# Project Documentation

## Overview

This project implements a **microservice-based vehicle detection system** using **YOLOv11** for object detection.
It combines a **FastAPI backend** (for API inference) and a **Gradio interface** (for UI interaction).

The solution meets the assessment requirements for:

1. An **AI backend service** - performing detection using a lightweight open-source model.
2. A **UI service** - allowing users to upload images and visualize detection results.

The entire system is **containerized using Docker** for easy setup and reproducibility.

**Core Technologies**

| Component | Technology | Purpose |
|---|---|---|
| **Model** | YOLOv11 (Ultralytics) | Vehicle detection (fast, lightweight, accurate) |
| **Frameworks** | **FastAPI** & **Gradio** | REST API + interactive web UI |
| **Containerization** | **Docker** | Environment isolation and portability |
| **Language** | **Python 3.10** | Core development language |

## Dataset

- The model is trained on the **Vehicle Detection Dataset from Roboflow Universe**.
- This dataset provides labeled images for **bus, car, microbus, motorbike, pickup-van, and truck** vehicles in various conditions.
- It was preprocessed in YOLO format using **Roboflow** utilities for compatibility with YOLOv11 training.

# Model Training

Training was performed using **YOLOv11** (Ultralytics framework).

The model was trained on the above dataset using default augmentation and optimization settings.

The **best-performing checkpoint (best.pt)** was saved under:

runs/detect/vehicle_detection/weights/best.pt

This trained model is bundled with the repository for immediate inference.

# Application Architecture

The application consists of two main parts:

## 1AI Backend - *FastAPI*

- Provides a /detect/ API endpoint.
- Accepts an uploaded image and performs inference using YOLOv11.
- Returns detection results (class names, bounding boxes, and confidence scores) as structured JSON.

## 2UI Service — *Gradio*

- Offers a simple drag-and-drop web interface for testing the model.
- Displays the original image with bounding boxes and detection labels.

Both services run together from main.py on port **8000**.

# Dockerization

- The project is fully containerized for easy deployment.
- You can build and run the container with:

```
docker build -t vehicle-detector .

docker run -p 8000:8000 vehicle-detector
```

This image includes:

- Python environment
- Required dependencies
- The trained YOLOv11 weights
- FastAPI + Gradio runtime

No additional setup is required - it runs out of the box.

# Inference Workflow

1. User uploads an image via Gradio or API.
2. The YOLOv11 model loads the weights from

   runs/detect/vehicle_detection/weights/best.pt.

3. The model performs inference and identifies vehicles.
4. Detected results (bounding boxes and confidence scores) are returned as JSON and saved as annotated images in inference_outputs/.

# Example API Usage

**curl -X POST "http://localhost:8000/detect/" -F "file=@test_image/car.jpg"**

**Response:**


```
[
  {
    "class_name": "car",
    "confidence": 0.6701048016548157,
    "box": {
      "x_min": 3268.811279296875,
      "y_min": 1646.0157470703125,
      "x_max": 4432.54248046875,
      "y_max": 2554.489990234375
    }
  }
]
```


## Deliverables

1. Fully containerized inference service
2. YOLOv11 model trained on Roboflow dataset
3. Web interface (Gradio) + REST API (FastAPI)
4. Pretrained model weights included
5. Output images and structured JSON responses
6. Documentation describing training and deployment

# References


- Dataset: Vehicle Detection – Roboflow Universe
- Model Framework: Ultralytics YOLOv11
- Web API: FastAPI Documentation
- Web Interface: Gradio


# Summary

This project demonstrates a complete, reproducible **vehicle detection microservice** using **YOLOv11**.

It integrates a modern inference API, a lightweight web UI, and Dockerized deployment — fulfilling all objectives of the AIMonk technical assessment