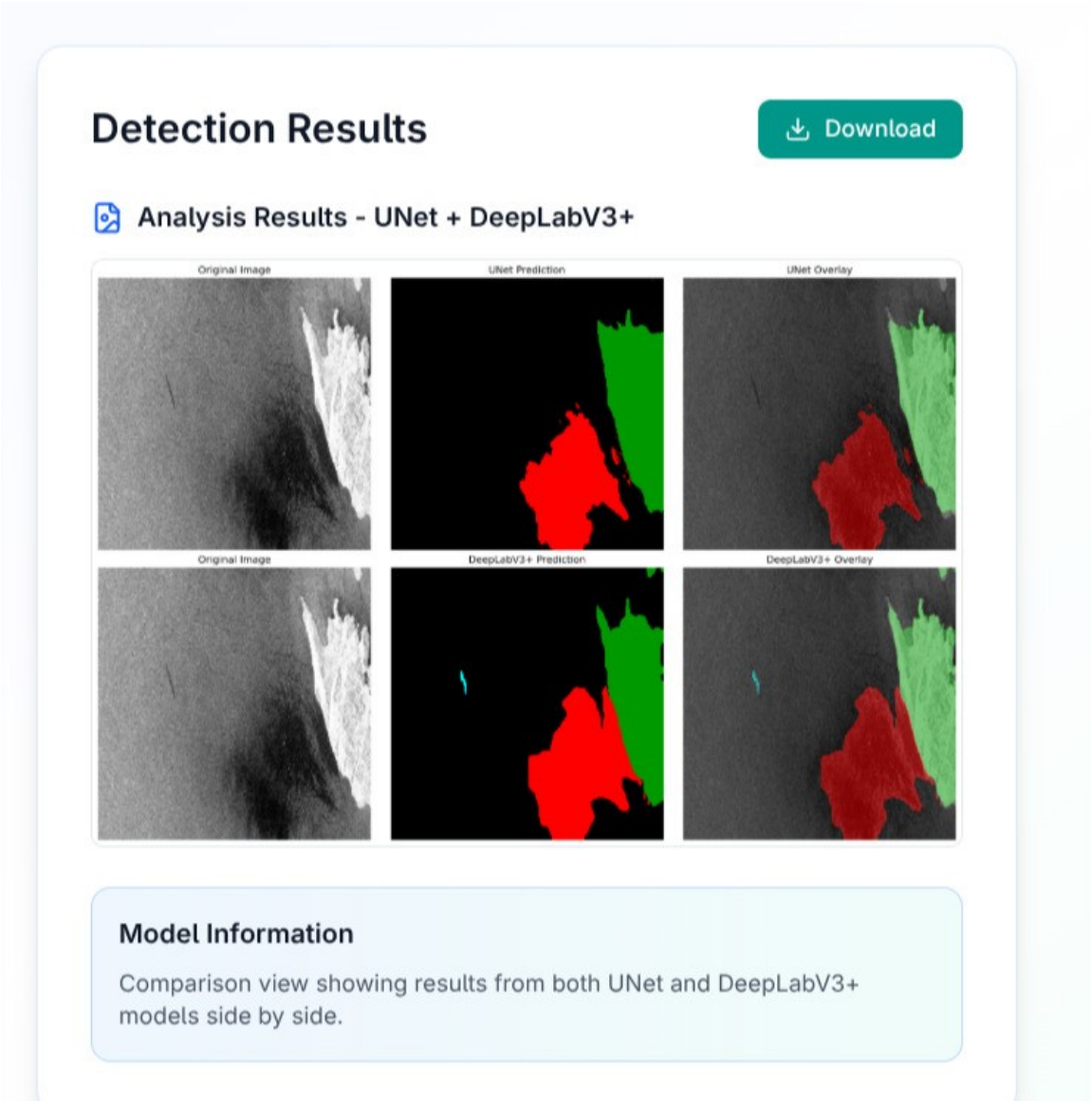


Figure 8.4 Segmented output of SAR images.

Results: Aerial Image Mask:

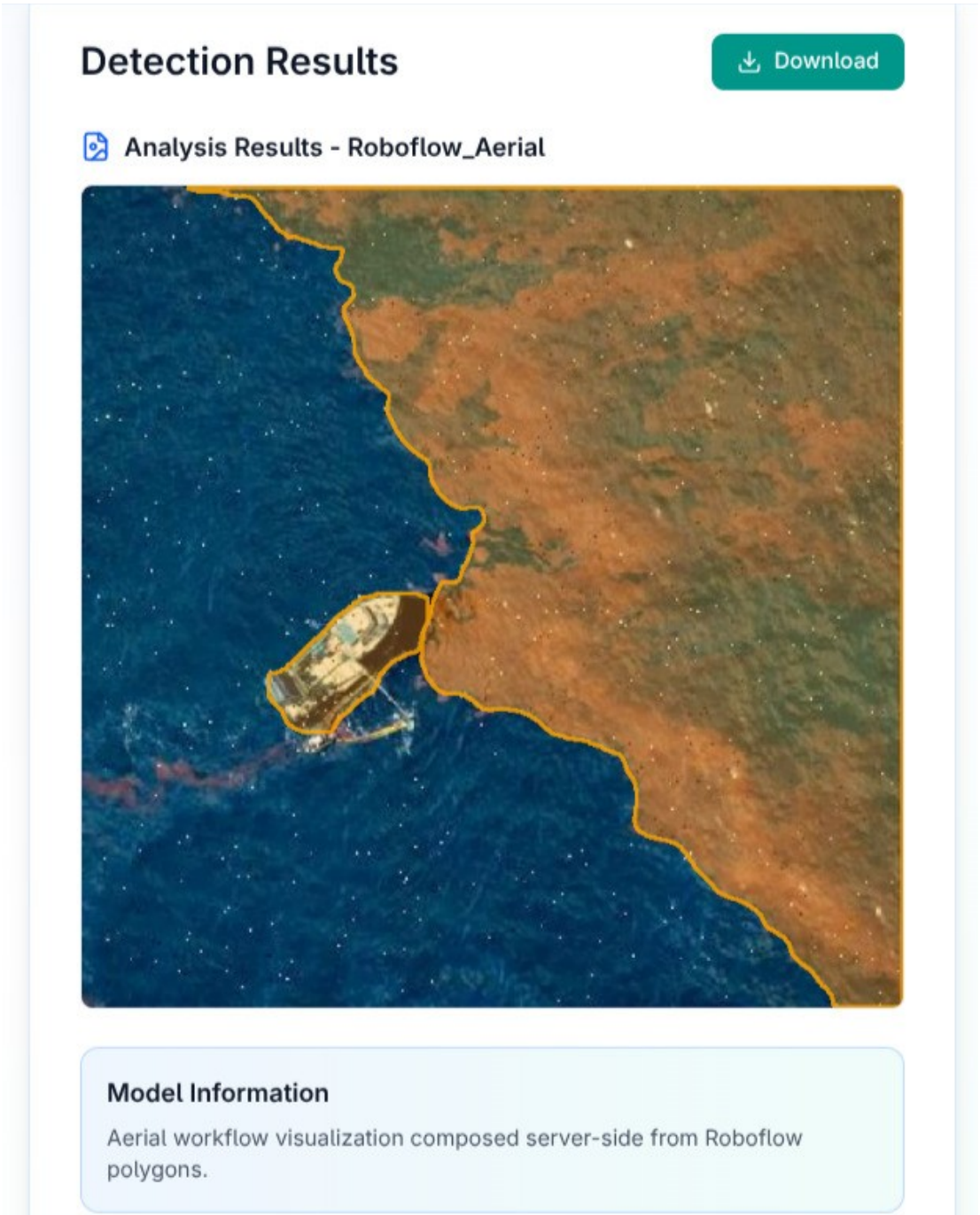


Figure 8.5 Segmented output of Aerial Image.

REFERENCES

[1] F. Del Frate, A. Petrocchi, J. Lichtenegger and G. Calabresi (2000). Neural networks for oil spill detection using ERS-SAR data.

https://doi.org/10.1109/36.868885

[2] Fingas, M.; Fieldhouse, B. (2003). Studies of the formation process of water-in-oil emulsions

https://doi.org/10.1016/S0025-326X(03)00212-1

[3] Krestenitis, M., Orfanidis, G., Ioannidis, K., Avgerinakis, K., Vrochidis, S., & Kompatsiaris, I. (2019). Oil Spill Identification from Satellite Images Using Deep Neural Networks.

https://doi.org/10.3390/rs11151762

[4] Rami Al-Ruzouq ,Mohamed Barakat A,.Gibril, Abdallah Shanableh, Abubakir KaisOsman HamedSaeed Al-Mansoori, Mohamad Ali Khalil (2020). Sensors, Features, and Machine Learning for Oil Spill Detection and Monitoring: A Review

https://doi.org/10.3390/rs12203338

[5] Bianchi, F. M., Espeseth, M. M., & Borch, N. (2020). Large-Scale Detection and Categorization of Oil Spills from SAR Images with Deep Learning.

https://doi.org/10.3390/rs12142260

[6]De Kerf, T.; Gladines, J.; Sels, S.; Vanlanduit, S. (2020). Oil spill detection using machine learning and infrared images.

https://doi.org/10.3390/rs12244090

[7] Fingas, M. (2021). Visual Appearance of Oil on the Sea.
https://doi.org/10.3390/jmse9010097

[8] Kiran Maharana, **Surajit Mondal**, **Bhushankumar Nemade (2022).** A review: Data pre-processing and data augmentation techniques
https://doi.org/10.1016/j.gltp.2022.04.020

[9] Zhang, J., Yang, P., & Ren, X. (2024). Detection of Oil Spill in SAR Image Using an Improved DeepLabV3+.
https://doi.org/10.3390/s24175460

[10] Zhen S., Qingshu Y., Nanyang Y., Siyu C., Jianhang Z., Jun Z., Shaojie S. (2024). Utilizing Deep Learning Algorithms for Automated Oil Spill Detection In Medium Resolution Optical Imagery.
https://doi.org/10.1016/j.marpolbul.2024.116777

[11] Kang, J., Yang, C., Yi, J., & Lee, Y. (2024). Detection of Marine Oil Spill from PlanetScope Images Using CNN and Transformer Models.
https://doi.org/10.3390/jmse12112095

[12] Gkountakos, K., Melitou, M., Ioannidis, K., Demestichas, K., Vrochidis, S., & Kompatsiaris, I. (2025). LADOS: Aerial Imagery Dataset for Oil Spill Detection, Classification, and Localization Using Semantic Segmentation. Data, 10(7), 117.
https://doi.org/10.3390/data10070117

[13] Zhang, Y., Xing, J., Chen, W. et al (2025). A novel YOLOv11-Driven deep learning algorithm for UAV multispectral oil spill detection in Inland lakes.
https://doi.org/10.1007/s44443-025-00117-z

[14] Samkhaniani, M., Khoshand, A., & Ezati, S. (2025). Deep Learning-Based Hyperspectral Oil Spill Detection for Marine Pollution Monitoring in the Gulf of Mexico: A Step Marine Pollution Monitoring and Sdg 14 Compliance.
https://dx.doi.org/10.213 9/ssrn.5354010

[15] Shi, jianting and Jiao, Tianyu and Ames, Daniel P. and Li, Zhijun. (2025). Improved Lightweight Marine Oil Spill Detection Algorithm of Yolov8.
http://dx.doi.org/10.2139/ssrn.5353307