IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511170

ParkSense: A Computer Vision-Based Parking Space Detection and Occupancy Prediction System Using YOLO and Machine Learning

Kalpesh Rajendra Patil[1], Tejas Narendra Borse[2], Krishna Chandrakant Patil[3], Prof. Kaminee S. Patil[4]

Student, Department of Computer Science & Engineering (Data Science), R. C. Patel Institute of Technology, Shirpur, India[1,2,3]

Guide, Department of Computer Science & Engineering (Data Science), R. C. Patel Institute of Technology, Shirpur, India[4]

Abstract: Finding a free parking bay is a daily problem on campuses, in commercial car parks and in city centres: drivers circle the lot without knowing where a bay is free or how many will be free when they arrive. Sensor-based solutions need one device for every bay and are costly to retrofit, while most camera-based approaches only classify the current state and give no forecast. This paper presents ParkSense, a web-based system that detects parking-slot occupancy from a single fixed camera and predicts the number of free slots for the near future. A YOLOv8 model fine-tuned from COCO weights detects vehicles in every frame. An occupancy engine compares each vehicle box with stored slot polygons using an overlap ratio and applies temporal smoothing, so that a slot changes state only after several consistent frames. The slot states are logged as a time series, and persistence and historical-average baselines, a random forest, XGBoost and an LSTM are compared to forecast free slots 15, 30 and 60 minutes ahead from calendar and recent-history features. A chronological, leakage-aware split by whole days ensures that all results come from days the models never saw. The prototype combines a FastAPI back end, a SQL database and a responsive dashboard that shows a live slot map, counters, a forecast with a range, the nearest free slot and alerts. Demonstration runs on a 396-slot test lot show that the image, video and live modes report consistent slot counts (about 68 % occupancy in the video and live runs), while formal accuracy and forecasting results on held-out days are left to future work. The system does not read number plates and does not store frames by default.

Keywords: Parking space detection, Occupancy prediction, YOLO, Computer vision, Time-series forecasting, Smart parking.

I. INTRODUCTION

Finding a free parking space is one of the most routine and most frustrating parts of urban life. Colleges, hospitals, shopping centres, offices and public car parks all see the same pattern: drivers enter a lot, circle through the aisles looking for a vacant bay, and leave when they cannot find one. A noticeable share of congestion in busy districts is caused by drivers cruising for parking [1]. The wasted time, extra fuel and added emissions are small for each driver but large when summed over a whole campus or city.

Most car parks in India still have no automatic way of telling a driver how many bays are free. Where a system exists, it usually relies on in-ground sensors, ultrasonic units above every bay, or loop detectors at the gates. These work, but they are costly to install, need wiring and maintenance, and give only a count at the entrance rather than the position of free bays. A single camera can watch many bays at once and most lots already have CCTV, and modern object detectors such as YOLO [2] can find vehicles in real time. A second question is just as useful to the driver: not only how many bays are free now? but how many will be free when I arrive?

ParkSense combines the two ideas. It uses a YOLO-based detector to find vehicles in camera frames, maps the detections to a configured set of parking slots to decide which are occupied, stores the result as a time series, and uses machine-learning models to predict the number of free slots for the next 15, 30 and 60 minutes. A web dashboard shows the live slot map, the forecast and the nearest free slot.

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511170

The main contributions of this paper are: (1) an integrated pipeline that joins vehicle detection, slot decision, temporal smoothing and short-term forecasting using cameras that already exist; (2) a slot-overlap rule with N-frame smoothing that removes flicker caused by passing vehicles; (3) a comparison of baseline, tree-based and recurrent forecasters that reports a range together with each forecast; (4) a chronological, leakage-aware evaluation protocol with honest baselines; and (5) a deployable web prototype with a REST interface, role-based access and privacy by design.

II. RELATED WORK

Parking occupancy detection has been studied for more than two decades. Early systems used magnetic loops, infrared or ultrasonic sensors fixed to each bay; vision-based approaches replaced the per-bay sensor by a camera and hand-crafted features classified with a support vector machine. De Almeida et al. released PKLot, a dataset of parking-lot images in several weather conditions, and showed that texture-based classifiers trained on one lot degrade on another [3]. Amato et al. ran a small convolutional network on a smart camera to classify every bay patch and introduced CNRPark+EXT to test generalisation across cameras [4]. Availability prediction is a separate line of work: Caicedo et al. forecast availability from the arrival and departure of vehicles at the entrances of a large car park [5], and Zheng et al. found regression trees the most accurate short-term forecasters on sensor data [6].

On the detection side, two-stage detectors such as R-CNN [7] and Faster R-CNN [8] are accurate but heavy, while single-stage detectors such as YOLO [2] and SSD [9] predict boxes in one pass. YOLOv7 [10] and YOLOv8 [11] are widely used for real-time applications. For forecasting, ARIMA and exponential smoothing [12], [13] handle seasonality but not extra inputs such as holidays; random forests [14] and gradient-boosted trees [15], [16] are strong on tabular features, and LSTM networks [17] learn temporal dependencies directly. Table I compares these families with the proposed system.

TABLE I SUMMARY OF RELATED WORK / GAP ANALYSIS

Approach	Data / Parameters	Algorithms / Techniques	Limitations and Gap
Per-bay sensors and entrance counters	Magnetic, ultrasonic, gate counts [5], [6]	Counting, regression on sensor data	One device per bay, cabling and maintenance; no camera pipeline
Patch-level bay classification	PKLot, CNRPark+EXT [3], [4]	SVM, mAlexNet on cropped bays	Manual cropping per camera; poor cross-camera transfer; no forecasting
Two-stage detectors	COCO / VOC-style labelled boxes	R-CNN [7], Faster R-CNN [8]	Accurate but heavier; slower on modest hardware
Single-stage detectors	COCO weights, fine-tuned	YOLO [2], SSD [9], YOLOv8 [11]	Detects vehicles only; no slot decision, smoothing or forecast by itself
Statistical and ML forecasting	Occupancy history, calendar	ARIMA [12], RF [14], XGBoost [15], LSTM [17]	Usually separate from detection; random splits and no uncertainty range
Proposed ParkSense	Fixed-camera frames, slot polygons, slot-state log	YOLOv8 + overlap rule + N-frame smoothing + RF / XGBoost / LSTM	No sensors or cropping; stable slot states; forecast with range; chronological evaluation

The gap addressed here is a lightweight, deployable workflow: slot detection, temporal stabilisation and short-term forecasting in one web application, a time-ordered evaluation protocol that avoids optimistic accuracy, forecasts reported with a range a driver can interpret, and low-cost deployment on existing CCTV for Indian campus and commercial lots.

III. PROBLEM IDENTIFICATION

Given a stream of images from a fixed camera covering a parking lot, together with a one-time definition of where each slot lies in the image, the system must detect the vehicles in every frame, decide for every slot whether it is free or occupied, keep a flicker-free record of slot states over time, predict the number of free slots for a short future horizon, present the result through a dashboard and a REST interface, and raise an alert when the lot is nearly full. The following practical difficulties were identified:

 This work is licensed under a Creative Commons Attribution 4.0 International License



International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026
DOI: 10.17148/IJARCCE.2026.1511170

- **Perspective and occlusion:** a tall vehicle hides part of the neighbouring bay in an oblique view, and a bay is a region of ground rather than an object.
- **Lighting and weather:** shadows, glare, rain and night-time lighting change the appearance of vehicles.
- **Flicker from moving vehicles:** a car driving past a slot is not parked in it, but a frame-by-frame decision would change the slot state.
- **Small, distant vehicles:** bays at the far end of the view appear only a few pixels wide.
- **Demand variation and data scarcity:** weekends, holidays and events change the daily pattern, and a new lot has little history.
- **Cost and optimistic evaluation:** per-bay sensors are expensive to retrofit, and random train/test splits of near-identical frames inflate reported accuracy.

IV. PROPOSED SYSTEM

ParkSense follows a layered, client–server architecture (Table II). A detection service reads frames from the camera, resizes and normalises them with OpenCV [18] and runs a fine-tuned YOLOv8 model. An occupancy engine compares the detected vehicle boxes with the stored slot polygons and applies temporal smoothing. The slot states are written to the database, a prediction service builds features from the stored history and returns the forecast, and a FastAPI back end exposes everything through REST endpoints [19] and serves the web dashboard.

TABLE II SOFTWARE ARCHITECTURE OF THE PROTOTYPE

Layer	Technology	Responsibility
Camera / input	IP camera, RTSP, file upload	Provides frames to the detection service
Computer vision	Ultralytics YOLOv8, OpenCV	Pre-processing, vehicle detection, NMS
Occupancy logic	Python, Shapely	Slot–vehicle overlap, thresholds, smoothing
Prediction	scikit-learn [20], XGBoost, Keras	Feature building, training, forecasting
API / business logic	FastAPI, Pydantic	Validation, endpoints, alerts, authentication
Persistence	SQLite / PostgreSQL, SQLAlchemy	Slots, occupancy log, forecasts, users
Presentation	HTML, CSS, JavaScript, Chart.js	Live map, counters, forecast chart

The design separates concerns: detection, occupancy logic, prediction and presentation are independent modules with narrow interfaces, and lots, slots and thresholds are stored as data, so a new lot needs no code change. Detection is a perception problem that benefits from a pretrained, fine-tuned network, whereas prediction is a time-series problem that benefits from calendar features and history; keeping them separate lets each be evaluated and replaced independently. The dashboard shows the lot name, four counters (total, occupied, free, occupancy percentage), a colour-coded live slot map, a forecast panel, the nearest free slot and an alert list; an administrator page is used to draw slot polygons and set thresholds.

V. DATASET AND DETECTION MODEL

A. Dataset Development

The dataset has two parts. The detection set consists of frames from the deployment camera annotated with vehicle boxes in YOLO format; to save effort the pretrained model first proposes boxes, which annotators then correct. PKLot [3] images can be added for variety, but deployment-camera frames remain the main source because final accuracy depends on matching the real viewpoint. The forecasting set is the log of slot states collected while the system runs. Table III gives the classes and the indicative collection targets. Frames should cover morning, afternoon, evening and night and clear, cloudy and rainy days; partly visible vehicles are boxed when more than about 30 % is visible; and own recordings are made with the permission of the lot owner with no personal identification retained.

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511170

TABLE III TARGET CLASSES AND INDICATIVE COLLECTION TARGETS

Class ID	Class	Examples covered by the annotation guide	Target boxes
0	Car	Hatchback, sedan, SUV, van, pick-up	2,500
1	Two-wheeler	Motorcycle, scooter, bicycle in a bay	1,200
2	Bus / truck	Mini-bus, bus, light truck	300
—	Background frames	Empty lot, shadows, trolleys, no vehicles	400 frames

B. Leakage-Aware Data Splitting

Parking frames captured seconds apart are nearly identical, so a random split would put almost the same image in both the training and the test set. For forecasting the problem is worse, because a random split lets the model see the future of the period it is tested on [13]. ParkSense therefore splits chronologically by whole recording days: the earliest days are used for training, the following days for validation and the latest days for testing, with a one-day gap between the parts. Model selection and threshold tuning use the validation part only, and the test part is used once for the final report.

C. Vehicle Detection with YOLO

YOLO [2] divides the image into a grid, and each cell predicts bounding boxes, objectness and class scores in a single forward pass. Several overlapping boxes may be predicted for one vehicle, so non-maximum suppression (NMS) [21] keeps the box with the highest score and discards boxes whose intersection over union (IoU) with it exceeds a threshold. IoU of two boxes A and B is

$$IoU(A, B) = Area(A \cap B) / Area(A \cup B) \quad (1)$$

YOLOv8s is the default model because it balances accuracy and speed, and YOLOv8m is trained for comparison when GPU time allows. The model is initialised with COCO weights [22] and fine-tuned with the AdamW optimiser [23]. Transfer learning [24] is used because training from scratch needs far more images than a project can collect: the early layers already detect edges, wheels, windows and body shapes, and fine-tuning adapts the later layers to the camera angle and vehicle mix. Augmentation [25] by brightness, contrast, colour, scale and translation changes imitates different times of day and small camera shifts; vertical flips are not used, and mosaic augmentation helps with small and partly hidden vehicles. The planned configuration is listed in Table IV.

TABLE IV PLANNED TRAINING CONFIGURATION

Parameter	Planned configuration
Base model	YOLOv8s pretrained on COCO (YOLOv8m as comparison)
Input size	960 × 960 pixels (to retain small, far bays)
Epochs / patience	100 epochs, early stopping after 20 epochs without improvement
Batch size	16 (reduced if GPU memory is limited)
Optimiser	AdamW, initial learning rate 0.001, cosine decay
Augmentation	HSV shifts, brightness/contrast, horizontal flip, mosaic, small scale and translation
Confidence / NMS IoU	0.40 / 0.50 at inference
Selection metric	mAP@50 on the validation days

VI. MATHEMATICAL MODEL OF SLOT OCCUPANCY AND FORECASTING

A. Slot–Vehicle Overlap

The detector outputs vehicle boxes, but the product needs a decision per slot. For a slot S with polygon area A_S and a detected vehicle box B, the overlap ratio is the share of the slot covered by the box (Fig. 1):

$$r(S, B) = Area(S \cap B) / Area(S) \quad (2)$$

A slot is occupied in a frame if at least one vehicle box with confidence above the threshold has $r \geq \tau$ (default $\tau = 0.35$):

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511170

$$o_t(S) = 1 \text{ if } \max_B r(S, B) \geq \tau, \text{ else } 0 \quad (3)$$

Using the share of the slot rather than IoU makes the rule less sensitive to vehicle size: a motorcycle covers little of the bay, but a car parked slightly across a line still covers a lot of it.

B. Temporal Smoothing

A per-slot counter c_t grows while the raw state o_t differs from the stored state s_{t-1} and is reset when they agree; the stored state flips only when the counter reaches N (default N = 5 consecutive frames):

$$c_t = c_{t-1} + 1 \text{ if } o_t \neq s_{t-1}, \text{ else } 0; \quad s_t = 1 - s_{t-1} \text{ if } c_t \geq N \quad (4)$$

This prevents a car that only drives past a slot from changing its state. Slots with persistently low detection confidence are flagged as possibly obstructed.

C. Occupancy Rate and Status

With O occupied slots out of a total of T, the occupancy rate is and the dashboard maps it to a label: 0–50 % Available, 50–80 % Filling, 80–95 % *Nearly full* and 95–100 % Full.

$$\rho = 100 \cdot O / T \ (\%) \quad (5)$$

D. Occupancy Forecasting

The task is to estimate the number of free slots h minutes ahead, with $h \in \{15, 30, 60\}$. Slot states are aggregated into one value per time step (for example every 5 minutes), forming a series y_t, and a model f maps the feature vector x_t (Table V) to the forecast:

$$\hat{y}_{t+h} = f(x_t), \quad h \in \{15, 30, 60\} \text{ min} \quad (6)$$

TABLE V FEATURES USED FOR OCCUPANCY PREDICTION

Group	Feature	Description
Calendar	hour_sin, hour_cos	Cyclic encoding of the time of day
Calendar	day_of_week, is_weekend	Weekly pattern
Calendar	is_holiday, event_flag	Public holidays and known events
History	lag_1 … lag_12	Occupancy at the previous time steps
History	rolling_mean_1h, rolling_std_1h	Level and volatility over the last hour
History	same_time_last_week	Occupancy at the same time one week earlier
Optional	rain_flag	Weather indicator if available

Five forecasters are compared (Table VI). The persistence baseline predicts that occupancy stays as it is now, $\hat{y}_{t+h} = y_t$; the historical-average baseline predicts the mean of the same weekday and time slot in earlier weeks. A random forest [14] averages many decision trees, XGBoost [15] builds trees sequentially so that each corrects the previous ones, and an LSTM [17] reads a window of the last 24 steps. Models are tuned on the validation days only and scored once on the test days; a model is adopted only if it beats persistence on MAE. For the tree models, an approximate prediction interval is obtained from the spread of the individual tree predictions (random forest) or from quantile regression, so that the dashboard can show a range such as "6 ± 2 free slots".

TABLE VI FORECASTING MODELS AND PLANNED SETTINGS

Model	Type	Planned settings
Persistence	Baseline	y(t+h) = y(t)
Historical average	Baseline	Mean of same weekday and time slot
Random forest	Ensemble of trees	200 trees, maximum depth tuned on validation
XGBoost	Gradient boosting	300 rounds, learning rate 0.05, depth 6

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511170

Model	Type	Planned settings
LSTM	Recurrent network	1–2 layers, 64 units, window 24, Adam, early stopping

E. Evaluation Metrics

Detection and slot classification use precision, recall and F1, with AP computed at IoU 0.5 (mAP@50) following the VOC protocol [26]. Slot accuracy is the fraction of (slot, frame) pairs whose stored state equals the manually labelled state. Forecast quality uses, with n test samples,

$$MAE = (1/n)\,\Sigma\,|y - \hat{y}|, \quad RMSE = \sqrt{[(1/n)\,\Sigma\,(y - \hat{y})^2]} \tag{7}$$

VII. ALGORITHM AND LOGIC

Fig. 1 shows the complete processing logic. The first two rows run for every frame; the last row aggregates the slot states and refreshes the forecast on a schedule (for example every five minutes). The same logic is applied to uploaded images, uploaded videos and live camera frames. The steps are listed below.

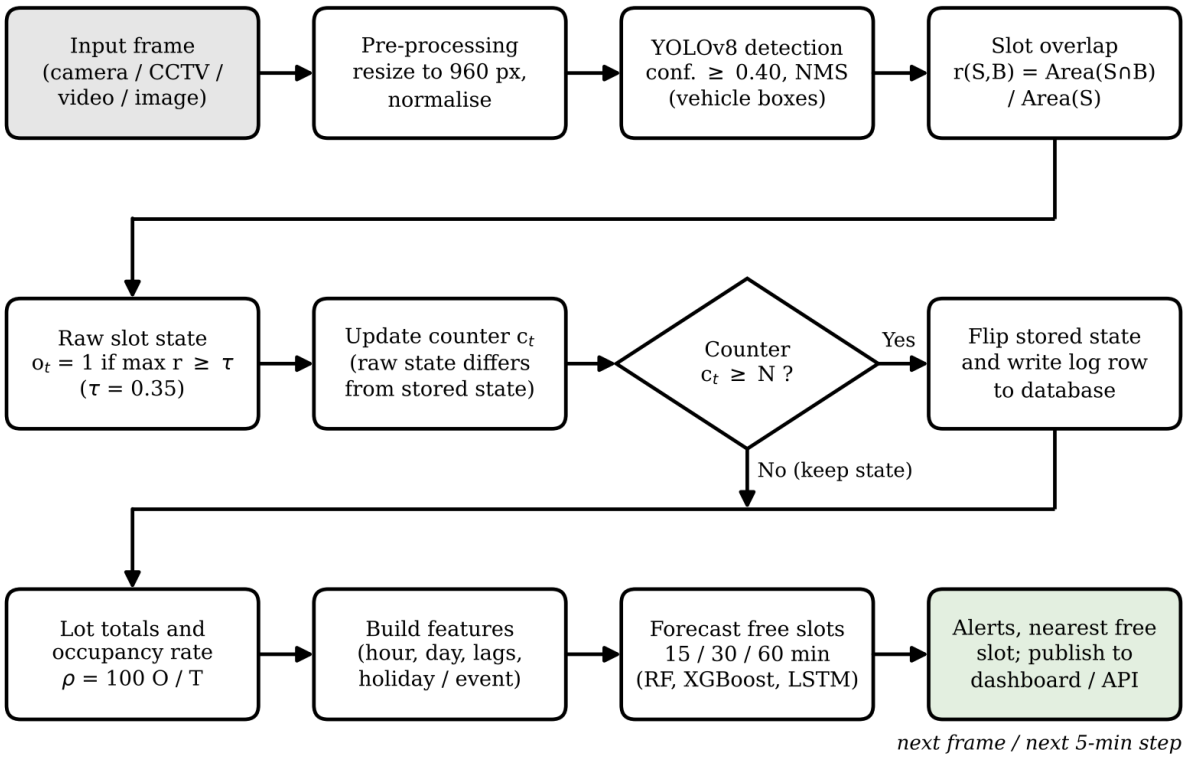


Fig. 1

Flowchart of the ParkSense processing logic: per-frame detection, slot decision and smoothing (rows 1 and 2), and aggregation, forecasting and publication (row 3).

1. Capture a frame from the camera (1–2 frames per second), resize it to the model input size and optionally denoise and normalise brightness.
2. Run YOLOv8 inference and keep vehicle boxes (car, motorcycle, bus, truck) with confidence ≥ 0.40 after NMS; drop boxes below a minimum area.
3. For each slot, compute r(S, B) for every box and take the maximum (Eq. 2).
4. Set the raw state to occupied if the maximum r ≥ τ, otherwise free (Eq. 3).
5. Update the slot counter; when it reaches N consecutive frames, change the stored state and write the change to the database (Eq. 4).

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511170

6. Compute the lot totals and the occupancy rate (Eq. 5), build the features of Table V, forecast the free slots for 15, 30 and 60 minutes (Eq. 6) and publish the state and forecast to the dashboard and API.

Two simple rule-based services use the slot states and the forecast. The nearest free slot service returns the free slot with the smallest walking distance to the entrance, using a distance value stored for every slot. The alert service compares the number of free slots and the 30-minute forecast with administrator-set thresholds and raises a message such as “Lot nearly full, expected full within 30 minutes”. No machine learning is used in these rules, which keeps their behaviour predictable and easy to explain. Table VII shows worked slot decisions.

TABLE VII WORKED SLOT-DECISION SCENARIOS

Scenario	Overlap r	Frames observed	Stored state
Car parked inside the bay	0.85	Stable	Occupied
Car stopped across the line	0.45	Stable	Occupied
Car passing through the lane	0.20 to 0.60	Fewer than N	Unchanged
Motorcycle in the bay	0.30	Stable	Free (below τ), occupied if τ is lowered
Bay empty, shadow only	0.05	Stable	Free
Vehicle leaves	0.00	≥ N frames	Free

VIII. IMPLEMENTATION

The back end is a FastAPI application. It loads the detection model and slot configuration at start-up, runs the capture worker as a background task and exposes REST endpoints for the lot list, current status, forecast with interval, occupancy history, image analysis, administrator slot configuration, model selection and login. Pydantic schemas validate requests and responses, and the generated OpenAPI page documents the API. Detection uses Ultralytics YOLOv8 with OpenCV [18]; forecasting uses scikit-learn [20], XGBoost and Keras. Training notebooks run on Google Colab, and the best weights are copied to the deployment machine, where the application is served by Uvicorn behind a reverse proxy.

The prototype uses SQLite through SQLAlchemy so that no server is needed; moving to PostgreSQL requires only a change of the connection string. Five tables are used: parking_lot, slot (polygon coordinates, type, distance to the entrance), occupancy_log (slot, timestamp, state, confidence), forecast (target time, predicted free slots, interval, model name) and user_account. Administrator passwords are hashed with bcrypt [27], login returns a signed token with a limited lifetime, administrator routes check the token and role, uploads are limited in size and type and decoded inside a try-block so that a corrupted file cannot crash the service, and cross-origin access is restricted to the dashboard origin.

The administrator dashboard (Figs. 2–4) is a web application whose AI Detection page offers Image, Video and Live modes, summary counters, a slot-by-slot status list with search and filters, and a report export. A separate user workspace shows live slot availability, find-and-book, recommendations, navigation and bookings at a fixed hourly rate, so that the detection output feeds the driver-facing map. After login, the administrator draws slot polygons by clicking on the corners of a reference frame and sets the overlap, smoothing and alert thresholds; changes are validated on the client and again on the server. Images are used only to find vehicle boxes: number plates and faces are not read and frames are not stored by default, so only slot states and aggregate counts are retained.

Testing is done at three levels: unit tests for the overlap computation, smoothing and feature building; API tests for the endpoints and their error handling; and scenario tests with recorded video for the whole pipeline. Representative cases cover a valid and an unsupported upload, a vehicle parked inside a bay, a vehicle passing for fewer than N frames, a vehicle leaving, a disconnected camera (lot shown offline with its last known state), a forecast with and without a trained model (a labelled baseline forecast is returned when no model exists), a wrong administrator password and the alert threshold. A health endpoint reports the age of the last processed frame so that monitoring can detect a stalled camera or worker.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511170

IX. RESULTS AND DISCUSSION

The prototype was demonstrated through the administrator dashboard in its three input modes: image analysis, video analysis and live stream. The video and live runs used a looping 1920 × 1080 test video of an aerial view of a parking lot with 396 configured slots (a test source, not real CCTV footage), and the image run used a single aerial photograph. Fig. 2 shows the live mode, Fig. 3 the image analysis and Fig. 4 the video analysis.

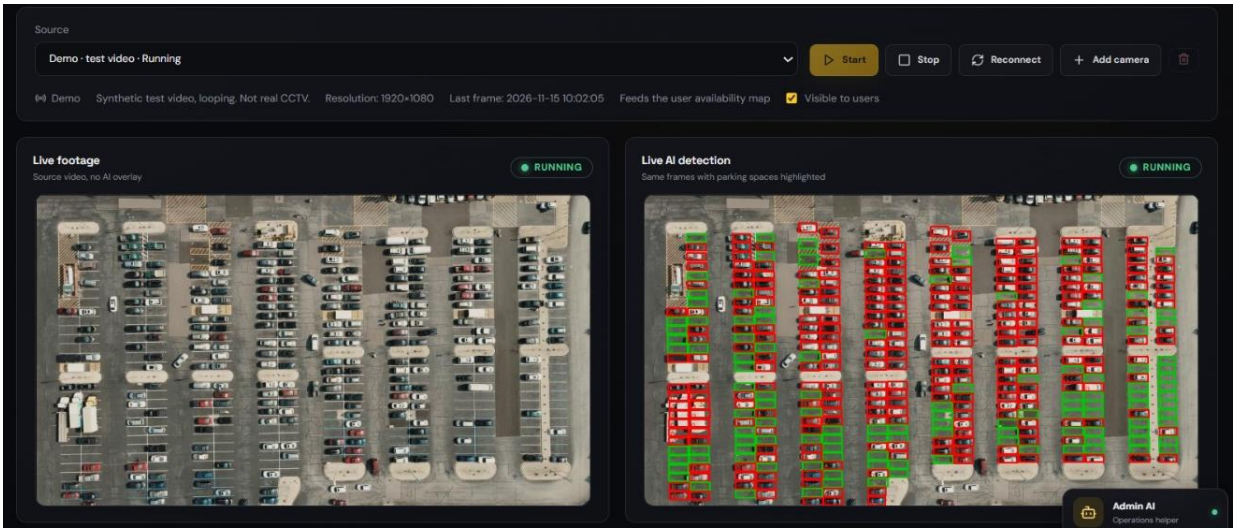


Fig. 2

Live mode of the AI Detection page: the source video (left) and the same frames with parking spaces highlighted (right); occupied slots are outlined in red and vacant slots in green.

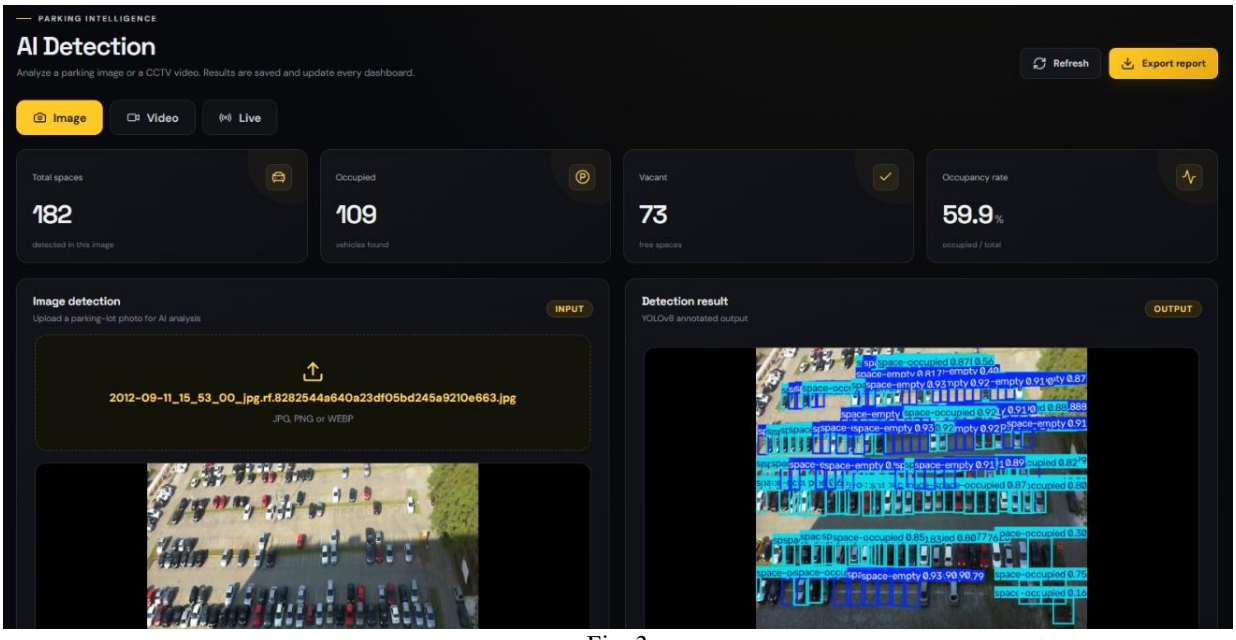


Fig. 3

Image analysis: counters for detected spaces, occupied, vacant and occupancy rate, the uploaded photograph (input) and the annotated output with class labels and confidence scores.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511170

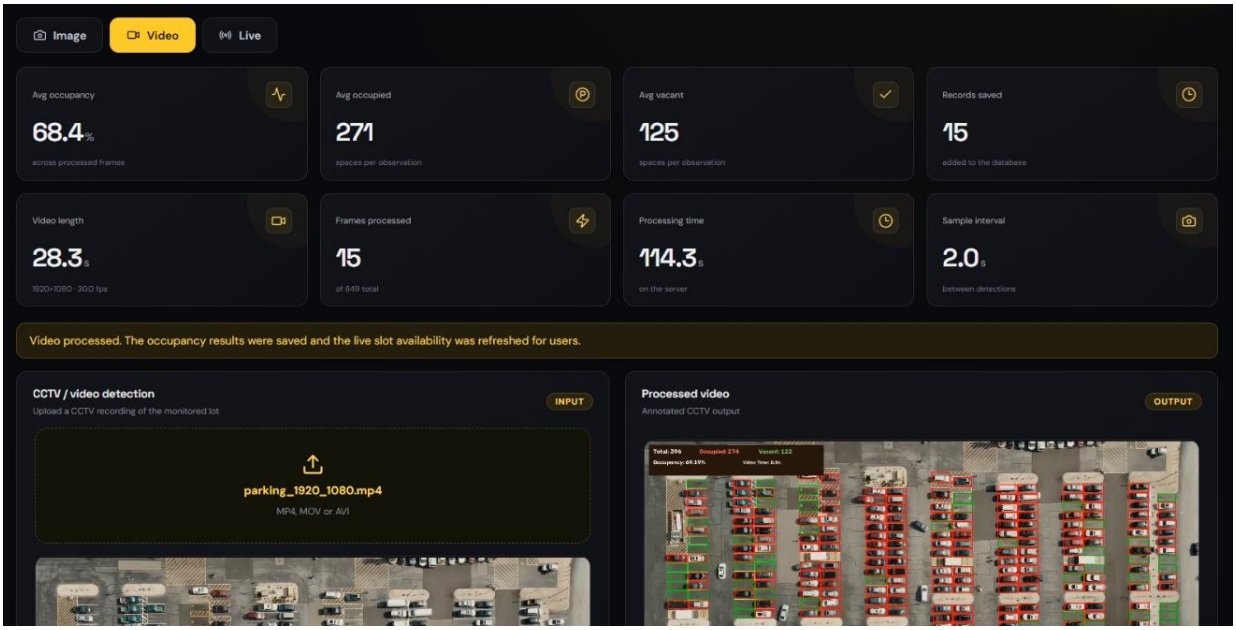


Fig. 4

Video analysis: summary counters (average occupancy, records saved, frames processed, processing time) and the annotated output video with slot boxes and an overlay of total, occupied and vacant counts.

Table VIII lists the values read from the outputs of the three runs.

TABLE VIII OBSERVED OUTPUTS OF THE DEMONSTRATION RUNS

Item	Image analysis	Video analysis	Live stream
Input	One aerial parking-lot photograph	28.3 s video, 1920 × 1080, 30 fps, 849 frames	Looping test video, 1920 × 1080
Total spaces	182 detected	396	396
Occupied	109	271 (average)	271
Vacant	73	125 (average)	125
Occupancy	59.9 %	68.4 % (average)	68.4 %
Frames processed	—	15 of 849 (every 2.0 s)	500 received, 11 inferences, 0.3 per s
Processing time	—	114.3 s on the server (about 7.6 s per sampled frame)	0 reconnects
Records saved	—	15 rows added to the database	State feeds the user availability map

The following observations can be made from these outputs.

- **Slot states are localised and shown on the frames.** In the live and video modes every slot is outlined in red (occupied) or green (vacant), and in the image mode each slot carries a label (space-empty or space-occupied) with a confidence score, most of the visible scores lying between about 0.8 and 0.93.
- **The three modes agree.** The live and the video runs report the same average counts (271 occupied, 125 vacant, 68.4 %), and the overlay on one video frame (274 occupied, 122 vacant, 69.19 %) differs from the average by only a few slots, which is the frame-to-frame variation expected from a changing scene.
- **The driver-facing view follows the detector.** The user workspace shows the same slot states (for example 127 available and 269 occupied, 67.93 %, from a later update) and, when the camera is stopped, displays the last known status with a notice, as intended in the error-handling design.

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511170

- **Throughput is modest but sufficient.** The server completed about 0.3 classifications per second, so the display is updated every few seconds rather than at video rate; since parking states change slowly, sampling every 2 s is adequate for guidance.
- **Dense scenes are visually cluttered.** In the image output the labels of neighbouring slots overlap, and a few low-confidence labels (0.16–0.30) are visible for small or partly hidden spaces.

Slot errors affect what the driver sees in a predictable way. For a lot with 20 slots of which 11 are truly occupied (55 %), Table IX shows the effect of different numbers of errors on the displayed free count and status label. The values follow directly from the definitions of Section VI and need no experiment.

TABLE IX EFFECT OF SLOT ERRORS ON DISPLAYED COUNTS (20-SLOT LOT, 11 OCCUPIED)

Error type	Displayed free	Displayed occupancy	Status label
No errors	9	55 %	Filling
1 occupied slot read as free	10	50 %	Available
2 occupied slots read as free	11	45 %	Available
1 free slot read as occupied	8	60 %	Filling
2 free slots read as occupied	7	65 %	Filling
1 missed + 1 false (cancel out)	9	55 %	Filling

A small number of slot errors changes the displayed count by the same small number and rarely changes the status label, which is one reason why a slot accuracy of about 95 % is acceptable for a guidance application. Compared with the prior work reviewed in Section II, the design uses a single fixed camera for many bays instead of per-bay sensors or entrance counters, decides the bay from detection and slot geometry instead of manually cropped bay images, smooths states over time, integrates forecasting with baselines and intervals, and uses a chronological split. The comparison is qualitative and no numerical claim of superiority is made.

X. LIMITATIONS AND FUTURE WORK

The evaluation presented here is observational. The demonstrations used a looping test video and a single photograph; no ground-truth annotation, precision, recall, mAP or slot-accuracy figures are reported, and the forecasting models have not yet been scored on held-out days. Results from one camera may not transfer to another viewpoint, a few weeks of data cannot capture seasonal changes such as examination periods or festivals, and manual labels of partly hidden vehicles are subjective. Accuracy decreases for heavily occluded bays and in very low light without supplementary lighting; forecasts describe the aggregate and cannot foresee unannounced events; the overlap rule assumes bays are at least partly visible; occupancy is detected for any vehicle without checking authorisation; and performance on a different lot may need retraining.

Future work will (i) record precision, recall, mAP@50, slot accuracy and forecasting error against the persistence baseline on held-out days of real camera footage, stratified by time of day, weather, occupancy level and day type, and run a cross-camera check on PKLot-style data [3], (ii) support multi-camera and multi-level lots, (iii) add multi-object tracking such as SORT [28] to measure dwell time, (iv) deploy a quantised YOLO model on an edge device at the camera, (v) enrich forecasts with weather, class timetables and event calendars, (vi) add a mobile application with navigation to a predicted free slot, and (vii) detect camera shift automatically and prompt for recalibration.

XI. CONCLUSION

A camera-based parking space detection and prediction system was presented. It detects vehicles with a fine-tuned YOLO model, decides the state of each configured slot with an overlap rule, stabilises the result by temporal smoothing and stores the history for forecasting. Baseline, tree-based and recurrent models predict the number of free slots for the next 15, 30 and 60 minutes, and a web dashboard and REST interface make the live state, forecast, nearest free slot and alerts available to drivers and administrators. The main contributions are the combination of detection, smoothing and forecasting in one deployable pipeline that uses existing cameras, a chronological, leakage-aware evaluation protocol with honest baselines, and a modular design in which the detector, the slot logic and the forecaster can each be replaced.

 This work is licensed under a Creative Commons Attribution 4.0 International License

DOI: 10.17148/IJARCCE.2026.1511170

The approach offers a low-cost route to giving drivers more than a count: a per-slot map and a short-term outlook. Its limits are equally clear: accuracy depends on camera placement and lighting, and forecasts describe patterns rather than individual behaviour.

ACKNOWLEDGMENT

The authors thank their guide, Prof. Kaminee S. Patil, the Project Coordinator, Dr. P. S. Sanjekar, the Head of the Department, Prof. Dr. U. M. Patil, and the Director, Prof. Dr. J. B. Patil, and the Department of Computer Science & Engineering (Data Science), R. C. Patel Institute of Technology, Shirpur, for guidance and facilities.

REFERENCES

[1]. D. C. Shoup, *The High Cost of Free Parking*. Chicago, IL, USA: APA Planners Press, 2005.
[2]. J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You only look once: Unified, real-time object detection", in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pp. 779-788, 2016.
[3]. P. R. L. de Almeida, L. S. Oliveira, A. S. Britto Jr., E. J. Silva Jr. and A. L. Koerich, "PKLot – A robust dataset for parking lot classification", *Expert Systems with Applications*, vol. 42, no. 11, pp. 4937-4949, 2015.
[4]. G. Amato, F. Carrara, F. Falchi, C. Gennaro, C. Meghini and C. Vairo, "Deep learning for decentralized parking lot occupancy detection", *Expert Systems with Applications*, vol. 72, pp. 327-334, 2017.
[5]. F. Caicedo, C. Blazquez and P. Miranda, "Prediction of parking space availability in real time", *Expert Systems with Applications*, vol. 39, no. 8, pp. 7281-7290, 2012.
[6]. Y. Zheng, S. Rajasegarar and C. Leckie, "Parking availability prediction for sensor-enabled car parks in smart cities", in *Proc. IEEE Int. Conf. Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pp. 1-6, 2015.
[7]. R. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation", in *Proc. IEEE CVPR*, pp. 580-587, 2014.
[8]. S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137-1149, 2017.
[9]. W. Liu et al., "SSD: Single shot multibox detector", in *Proc. European Conf. Computer Vision (ECCV)*, pp. 21-37, 2016.
[10]. C.-Y. Wang, A. Bochkovskiy and H.-Y. M. Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors", in *Proc. IEEE/CVF CVPR*, pp. 7464-7475, 2023.
[11]. G. Jocher, A. Chaurasia and J. Qiu, "Ultralytics YOLOv8", 2023. [Online]. Available: https://github.com/ultralytics/ultralytics
[12]. G. E. P. Box, G. M. Jenkins, G. C. Reinsel and G. M. Ljung, *Time Series Analysis: Forecasting and Control*, 5th ed. Hoboken, NJ, USA: Wiley, 2015.
[13]. R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*, 3rd ed. Melbourne, Australia: OTexts, 2021.
[14]. L. Breiman, "Random forests", *Machine Learning*, vol. 45, no. 1, pp. 5-32, 2001.
[15]. T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system", in *Proc. ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, pp. 785-794, 2016.
[16]. J. H. Friedman, "Greedy function approximation: A gradient boosting machine", *Annals of Statistics*, vol. 29, no. 5, pp. 1189-1232, 2001.
[17]. S. Hochreiter and J. Schmidhuber, "Long short-term memory", *Neural Computation*, vol. 9, no. 8, pp. 1735-1780, 1997.
[18]. G. Bradski, "The OpenCV Library", *Dr. Dobb's Journal of Software Tools*, 2000.
[19]. R. T. Fielding, "Architectural styles and the design of network-based software architectures", Ph.D. dissertation, Univ. of California, Irvine, 2000.
[20]. F. Pedregosa et al., "Scikit-learn: Machine learning in Python", *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011.
[21]. A. Neubeck and L. Van Gool, "Efficient non-maximum suppression", in *Proc. Int. Conf. Pattern Recognition (ICPR)*, vol. 3, pp. 850-855, 2006.
[22]. T.-Y. Lin et al., "Microsoft COCO: Common objects in context", in *Proc. ECCV*, pp. 740-755, 2014.
[23]. D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", in *Proc. Int. Conf. Learning Representations (ICLR)*, 2015.
[24]. S. J. Pan and Q. Yang, "A survey on transfer learning", *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345-1359, 2010.

 This work is licensed under a Creative Commons Attribution 4.0 International License



International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026
DOI: 10.17148/IJARCCE.2026.1511170

[25]. C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning", *Journal of Big Data*, vol. 6, Art. 60, 2019.
[26]. M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn and A. Zisserman, "The PASCAL Visual Object Classes (VOC) challenge", *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303-338, 2010.
[27]. N. Provos and D. Mazières, "A future-adaptable password scheme", in *Proc. USENIX Annual Technical Conf. (FREENIX Track)*, pp. 81-91, 1999.
[28]. A. Bewley, Z. Ge, L. Ott, F. Ramos and B. Upcroft, "Simple online and realtime tracking", in *Proc. IEEE Int. Conf. Image Processing (ICIP)*, pp. 3464-3468, 2016.