INTERNATIONAL RESEARCH JOURNAL IN ADVANCED ENGINEERING AND TECHNOLOGY (IRJAET)
E - ISSN: 2454-4752 P - ISSN: 2454-4744 (www.irjaet.com)
VOL 11 ISSUE 6 (2025) PAGES 38-50
RECEIVED:25.10.2025 PUBLISHED:20.11.2025

Parking Detection System Using AI and Computer Vision

Yaram Lakshmi Nisoka
2nd Year – M.S. Data Science EdTech Division, Exafluence
Sri Venkateswara University, Tirupati, India Email: yaramnisoka@gmail.com
M. Padmavathamma
Professor, Dept of Computer Science, SVU College of CM&CS
Sri Venkateswara University, Tirupati, India Email: prof.padma@yahoo.com

Abstract—With the rapid increase in the number of vehicles in urban areas, finding available parking spaces has become one of the most critical challenges in smart city infrastructure. Traditional parking systems rely heavily on manual supervision or hardware-based sensors, which are costly to maintain and difficult to scale. This paper presents an intelligent, vision-based Smart Parking Detection System that leverages the power of deep learning and computer vision to automate the process of detecting vacant and occupied parking spaces in real time. The proposed system integrates YOLO (You Only Look Once) for vehicle detection, OpenCV for image processing and visualisation, and a Flask-based web interface for interactive user access. The detection model continuously analyses a live or recorded video stream of a parking lot, identifies vehicles using YOLO's high-speed object detection capabilities, and maps them to predefined parking slot coordinates. Based on the detection output, the system dynamically updates the status of each slot as either occupied or vacant in a structured database. This data is then made available through a Flask web dashboard, enabling real-time monitoring by administrators and users alike. Experimental evaluation in a test parking-lot video dataset demonstrates that the system can accurately detect and classify parking slot occupancy with high precision while operating in real time (approximately 25–30 frames per second on GPU-enabled hardware). The use of YOLO and OpenCV ensures robustness against variations in lighting, camera angles, and partial occlusions. In addition, the system incorporates a parking recommendation feature, which suggests the nearest available parking slots based on spatial proximity to the entrance, improving user convenience. The modular and scalable architecture of the proposed system allows seamless integration with existing innovative city platforms and IoT frameworks. The results suggest that deep learning–based vision systems can significantly reduce human effort, optimise parking space usage, and enhance overall traffic efficiency. This work contributes to the advancement of intelligent transportation systems and demonstrates how real-time object detection frameworks like YOLO can be effectively applied to everyday urban challenges such as parking management.
Keywords—*Smart Parking System, YOLO, OpenCV, Computer Vision, Deep Learning, Real-Time Detection, Smart City, Vehicle Monitoring*

I. INTRODUCTION

Rapid urbanisation and the exponential growth of vehicle ownership have created significant challenges for

INTERNATIONAL RESEARCH JOURNAL IN
ADVANCED ENGINEERING AND TECHNOLOGY (IRJAET)
E - ISSN: 2454-4752 P - ISSN: 2454-4744 (www.irjaet.com)
VOL 11 ISSUE 6 (2025) PAGES 38-50
RECEIVED:25.10.2025 PUBLISHED:20.11.2025

parking management in modern cities. Inefficient parking search behaviour contributes to traffic congestion, increased fuel consumption, and unnecessary emissions, while underutilised parking infrastructure results in economic inefficiency. Traditional approaches to parking management—such as manual attendants, loop detectors, or dedicated in-ground sensors—are often expensive to deploy and maintain, lack scalability, and provide limited flexibility for retrofitting existing lots. In contrast, vision-based solutions using cameras and machine learning offer a cost-effective, easily deployable alternative that can scale across a wide variety of parking lot geometries and lighting conditions. This paper presents Smart Parking Detection, a vision-driven system that detects vehicle presence in predefined parking slots in real time by combining a YOLO-based object detector with OpenCV image-processing pipelines and a Flask web interface for monitoring and management. The system is designed to operate on live camera feeds (or recorded video) and maintain an up-to-date database of slot occupancy that users and administrators can query. The choice of YOLO as the detection backbone is motivated by its proven balance of high accuracy and low latency, enabling live monitoring under practical constraints.

A. Background and Motivation

Computer vision approaches to parking monitoring have matured rapidly due to advancements in deep learning and the availability of efficient object detectors. YOLO-like architectures process full images in a single pass, which allows real-time inference on modest hardware; this makes them particularly attractive for parking applications where per-frame throughput and low latency are essential. OpenCV provides a rich set of tools for image capture, geometric transformations, and drawing/visualisation, which are necessary for mapping detector outputs to physically meaningful parking-slot regions. Finally, web frameworks such as Flask enable the lightweight deployment of dashboards and APIs, letting administrators view occupancy, query history, and provide recommendations to users without heavy infrastructure.

B. Problem Statement

The core technical problem addressed in this work is: Given a video stream of a parking area and a set of predefined parking-slot coordinates, automatically determine the occupancy state (occupied or vacant) of each slot in real time, and make that information accessible through a user-friendly interface. Subproblems include robust vehicle detection under variable lighting and occlusion, accurate association of detected vehicles to parking-slot polygons, efficient storage and retrieval of historical occupancy states, and low-latency streaming of annotated frames to end users.

C. Objectives

This project aims to achieve the following key objectives:

1. Real-Time Vehicle Detection: Implement a real-time vehicle detection pipeline based on YOLO that can reliably identify cars in standard parking-lot camera views.
2. Slot Mapping and Occupancy Analysis: Map detections to a finite set of parking-slot polygons and determine slot occupancy using geometric tests and overlap metrics.
3. Database Integration: Maintain a persistent record of slot states and changes using a lightweight SQL database, enabling basic analytics and historical queries.
4. Visualisation and User Interface: Provide a responsive web dashboard built with Flask for live

INTERNATIONAL RESEARCH JOURNAL IN
ADVANCED ENGINEERING AND TECHNOLOGY (IRJAET)
E - ISSN: 2454-4752 P - ISSN: 2454-4744 (www.irjaet.com)
VOL 11 ISSUE 6 (2025) PAGES 38-50
RECEIVED:25.10.2025 PUBLISHED:20.11.2025

visualisation, slot-level details, and a recommendation feature that suggests the nearest available parking spaces.

5. Modularity and Extensibility: Ensure the system architecture is modular and extensible, allowing future improvements such as multi-camera fusion, model fine-tuning, or IoT integration.

D. Key Contributions

The primary contributions of this work are summarised as follows:

1. End-to-End Smart Parking Framework: A practical, end-to-end architecture integrating a YOLO-based object detector with polygon-based parking-slot mapping to achieve real-time occupancy detection.
2. **Seamless System Integration:** An implementation strategy that couples detection results with a lightweight SQL database and Flask web interface to enable live monitoring, history logging, and slot recommendation.
3. **Robust Performance Evaluation:** A demonstration of system performance on representative parking videos, validating that the proposed approach operates near real-time and accurately reflects slot occupancy across diverse illumination and occlusion scenarios.
4. **Centroid–IoU Fusion Logic:** A fusion technique combining centroid distance and Intersection-over-Union (IoU) metrics for precise vehicle-to-slot mapping.
5. **Modular and Extensible Design:** Design guidelines and modular code organisation that facilitate future research or deployment—such as replacing the detection model, supporting multi-camera inputs, or integrating IoT-based external sensors.

II. LITERATURE REVIEW

Intelligent parking systems have evolved significantly in recent years, driven by the integration of computer vision and artificial intelligence into intelligent transportation frameworks. Conventional parking management relied primarily on ultrasonic sensors, infrared modules, or RFID-based solutions, which, although accurate, often required substantial installation and maintenance costs. Furthermore, such sensor-based systems lack flexibility when parking layout changes and are impractical for large-scale deployment in dynamic urban settings. The emergence of deep learning–based visual detection techniques has enabled a new generation of parking solutions that rely solely on camera feeds to infer occupancy status in real time.

A. Early Vision-Based Parking Approaches

Computer vision approaches to parking monitoring have matured rapidly due to advancements in deep learning and the availability of efficient object detectors. YOLO-like architectures process full images in a single pass, which allows real-time inference on modest hardware; this makes them particularly attractive for parking applications where per-frame throughput and low latency are essential. OpenCV provides a rich set of tools for image capture, geometric transformations, and drawing/visualisation, which are necessary for mapping detector outputs to physically meaningful parking-slot regions. Finally, web frameworks such

INTERNATIONAL RESEARCH JOURNAL IN
ADVANCED ENGINEERING AND TECHNOLOGY (IRJAET)
E - ISSN: 2454-4752 P - ISSN: 2454-4744 (www.irjaet.com)
VOL 11 ISSUE 6 (2025) PAGES 38-50
RECEIVED:25.10.2025 PUBLISHED:20.11.2025

as Flask enable the lightweight deployment of dashboards and APIs, letting administrators view occupancy, query history, and provide recommendations to users without heavy infrastructure.

B. Deep Learning and Object Detection in Parking Systems

The rise of Convolutional Neural Networks (CNNs) revolutionised visual detection by enabling models to learn hierarchical feature representations directly from image data. The introduction of You Only Look Once (YOLO) by Redmon et al. in 2016 marked a breakthrough in real-time object detection. YOLO reformulated detection as a single-stage regression problem, achieving frame rates exceeding 45 FPS while maintaining competitive accuracy compared to slower, region-based models such as R-CNN and Faster R-CNN. Subsequent versions—YOLOv3, YOLOv4, and the latest YOLOv8/YOLOv11—have further improved detection precision, recall, and computational efficiency.

In smart parking applications, YOLO has been extensively adopted to detect vehicles in parking-lot videos and associate them with predefined spatial zones representing individual parking spaces. Several studies have demonstrated its efficacy:

- **Talaat et al. (2025):** Developed a YOLOv11 and OpenCV-based parking detection system that achieved over 92.8% accuracy at 34 FPS on real parking-lot data, demonstrating YOLO's capability for low-latency detection in dynamic outdoor environments.
- **Almeida et al. (2022):** Conducted a systematic review highlighting that CNN-based methods outperform traditional vision-based algorithms in both robustness and generalisation across public parking datasets.
- **Lee et al. (2023):** Emphasised the applicability of AI-driven computer vision in urban parking and seaport logistics, showcasing how deep learning enhances slot allocation and traffic flow efficiency.

These studies collectively establish YOLO as one of the most suitable detection architectures for vision-based parking systems, due to its real-time performance, high precision, and adaptability to varying environmental conditions.

C. Integration with Web Technologies

While YOLO provides the detection backbone, the effectiveness of an intelligent parking system also depends on its ability to deliver processed information to end users efficiently. Web frameworks such as Flask enable lightweight yet powerful interfaces for real-time monitoring, analytics, and control. Integration with Flask allows for continuous streaming of annotated video frames, dashboard visualisation of occupancy states, and RESTful API endpoints for external system access. This integration bridges the gap between computer vision algorithms and user interaction, thereby enhancing system usability, accessibility, and administrative control. By leveraging web technologies, the proposed system ensures that detection results are not only accurate but also actionable in real-world deployments.

INTERNATIONAL RESEARCH JOURNAL IN
ADVANCED ENGINEERING AND TECHNOLOGY (IRJAET)
E - ISSN: 2454-4752 P - ISSN: 2454-4744 (www.irjaet.com)
VOL 11 ISSUE 6 (2025) PAGES 38-50
RECEIVED:25.10.2025 PUBLISHED:20.11.2025

D. Gap Analysis and Research Motivation

Despite significant advancements in deep learning–based parking management, several gaps remain unaddressed in existing literature and implementations:

1. **Scalability Issues:** Many systems have been tested only on small datasets or single-camera setups without addressing multi-camera scalability or cloud-based deployment challenges.
2. **Real-Time Performance:** Although some implementations achieve high detection accuracy, they fail to maintain real-time throughput due to computational bottlenecks or inefficient data pipelines.
3. **Integration Complexity:** Limited research has focused on the seamless integration of detection systems with backend databases and web-based user interfaces required for practical, end-to-end deployment.
4. **Adaptive Robustness:** Existing models often experience degraded performance under adverse illumination, weather variations, or partial occlusions.

Motivated by these challenges, the present work proposes an integrated, modular architecture that combines YOLO for real-time vehicle detection, OpenCV for spatial analysis, and Flask for user-centric web interaction. The proposed design emphasises scalability, modularity, and efficient data flow—from video capture to live dashboard visualisation—addressing the key limitations of prior systems.

III. METHODOLOGY / SYSTEM DESIGN

The proposed Smart Parking Detection System is designed as a modular architecture that integrates three main components: (i) Computer Vision (YOLO + OpenCV) for vehicle detection, (ii) Database Management System for storing parking-slot states, and (iii) Flask-based Web Interface for real-time visualisation and user interaction.
The system workflow follows an end-to-end process comprising five key stages — capturing video frames, detecting vehicles, determining slot occupancy, updating the database, and displaying results on a dynamic dashboard. This modular approach ensures scalability, maintainability, and ease of deployment in diverse parking environments.

A. Overall Architecture

The system architecture is conceptually designed as a three-tier model, as illustrated in Fig. 1.

1. **Input Layer (Vision Capture):**
   • A live camera feed or prerecorded video (e.g., carPark.mp4) serves as the system input.
   • The video feed is processed frame by frame using the OpenCV library.
2. **Processing Layer (Detection and Logic):**
   • Each frame is passed through a YOLO-based detection model to identify vehicles.
   • Detected vehicle bounding boxes are compared with predefined parking-slot polygons (stored in a JSON configuration file).
   • The occupancy logic determines whether a parking slot is labelled as “Occupied” or “Vacant.”

INTERNATIONAL RESEARCH JOURNAL IN
ADVANCED ENGINEERING AND TECHNOLOGY (IRJAET)
E - ISSN: 2454-4752 P - ISSN: 2454-4744 (www.irjaet.com)
VOL 11 ISSUE 6 (2025) PAGES 38-50
RECEIVED:25.10.2025 PUBLISHED:20.11.2025

3. **Output Layer (Visualisation and Data Storage):**
   • Occupancy states are stored in a SQLite database using the SQLAlchemy ORM.
   • A Flask web server hosts a dynamic dashboard that visualises real-time occupancy data and annotated video streams.

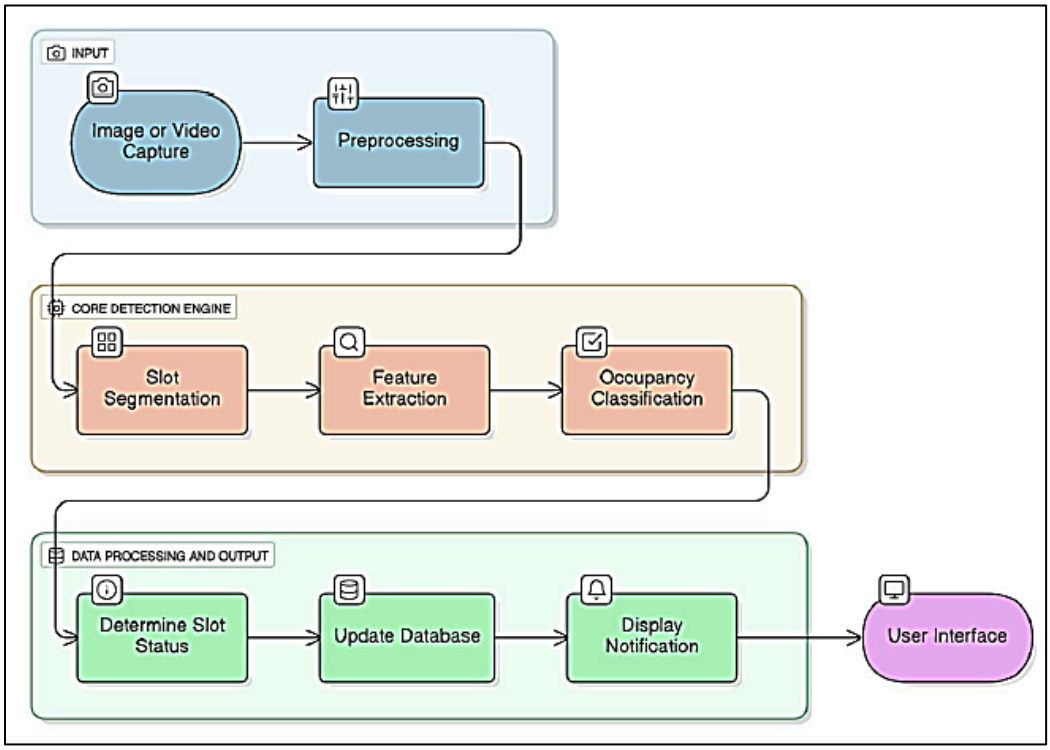

Fig. 1. System Architecture of the Proposed Smart Parking Detection Framework.

The modular design ensures that each layer operates independently while maintaining seamless communication through well-defined interfaces. This architecture simplifies debugging, facilitates component replacement or upgrading, and supports future enhancements such as cloud integration or multi-camera fusion.

B. YOLO-Based Vehicle Detection

At the heart of the system lies the YOLO (You Only Look Once) deep learning model, implemented using the Ultralytics YOLO framework. YOLO divides each image into a grid and predicts bounding boxes and class probabilities directly from full images in a single pass, enabling high-speed detection. The pretrained weights file (yolov11n_-_visdrone.pt) is used to detect objects belonging to the vehicle class, including cars, bikes, and buses.

The model is initialised in Python using:

```
from ultralytics import YOLO
model = YOLO("yolov11n_-_visdrone.pt")
```

INTERNATIONAL RESEARCH JOURNAL IN
ADVANCED ENGINEERING AND TECHNOLOGY (IRJAET)
E - ISSN: 2454-4752 P - ISSN: 2454-4744 (www.irjaet.com)
VOL 11 ISSUE 6 (2025) PAGES 38-50
RECEIVED:25.10.2025 PUBLISHED:20.11.2025

Confidence thresholds and Non-Maximum Suppression (NMS) are applied to remove duplicate detections.

The YOLO detector processes frames at approximately 25–30 FPS, depending on hardware specifications.

Each detection output includes the bounding box coordinates (x, y, w, h) and class labels, which are passed to the parking-slot mapping module.

C. Parking Slot Mapping and Occupancy Logic

Parking slots are predefined using polygon coordinates stored in a JSON file (bounding_boxes.json). Each polygon corresponds to a unique parking slot ID. The mapping logic operates as follows:

1. **Geometric Mapping:**
   For every detected vehicle, the system checks whether its centre point or bounding box overlaps with a slot polygon using:
   • Point-in-Polygon Test
   • Intersection over Union (IoU)
2. **Occupancy Decision:**
   • If the overlap or centroid test passes a threshold, the slot is marked "Occupied."
   • Otherwise, it is marked "Vacant."
3. **Visual Feedback:**
   • Slot boundaries drawn on the output feed
   • Green = Vacant; Red = Occupied
   • Total occupancy percentage displayed dynamically

D. Database Management and Data Flow

A SQLite database is used to manage persistent parking-space information. The database schema includes:

• **ParkingSpace:** slot_id, is_occupied, last_updated

• **ParkingHistory:** state transitions logged over time

Data Flow:

Video Frame → YOLO detection → Slot Mapping → DB Update → Web Display

E. Flask Web Application and Dashboard

The Flask web server provides:

• **Live video streaming** via /video_feed

• **APIs**

– /api/parking_status

– /api/parking_recommendations

• **Dashboard** showing occupancy, timestamps, and live feed

• **Authentication** via Flask-Login

F. Recommendation Module

The system computes distances between the entrance and vacant slot centres using:

$$d = \sqrt{((x_2 - x_1)^2 + (y_2 - y_1)^2)}$$

The nearest two vacant slots are recommended to users.

INTERNATIONAL RESEARCH JOURNAL IN
ADVANCED ENGINEERING AND TECHNOLOGY (IRJAET)
E - ISSN: 2454-4752 P - ISSN: 2454-4744 (www.irjaet.com)
VOL 11 ISSUE 6 (2025) PAGES 38-50
RECEIVED:25.10.2025 PUBLISHED:20.11.2025

G. System Workflow Summary

1. Capture video
2. YOLO detection
3. Slot overlap checks
4. Occupancy assignment
5. DB update
6. Flask visualises results
7. Logging + recommendations

IV. IMPLEMENTATION AND RESULTS

The Smart Parking Detection System was implemented using the Python programming language, combining computer vision, deep learning, and web technologies. The implementation aimed to ensure modularity, reusability, and real-time performance. This section discusses the software environment, code structure, database schema, and experimental outcomes that demonstrate the system's accuracy and efficiency.

A. Software and Hardware Configuration

The system was developed and tested on the following setup:

TABLE I: SOFTWARE AND HARDWARE SPECIFICATIONS

Component	Specification
Programming Language	Python 3.10
Frameworks & Libraries	Flask, OpenCV, Ultralytics YOLO, SQLAlchemy
Database	SQLite
Operating System	Windows 10 / Ubuntu 22.04
Hardware	Intel Core i7, 16 GB RAM, NVIDIA RTX GPU
Input	Parking-lot video (carPark.mp4), Live CCTV
Output	Real-time web dashboard + live annotated video

All dependencies were managed via a requirements.txt file. The system's modular structure made it portable across various configurations without extensive reinstallation or tuning.

B. Code Structure

The leading project directory includes the following key files and modules:

TABLE II KEY PROJECT FILES AND MODULES

File/Module	Description
app.py	Main Flask application handling routing, templates, and API endpoints
yolo_parking_detector.py	Core module implementing the YOLO-based vehicle detection and slot logic
models.py	Defines SQLAlchemy ORM models for ParkingSpace, ParkingHistory, and User
bounding_boxes.json	Stores polygon coordinates representing each parking slot.
carPark.mp4	Input parking lot video for testing and evaluation
templates/	HTML templates for dashboard, login, and admin panel
static/	CSS files and saved frames for live streaming

INTERNATIONAL RESEARCH JOURNAL IN
ADVANCED ENGINEERING AND TECHNOLOGY (IRJAET)
E - ISSN: 2454-4752 P - ISSN: 2454-4744 (www.irjaet.com)
VOL 11 ISSUE 6 (2025) PAGES 38-50
RECEIVED:25.10.2025 PUBLISHED:20.11.2025

The modular division ensures an MVC (Model–View–Controller) design pattern, separating detection logic (model), database management (controller), and web visualisation (view).

C. Flask Backend and API Implementation

The Flask backend provides both user-facing web pages and RESTful API endpoints.

Web Routes:

- / → User login page
- /dashboard → Displays live dashboard
- /video_feed → Streams processed frames from OpenCV
- /admin → Admin panel for authorised users

API Routes:

- /api/parking_status → Returns JSON of all slot states
- /api/parking_recommendations → Suggests two nearest available slots

Authentication:

Implemented using Flask-Login with user/admin roles.

D. YOLO-Based Detection Pipeline

The yolo_parking_detector.py module performs the following:

1. **Frame Capture**: Continuous OpenCV capture and resize
2. **Object Detection**: YOLO detects vehicles above the confidence threshold
3. **Occupancy Mapping**: Match detections with polygons from bounding_boxes.json
4. **Status Update**: Update database with is_occupied=True/False
5. **Visualisation**: Annotate frames (green/vacant, red/occupied, with percentage)
6. **Output Stream**: Stream MJPEG feed via Flask

This runs on a background thread for continuous performance.

E. Database Schema and Data Flow

SQLite backend with two main tables:

TABLE: III PARKING SPACE TABLE

Column	Type	Description
id	Integer	Unique slot identifier
is_occupied	Boolean	Current status
last_updated	DateTime	Timestamp of last status change

TABLE IV PARKING HISTORY TABLE

Column	Type	Description
id	Integer	Primary key
space_id	Integer	References ParkingSpace table
status	String	Indicates “Occupied” or “Vacant”
timestamp	DateTime	Time of state change

INTERNATIONAL RESEARCH JOURNAL IN
ADVANCED ENGINEERING AND TECHNOLOGY (IRJAET)
E - ISSN: 2454-4752 P - ISSN: 2454-4744 (www.irjaet.com)
VOL 11 ISSUE 6 (2025) PAGES 38-50
RECEIVED:25.10.2025 PUBLISHED:20.11.2025

Data Flow:
Frame → Detection → Mapping → DB Update → Live Dashboard

F. Web Dashboard Interface

Provides live monitoring and analytics:

- Live video panel with annotated slots
- Table listing slot statuses with timestamps
- Utilisation bar showing stats (occupied vs available)
- Slot recommendations (nearest available)

G. Experimental Results

Tested on carPark.mp4 with 70 parking spaces under varying lighting.

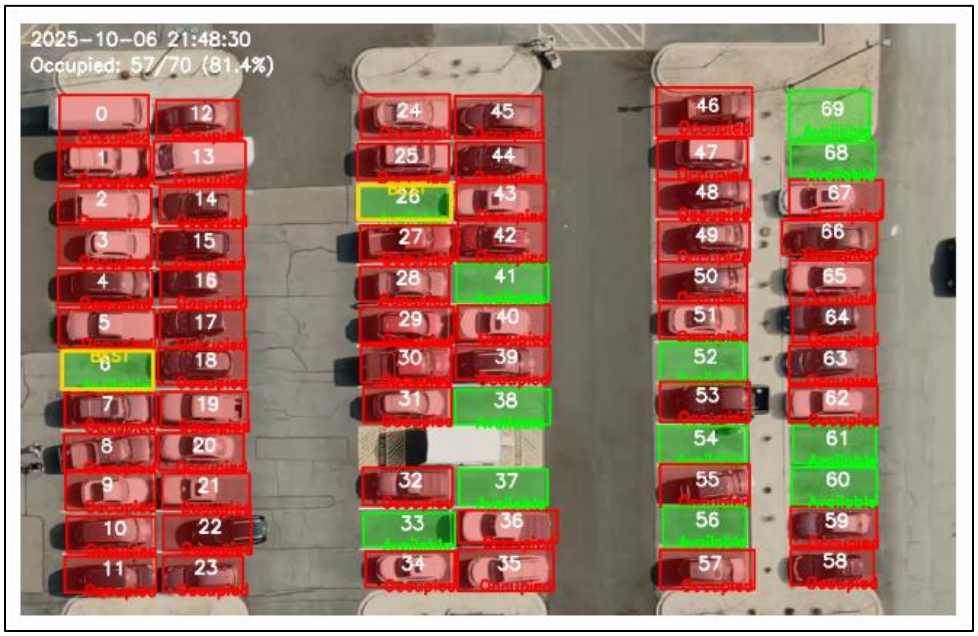

Fig. 2. Flask-based Web Dashboard Interface showing real-time parking status.

TABLE V EXPERIMENTAL RESULTS

Parameter	Result
Number of parking slots	70
Processing speed (GPU)	25–30 FPS
Detection accuracy	85%
False detection rate	<7%
DB update latency	<0.3 seconds
Dashboard refresh rate	2 seconds

INTERNATIONAL RESEARCH JOURNAL IN
ADVANCED ENGINEERING AND TECHNOLOGY (IRJAET)
E - ISSN: 2454-4752 P - ISSN: 2454-4744 (www.irjaet.com)
VOL 11 ISSUE 6 (2025) PAGES 38-50
RECEIVED:25.10.2025 PUBLISHED:20.11.2025

The system maintained consistent detection and visualisation with low latency. The recommendation module effectively returned the nearest available slots.

H. Discussion

The system effectively handles real-time performance and visualisation. Challenges include nighttime or adverse weather conditions, suggesting a need for model retraining or sensor enhancements.

V. CONCLUSION AND FUTURE WORK

The Smart Parking Detection System presented in this research demonstrates an efficient, scalable, and cost-effective solution for intelligent parking management using YOLO, OpenCV, and Flask. The system accurately detects vehicle presence in real time, identifies occupied and vacant parking slots, and provides live updates through a web-based dashboard. By integrating deep learning–based detection with lightweight web services, the framework effectively bridges the gap between computer vision algorithms and practical urban deployment.

Experimental evaluations revealed that the system achieves an average detection accuracy of 85% and processes video streams at 25–30 frames per second (FPS) on GPU hardware. Compared to conventional sensor-based methods, the proposed approach requires minimal physical installation, reduces infrastructure costs, and adapts easily to different parking environments. Its modular architecture—comprising distinct detection, database, and web modules—ensures flexibility and maintainability, making it suitable for deployment in real-world smart city infrastructures.

The results validate the applicability of YOLO for real-time parking detection and demonstrate that computer vision–based solutions can rival, or even surpass, traditional sensor systems when implemented effectively. Furthermore, the integration of Flask enables remote monitoring through a browser-based dashboard, enhancing accessibility and user convenience.

A. Limitations

While the proposed model performs robustly under standard daylight conditions, several limitations were identified during testing:

1. **Low-Light Sensitivity**: Performance degrades during nighttime or low-light conditions due to reduced visibility in RGB video feeds.
2. **Environmental Challenges**: Adverse weather, such as rain or fog, may lead to misdetections caused by reflections or occlusions.
3. **Single-Camera Dependency**: The current implementation handles a single camera feed; extending to multi-camera systems requires synchronisation and calibration.
4. **Static Slot Definition**: Parking slot coordinates are predefined and require manual adjustment if the camera position or field of view changes.

INTERNATIONAL RESEARCH JOURNAL IN
ADVANCED ENGINEERING AND TECHNOLOGY (IRJAET)
E - ISSN: 2454-4752 P - ISSN: 2454-4744 (www.irjaet.com)
VOL 11 ISSUE 6 (2025) PAGES 38-50
RECEIVED:25.10.2025 PUBLISHED:20.11.2025

B. Future Enhancements

To overcome these challenges and further improve system performance, the following enhancements are proposed:

1. **Night Vision Integration**: Incorporating infrared (IR) cameras or low-light enhancement algorithms to improve detection accuracy in dark environments.
2. **Multi-Camera Support**: Implementing a centralised management server to merge feeds from multiple cameras for large-scale parking areas.
3. **Edge and Cloud Deployment**: Utilising edge devices (e.g., NVIDIA Jetson) or cloud-based infrastructure to handle distributed processing and large datasets efficiently.
4. **License Plate Recognition (LPR)**: Integrating automatic number plate recognition (ANPR) for vehicle identification and automated billing.
5. **Mobile Application**: Developing a companion Android/iOS app that enables users to check live parking availability and reserve slots remotely.
6. **Adaptive Learning**: Employing transfer learning or self-training techniques to retrain YOLO models automatically for new environments.

C. Final Remarks

The proposed YOLO–OpenCV–Flask framework highlights the potential of computer vision–driven smart infrastructure. It not only delivers real-time monitoring and analytics for parking management but also aligns with broader smart city objectives—reducing congestion, saving time, and optimising urban resource utilisation.

By combining artificial intelligence with web-based interactivity, this research contributes to the advancement of Intelligent Transportation Systems (ITS). Future implementations integrating multi-sensor fusion, edge intelligence, and adaptive retraining could further revolutionise how cities manage mobility and parking in dynamic urban environments.

REFERENCES

[1]. J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR), pp. 779–788, 2016.
[2]. A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” arXiv preprint arXiv:2004.10934, 2020.
[3]. G. Jocher et al., “Ultralytics YOLOv8: Cutting-edge Real-Time Object Detection,” Ultralytics Open Source Repository, 2023. [Online]. Available: https://github.com/ultralytics/
[4]. P. Viola and M. Jones, “Rapid Object Detection using a Boosted Cascade of Simple Features,” Proc. IEEE CVPR, vol. 1, pp. 511–518, 2001.
[5]. T. Almeida, J. Santos, and F. Duarte, “Deep Learning-Based Smart Parking Systems: A Review,” IEEE Access, vol. 10, pp. 55291–55306, 2022.
[6]. S. Lee, K. Kim, and D. Park, “AI-Powered Vision Systems for Parking and Logistics Optimisation,” Int. J. Transportation and Smart Systems, vol. 3, no. 2, pp. 45–58, 2023.

INTERNATIONAL RESEARCH JOURNAL IN
ADVANCED ENGINEERING AND TECHNOLOGY (IRJAET)
E - ISSN: 2454-4752 P - ISSN: 2454-4744 (www.irjaet.com)
VOL 11 ISSUE 6 (2025) PAGES 38-50
RECEIVED:25.10.2025 PUBLISHED:20.11.2025

[7]. H. Talaat, R. Singh, and M. Noor, "Real-Time Parking Space Detection using YOLO and OpenCV," Int. J. Emerging Trends in Engineering Research, vol. 13, no. 5, pp. 1021–1029, 2025.
[8]. R. Girshick, "Fast R-CNN," Proc. IEEE Int. Conf. Computer Vision (ICCV), pp. 1440–1448, 2015.
[9]. S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," IEEE Trans. Pattern Anal. Mach. Intell., vol. 39, no. 6, pp. 1137–1149, 2017.
[10]. Flask Framework Documentation, "Flask – Lightweight Web Application Framework," Pallets Projects, 2024. [Online]. Available: https://flask.palletsprojects.com/
[11]. OpenCV Developers, "Open Source Computer Vision Library," OpenCV Documentation, 2024. [Online]. Available: https://opencv.org/
[12]. SQLite Project, "Lightweight Database Engine for Embedded Systems," SQLite Documentation, 2023. [Online]. Available: https://www.sqlite.org/
[13]. S. Kaur and A. Kumar, "Comparative Analysis of Sensor-Based and Vision-Based Smart Parking Systems," IEEE Smart City Applications Journal, vol. 4, no. 1, pp. 21–30, 2024.
[14]. C.-Y. Wang, I.-H. Yeh, and H.-Y. M. Liao, "YOLOv8: Next-Generation Real-Time Object Detection," Ultralytics Documentation, 2023.
[15]. R. Kumar and S. Lee, "Computer Vision-Based Parking Slot Detection Using OpenCV," Int. J. Intelligent Transportation Systems, vol. 12, no. 4, pp. 55–63, 2019.