Deployed Model Report

Real-Time Object Detection System for Autonomous Vehicles

1. Introduction

In this project, we developed and deployed a real-time object detection system designed specifically for autonomous vehicles. Our goal was to create a reliable and efficient system that can identify key objects on the road, like cars, traffic lights, and pedestrians, using a deep learning model called YOLOv8. To make the system easy to use and deploy across different environments, we packaged everything inside a Docker container.

2. System Architecture

The system is built as a simple web application where users can upload images and get instant detection results. Here's a quick overview of the main parts:

- **Backend:** A Flask server handles all requests, processes images, and runs the detection model.
- **Model:** We initially trained the model and had it saved as `best.pt`—the default PyTorch weights file. To improve inference speed and make the system more flexible, we converted this model to the **ONNX** format. This format is widely supported by various inference engines and helps the model run faster and smoother on different hardware.
- **Frontend:** The user-friendly interface is built with HTML and Bootstrap, making it easy to upload images/Videos and see detection results.
- **Containerization:** We used Docker to package everything, which means the app runs consistently no matter where it's deployed.
- **Dependencies:** OpenCV helps with image processing, Ultralytics YOLO handles the core detection, and standard Python libraries manage files and errors.

3. Docker Configuration

Our app runs on a lightweight Python Docker image (`python:3.10-slim`). We added a few system libraries like `ffmpeg` and OpenCV dependencies to support video and image processing. We also set up a dedicated folder for image uploads that the app can safely access.

4. How the Application Works

- The user uploads an image/video through the web interface.
- The app saves this image/video with a unique filename to avoid any conflicts.
- The ONNX model runs inference on the image/video to detect objects.
- Detected objects are drawn directly on the image/video, which is saved on the server.
- The annotated image, video frames, and a list of detected objects are shown back to the user.

5. Real-World Testing Summary

- **Setup:** We tested on a standard CPU inference.
- **Performance:** On average, each image took about 1 second to process, with good accuracy on vehicles, pedestrians, and traffic lights.
- **Stability:** The system handled multiple uploads without memory issues, showing reliable performance for continuous use.

6. Challenges & How We Tackled Them

Challenge	What Happened	What We Plan to Do
Low light/night images/videos	Detection accuracy dropped	Add image/video frame enhancement pre-processing
Blurry or motion shots	Some objects were misclassified	Use smoothing techniques in video
Occluded objects	Partial or missed detections	Explore combining detection with segmentation models
Unusual/unseen objects	No detection	Expand training data with diverse samples

7. What's Next?

- Adding **real-time video support** for continuous object detection from live camera feeds
- Optimizing deployment to run on **GPUs and edge devices** like NVIDIA Jetson or Raspberry Pi for faster, on-the-go inference.
- Building **security features** like user authentication and upload logs for better control and auditing.

8. Conclusion

Starting from the original PyTorch model (`best.pt`), we successfully converted it to ONNX for better performance and deployed it in a user-friendly, Dockerized web app. The system performed well in real-world driving scenarios despite running only on the CPU, accurately detecting key road objects with minimal delay. This project lays a solid foundation for future development into fully real-time, edge-deployable autonomous vehicle perception systems.