A Real-time Research Project / Societal Related Project

Report on

ATLAS AI - Automated interview questions Generator using AI

Submitted in Partial fulfillment of requirements for B.Tech II Year II Semester course

By

N.Sheshi Vardhan	24BD5A0506
K.Pruthvidhar Reddy	23BD1A053N
D.Jayanth	23BD1A052X
K.Syanthan Kumar Reddy	23BD1A0534
K.Mohan Vamsi	23BD1A0536

Under the guidance of

Mrs.Swetha
Assistant Professor, CSE Dept
& Mrs.Manasa
Assistant Professor, IT Dept

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
(AN AUTONOMOUS INSTITUTION)
Accredited by NBA & NAAC, Approved by AICTE, Affiliated to JNTUH.
Narayanaguda, Hyderabad, Telangana-29
2024-25

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
(AN AUTONOMOUS INSTITUTION)
Accredited by NBA & NAAC, Approved by AICTE, Affiliated to JNTUH
Narayanaguda, Hyderabad, Telangana-29

CERTIFICATE

This is to certify that this is a bonafide record of the project report titled "ATLAS" which is being presented as the Real-time Research Project / Societal Related Project report by

1.N.Sheshi Vardhan	24BD5A0506
2.K.Pruthvidhar Reddy	23BD1A053N
3.D.Jayanth	23BD1A052X
4.K.Syanthan Kumar Reddy	23BD1A0534
5.K.Mohan Vamsi	23BD1A0536

In partial fulfillment for the II Year II Semester Course RTRP in KMIT affiliated to the Jawaharlal Nehru Technological University Hyderabad, Hyderabad

Mentors
Mrs.Swetha
Mrs.Manasa

Program Coordinator

(Mr Shailesh Gangakhedkar)

Submitted for Final Project Review held on _____

Vision & Mission of KMIT

Vision of KMIT

- To be the fountainhead in producing highly skilled, globally competent engineers
- . Producing quality graduates trained in the latest software technologies and related tools and striving to make India a world leader in software products and services.

Mission of KMIT

- To provide a learning environment that inculcates problem solving skills, professional, ethical responsibilities, lifelong learning through multi modal platforms and prepares students to become successful professionals.
- To establish an industry institute Interaction to make students ready for the industry.
- To provide exposure to students on the latest hardware and software tools.
- To promote research-based projects/activities in the emerging areas of technology convergence.
- To encourage and enable students to not merely seek jobs from the industry but also to create new enterprises.
- To induce a spirit of nationalism which will enable the student to develop, understand India's challenges and to encourage them to develop effective solutions.
- To support the faculty to accelerate their learning curve to deliver excellent service to students.

PROGRAM OUTCOMES (POs)

PO1. Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2. Problem Analysis: Identify formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences

PO3. Design/Development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4. Conduct Investigations of Complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5. Modern Tool Usage: Create select, and, apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6. The Engineer and Society: Apply reasoning informed by contextual knowledge to societal, health, safety. Legal und cultural issues and the consequent responsibilities relevant to professional engineering practice.

PO7. Environment and Sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts and demonstrate the knowledge of, and need for sustainable development.

PO8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9. Individual and Team Work: Function effectively as an individual, and as a member or leader in diverse teams and in multidisciplinary settings.

PO10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11. Project Management and Finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12. Life-Long Learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROJECT OUTCOMES

P1:Creating a Seamless web application for job aspirants

P2:A robust engineered backend services of best performace

P3: Creating various tool like Resume Builder ,Interview Questions Generation etc

P4 : Secured authentication to controll the access to the application

MAPPING PROJECT OUTCOMES WITH PROGRAM OUTCOMES

PO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
P1	M	H	H	M	H	L	L	M	M	H	L	M
P2	L	H	H	L	H	M	L	M	H	H	M	M
P3	L	M	H	L	H	M	L	H	M	H	H	M
P4	M	M	M	L	M	H	L	H	M	H	L	H

L – LOW M –MEDIUM H– HIGH

DECLARATION

We hereby declare that the results embodied in the dissertation entitled “SAR Marine surveillance ” has been carried out by us together during the academic year 2024-25 as a partial fulfillment of the B.Tech III Year I Semester Course “Real-time Research Project / Societal Related Project”. We have not submitted this report to any other Course/College.

Student Name	Roll no.
1.N.Sheshi Vardhan	24BD5A0506
2.More Omkar	24BD5A0504
3.Sharath Kumar	24BD5A0502
4.Mahesh Kiran	24BD5A0509
5.K.Mohan Vamsi	23BD1A0536

ACKNOWLEDGEMENT

We take this opportunity to thank all the people who have rendered their full support to our project work. We render our thanks to Dr. B L Malleswari, Principal who encouraged us to do the Project.

We are grateful to Mr. Neil Gogte, Founder & Director and Mr. S. Nitin, Director, for facilitating all the amenities required for carrying out this project.

We express our sincere gratitude to Ms. Deepa Ganu, Director Academic for providing an excellent environment in the college.

We are also thankful to Mr.Shailesh Gangakhedkar, Real-Time Research Project Program Coordinator for providing us with time to make this project a success within the given schedule.

We are also thankful to our Project Mentors Dr. Bhanu Mahesh & Ms. Khusi, for their valuable guidance and encouragement given to us throughout the project work.

We would like to thank the entire KMIT faculty, who helped us directly and indirectly in the completion of the project.

We sincerely thank our friends and family for their constant motivation during the project work.

1.N.Sheshi Vardhan	24BD5A0506
2.Pruthvidhar Reddy	23BD1A053N
3.D.Jayanth	23BD1A052X
4.K.Syanthan Kumar Reddy	23BD1A0534
5.K.Mohan Vamsi	23BD1A0536

ABSTRACT

In this project, a prototype and implementation of an intelligent SAR (Synthetic Aperture Radar) Marine Surveillance System is demonstrated. The proposed system focuses on large-scale ship and oil spill detection using advanced deep learning models such as TransUNet for segmentation and Deformable DETR for object detection. The backend leverages FastAPI for efficient model serving and MongoDB for storing geospatial and detection metadata, while the frontend interface, developed using React, visualizes real-time surveillance data and detection outputs.This platform enables automated identification and classification of marine targets from satellite imagery, supporting critical applications in environmental monitoring, maritime security, and illegal activity detection.

LIST OF FIGURES

OBJECTIVE

The main objective of this project is to design and develop an intelligent SAR Marine Surveillance System capable of accurately detecting and monitoring ships and oil spills over large oceanic areas using Synthetic Aperture Radar (SAR) imagery. The system aims to leverage deep learning techniques — specifically TransUNet for segmentation and Deformable DETR for object detection — to achieve high precision and robustness under challenging maritime conditions.

TECH STACK

1.Model Development Layer

Purpose: Training and experimentation of deep learning models for ship and oil spill detection.

1.PyTorch – For developing and training deep learning models (TransUNet & Deformable DETR).

2.Google Colab – For GPU-accelerated training and prototyping of large SAR datasets.

3.NumPy, OpenCV, Matplotlib – For preprocessing, visualization, and evaluation of training results.




2,Backend Layer

Purpose: Serving trained models, handling data processing, and managing API endpoints.

1.FastAPI – Backend framework for building high-performance APIs and model inference endpoints.

2.MongoDB – NoSQL database for storing detection results, user data, and metadata.

3.PyVips – For fast and memory-efficient image processing of large SAR images.

4.DZI (Deep Zoom Image Tools – For generating deep zoom tiles for visualization.

5.Cloudinary – For secure cloud-based storage and delivery of high-resolution images.



TECH STACK

3.Frontend Layer

Purpose: Providing an interactive and intuitive visualization of detection results.

1.React.js – For building the dynamic, component-based user interface.

2.TailwindCSS – For rapid UI styling and responsive design**.**

3.OpenSeadragon – For high-resolution image viewing and zoomable map interactions.

4.Firebase Authentication – For secure and seamless user login and access control.




4.Integration & Deployment

Purpose: To ensure scalability, performance, and ease of access.

1.Cloudinary – Image and asset management for large-scale data.

2.FastAPI + React Integration – Unified API and frontend communication for real-time visualization**.**

3.GitHub / Render / Vercel (optional) – For version control and deployment**.**



DATASET DESCRIPTION

1.Ship Detection- SARscope: Synthetic Aperture Radar Maritime Images

SARscope is a high-quality, curated dataset created by combining HRSID and OPEN-SSDD sources, designed specifically for ship detection and instance segmentation in Synthetic Aperture Radar (SAR) imagery.It simplifies access to robust maritime data for research, analysis, and model training.

Highlights

1.Unified Dataset for Dual Tasks - Combines two leading SAR datasets—HRSID and OPEN-SSDD—allowing models to be trained and tested on both ship detection and instance segmentation tasks simultaneously.

2.High-Resolution Imagery - Includes 5,719 SAR images (640×640) with 17,708 labeled ship instances, covering multiple radar polarizations (VV and VH) for better generalization.

3.Optimized for Efficiency - All images are resized to 640×640, balancing detail and processing speed for faster model training and inference.

4.Data Integrity & Compression - Leveraging Roboflow's lossless compression, SARscope ensures consistent data quality while reducing download size and improving accessibility.

5.Well-Balanced Splits - Offers a 70-20-10 train-validation-test split to support robust evaluation and fair benchmarking across different ML frameworks.

Format & Compatibility

1.Annotation Format: COCO-MMDetection compatible
2.Ideal For: PyTorch, MMDetection, and other computer vision research frameworks
3.Total Image Count: 6,735
4.Primary Tasks: Ship Detection, Instance Segmentation

DATASET DESCRIPTION

2.Oil Spill Detection- Synthetic Aperture Radar Oil Spill Segmentation

The Deep-SAR Oil Spill (SOS) dataset is a high-quality, curated benchmark resource created by the High-Performance Spatial Computational Intelligence Lab (HPSCIL) of China University of Geosciences, Wuhan, and RS-IDEA of Wuhan University. It is specifically designed for oil spill detection and instance segmentation in Synthetic Aperture Radar (SAR) imagery, advancing state-of-the-art algorithms for maritime environmental monitoring and disaster response.

Highlights

1. Dual Geographic Coverage
Encompasses two critical oil spill regions—the Gulf of Mexico and the Persian Gulf—providing diverse environmental conditions and spill characteristics for robust model training and evaluation.

2. Multi-Satellite Imagery
Includes SAR images from multiple satellite platforms (ALOS and Sentinel-1A) covering the Deepwater Horizon disaster (2010) and the Kuwait coastal spill (2017), offering temporal and sensor diversity.

3. Data Augmentation Pipeline
Original dataset extended through systematic data enhancement techniques including cropping, rotation, and noise injection to improve model generalization and handle limited real-world oil spill samples.

4. Balanced Train-Test Splits
Provides separate training and validation sets for both geographic regions with 4:1 split ratios, supporting fair benchmarking across different deep learning frameworks

5. Ground Truth Annotations
All images include pixel-level ground truth masks for precise segmentation tasks, enabling supervised learning for oil spill boundary delineation and area estimation.

Format & Compatibility
1.Annotation Format: Pixel-level segmentation masks
2.Ideal For: PyTorch, MMDetection, and other computer vision research frameworks
3.Total Image Count: 8070
4.Primary Tasks: Oil Spill Detection, Instance Segmentation

DATA PREPROCESSING

The data processing pipeline transforms massive satellite images (often 160MB+ per image) into manageable tiles, runs deep learning detection models on each tile independently, and aggregates all predictions into a single detections.json file containing global coordinates and metadata. This approach enables efficient processing of high-resolution SAR imagery that exceeds GPU memory constraints while maintaining spatial accuracy across the entire scene.

1.Image Tiling

Large SAR images are divided into smaller, fixed-size tiles to enable efficient processing within GPU memory constraints. Tiles overlap slightly to ensure ships at tile boundaries are not missed during detection.

Configuration: Tile size of 640×640 or 800×800 pixels with 64-128 pixel overlap between adjacent tiles. Each tile stores spatial metadata including its position in the original image and geographic bounds.

2.Ship Detection

The detection model processes each tile independently to identify ships and generate bounding boxes with confidence scores. Common models include Deformable DETR, YOLO, or specialized SAR ship detectors trained on maritime datasets.

Quality Control: Predictions are filtered using a minimum confidence threshold (typically 0.45) and Non-Maximum Suppression removes overlapping detections within each tile.

3.Coordinate Transformation

Detections from individual tiles are converted to a unified coordinate system. First, tile-local coordinates are adjusted by adding the tile's offset to obtain global pixel positions within the original image. Then, affine transformation matrices convert pixel coordinates to geographic coordinates (latitude/longitude).

Duplicate Removal: Since overlapping tiles may detect the same ship multiple times, global Non-Maximum Suppression identifies and removes duplicate detections across all tiles, keeping only the highest confidence instance.

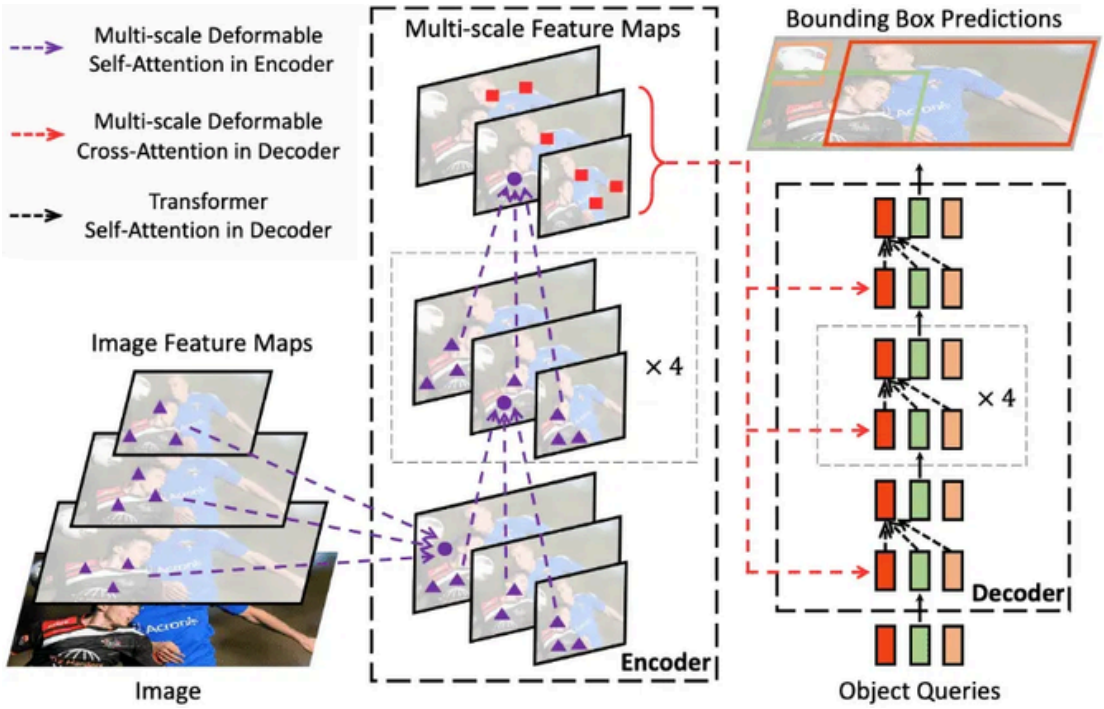
4.Global Detections JSON

All ship detections are aggregated into a single JSON file containing metadata about the source image and processing configuration, individual detection records with both pixel and geographic coordinates, and summary statistics on detection quality and distribution.

Each detection includes a unique identifier, confidence score, bounding box coordinates, geographic centroid, and physical properties such as estimated ship length and width in meters.

ARCHITECTURE

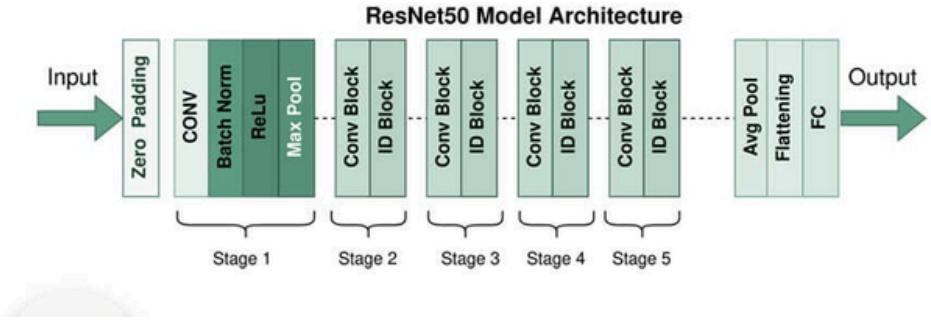
1.Ship Detection- Deformable DETR



Our model is based on Deformable DETR (Deformable DEtection TRansformer) — an improved version of the DETR architecture designed for efficient object detection with faster convergence and better handling of multi-scale features.

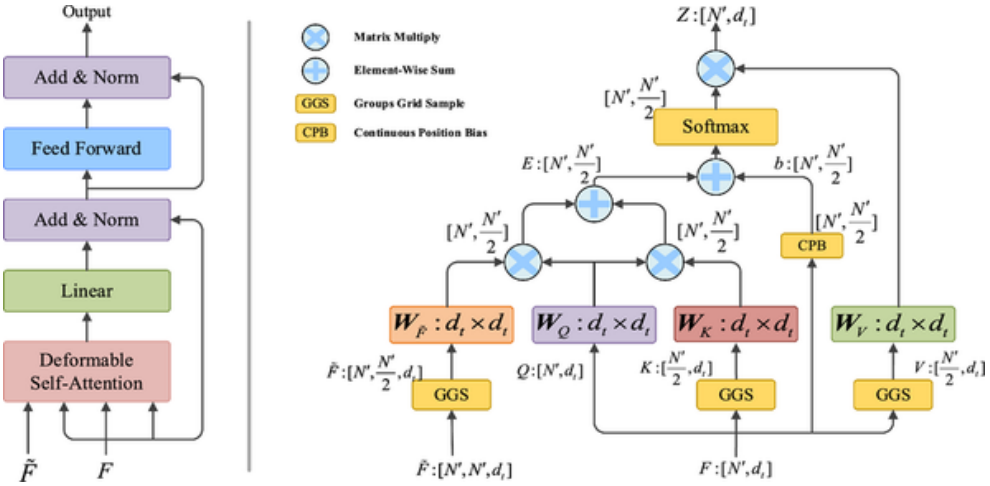
1. Image Feature Extraction

The input SAR images are first passed through a CNN backbone (ResNet-50) to extract features from different layers.These multi-scale feature maps help the model detect both small and large ships clearly.



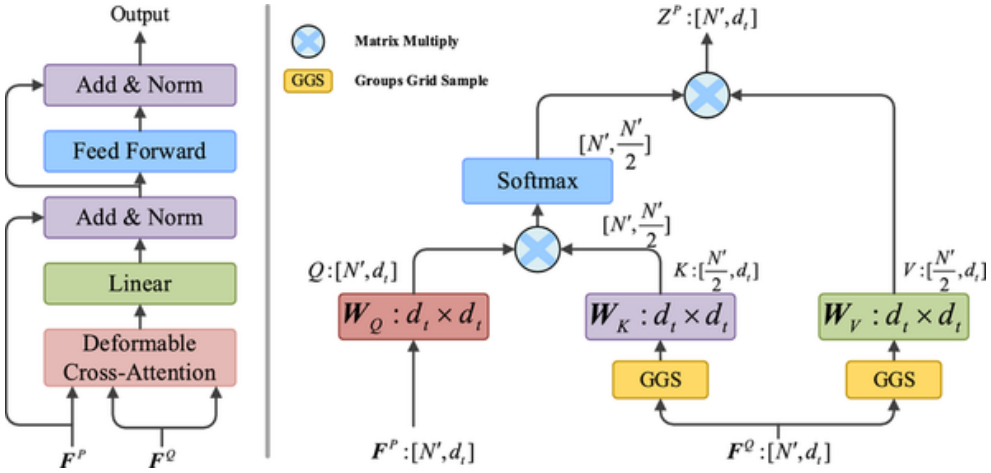
2. Encoder

The encoder uses multi-scale deformable attention, which allows it to look only at a few key points instead of the whole image.This makes the model faster and more accurate in learning useful information from SAR images.



3. Decoder

The decoder receives a fixed set of object queries, each representing a possible ship.It uses cross-attention to match these queries with important areas in the image and then refines them to make precise predictions.



4. Output Prediction

For each object query, the model predicts:

-A bounding box around the ship

-A label (ship or background)

Optionally, it can also produce segmentation masks to show exact ship boundaries.

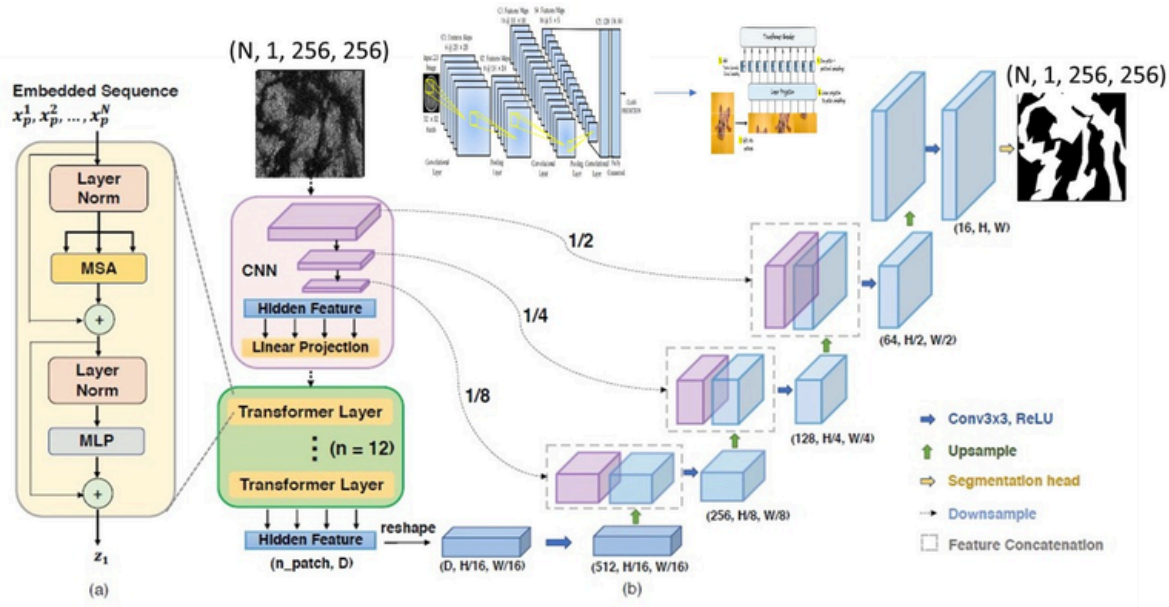
5. Training

The model was trained on Google Colab using PyTorch and the SARscope dataset.

We used a combination of losses — L1, GIoU, and Cross-Entropy — to improve both detection accuracy and box precision.

ARCHITECTURE

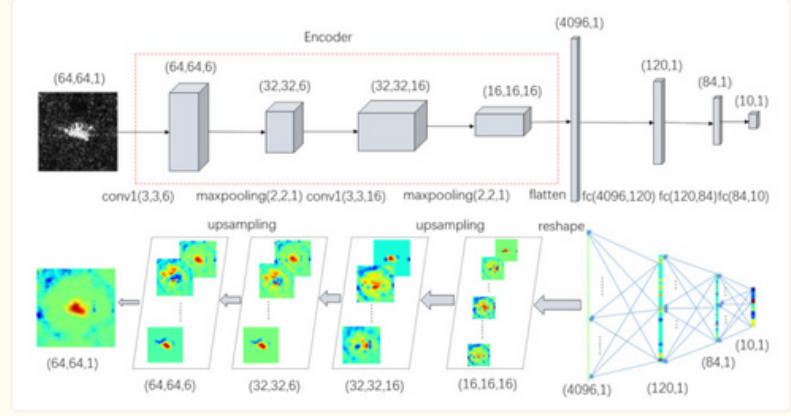
2.Oil Spill Detection- TransUNet



TransUNet is a hybrid CNN-Transformer architecture that combines a ResNet CNN encoder for extracting detailed spatial features with a 12-layer Vision Transformer encoder to capture global context and long-range dependencies across image patches. The encoded features are then progressively upsampled through a U-Net-style decoder with skip connections from the CNN layers, fusing high-resolution spatial details with global semantic information to produce precise pixel-level segmentation masks. This design addresses the limitation of pure Transformers (which lose fine spatial details) and pure CNNs (which struggle with global context), making it particularly effective for medical image and SAR oil spill segmentation tasks

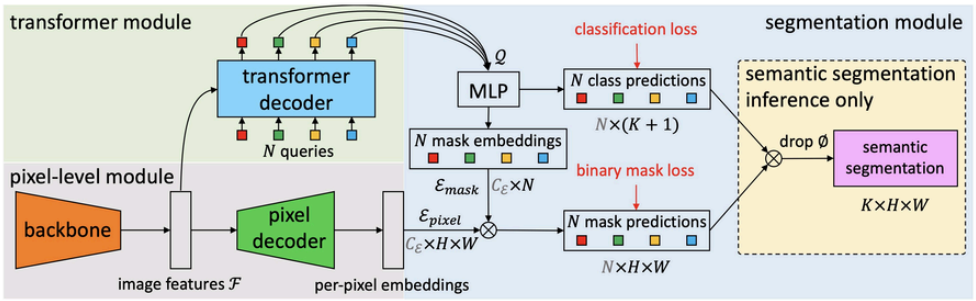
1. CNN Encoder

Extracts basic features like edges, textures, and patterns from the input SAR image using convolutional layers. Acts as the foundation by converting raw pixel data into meaningful spatial representations that the transformer can process.



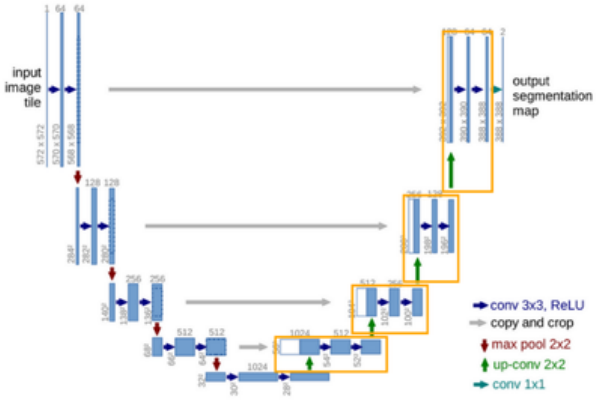
2. Transformer Layers (n=12)

Captures long-range dependencies and global context across the entire image using self-attention mechanisms. These 12 stacked layers learn relationships between distant oil spill regions that CNNs alone would miss



3. Multi-Scale Decoder (1/8, 1/4, 1/2)

Progressively upsamples low-resolution features back to full image size while fusing information from different scales. Combines coarse semantic understanding with fine-grained spatial details for precise boundary detection



4. Feature Concatenation

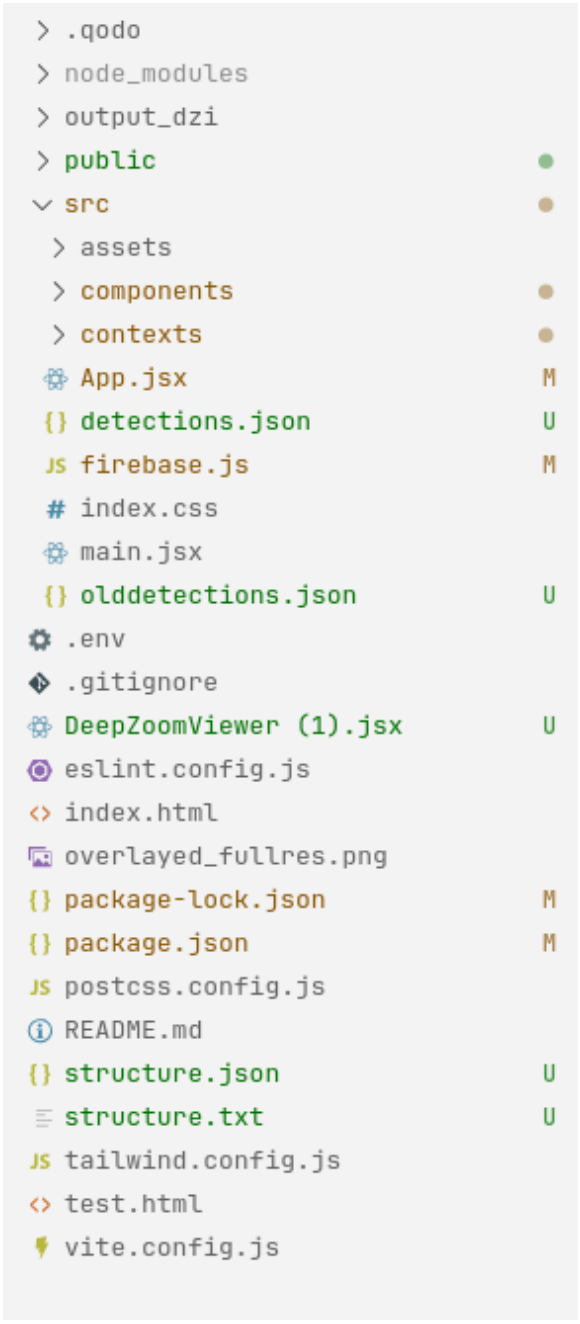
Merges features from the encoder's skip connections with upsampled decoder features at matching resolutions. Preserves both high-level context and low-level details, ensuring the model doesn't lose important spatial information during downsampling.

5. Segmentation Head

Final output layer that produces the binary mask (oil spill vs background) for each pixel in the 256×256 image. Converts learned features into actionable segmentation results that can be overlaid on the original SAR imagery.

FRONTEND

Project structure



The SARClient project uses a modular React setup where the src/ folder holds the main code.

- components/ – reusable UI and viewer components.
- contexts/ – global state management.
- detections.json – bounding box data for visualization.
- firebase.js – handles authentication.
- DeepZoomViewer.jsx – displays DZI images using OpenSeadragon.
- Configuration files like vite.config.js and tailwind.config.js manage build and styling.

Code - App

```
// src/App.jsx
import { BrowserRouter as Router, Routes, Route, Navigate } from "react-router-dom";
import { useState, useEffect } from "react";
import { AuthProvider, useAuth } from "./contexts/AuthContext";
import SignInPage from "./components/SignIn";
import SignUpPage from "./components/SignUp";
import PhoneAuthPage from "./components/PhoneAuthPage";
import Dashboard from "./components/Dashboard";
import LandingPage from "./components/LandingPage";
import SarLoader from "./components/SARLoader";
import ImageProcessingPage from "./components/ImageUploadingPage";
import SarFooter from "./components/Footer";
function AppContent() {
 const { currentUser, loading } = useAuth();
 const [phoneAuth, setPhoneAuth] = useState(false);
 const [checkingPhoneAuth, setCheckingPhoneAuth] = useState(true);
 useEffect(() => {
  const checkPhoneAuth = async () => {
   try {
    const res = await fetch("http://localhost:3001/api/auth/status", { credentials: "include" });
    if (res.ok) {
     const data = await res.json();
     setPhoneAuth(data.authenticated);
    }
   } catch (e) {
    console.error(e);
   } finally {
    setCheckingPhoneAuth(false);
   }
  };
  if (!loading) checkPhoneAuth();
 }, [loading]);
 if (loading || checkingPhoneAuth) return <SarLoader />;
 const isAuthenticated = currentUser || phoneAuth;
 const authType = currentUser ? "firebase" : phoneAuth ? "phone" : null;
 return (
  <div className="w-screen h-full bg-gradient-to-br from-slate-900 via-blue-900 to-indigo-900">
   <Router>
    <Routes>
     <Route path="/sign-in" element={!isAuthenticated ? <SignInPage /> : <Navigate to="/dashboard" />} />
     <Route path="/sign-up" element={!isAuthenticated ? <SignUpPage /> : <Navigate to="/dashboard" />} />
          <Route path="/phone-auth" element={!isAuthenticated ? <PhoneAuthPage onAuthSuccess={() => setPhoneAuth(true)} /> :
<Navigate to="/dashboard" />} />
       <Route path="/dashboard" element={isAuthenticated ? <Dashboard authType={authType} onLogout={() => setPhoneAuth(false)}
/> : <Navigate to="/" />} />
     <Route path="/upload" element={<ImageProcessingPage />} />
     <Route path="/" element={isAuthenticated ? <Navigate to="/dashboard" /> : <LandingPage />} />
    </Routes>
   </Router>
   <SarFooter />
  </div>
 );
}

function App() {
 return (
  <AuthProvider>
   <AppContent />
  </AuthProvider>
 );
}

export default App;
```

Code - Ship viewer

/src/ShipViewer.js

```
import React, { useEffect, useRef } from "react";
import OpenSeadragon from "openseadragon";
import detections from "../detections.json";

const ShipViewer = () => {
  const viewer1Ref = useRef(null), viewer2Ref = useRef(null);
  useEffect(() => {
    if (viewer1Ref.current || viewer2Ref.current) return;
    const viewer1 = OpenSeadragon({
      id: "osd-viewer-1",
      prefixUrl: "https://openseadragon.github.io/openseadragon/images/",
      tileSources: "/output_dzi.dzi",
      showNavigator: true,
    });
    const viewer2 = OpenSeadragon({
      id: "osd-viewer-2",
      prefixUrl: "https://openseadragon.github.io/openseadragon/images/",
      tileSources: "/output_dzi.dzi",
      showNavigator: true,
    });
    viewer1Ref.current = viewer1;
    viewer2Ref.current = viewer2;

    const sync = (src, dest) => {
      const center = src.viewport.getCenter();
      const zoom = src.viewport.getZoom();
      dest.viewport.panTo(center);
      dest.viewport.zoomTo(zoom);
    };
    viewer1.addHandler("viewport-change", () => sync(viewer1, viewer2));
    viewer2.addHandler("viewport-change", () => sync(viewer2, viewer1));

    viewer1.addHandler("open", () => {
      detections.forEach((b) => {
        const box = document.createElement("div");
        box.className = "bounding-box";
        box.style.border = "1px solid red";
        viewer1.addOverlay({
          element: box,
          location: viewer1.viewport.imageToViewportRectangle(
            new OpenSeadragon.Rect(b.x, b.y, b.w, b.h)
          ),
        });
      });
    });
  }, []);
  return (
    <div style={{ display: "flex" }}>
      <div id="osd-viewer-1" style={{ width: "50%", height: "90vh" }} />
      <div id="osd-viewer-2" style={{ width: "50%", height: "90vh" }} />
      <style>{`
        .bounding-box {
          position: absolute;
          pointer-events: none;
          box-sizing: border-box;
        }
      `}</style>
    </div>
  );
};

export default ShipViewer;
```

Code - Oil Spill Viewer

```
import React, { useEffect, useRef } from 'react';
import OpenSeadragon from 'openseadragon';

const OilSpillViewer = () => {
  const v1 = useRef(null), v2 = useRef(null), v3 = useRef(null);

  useEffect(() => {
    if (v1.current) return;

    v1.current = OpenSeadragon({
      id: "viewer-1",
      prefixUrl: "https://cdn.jsdelivr.net/npm/openseadragon@4.1/build/openseadragon/images/",
      tileSources: { type: 'image', url: '/actual.png' },
    });
    v2.current = OpenSeadragon({
      id: "viewer-2",
      prefixUrl: "https://cdn.jsdelivr.net/npm/openseadragon@4.1/build/openseadragon/images/",
      tileSources: { type: 'image', url: '/merged_mask.png' },
    });
    v3.current = OpenSeadragon({
      id: "viewer-3",
      prefixUrl: "https://cdn.jsdelivr.net/npm/openseadragon@4.1/build/openseadragon/images/",
      tileSources: { type: 'image', url: '/overlayed_fullres.png' },
    });

    const sync = (src, t1, t2) => {
      t1.viewport.panTo(src.viewport.getCenter());
      t1.viewport.zoomTo(src.viewport.getZoom());
      t2.viewport.panTo(src.viewport.getCenter());
      t2.viewport.zoomTo(src.viewport.getZoom());
    };

    v1.current.addHandler("viewport-change", () => sync(v1.current, v2.current, v3.current));
    v2.current.addHandler("viewport-change", () => sync(v2.current, v1.current, v3.current));
    v3.current.addHandler("viewport-change", () => sync(v3.current, v1.current, v2.current));
  }, []);

  return (
    <div style={{ display: 'flex', gap: '1rem' }}>
      <div id="viewer-1" style={{ flex: 1, height: '600px' }} />
      <div id="viewer-2" style={{ flex: 1, height: '600px' }} />
      <div id="viewer-3" style={{ flex: 1, height: '600px' }} />
    </div>
  );
};

export default OilSpillViewer;
```

Code - Auth Context

```
import { createContext, useContext, useEffect, useState } from 'react';
import {
  createUserWithEmailAndPassword, signInWithEmailAndPassword, signOut,
  onAuthStateChanged, GoogleAuthProvider, signInWithPopup,
  sendPasswordResetEmail, updateProfile, RecaptchaVerifier,
  signInWithPhoneNumber
} from 'firebase/auth';
import { auth } from '../firebase';

const AuthContext = createContext({});
export const useAuth = () => useContext(AuthContext);

export const AuthProvider = ({ children }) => {
  const [currentUser, setCurrentUser] = useState(null);
  const [loading, setLoading] = useState(true);

  const signup = async (email, password, displayName) => {
    const cred = await createUserWithEmailAndPassword(auth, email, password);
    if (displayName) await updateProfile(cred.user, { displayName });
    return cred;
  };

  const login = (email, password) => signInWithEmailAndPassword(auth, email, password);

  const loginWithGoogle = () => signInWithPopup(auth, new GoogleAuthProvider());

  const logout = () => signOut(auth);

  const resetPassword = (email) => sendPasswordResetEmail(auth, email);

  const sendOTP = async (phoneNumber) => {
    if (!window.recaptchaVerifier) {
      window.recaptchaVerifier = new RecaptchaVerifier(auth, 'recaptcha-container', { size: 'invisible' });
    }
    return await signInWithPhoneNumber(auth, phoneNumber, window.recaptchaVerifier);
  };

  const verifyOTP = (confirmationResult, code) => confirmationResult.confirm(code);

  useEffect(() => {
    return onAuthStateChanged(auth, (user) => {
      setCurrentUser(user);
      setLoading(false);
    });
  }, []);

  return (
    <AuthContext.Provider value={{ currentUser, signup, login, loginWithGoogle, logout, resetPassword, sendOTP, verifyOTP }}>
      {!loading && children}
    </AuthContext.Provider>
  );
};
```

Code - Auth Context

```
import { createContext, useContext, useEffect, useState } from 'react';
import {
  createUserWithEmailAndPassword, signInWithEmailAndPassword, signOut,
  onAuthStateChanged, GoogleAuthProvider, signInWithPopup,
  sendPasswordResetEmail, updateProfile, RecaptchaVerifier,
  signInWithPhoneNumber
} from 'firebase/auth';
import { auth } from '../firebase';

const AuthContext = createContext({});
export const useAuth = () => useContext(AuthContext);

export const AuthProvider = ({ children }) => {
  const [currentUser, setCurrentUser] = useState(null);
  const [loading, setLoading] = useState(true);

  const signup = async (email, password, displayName) => {
    const cred = await createUserWithEmailAndPassword(auth, email, password);
    if (displayName) await updateProfile(cred.user, { displayName });
    return cred;
  };

  const login = (email, password) => signInWithEmailAndPassword(auth, email, password);

  const loginWithGoogle = () => signInWithPopup(auth, new GoogleAuthProvider());

  const logout = () => signOut(auth);

  const resetPassword = (email) => sendPasswordResetEmail(auth, email);

  const sendOTP = async (phoneNumber) => {
    if (!window.recaptchaVerifier) {
      window.recaptchaVerifier = new RecaptchaVerifier(auth, 'recaptcha-container', { size: 'invisible' });
    }
    return await signInWithPhoneNumber(auth, phoneNumber, window.recaptchaVerifier);
  };

  const verifyOTP = (confirmationResult, code) => confirmationResult.confirm(code);

  useEffect(() => {
    return onAuthStateChanged(auth, (user) => {
      setCurrentUser(user);
      setLoading(false);
    });
  }, []);

  return (
    <AuthContext.Provider value={{ currentUser, signup, login, loginWithGoogle, logout, resetPassword, sendOTP, verifyOTP }}>
      {!loading && children}
    </AuthContext.Provider>
  );
};
```

BACKEND

Code - App

```
from fastapi import FastAPI
from routes import dzi_routes, detection_routes

app = FastAPI()

# Register routes
app.include_router(dzi_routes.router)
app.include_router(detection_routes.router)
```

Creates a FastAPI application instance and registers route blueprints for DZI file handling and ship detection functionality. This serves as the main entry point that connects all API routes together.

Code - config.py

```
from pathlib import Path
import os

BASE_DIR = Path(__file__).resolve().parent.parent

UPLOADS_DIR = Path(os.getenv('UPLOADS_DIR', BASE_DIR / "shared/uploads"))
TILES_DIR = Path(os.getenv('TILES_DIR', BASE_DIR / "shared/tiles"))
OUTPUTS_DIR = Path(os.getenv('OUTPUTS_DIR', BASE_DIR / "shared/outputs"))

# Paths to models
SHIP_MODEL_PATH = Path(os.getenv('SHIP_MODEL_PATH', BASE_DIR / "model_server/models"))
OILSPILL_MODEL_PATH = Path(os.getenv('OILSPILL_MODEL_PATH', BASE_DIR / "model_server/models/oil_spill.pth"))
```

Defines the directory structure and file paths for the project using pathlib, establishing where uploads, tiles, outputs, and ML model files are stored. Sets up the base directory relative to the config file location and creates organized paths for data processing.

Code - dzi_service.py

```
import pyvips

def generate_dzi(input_path, output_prefix, tile_size=256):
    image = pyvips.Image.new_from_file(str(input_path), access='sequential')
    image.dzsave(str(output_prefix), tile_size=tile_size)
```

Converts large input images into Deep Zoom Image (DZI) format using PyVIPS, which creates a pyramid of tiles at different zoom levels for efficient web viewing. The function takes an image path and output prefix, then generates tiled versions at a specified size (256px) for progressive loading in viewers like OpenSeadragon.

Code - Ship detector.py

```
import os
import torch
from PIL import Image
from transformers import DeformableDetrForObjectDetection, DeformableDetrImageProcessor
import torchvision.ops as ops
from config import SHIP_MODEL_PATH

TILE_SIZE = 512
OVERLAP = 1
SCORE_THRESHOLD = 0.5
IOU_THRESHOLD = 0.5
DEVICE = "cuda" if torch.cuda.is_available() else "cpu"

model = DeformableDetrForObjectDetection.from_pretrained(SHIP_MODEL_PATH).to(DEVICE).eval()
processor = DeformableDetrImageProcessor.from_pretrained(SHIP_MODEL_PATH)
id2label = model.config.id2label if hasattr(model.config, 'id2label') else {0: "object"}

def detect_ships(tile_folder: str, zoom_level: str = "15") -> list:
    zoom_path = os.path.join(tile_folder, zoom_level)
    detections = []
    for tile_file in os.listdir(zoom_path):
        if not tile_file.endswith(".jpeg"):
            continue
        x_idx, y_idx = map(int, tile_file.replace(".jpeg", "").split("_"))
        tile_path = os.path.join(zoom_path, tile_file)
        tile_img = Image.open(tile_path).convert("RGB")
        offset_x = x_idx * TILE_SIZE - (OVERLAP if x_idx > 0 else 0)
        offset_y = y_idx * TILE_SIZE - (OVERLAP if y_idx > 0 else 0)
        detections.extend(detect_on_tile(tile_img, (offset_x, offset_y), tile_img.size[0] - 2 * OVERLAP, tile_img.size[1] - 2 * OVERLAP))
    return apply_nms(detections)

def detect_on_tile(tile_img, offset, content_w, content_h):
    with torch.no_grad():
        outputs = model(**processor(images=tile_img, return_tensors="pt").to(DEVICE))
    results = processor.post_process_object_detection(outputs, threshold=SCORE_THRESHOLD,
                                      target_sizes=torch.tensor([[tile_img.size[1], tile_img.size[0]]]).to(DEVICE))[0]

    detections = []
    for box, label, score in zip(results["boxes"], results["labels"], results["scores"]):
        x1, y1, x2, y2 = box.tolist()
        if OVERLAP <= x1 and OVERLAP <= y1 and x2 <= OVERLAP + content_w and y2 <= OVERLAP + content_h:
            detections.append({
                "x": offset[0] + (x1 - OVERLAP),
                "y": offset[1] + (y1 - OVERLAP),
                "w": x2 - x1,
                "h": y2 - y1,
                "label": id2label.get(int(label), str(label)),
                "score": score.item()
            })
    return detections

def apply_nms(detections):
    if not detections:
        return []
    boxes = torch.tensor([[d["x"], d["y"], d["x"] + d["w"], d["y"] + d["h"]] for d in detections])
    scores = torch.tensor([d["score"] for d in detections])
    return [detections[i] for i in ops.nms(boxes, scores, IOU_THRESHOLD)]
```

This code processes large satellite/SAR images by splitting them into 512×512 tiles with 1-pixel overlap, runs a Deformable DETR model on each tile to detect ships, then merges all detections using NMS to eliminate duplicates across tile boundaries.

Code - oil_spill_detector.py

```
import os
from pathlib import Path
import shutil
import torch
from services.oilspill_util import VisionTransformer, get_r50_b16_config, ResizeToTensor, predict_single_image
from services.stitch import stitch_predicted_folder
from config import OILSPILL_MODEL_PATH, OUTPUTS_DIR
from services.dzi_service import generate_dzi

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
config = get_r50_b16_config()
model = VisionTransformer(config, img_size=(224, 224), num_classes=config.n_classes).to(device)
model.load_state_dict(torch.load(str(OILSPILL_MODEL_PATH), map_location=device))
model.eval()
transform = ResizeToTensor(size=(224, 224))

def detect_oilspill(tile_folder: str, zoom_level: str = "15", image_id: str = None) -> dict:
    zoom_path = Path(tile_folder) / zoom_level
    if not zoom_path.exists():
        raise FileNotFoundError(f"Zoom-level folder not found: {zoom_path}")

    image_id = image_id or os.path.basename(tile_folder.rstrip('/\\'))
    pred_tiles_dir = Path(OUTPUTS_DIR) / "oilspill" / image_id / "pred_tiles" / zoom_level
    shutil.rmtree(pred_tiles_dir, ignore_errors=True)
    pred_tiles_dir.mkdir(parents=True, exist_ok=True)

    for tile_name in os.listdir(zoom_path):
        if tile_name.lower().endswith((".jpeg", ".jpg", ".png", ".tiff", ".tif", ".bmp")):
            tile_path = zoom_path / tile_name
            out_mask = pred_tiles_dir / f"{Path(tile_name).stem}_mask.png"
            predict_single_image(str(tile_path), str(out_mask), model, transform, device)

    xml_path = str(xml_candidate) if (xml_candidate := Path(tile_folder) / "vips-properties.xml").exists() else None
    stitched_dir = Path(OUTPUTS_DIR) / "oilspill" / image_id
    stitched_dir.mkdir(parents=True, exist_ok=True)
    stitched_path = stitched_dir / f"{image_id}_oilspill_mask.png"
    stitch_predicted_folder(str(pred_tiles_dir), str(stitched_path), xml_path=xml_path)
    generate_dzi(input_path=str(stitched_path), output_prefix=str(stitched_dir), tile_size=256)

    return {
        "stitched_mask": str(stitched_path),
        "dzi_path": str(stitched_dir),
        "dzi_folder": str(stitched_dir / f"{Path(stitched_dir).stem}_files")
    }
```

This code detects oil spills in large satellite/SAR images by processing tiled images through a Vision Transformer model trained to classify oil slicks versus background, generating segmentation masks for each tile. After predicting all tiles at a specified zoom level, it stitches the individual mask predictions back into a full-resolution output image using the original image dimensions from DZI metadata.

API

Code - index.js

```
require('dotenv').config();
const express = require('express');
const cors = require('cors');
const imageRoutes = require('./routes/imageRoutes');
const dziRoutes = require('./routes/dziRoutes');
const detectionRoutes = require('./routes/detectionRoutes');
const path = require('path');


const app = express();
const PORT = process.env.PORT || 3000;
app.use(cors());

// Middleware to parse JSON bodies
app.use(express.json());
app.use(express.urlencoded({ extended: true }));


// Serve oil spill DZI tiles
app.use('/tiles/oilspill', express.static(path.join(__dirname, '../shared/tiles/oilspill')));

// Serve ship detection DZI tiles
app.use('/tiles/ship', express.static(path.join(__dirname, '../shared/tiles/ship')));
app.use('/outputs/oilspill',express.static(path.join(__dirname,'../shared/outputs/oilspill')))

app.use('/api/auth', authRoutes);

// uploading images and gettiing list of images
app.use('/api/images', imageRoutes);
// DZI generation routes for uploaded images
app.use('/api/dzi', dziRoutes);
// Detection routes for ship and oil spill
app.use('/api/detect', detectionRoutes);

app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

This Express.js server sets up a maritime surveillance API with CORS-enabled endpoints for image uploads, DZI tile generation, and ship/oil spill detection, while serving static files for DZI tiles and detection outputs from shared directories. It configures middleware for JSON parsing and URL-encoded data, registers three main route modules (/api/images, /api/dzi, /api/detect), and listens on port 3000 to coordinate the frontend-backend communication for the SAR image processing pipeline.

API

Code - auth.js

```
const express = require('express');
const twilio = require('twilio');
const router = express.Router();

const client = twilio(process.env.TWILIO_ACCOUNT_SID, process.env.TWILIO_AUTH_TOKEN);
const SERVICE_SID = process.env.TWILIO_VERIFY_SID;

// Send OTP
router.post('/send-otp', async (req, res) => {
  try {
    await client.verify.v2.services(SERVICE_SID).verifications.create({
      to: req.body.phoneNumber,
      channel: 'sms'
    });
    res.json({ success: true });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

// Verify OTP
router.post('/verify-otp', async (req, res) => {
  try {
    const check = await client.verify.v2.services(SERVICE_SID).verificationChecks.create({
      to: req.body.phoneNumber,
      code: req.body.code
    });
    res.json({ verified: check.status === 'approved' });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

module.exports = router;
```

This Express.js server sets up a maritime surveillance API with CORS-enabled endpoints for image uploads, DZI tile generation, and ship/oil spill detection, while serving static files for DZI tiles and detection outputs from shared directories. It configures middleware for JSON parsing and URL-encoded data, registers three main route modules (/api/images, /api/dzi, /api/detect), and listens on port 3000 to coordinate the frontend-backend communication for the SAR image processing pipeline.

Code - dziController.js

```
const path = require('path');
const fs = require('fs');
const axios = require('axios');

const uploadsDir = path.join(__dirname, '../../shared/uploads');
const dziBaseDir = path.join(__dirname, '../../shared/tiles');

const allowedTypes = ['ship', 'oilspill'];

// Helper: Check if DZI file exists
function dziExists(type, imageId) {
  const dziFile = path.join(dziBaseDir, type, imageId, `${imageId}.dzi`);
  return fs.existsSync(dziFile);
}

exports.generateDZI = async (req, res) => {
  const { type, imageId } = req.params;

  if (!allowedTypes.includes(type)) {
    return res.status(400).json({ error: 'Invalid type. Must be "ship" or "oilspill".' });
  }

  try {
    // Optional: check if already exists
    if (dziExists(type, imageId)) {
      return res.status(200).json({
        message: 'DZI already exists',
        dziUrl: `/tiles/${type}/${imageId}/${imageId}.dzi`
      });
    }

    //  Call Python backend with type and imageId
    const response = await axios.post(`http://localhost:8000/api/generate_dzi/${type}/${imageId}`);

    return res.status(response.status).json(response.data);

  } catch (err) {
    console.error('DZI generation error:', err.message);
    return res.status(500).json({ error: 'Failed to generate DZI via Python service' });
  }
};
```

This Express route handler checks if a DZI file already exists for a given image type (ship or oilspill) and image ID, returning the existing path if found. If the DZI doesn't exist, it forwards the request to a Python FastAPI backend service running on port 8000 to generate the DZI tiles and returns the result.

DOCKERIZATION

Dockerfile

```
FROM python:3.11-slim

ENV PYTHONUNBUFFERED=1 PYTHONDONTWRITEBYTECODE=1
WORKDIR /app


RUN apt-get update \
   && apt-get install -y --no-install-recommends \
      git \
      build-essential \
      libsndfile1 \
   && apt-get clean \
   && rm -rf /var/lib/apt/lists/*


COPY requirements.txt .


RUN pip install --no-cache-dir --upgrade pip \
 && pip install --no-cache-dir -r requirements.txt


COPY . .


EXPOSE 8000


CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "8000"]
```

This Dockerfile containerizes our Python FastAPI backend that powers the maritime surveillance system's DL inference capabilities. It packages my entire detection pipeline—including the Deformable DETR model for ship detection and Vision Transformer - TransUNet for oil spill classification—along with PyTorch, transformers, and pyvips libraries needed to process those massive 160MB satellite images.

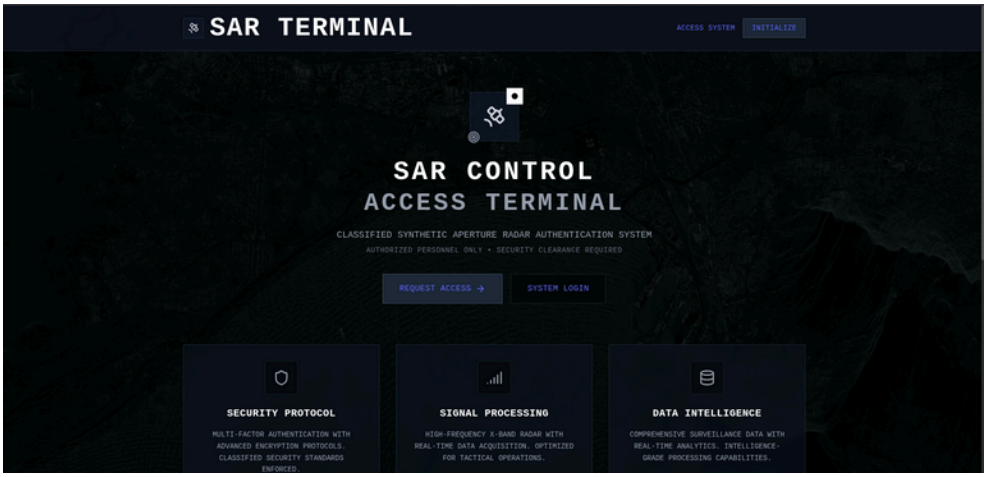
```
# Build the image
docker build -t nsvoriginals/maritime-surveillance-api:latest .

# Push latest version
docker push nsvoriginals/maritime-surveillance-api:latest
```

Link → https://hub.docker.com/repository/docker/nsvoriginals/sarv1

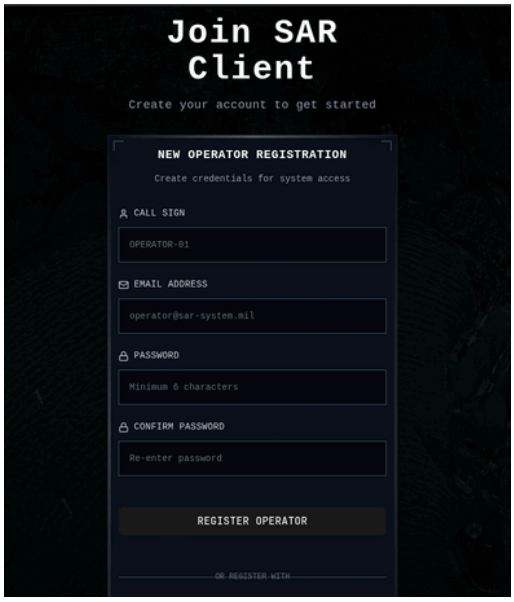
SCREENSHOTS

Frontend

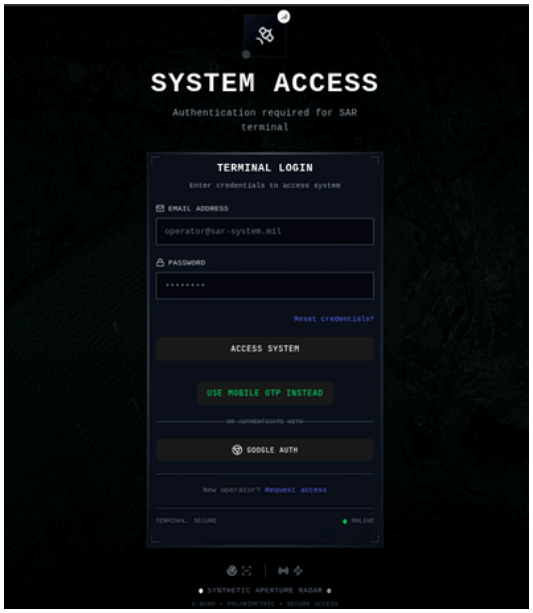
Landing Page



Register Page



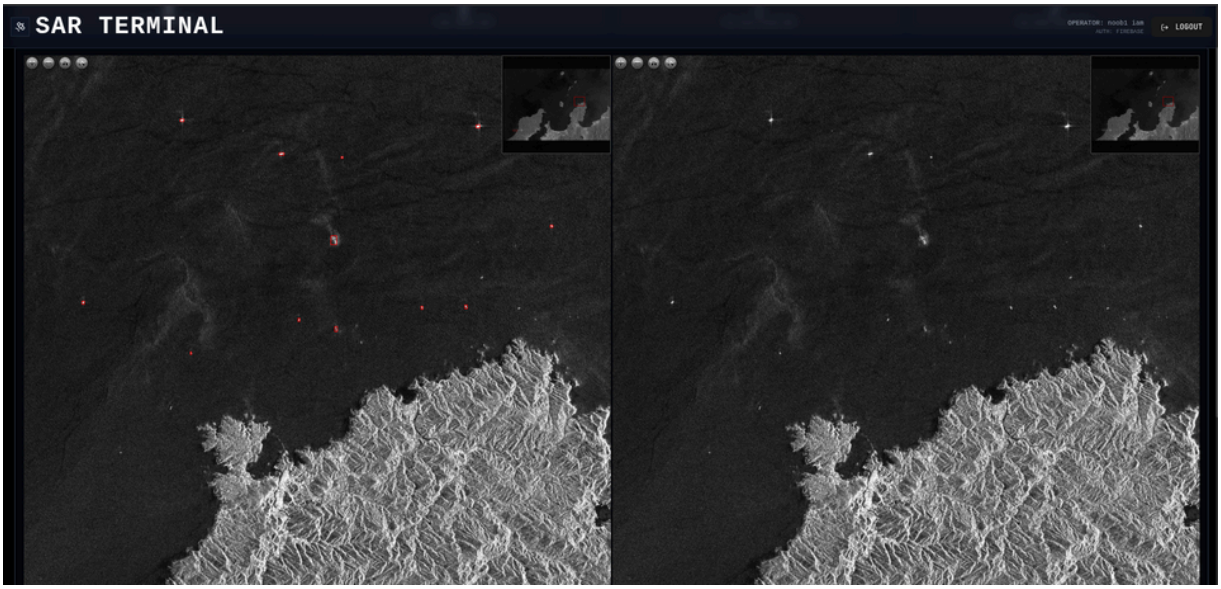
Login Page



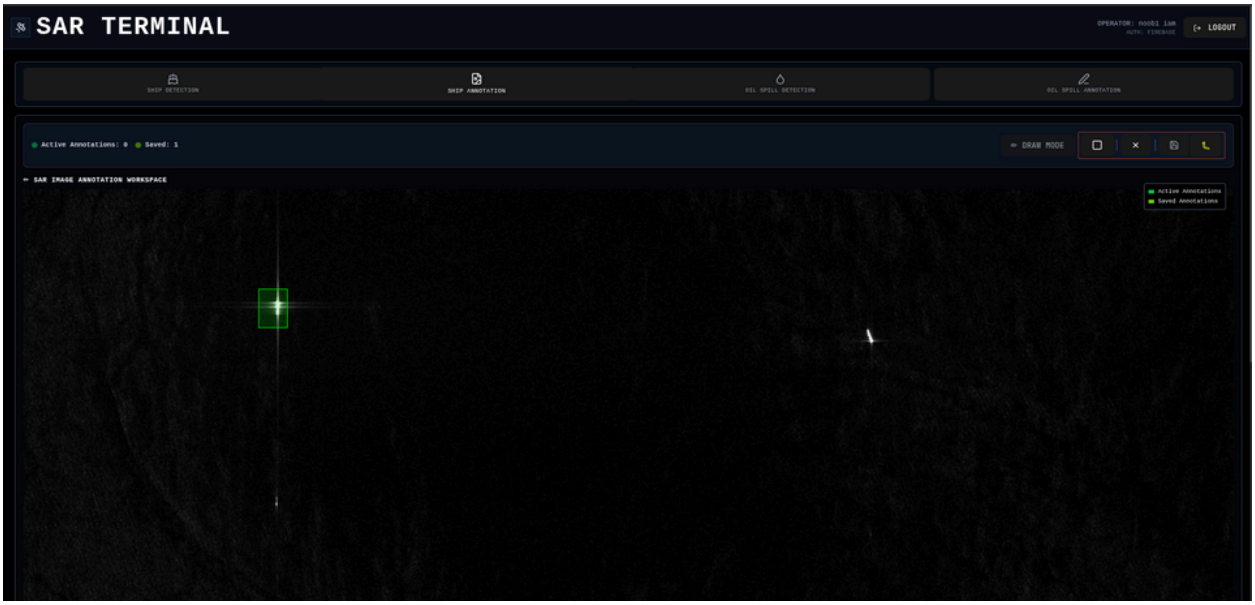
SCREENSHOTS

Frontend

SAR Ship Detection Viewer



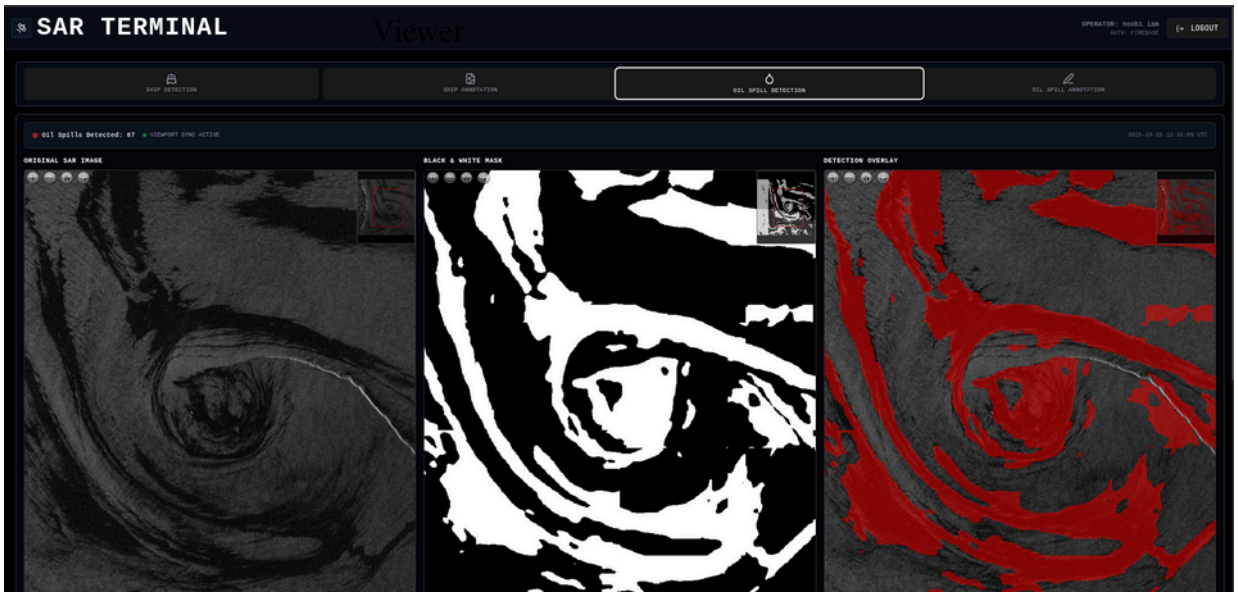
SAR Ship Annotation Viewer



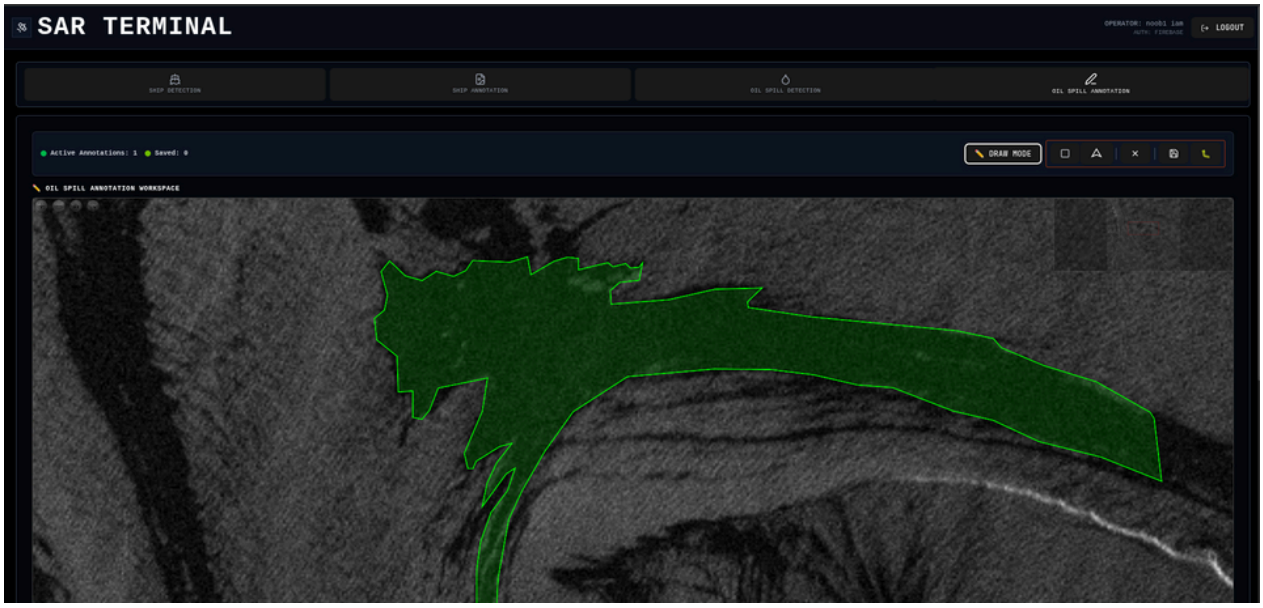
SCREENSHOTS

Frontend

SAR Oil SPill Detection Annotation Viewer

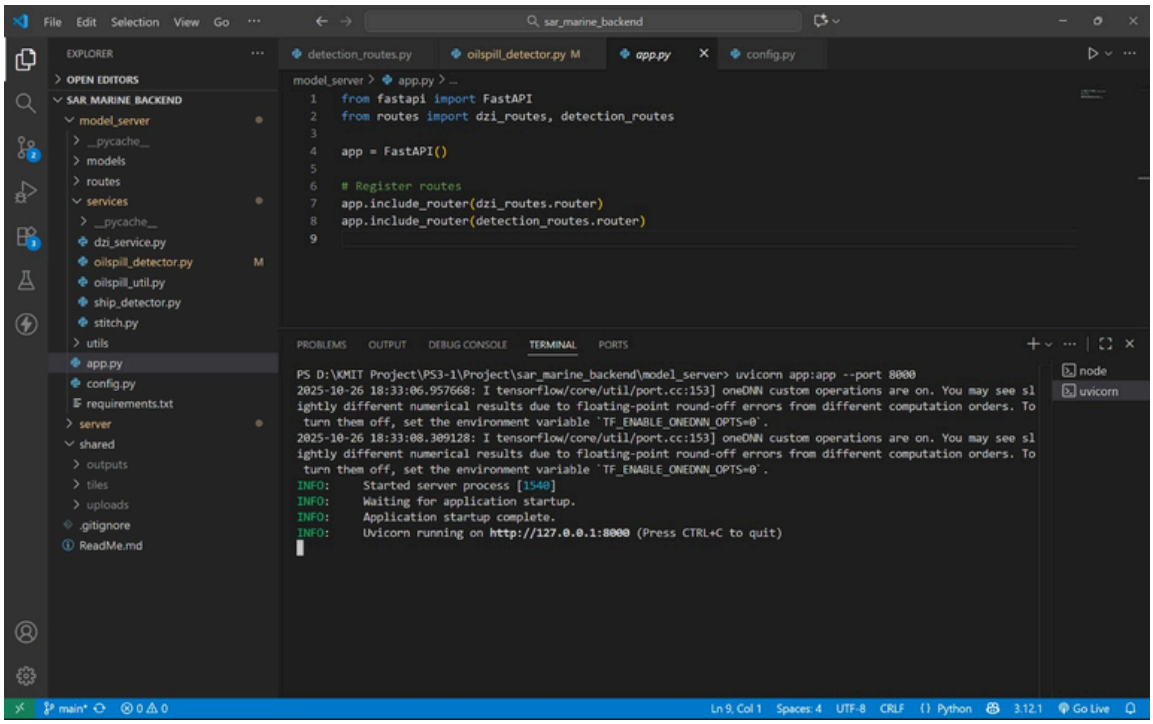


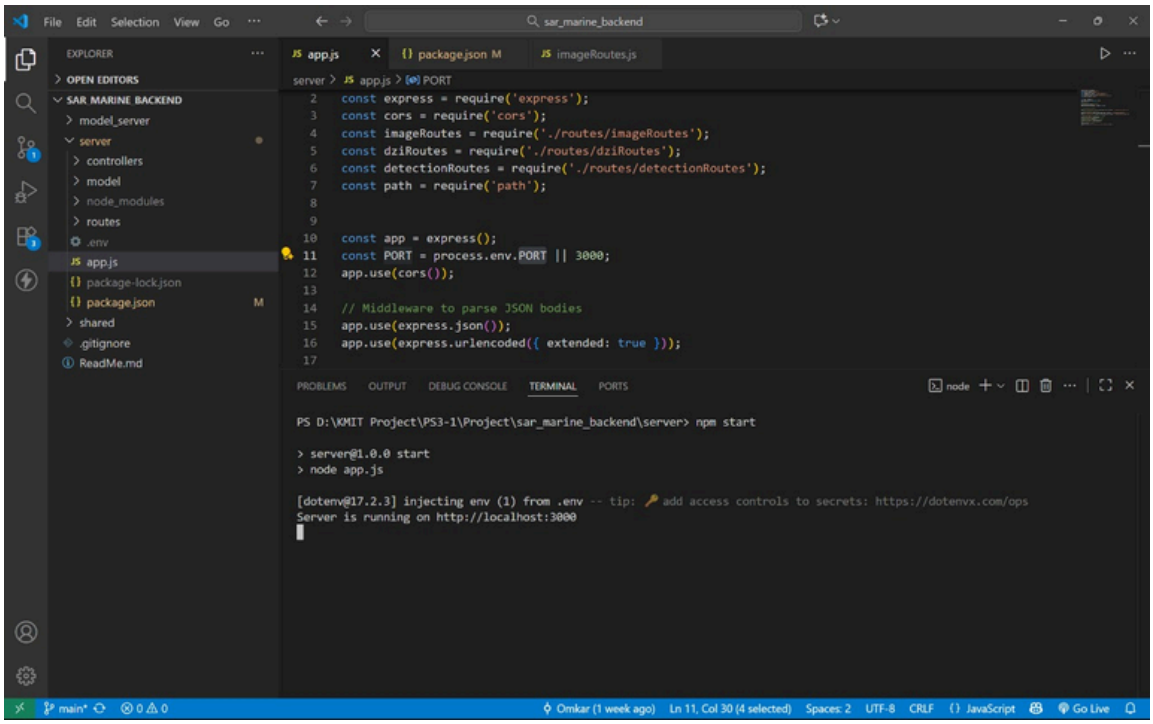
SAR Oil Spill Annotation Viewer



SCREENSHOTS

Backend

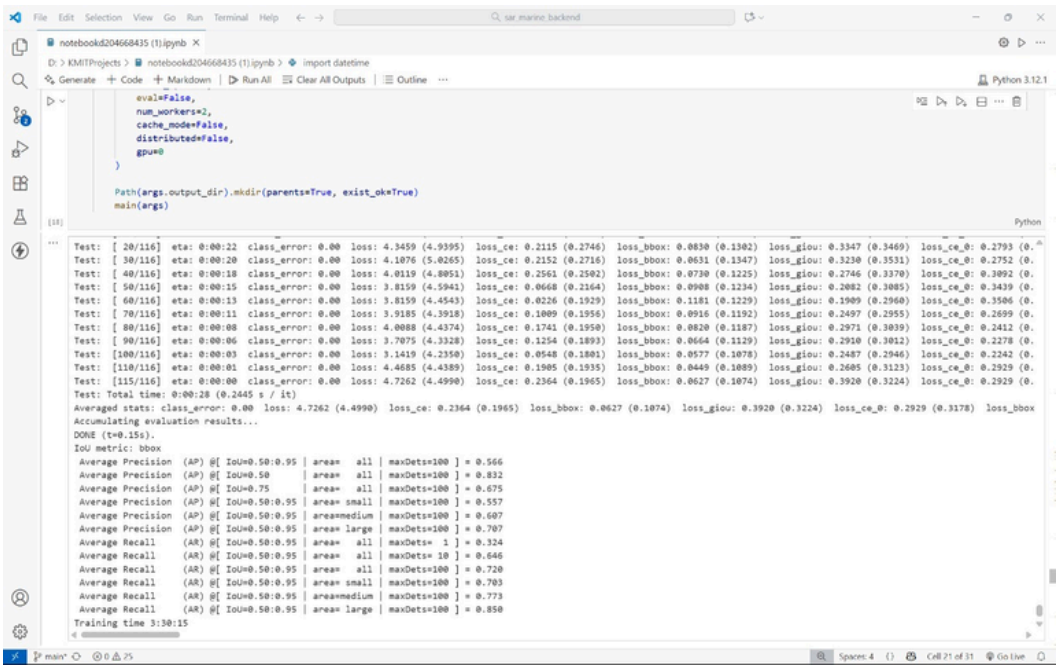


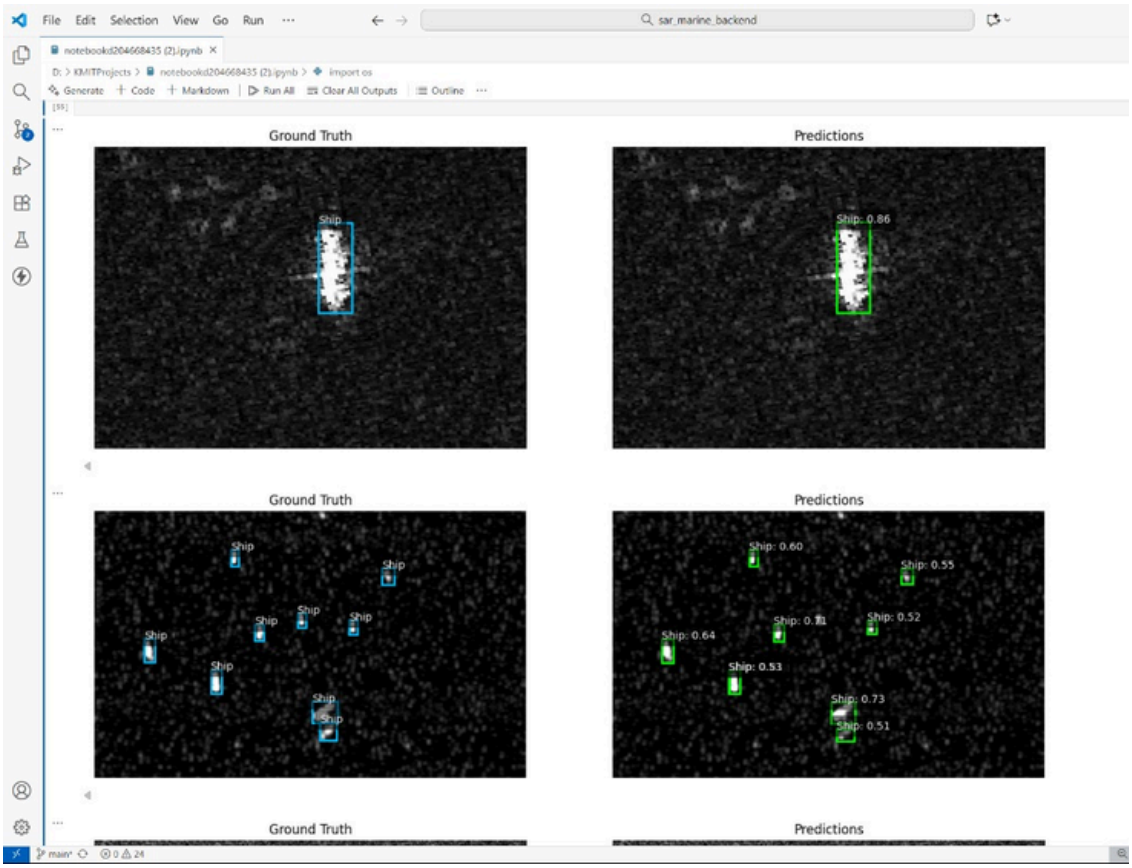


SCREENSHOTS

Model

Deformable DETR



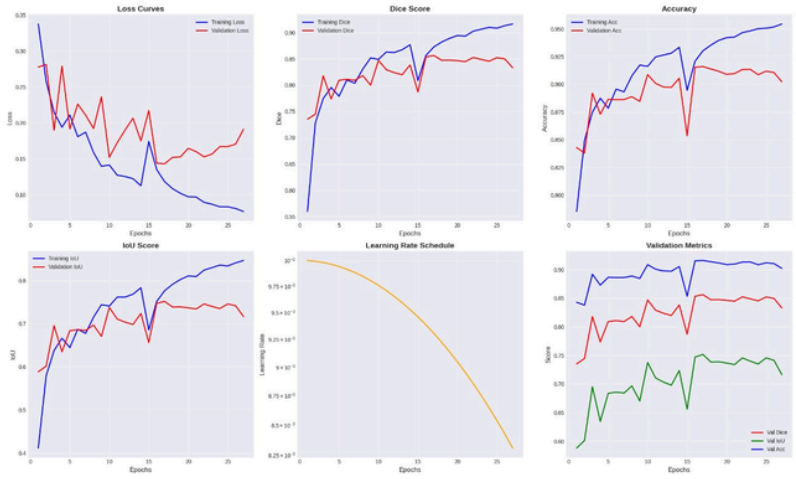


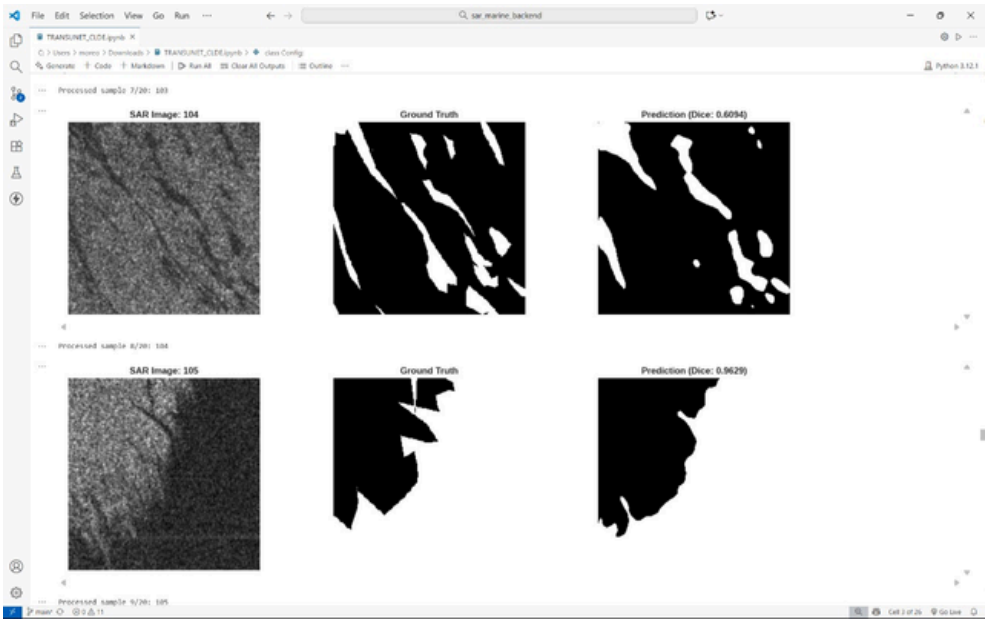
SCREENSHOTS

Model

TransUNet

```
============================================================
Epoch 27/100
============================================================
Training Epoch 27: 100%|██████████| 60/60 [00:29<00:00,  2.01it/s, loss=0.0734, dice=0.9040]
Validation Epoch 27: 100%|██████████| 15/15 [00:02<00:00,  5.64it/s, loss=0.1732, dice=0.8673]

Results:
  Train -> Loss: 0.0763, Dice: 0.9162, IoU: 0.8467, Acc: 0.9544
  Val   -> Loss: 0.1909, Dice: 0.8330, IoU: 0.7164, Acc: 0.9022
  Learning Rate: 0.008307

  No improvement. Patience: 10/10

X Early stopping triggered at epoch 27

============================================================
Training completed!
  Best validation Dice: 0.8564
  Final model saved to: /content/model/TU_OilSpill224/final_model.pth
============================================================
```





REFERENCES

1.Deformable DETR for Ship Detection
Link: https://www.mdpi.com/2077-1312/11/8/1552
Deformable DETR is an advanced deep learning model using transformer architecture with multi-scale deformable attention mechanisms for detecting ships in SAR imagery. The model effectively handles ships with varying sizes and orientations in complex maritime environments by focusing on relevant regions of large-scale satellite images.

2.OpenSeadragon Documentation
Link: https://openseadragon.github.io/docs/
OpenSeadragon is an open-source JavaScript library for viewing high-resolution, zoomable images in web browsers using Deep Zoom Images (DZI) and tile-based formats. It enables users to pan and zoom through large SAR satellite images with detection overlays without loading entire high-resolution files at once.

3.FastAPI Documentation
Link: https://fastapi.tiangolo.com
FastAPI is a modern, high-performance Python web framework for building APIs with automatic validation and interactive documentation. For this project, it powers the backend API that processes ship detection requests, manages MongoDB database operations, and serves results with auto-generated docs at /docs and /redoc endpoints.

4.React Documentation
Link: https://react.dev
React is a JavaScript library for building interactive user interfaces through reusable components. It powers the frontend where users upload SAR images, view tiled displays with OpenSeadragon, and interact with detection results through modular features like authentication, image uploads, and visualization panels.

5.Marine Oil Spill Detection from SAR Images
Link: https://www.mdpi.com/2077-1312/11/8/1552
This research paper by Dong et al. (2023) presents a deep learning approach using Deformable DETR for detecting marine oil spills from low-quality SAR remote sensing images. The method addresses challenges of high noise, low contrast, and irregular oil spill boundaries in SAR imagery, making it applicable to environmental monitoring and maritime pollution detection.

SUMMARY

This project is a web-based ship detection system that processes large-scale Synthetic Aperture Radar (SAR) satellite images to automatically identify and locate maritime vessels. The system uses Deformable DETR, a transformer-based deep learning model trained on the SARscope dataset (combining HRSID and OPEN-SSDD sources), to detect ships in SAR imagery with high accuracy. Large satellite images averaging 160MB each are divided into tiles, processed through the detection model, and aggregated into a unified coordinate system with geographic locations.

The technology stack includes a React frontend with OpenSeadragon for interactive visualization of high-resolution images, FastAPI backend for processing and API services, MongoDB for data storage, and Clerk for user authentication. PyVIPS handles conversion of large TIFF images to Deep Zoom Image (DZI) format, enabling seamless browser-based viewing of massive satellite imagery with ship detection overlays. The complete pipeline produces a single JSON file containing all ship detections with bounding boxes, confidence scores, geographic coordinates, and estimated physical properties.

This system serves maritime surveillance applications including vessel traffic monitoring, illegal fishing detection, search and rescue operations, and port management. The automated detection capabilities significantly reduce manual analysis time while providing comprehensive coverage of large maritime regions with scalable processing that handles satellite images of any size through efficient tiling and batching techniques.