Smart Parking & Surveillance AI Model Challenge

Model Architecture & System Logic Report

1. Overview

This report outlines the architecture, training process, and deployment logic of the AI system developed for the Smart Parking & Surveillance AI Model Challenge. The goal was to create a real-time vehicle detection system using video input, capable of identifying vehicles and potentially detecting parking slot status, with deployment support on mobile devices.

2. Model Selection & Justification

We utilized the YOLOv8s (You Only Look Once – small variant) model developed by Ultralytics, known for its speed and performance on real-time object detection tasks. It provides a good trade-off between inference speed and accuracy, making it suitable for edge deployment.

- Base Architecture: YOLOv8s (Ultralytics)

- Pretrained Weights: COCO dataset

- Fine-tuned: Yes, on a mix of CARPK dataset and a custom top-down vehicle dataset

- Reason for Selection: Lightweight, accurate, supports ONNX and TensorFlow Lite conversion for mobile deployment

3. Dataset & Annotation

- Dataset Used:

- CARPK Dataset (1567 top-down vehicle images)

- A small custom dataset with additional top-view vehicle images

- Annotation Tool: Roboflow

- Classes: Single class - `0` (used instead of label names like "car" or "vehicle" for simplicity)

- Image Input Size: 640x640

Note: For parking slot detection (in desktop-based implementation), polygon coordinates were defined using Ultralytics' annotation UI and exported as a JSON file

4. Training Pipeline

- Training Platform: Google Colab (with GPU support)

- Epochs: 50

- Batch Size: 16

- Input Image Size: 640x640

- Data Augmentation: Enabled (flipping, brightness/contrast adjustment, scaling)

- Training Output: Model weights saved as `best.pt` based on best validation metrics

5. Inference & Parking Slot Logic (Desktop Version)

The trained YOLOv8s model was used to analyze each video frame and estimate vehicle occupancy in parking slots.

Steps:

1. Frame Inference: YOLOv8 processes each frame to detect vehicles and returns bounding boxes.

2. Centroid Calculation: The center point of each detected box is calculated.

3. Slot Check (via JSON):

- Predefined parking slot polygons are loaded from a JSON file.

- OpenCV's **cv2.pointPolygonTest()** checks if centroids fall within the polygon areas.

- Slots with overlapping centroids are marked as "Occupied".

Note: This logic was implemented in the PC version only. It was not integrated into the Android app due to the need for re-calibrating slot coordinates for different environments.

6. Deployment & Optimization

- Deployment Target: Android smartphone

- Tools & Libraries Used:

- Ultralytics YOLOv8 (for training and model export)

- ONNX (for intermediate model format)

- TensorFlow Lite Converter (for mobile deployment)

- Android Studio (for app development and testing)

Steps Followed:

1. Converted YOLOv8 weights (best.pt) → ONNX → TFLite (best_float32.tflite)

2. Integrated .tflite model into an open-source Android app repository

3. Updated paths and logic in the Kotlin-based app to run real-time inference using the phone camera

7. Visual Feedback & Deployment Notes

- On Android App (Mobile Deployment):

- Bounding boxes are drawn around detected vehicles

- Each box shows class label `0`

- No polygon overlays, live slot counts, or color-coded status (those were only available in the PC version)

8. Challenges & Mitigations

Challenge: Slot detection inaccuracies due to perspective **Solution:** Manual calibration and tuning of polygon coordinates
Challenge: Maintaining real-time performance **Solution:** Chose YOLOv8s for lightweight size, converted to TFLite
Challenge: Integration into mobile UI **Solution:** Adapted open-source Android app and customized it for our use case

9. Summary & Outcomes

- Successfully trained and deployed a YOLOv8s-based vehicle detection model

- Mobile app integration completed using .tflite model

- Live slot counting and color overlays are only for PC not included in mobile app because of re-calibrating slot coordinates for different environments.

10. One-Slide Summary

Component	Details
Model	YOLOv8s, fine-tuned on CARPK and my own customized Dataset
Detection Target	Vehicles (single class)
Slot Detection	Based on JSON-defined polygons
Deployment	TensorFlow Lite on Android phone
UI Features	Real-time overlay, slot labels
Tools Used	Ultralytics, Roboflow, OpenCV, Android Studio