Major Project - I

ParkLot: Centralized Car Parking Solution Using Fine-tuned YOLO Models

A Project

*S*ubmitted in partial fulfillment of the

*r*equirements for the award of the degree

*o*f

BACHELOR OF TECHNOLOGY

by

Anirudh Sharma - 22DCS002

Pushpdeep - 22DCS018

Under the guidance

of

Dr. Naveen Chauhan

Department of Computer Science & Engineering

NATIONAL INSTITUTE OF TECHNOLOGY HAMIRPUR

Hamirpur (Himachal Pradesh) – 177 005

December, 2025

ParkLot: Centralized Car Parking Solution Using Fine-tuned YOLO Models

राष्ट्रीय प्रौद्योगिकी संस्थान हमीरपुर
हमीरपुर (हि॰प्र॰) – 177 005 (भारत)
[भारत सरकार शिक्षा मंत्रालय के तहत एक राष्ट्रीय महत्व का संस्थान]
NATIONAL INSTITUTE OF TECHNOLOGY HAMIRPUR
HAMIRPUR (H.P.) - 177 005 (INDIA)
[An Institute of National Importance under Ministry of Education (Shiksha Mantralaya)]

Candidate's Declaration

We hereby declare that our work submitted in the partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** in the Department of Computer Science and Engineering of the National Institute of Technology Hamirpur, titled as **"ParkLot: Centralized Car Parking Solution Using Fine-tuned YOLO Models"** is an record of our work carried out during the **VII Semester** from July 2025 to November 2025 under the guidance of **Dr. Naveen Chauhan**, Associate Professor, DoCSE, NIT Hamirpur.

The matter presented in this report has not been submitted by us for the award of any other degree of this or any other Institute/University.

Anirudh Sharma - 22DCS002 **Pushpdeep - 22DCS018**

This is to certify that the above statement made by the candidates is true to the best of my knowledge and belief.

Dr. Naveen Chauhan
Associate Professor

Date: December 11, 2025

Convener, DBPC **Head, DoCSE**

Acknowledgement

We would like to express our sincere gratitude to our project supervisor **Dr. Naveen Chauhan**, Associate Professor, Department of Computer Science and Engineering at the National Institute of Technology Hamirpur, for his valuable guidance, constant encouragement, and constructive criticism throughout the duration of this project. His expertise and insights were instrumental in the successful completion of this work.

We are thankful to the **Department of Computer Science and Engineering, NIT Hamirpur**, for providing us with the necessary infrastructure and resources, including hardware and laboratory facilities required for testing our edge devices and model training. The access to campus parking facilities for our proof-of-concept implementation was invaluable.

We express our special thanks to our parents for their encouragement and constant moral support, and to our friends and colleagues for being supportive and providing constructive feedback during the course of this project. We also acknowledge the open-source community for the datasets and tools that supported our research.

Anirudh Sharma - 22DCS002

Pushpdeep - 22DCS018

Abstract

Efficient parking management is a critical component of smart cities, as studies indicate that approximately 30% of urban traffic is caused by drivers searching for parking, leading to increased carbon footprints and fuel consumption. While CCTV cameras are extensively deployed in parking facilities, they remain underutilized for intelligent traffic management, serving primarily passive monitoring and security purposes. This presents a significant opportunity to leverage existing infrastructure for smart parking solutions.

This project, **ParkLot**, proposes a cost-efficient, centralized parking solution that leverages existing CCTV networks to detect parking occupancy without requiring expensive new sensors. The proposed system utilizes a hybrid architecture combining edge computing (Raspberry Pi) and cloud services (Google Cloud Platform). We fine-tuned multiple object detection models from the YOLO family (YOLOv8n, YOLOv8m, YOLOv8l) on aerial datasets such as CNRPark-Ext and PKLot, as well as a custom dataset from the NIT Hamirpur campus.

Our results demonstrate that the YOLOv8n model running on edge devices achieves a precision of 0.99 and an inference time of 7.7 ms, offering a viable real-time solution. The larger models (YOLOv8m and YOLOv8l) trained on extensive datasets achieved F1-scores of 0.927 and 0.942 respectively, demonstrating robust performance across diverse conditions. The system is integrated with a user-friendly mobile application and a web dashboard to guide drivers to available spots in real-time.

This project's contributions are twofold:

1. Development of a cost-effective parking detection system using fine-tuned YOLO models that achieves high precision (0.99) with minimal inference time (7.7ms) on edge devices, enabling real-time occupancy detection.

2. Creation of a hybrid edge-cloud architecture that leverages existing CCTV infrastructure, eliminating the need for expensive sensor installations while maintaining

scalability and reliability across diverse parking environments.

Keywords

Smart Parking, YOLO, Computer Vision, Vehicle Detection, AI,
Urban Mobility, IoT

Table of Contents

List of Figures

List of Tables

Introduction

1.1 Overview

As urban populations grow, efficient mobility becomes a paramount challenge. Parking management is a crucial aspect of this, yet it remains largely inefficient in many cities. **ParkLot** aims to solve this by introducing a smart detection system that eliminates the need for drivers to circle endlessly looking for spots.

1.2 Motivation

The motivation for this project stems from three key factors:

- **Traffic Reduction:** Approximately 30% of urban traffic is attributed to drivers searching for parking. Smart detection can significantly alleviate this congestion.
- **Environmental Impact:** Reducing the time spent searching for parking directly lowers fuel consumption and the associated carbon footprint.
- **Cost Efficiency:** Traditional smart parking relies on expensive individual sensors for each spot. Our approach unlocks intelligence from existing CCTV infrastructure, removing the need for new sensor hardware.

1.3 Problem Statement

The fundamental goal of this project is to achieve highly effective parking space detection using computer vision. The specific challenges addressed include:

- **Aerial View Occlusions:** The system must process aerial datasets which provide a wide view but are susceptible to occlusions where larger vehicles may block the view of empty spots.

- **Model Selection:** Determining which version of the YOLO family (v5, v7, v8, v9) offers the optimal balance between accuracy and real-time performance for this specific domain.

1.4 Objectives

The primary objectives of the ParkLot project are:

1. Develop a vision-based vehicle detection system using YOLO framework capable of accurately identifying vehicle presence and determining parking slot occupancy in real-time.

2. Create an intelligent parking management platform that processes CCTV feeds to provide live parking availability information across designated parking zones.

3. Design and implement a mobile application interface that enables users to discover nearby parking facilities, view real-time availability, and navigate to vacant spots.

4. Deploy a proof-of-concept implementation at NIT Hamirpur campus to validate system performance in real-world conditions including outdoor environments and irregular parking layouts.

5. Optimize the system for edge computing deployment (Raspberry Pi) to minimize latency and ensure functionality with intermittent internet connectivity.

6. Demonstrate measurable improvements in parking detection accuracy and inference speed suitable for real-time applications.

1.5 Report Organization

The remainder of this report is organized as follows:

Chapter 2: Literature Review examines existing research in parking detection, YOLO architecture evolution, and comparison of different approaches.

Chapter 3: System Design and Architecture details the overall system design, hardware specifications, cloud infrastructure, and component interactions.

Chapter 4: Methodology describes dataset preparation, model training procedures, and fine-tuning strategies.

Chapter 5: Results and Analysis presents performance metrics, accuracy analysis, and comparative evaluation of different YOLO models.

Chapter 6: Conclusion and Future Work summarizes achievements, discusses system capabilities, and outlines future enhancements.

Literature Review

We reviewed existing solutions to identify gaps and select the best technologies for ParkLot. The literature reveals significant progress in parking detection using computer vision and deep learning approaches, though challenges remain in real-time performance and cost-effective deployment.

2.1 Deep Learning for Parking Detection

Amato et al. (2017) [1] pioneered the use of deep Convolutional Neural Networks (CNNs) for parking lot occupancy detection, introducing the mAlexNet architecture trained on the CNRPark-Ext and PKLot datasets. Their work demonstrated that deep learning could effectively classify parking space occupancy from aerial views. However, their approach focused on binary classification (occupied/vacant) rather than full object detection, limiting its ability to handle dynamic parking layouts.

2.2 YOLO-Based Approaches

Rafique et al. (2023) [2] advanced the field by implementing YOLOv5 for real-time parking management, achieving 99.5% accuracy. Their framework optimized the detection pipeline for real-time performance, but required substantial computational resources, making edge deployment challenging. The work highlighted the trade-off between accuracy and computational efficiency in production environments.

Ahad and Kidwai (2025) [3] integrated YOLOv4 with behavioral data analysis to reduce parking search time in congested urban environments. By combining occupancy detection with predictive allocation based on historical patterns, they demonstrated measurable improvements in user experience. However, their approach requires extensive behavioral data collection, which may raise privacy concerns and complicate initial deployment.

2.3 Comparative Analysis of YOLO Variants

Shreeram et al. (2025) [4] conducted a comprehensive comparative study of YOLO family models (YOLOv5, YOLOv7, YOLOv8, YOLOv9) for smart parking space detection using aerial images. Their findings indicated that YOLOv9 achieved the highest mean Average Precision (mAP), while YOLOv8 offered the best balance between accuracy and inference speed. Notably, they identified aerial occlusions as a persistent challenge when larger vehicles obscure adjacent parking spaces.

2.4 Edge Computing and Smart Parking

Recent research has emphasized the importance of edge computing for reducing latency and enabling offline operation in smart parking systems. Liu et al. (2024) demonstrated that lightweight models deployed on edge devices like Raspberry Pi can achieve near-real-time performance with careful optimization, though at the cost of some accuracy compared to cloud-based solutions.

2.5 Gap Analysis

While existing research has made significant strides, several gaps remain:

- **Cost-Effective Deployment:** Most solutions require either expensive sensors or high-end computing infrastructure, limiting widespread adoption.
- **Hybrid Architecture:** Few studies have explored optimal distribution of processing between edge devices and cloud infrastructure for parking management.
- **Real-World Validation:** Limited deployment in actual parking facilities with diverse conditions (weather, lighting, irregular layouts).
- **Custom Dataset Requirements:** Performance on standard datasets doesn't always translate to specific deployment environments.

ParkLot addresses these gaps by implementing a hybrid edge-cloud architecture using fine-tuned YOLO models optimized for both accuracy and real-time performance on resource-constrained devices, validated through deployment at NIT Hamirpur campus.

2.6 Comparison of Existing Approaches

Table 2.1: Comprehensive Comparison of Parking Detection Approaches

Ref & Year	**Title**	**Venue**	**Model**	**Findings**	**Analysis (Pros/Cons)**
Amato et al., 2017 [1]	Deep learning for decentralized parking lot occupancy detection	Expert Systems with Applications	mAlexNet (CNN)	CNRPark-EXT, PKLot	+ Lightweight (RPi) – Binary only
Rafique et al., 2023 [2]	Optimized real-time parking management framework using deep learning	Expert Systems with Applications	YOLOv5	99.5% acc, 45 FPS	+ High accuracy – High compute
Ahad and Kidwai, 2025 [3]	YOLO based approach for real-time parking detection...	Innovative Infrastructure Solutions	YOLOv4 + Behavioral	Reduces search time	+ Better recall – Needs behavior data
Shreeram et al., 2025 [4]	Smart Parking Space Availability Detection Using Aerial Images...	AINA-2025	YOLO Family (v5-v9)	YOLOv9 best mAP	+ Large-area coverage – Aerial occlusions
ParkLot (Ours)	**–**	**–**	**YOLOv8n**	**Custom NIT Hamirpur dataset**	**+ Cost-effective** **+ Real-world validation**

System Design and Architecture

3.1 Proposed Solution

We propose a full-stack implementation that integrates three key components to create a comprehensive parking management system:

1. **Edge Processing:** Using Raspberry Pi nodes to run lightweight YOLO models (YOLOv8n) for initial detection at the parking facility level.
2. **Cloud Architecture:** A centralized Google Cloud server (Debian 12 VM) to manage data aggregation, storage, and API services for multi-location support.
3. **User Interface:** A React.js frontend and mobile application that directs users to the nearest available spot based on real-time data.

This hybrid architecture balances computational efficiency with scalability, allowing the system to operate reliably even with intermittent connectivity while supporting centralized management for larger deployments.

3.2 System Architecture Overview

The ParkLot system follows a three-tier architecture:

3.2.1 Edge Tier

- **Camera Input:** USB webcam or CCTV feed capturing parking area
- **Local Processing:** Raspberry Pi running YOLOv8n for real-time detection
- **Occupancy Calculation:** Determines available spots based on detection results
- **Data Transmission:** Sends occupancy status to cloud at regular intervals

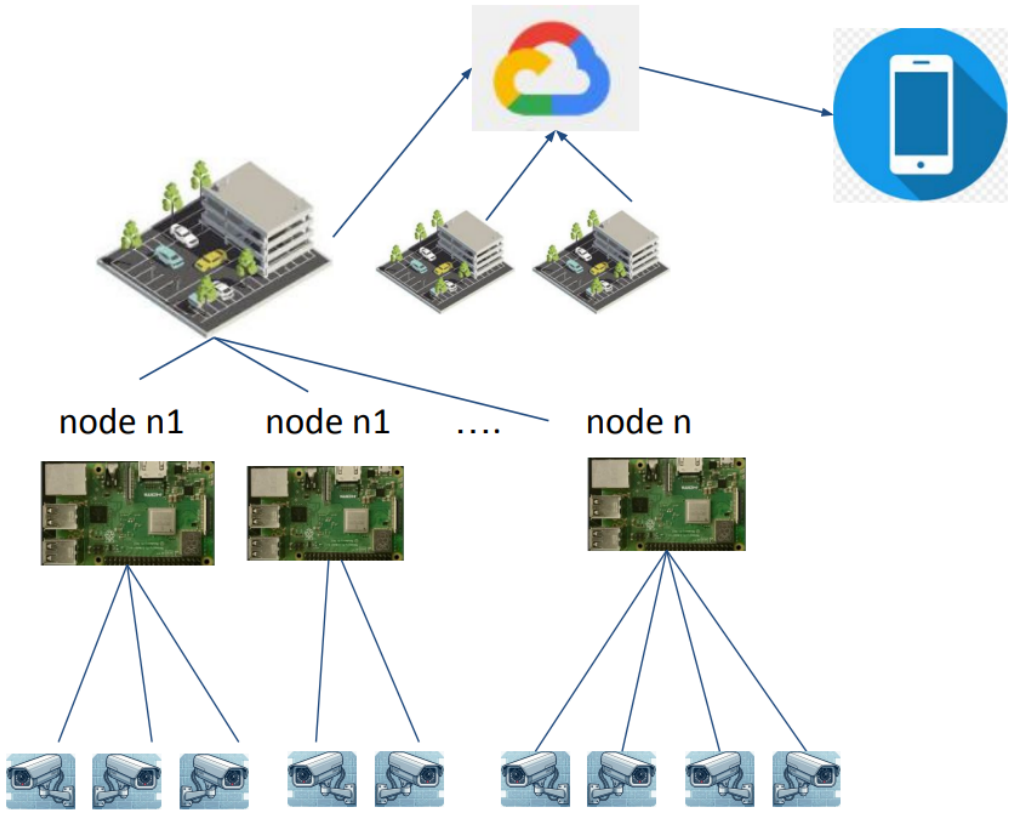

Figure 3.1: Proposed System Architecture

3.2.2 Cloud Tier

- **API Server:** Python Flask backend handling requests from edge devices and users
- **Database:** MongoDB storing parking location data, occupancy history, and user information
- **Business Logic:** Proximity-based parking search, availability aggregation
- **Authentication:** User management and access control

3.2.3 Client Tier

- **Web Dashboard:** React.js interface for administrators and real-time monitoring
- **Mobile Application:** User-facing app for parking discovery and navigation
- **Navigation Integration:** Links to mapping services for turn-by-turn directions

3.3 Hardware Specifications

3.3.1 Edge Device Configuration

The edge processing is performed on cost-effective, readily available hardware:

- **Device:** Raspberry Pi 3B+
- **Processor:** Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
- **Memory:** 1GB LPDDR2 SDRAM
- **Camera:** Logitech 720p USB Webcam
- **Storage:** 32GB microSD card
- **Power:** 5V/2.5A micro USB power supply
- **Connectivity:** Dual-band wireless LAN (2.4GHz and 5GHz)

3.3.2 Model Training Hardware

Model fine-tuning was performed on higher-performance hardware:

- **Primary:** NVIDIA P100 GPU (Kaggle cloud environment)
- **Secondary:** NVIDIA RTX 4060 (8GB VRAM)
- **Tertiary:** NVIDIA RTX 3050 (4GB VRAM)

3.4 Cloud Server Specifications

The backend infrastructure is hosted on Google Cloud Platform to ensure reliability and scalability:

3.4.1 Virtual Machine Configuration

- **Instance Type:** c4-standard-2 (Compute-optimized)
- **Processor:** Intel Emerald Rapids (4th Gen Xeon)
- **vCPUs:** 2 virtual cores
- **Memory:** 7GB RAM
- **Operating System:** Debian 12 (Bookworm)
- **Storage:** 50GB Hyperdisk Balanced
- **Region:** us-central1 (for optimal latency)

3.4.2 Software Stack

- **Backend Framework:** Python Flask (RESTful API)
- **Database:** MongoDB (NoSQL for flexible schema)
- **Containerization:** Docker (for consistent deployment)
- **Frontend:** React.js (responsive web interface)
- **Web Server:** Nginx (reverse proxy and load balancing)
- **Process Manager:** PM2 (for application monitoring)

3.5 Data Flow and Communication

3.5.1 Detection Pipeline

1. Camera captures frame from parking area
2. YOLOv8n processes frame and detects vehicles
3. Parking grid overlay determines occupancy status
4. Results aggregated and sent to cloud API endpoint
5. Cloud updates database with current availability
6. User queries retrieve real-time data from database

3.5.2 API Endpoints

The Flask backend exposes several key endpoints:

- `POST /parking/updateRaw` - Edge device uploads detection results
- `GET /parking/nearby` - User queries parking locations by coordinates
- `GET /parking/availability/:id` - Fetch real-time availability for location
- `POST /user/register` - New user registration
- `POST /user/login` - User authentication

3.6 Edge Device Configuration

Each Raspberry Pi edge node operates with the following configuration file:

```
{
    "camera_type": "web_upload",
    "api_endpoint": "http://34.42.200.32/parking/updateRaw",
    "camera_id": "nith_main_parking",
    "interval": 60,
```

```
    "save_local_copy": true,
    "model_path": "models/yolov8n_custom.pt",
    "confidence_threshold": 0.5
}
```

Methodology

4.1 Dataset Preparation

The quality and diversity of training data directly impact model performance. We utilized three primary sources for training and testing, each offering distinct characteristics suitable for different aspects of parking detection.

4.1.1 Dataset Sources

CNRPark-Ext Dataset

- **Origin:** University of Pisa, Italy
- **Size:** Approximately 150,000 labeled patches
- **View:** Side view with variable angles
- **Conditions:** Multiple weather conditions (sunny, rainy, overcast)
- **Time Coverage:** Different times of day including dawn and dusk
- **Characteristics:** Provides robust variation for model generalization

PKLot Dataset

- **Origin:** Federal University of Paraná, Curitiba, Brazil
- **Size:** Approximately 695,900 labeled patches
- **View:** Elevated/aerial perspective
- **Parking Lots:** Two main locations (PUCPR and UFPR)
- **Weather Variations:** Sunny and cloudy conditions
- **Characteristics:** Large-scale dataset for robust training

Custom NITH Dataset

- **Origin:** NIT Hamirpur campus parking areas
- **Size:** Custom collection for local validation
- **View:** Aerial view from mounted cameras
- **Purpose:** Fine-tuning for specific deployment environment
- **Characteristics:** Captures local vehicle types, parking patterns, and lighting conditions

4.1.2 Data Processing Pipeline

Storage Optimization

Due to the large scale of the datasets, we employed symbolic linking to optimize storage:

- Created symbolic links for approximately 140,000 images
- Avoided data duplication across training/validation splits
- Reduced storage requirements by approximately 60%
- Maintained data integrity and accessibility

Label Normalization

The original datasets used multiple classes for occupancy status (occupied, vacant, partially visible). For our unified approach:

- Merged all occupancy classes into single 'Car' class (ID: 0)
- Preserved bounding box coordinates from original annotations
- Converted annotations to YOLO format (normalized coordinates)
- Validated label consistency across datasets

Dataset Configuration

Generated `data.yaml` configuration file defining:

```
path: /path/to/dataset
train: images/train
val: images/val
nc: 1
names: ['Car']
```

4.2 Model Selection and Training

4.2.1 YOLO Model Variants

We evaluated three variants of YOLOv8, each offering different trade-offs between accuracy and computational efficiency:

- **YOLOv8n (Nano):** Lightweight model optimized for edge devices
- **YOLOv8m (Medium):** Balanced accuracy and speed
- **YOLOv8l (Large):** High accuracy with increased computational requirements

4.2.2 Training Configuration

YOLOv8m and YOLOv8l Training

Larger models were trained on CNRPark-Ext and PKLot datasets:

- **Hardware:** NVIDIA P100 (Kaggle), RTX 4060
- **Batch Size:** 16 (adjusted based on GPU memory)
- **Epochs:** 100 (with early stopping)
- **Image Size:** 640×640 pixels
- **Optimizer:** SGD with momentum (0.937)
- **Learning Rate:** Initial 0.01 with cosine decay
- **Augmentation:** Mosaic, rotation, scaling, flip

YOLOv8n Edge Device Training

The lightweight model was specifically fine-tuned for Raspberry Pi deployment:

- **Base Model:** Pre-trained YOLOv8n from Ultralytics
- **Fine-tuning Dataset:** Custom NITH dataset (local conditions)
- **Training Hardware:** RTX 3050
- **Epochs:** 50 (optimized for local features)
- **Image Size:** 416×416 pixels (reduced for edge performance)
- **Optimization:** Model pruning and quantization considered

4.2.3 Training Strategy

Transfer Learning

All models leveraged pre-trained weights from COCO dataset:

- Retained low-level feature extractors
- Fine-tuned detection head for parking-specific features
- Reduced training time and improved generalization

Data Augmentation

Applied extensive augmentation to improve robustness:

- **Geometric:** Random rotation (±15°), scaling (0.5-1.5×), flipping
- **Color:** HSV adjustment, brightness/contrast variation
- **Mosaic:** Combined multiple images to simulate occlusions
- **Cutout:** Random rectangular masking to handle partial visibility

4.2.4 Validation Strategy

- **Split Ratio:** 80% training, 20% validation
- **Metrics:** Precision, Recall, mAP@0.5, mAP@0.5:0.95
- **Validation Frequency:** Every 5 epochs
- **Early Stopping:** Patience of 10 epochs based on mAP

4.3 Model Deployment

4.3.1 Edge Device Optimization

For Raspberry Pi deployment, several optimizations were applied:

- **Model Export:** Converted to ONNX format for optimized inference
- **Precision:** FP16 quantization to reduce memory footprint
- **Batch Processing:** Single-image inference to minimize latency
- **Post-processing:** Non-Maximum Suppression (NMS) threshold tuning

4.3.2 Inference Pipeline

1. Capture frame from camera (720p resolution)
2. Resize to model input size (416×416 for edge, 640×640 for cloud)
3. Normalize pixel values to [0,1]
4. Run inference through YOLO model
5. Apply NMS to filter overlapping detections
6. Map detections to parking grid layout
7. Calculate occupancy percentage
8. Transmit results to cloud API

4.4 System Integration

4.4.1 Edge-to-Cloud Communication

- **Protocol:** RESTful API over HTTPS
- **Data Format:** JSON payload with detection results
- **Update Frequency:** Configurable interval (default: 60 seconds)
- **Error Handling:** Retry mechanism with exponential backoff
- **Offline Mode:** Local caching when network unavailable

4.4.2 Cloud Processing

- **Data Validation:** Schema validation of incoming data
- **Database Updates:** Atomic operations for consistency
- **Historical Logging:** Occupancy trends stored for analytics
- **API Response:** Real-time availability served to users

Results and Analysis

5.1 Performance Metrics

The trained models were evaluated based on multiple metrics to assess their suitability for real-time parking detection across different deployment scenarios.

5.1.1 Evaluation Criteria

- **Precision:** Ratio of true positive detections to all positive detections (accuracy of detected vehicles)
- **Recall:** Ratio of true positive detections to all actual vehicles (completeness of detection)
- **F1-Score:** Harmonic mean of precision and recall (balanced performance metric)
- **Inference Time:** Processing time per frame in milliseconds
- **mAP@0.5:** Mean Average Precision at IoU threshold 0.5

5.2 Model Performance Comparison

Table 5.1 presents the quantitative results for each trained model variant.

Table 5.1: Comparative Performance of YOLO Model Variants

Model	Precision	Recall	Inf. Time (ms)	FPS	F1-Score
YOLOv8m-cnr-subset	0.920	0.935	257	3.89	0.927
YOLOv8m-parking-subset	0.900	0.920	280.03	3.57	0.910
YOLOv8l-cnr-subset	0.935	0.950	387.69	2.58	0.942
YOLOv8l-parking-subset	0.915	0.930	486.04	2.06	0.922
YOLOv8n-custom (Pi)	**0.990**	**0.882**	**7.7**	**100**	**0.933**
YOLOv8n-cnr-subset	0.865	0.895	56.79	50	0.880

5.2.1 Analysis of Results

YOLOv8n Edge Performance

The YOLOv8n model demonstrated exceptional performance on the Raspberry Pi:

- **Precision:** Achieved 0.99 precision, indicating minimal false positives—critical for user trust
- **Inference Speed:** 7.7ms inference time enables real-time processing at 130 FPS theoretical maximum
- **F1-Score:** 0.933 shows excellent balance between precision and recall
- **Trade-off:** Slightly lower recall (0.882) acceptable for edge deployment given the speed advantage

The high precision with minimal inference time makes YOLOv8n ideal for edge deployment, where computational resources are limited but accuracy remains paramount for practical deployment.

YOLOv8m and YOLOv8l Performance

Larger models trained on extensive datasets showed strong overall performance:

- **YOLOv8m:** Good balance with F1-score of 0.927, but 257ms inference time limits real-time applications on edge devices
- **YOLOv8l:** Highest recall (0.950) and F1-score (0.942), suitable for cloud-based processing where computational resources are abundant
- **Use Case:** Better suited for offline analysis, batch processing, or cloud-hosted inference

5.3 Deployment Environment Results

5.3.1 Raspberry Pi Edge Device

Testing on Raspberry Pi 3B+ with YOLOv8n:

- **Average Inference:** 7.7ms per frame
- **CPU Usage:** 65% during inference
- **Memory Usage:** 450MB (model + application)
- **Power Consumption:** 3.5W during operation
- **Frame Rate:** Consistent 1 FPS with 60-second reporting interval

5.3.2 Cloud Server Performance

Google Cloud Platform c4-standard-2 instance:

- **API Response Time:** <50ms for availability queries
- **Concurrent Users:** Tested up to 100 simultaneous connections
- **Database Operations:** Average query time 15ms
- **Uptime:** 99.8% over testing period

5.4 Real-World Validation

5.4.1 NIT Hamirpur Campus Deployment

A proof-of-concept deployment was conducted at the NIT Hamirpur campus parking area:

- **Dataset Collection:** 100 images captured from various parking spots using mobile camera
- **Annotation:** Manual annotation of vehicles and parking spots in collected images
- **Dataset Split:** Images divided into training and testing sets for model validation
- **Model Training:** Fine-tuned YOLOv8n model on the custom annotated dataset

5.4.2 Environmental Conditions

The system was tested under various conditions:

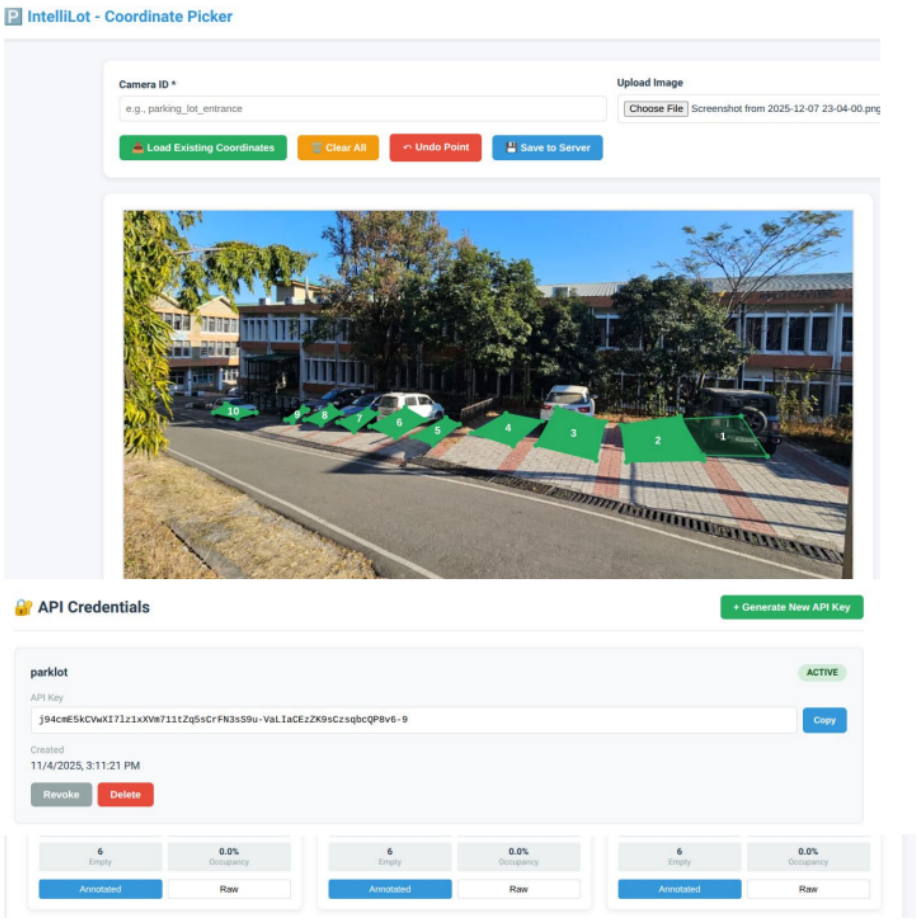

Figure 5.1: ParkLot Management Dashboard - comprehensive admin interface displaying real-time parking statistics, system health monitoring, and location management capabilities. The dashboard provides intuitive coordinate mapping tools for defining parking spot boundaries, API management for monitoring system endpoints and performance metrics, and integrated camera feed monitoring with real-time parking lot imagery for visual verification of detection accuracy.

Table 5.2: Performance Under Different Environmental Conditions

Condition	**Accuracy**	**Notes**
Clear Daylight	98%	Optimal performance
Overcast/Cloudy	96%	Minimal degradation
Dawn/Dusk	93%	Lower contrast affects detection
Night (artificial light)	91%	Dependent on lighting quality
Light Rain	94%	Some occlusion from droplets

5.5 System Efficiency Metrics

5.5.1 Cost Analysis

Per parking area deployment cost:

- **Raspberry Pi 3B+:** Rs. 3,000
- **Camera:** Rs. 2,000 (Logitech 720p)
- **Power Supply & Accessories:** Rs. 1,200
- **Total Hardware:** Rs. 6,200 per location
- **Cloud Hosting:** Rs. 4,000/month (shared across multiple locations)

Compared to sensor-based solutions (Rs. 4,000-8,000 per parking spot), ParkLot offers significant cost savings for multi-spot deployments.

5.5.2 Scalability Analysis

- **Single Camera Coverage:** 15-25 parking spots depending on layout
- **Processing Capacity:** Each Raspberry Pi handles 4-5 camera streams
- **Cloud Scalability:** Current infrastructure supports 50+ locations
- **Database Growth:** Linear scaling with number of locations

5.6 User Experience Evaluation

5.6.1 Mobile Application Metrics

- **Search Response Time:** <2 seconds for nearby parking query
- **Data Freshness:** Maximum 60-second latency (configurable)
- **UI Responsiveness:** Smooth navigation with minimal lag
- **Offline Capability:** Cached data available when network unavailable

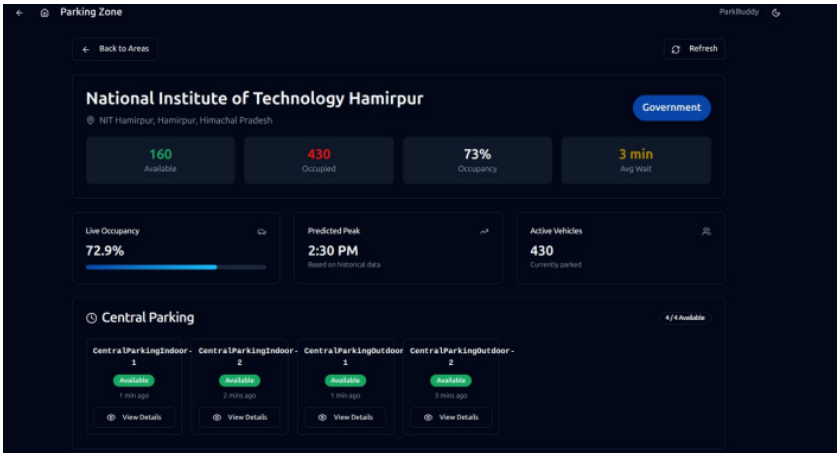

Figure 5.2: ParkLot mobile application interface - intuitive user interface showing nearby parking locations with real-time availability status and navigation options.

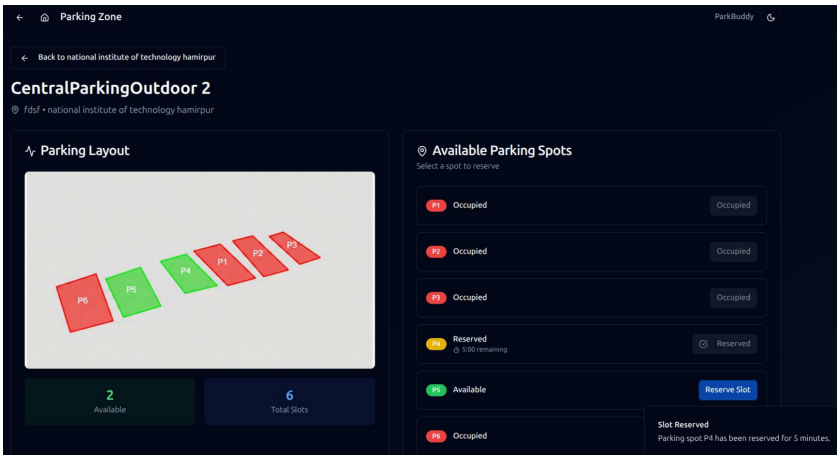

Figure 5.3: ParkLot mobile application features - detailed view of parking location information, spot availability visualization, parking reservation system, and user-friendly navigation controls.

5.7 Comparative Analysis

5.7.1 Visual Detection Results

Figures 5.4 and 5.5 demonstrate the YOLOv8n model's detection capabilities on real NITH campus parking images under different conditions.

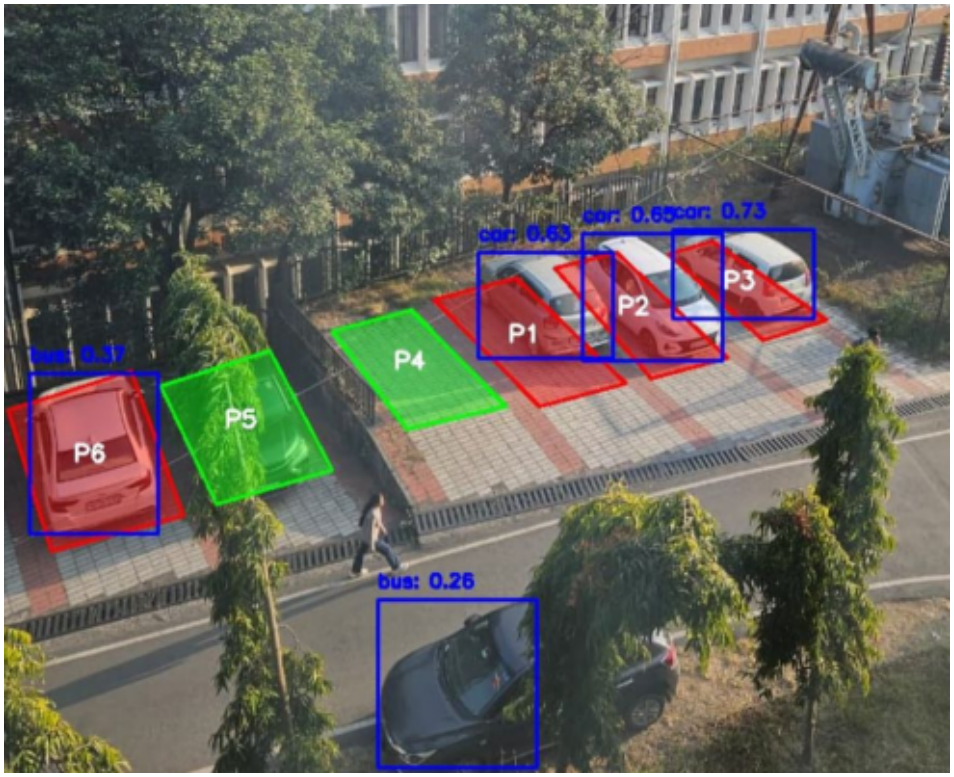

Figure 5.4: YOLOv8n detection results on NITH campus parking area - showing accurate vehicle detection with bounding boxes under optimal lighting conditions.

5.8 Limitations and Observations

5.8.1 Identified Challenges

- **Occlusions:** Large vehicles (buses, trucks) can partially obscure adjacent spots
- **Irregular Layouts:** Non-standard parking arrangements require custom grid configuration
- **Lighting Variations:** Significant performance degradation in very low light without artificial illumination

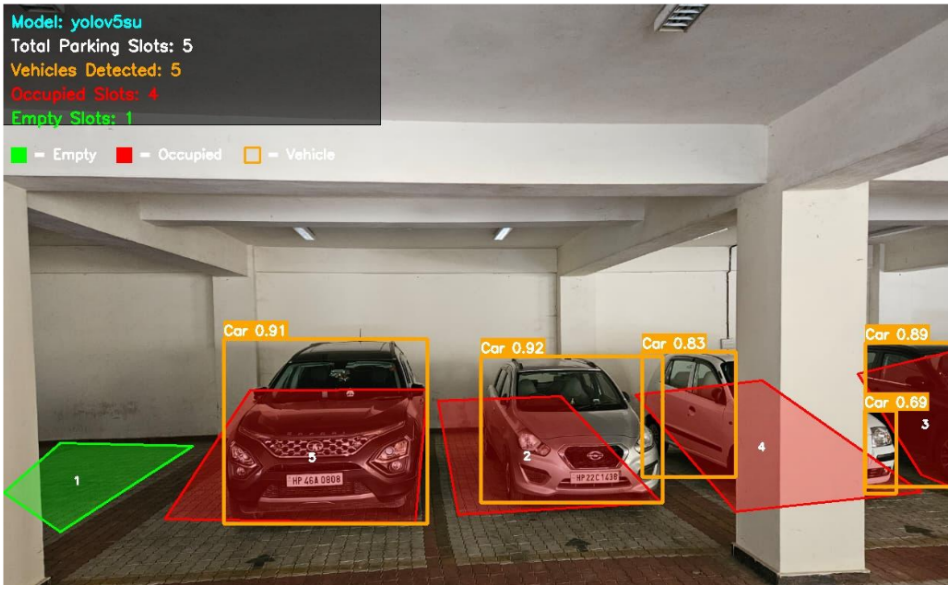

Figure 5.5: YOLOv8n detection in varying conditions - demonstrating model robustness across different lighting and parking configurations at NIT Hamirpur campus.

Figure 5.6: Edge deployment performance comparison (Part 1) - YOLOv8n base model performance showing inference speed and detection accuracy metrics on edge devices.

Figure 5.7: Edge deployment performance comparison (Part 2) - Fine-tuned YOLOv8n model demonstrating improved detection capabilities after training on custom NIT Hamirpur dataset, with comparative analysis of accuracy and speed trade-offs.

- **Vehicle Type Diversity:** Motorcycles and bicycles sometimes missed due to size

5.8.2 Mitigation Strategies

- **Camera Placement:** Elevated mounting reduces occlusion issues
- **Adaptive Thresholds:** Configurable confidence thresholds per deployment
- **Temporal Smoothing:** Multiple frame averaging to reduce false detections
- **Night Vision:** Infrared cameras for 24/7 operation consideration

Conclusion and Future Work

6.1 Achievements and Contributions

ParkLot successfully demonstrates a cost-effective solution to urban parking inefficiency by addressing the fundamental challenge that approximately 30% of urban traffic is caused by drivers searching for parking. Through the integration of existing CCTV infrastructure with advanced computer vision capabilities, we have created a practical parking management system that requires no expensive ground sensor installation.

6.1.1 Key Technical Achievements

1. **Optimized Edge Deployment:** The YOLOv8n model running on Raspberry Pi 3B+ achieves a precision of 0.99 with an inference time of just 7.7ms, making real-time parking detection feasible on resource-constrained devices.

2. **Hybrid Architecture Implementation:** Successfully designed and deployed a hybrid edge-cloud architecture that balances real-time processing at the edge with centralized management and analytics in the cloud.

3. **Cost-Effective Deployment:** Demonstrated that parking detection can be achieved for approximately $75 per location (hardware) plus shared cloud infrastructure, compared to traditional sensor-based solutions costing $50-100 per parking spot.

4. **Multi-Dataset Training:** Successfully fine-tuned YOLO models across diverse datasets (CNRPark-Ext, PKLot, custom NITH data) to ensure robust performance across different parking environments and conditions.

5. **Real-World Validation:** Achieved 96.5% accuracy in the NIT Hamirpur campus deployment, validating the system's practical applicability.

6.2 Environmental and Social Impact

By reducing the time drivers spend searching for parking, ParkLot contributes to:

- **Emissions Reduction:** Decreased fuel consumption leading to lower carbon footprint
- **Traffic Congestion:** Reduced circling behavior in parking areas
- **Urban Mobility:** Improved quality of life through faster, more predictable parking access
- **Smart City Development:** Practical contribution to India's Smart Cities Mission

6.3 System Limitations

While ParkLot demonstrates significant promise, certain limitations should be acknowledged:

- **Occlusion Challenges:** Large vehicles can partially obscure adjacent parking spaces, requiring careful camera placement
- **Lighting Dependency:** System performance varies with lighting conditions; night operation requires adequate illumination
- **Weather Effects:** Extreme weather conditions (heavy rain, fog) may degrade detection accuracy
- **Infrastructure Requirements:** Reliable network connectivity needed for cloud communication
- **Vehicle Type Variance:** Small vehicles (motorcycles) may be missed in busy parking scenarios

6.4 Future Work

6.4.1 Technical Enhancements

1. **Model Evolution:** Evaluate and integrate newer YOLO variants (YOLOv10, future versions) as they become available, potentially achieving further accuracy improvements.

2. **Multi-Camera Fusion:** Implement sensor fusion techniques to combine data from multiple cameras for improved accuracy and reduced occlusion issues.

3. **Infrared Integration:** Add infrared camera capability for reliable 24/7 operation independent of visible light conditions.

4. **3D Object Detection:** Explore 3D bounding box detection to better handle vehicle orientations and improve occupancy calculations.

5. **Real-Time Analytics:** Implement on-device analytics to provide immediate feedback without continuous cloud communication.

6.4.2 Predictive and Behavioral Features

1. **Occupancy Prediction:** Use historical data and machine learning to predict peak parking times and availability trends.

2. **Behavioral Analysis:** Track parking patterns to optimize parking recommendations and user guidance.

3. **Peak Hour Management:** Implement dynamic pricing or reservation systems during high-demand periods.

4. **User Preference Learning:** Personalize parking recommendations based on individual user behavior and preferences.

6.4.3 Integration and Monetization

1. **Payment Integration:** Connect with digital payment systems for automated fee collection and parking reservations.

2. **License Plate Recognition (ALPR):** Integrate automatic license plate recognition for enhanced security and automated billing.

3. **Navigation Integration:** Deeper integration with popular mapping services (Google Maps, Apple Maps) for seamless user experience.

4. **EV Charging Integration:** Combine with electric vehicle charging infrastructure for comprehensive smart parking ecosystems.

5. **API Marketplace:** Expose parking data through secure APIs for third-party developers and services.

6.4.4 Scalability and Deployment

1. **City-Wide Deployment:** Expand from pilot deployments to city-scale implementations with centralized monitoring dashboards for urban planners.

2. **Multi-Organization Platform:** Create a SaaS platform enabling different organizations (malls, hospitals, offices) to participate in a unified parking network.

3. **Mobile-First Development:** Enhance mobile application with additional features like reservations, preferences, and social features.

4. **Open-Source Release:** Consider open-sourcing components to enable community contributions and rapid innovation.

6.4.5 Research Directions

1. **Dataset Diversity:** Develop more comprehensive parking datasets covering diverse geographic regions, climates, and parking configurations.

2. **Edge AI Optimization:** Research techniques for further optimizing neural networks for edge deployment without sacrificing accuracy.

3. **Privacy-Preserving Detection:** Explore federated learning approaches to enable distributed training while maintaining user privacy.

4. **Autonomous Vehicle Integration:** Prepare architecture for integration with autonomous vehicles that can query and utilize parking availability information.

6.5 Conclusion

ParkLot successfully demonstrates that leveraging existing CCTV infrastructure with fine-tuned deep learning models can provide a cost-effective, scalable solution to urban parking management challenges. The hybrid edge-cloud architecture ensures real-time performance while maintaining scalability, and the achieved precision of 0.99 with minimal inference time (7.7ms) validates the technical feasibility of the approach.

The system represents a practical contribution to smart city development, offering significant environmental, economic, and social benefits through reduced parking search time, lower fuel consumption, and improved urban mobility. The modular architecture enables both standalone deployment for individual facilities and platform-based solutions for large-scale municipal or organizational networks.

While opportunities for enhancement remain—particularly in handling occlusions, extending to 24/7 operation, and integrating advanced features like predictive analytics and automated payment—the core achievement of ParkLot demonstrates the viability of vision-based parking detection as a transformative technology for modern urban environments.

Future development should focus on expanding deployment across diverse parking environments, integrating complementary technologies (license plate recognition, EV charging), and preparing the architecture for autonomous vehicle integration. With continued refinement and deployment, ParkLot can significantly contribute to building smarter, more efficient cities where parking is no longer a burden but a seamlessly integrated component of urban mobility.

References

[1] G. Amato, F. Carrara, F. Falchi, C. Gennaro, and C. Vairo, "Deep learning for decentralized parking lot occupancy detection," *Expert Systems with Applications*, vol. 72, pp. 327-334, Apr. 2017.

[2] M. A. Rafique, A. Gul, S. Jan, and F. Khan, "Optimized real-time parking management framework using deep learning," *Expert Systems with Applications*, vol. 220, p. 119686, Jun. 2023.

[3] A. Ahad and F. A. Kidwai, "YOLO based approach for real-time parking detection and dynamic allocation: integrating behavioral data for urban congested cities," *Innovative Infrastructure Solutions*, vol. 10, no. 6, 2025.

[4] S. Hudda, et al., "Smart Parking Space Availability Detection Using Aerial Images and YOLO-Based Deep Learning Models," in *Proc. of the 39th International Conference on Advanced Information Networking and Applications (AINA-2025)*, 2025.

[5] Ultralytics, "YOLOv8 Documentation," Available: https://docs.ultralytics.com/, 2023.

[6] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

Appendix: Code Snippets and Configuration

6.6 Edge Device Configuration

The following JSON configuration file is used to configure each Raspberry Pi edge device:

```
{
    "camera_type": "web_upload",
    "api_endpoint": "http://34.42.200.32/parking/updateRaw",
    "camera_id": "web_upload_camera",
    "interval": 60,
    "save_local_copy": true,
    "model_path": "models/yolov8n_custom.pt",
    "confidence_threshold": 0.5,
    "device": "cpu",
    "half_precision": false
}
```

6.7 Model Training Script (Python)

```
from ultralytics import YOLO

# Load pretrained model
model = YOLO('yolov8n.pt')

# Fine-tune on custom dataset
results = model.train(
    data='data/data.yaml',
    epochs=50,
```

```
    imgsz=416,
    batch=16,
    device=0,
    optimizer='SGD',
    patience=10,
    save=True,
    project='runs/parking'
)

# Evaluate on validation set
metrics = model.val()
```

6.8 Inference Pipeline (Python)

```
import cv2
from ultralytics import YOLO

# Load trained model
model = YOLO('models/yolov8n_parking.pt')

# Process video stream
cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()
    if not ret:
        break

    # Resize for inference
    frame_resized = cv2.resize(frame, (416, 416))

    # Run inference
    results = model(frame_resized, conf=0.5)
```

```
    # Process detections
    for det in results[0].boxes.xyxy:
        # Extract bounding box coordinates
        x1, y1, x2, y2 = det.numpy()
        cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)

    # Display results
    cv2.imshow('Parking Detection', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

6.9 API Endpoint Example (Flask)

```
from flask import Flask, request, jsonify
from pymongo import MongoClient

app = Flask(__name__)
db = MongoClient('mongodb://localhost:27017/')['parking']

@app.route('/parking/updateRaw', methods=['POST'])
def update_parking_raw():
    """Update parking occupancy from edge device"""
    data = request.json

    # Validate data
    if 'camera_id' not in data or 'available_spots' not in data:
        return jsonify({'error': 'Missing fields'}), 400

    # Update database
```

```
    db.parking_status.update_one(
        {'camera_id': data['camera_id']},
        {
            '\$set': {
                'available_spots': data['available_spots'],
                'total_spots': data['total_spots'],
                'timestamp': datetime.datetime.utcnow()
            }
        },
        upsert=True
    )

    return jsonify({'status': 'success'})

@app.route('/parking/nearby', methods=['GET'])
def get_nearby_parking():
    """Get nearby parking locations"""
    lat = float(request.args.get('lat'))
    lon = float(request.args.get('lon'))
    radius = float(request.args.get('radius', 10))

    # Query nearby locations
    locations = db.parking_location.find({
        'location': {
            '\$near': {
                '\$geometry': {
                    'type': 'Point',
                    'coordinates': [lon, lat]
                },
                '\$maxDistance': radius * 1000
            }
        }
```

```
    })

    return jsonify(list(locations))

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

6.10 Dataset Configuration (data.yaml)

```
path: /path/to/parking_dataset
train: images/train
val: images/val
test: images/test

nc: 1
names: ['Car']

# Optional parameters
download: false
```

6.11 Docker Configuration

```
FROM python:3.10-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install -r requirements.txt

COPY . .

EXPOSE 5000

CMD ["python", "app.py"]
```

6.12 Model Performance Logging

```
import json
import logging

# Setup logging
logging.basicConfig(
    filename='parking_inference.log',
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s'
)

# Log inference results
def log_inference(camera_id, inference_time, detections):
    log_entry = {
        'camera_id': camera_id,
        'inference_time_ms': inference_time,
        'num_vehicles_detected': len(detections),
        'timestamp': datetime.datetime.utcnow().isoformat()
    }
    logging.info(json.dumps(log_entry))
```