Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.084246

ARTICLE

Edge-Optimized Automatic Number Plate Recognition for IoT-Based Smart Parking Using YOLOv8

Mohammad Ebrahimishadman and Alireza Souri*

Department of Computer Engineering, Haliç University, Istanbul, Turkey

*Corresponding Author: Alireza Souri. Email: alirezasouri@halic.edu.tr

Received: 18 April 2026; Accepted: 11 June 2026; Published: 23 July 2026

ABSTRACT: Automatic Number Plate Recognition (ANPR) is widely used in Intelligent Transportation Systems (ITS) and smart parking applications, but running deep learning-based ANPR directly on low-power edge devices remains difficult because of computation time, memory, and latency limitations. In this study, we develop an edge-oriented ANPR pipeline for an Internet of Things (IoT)-based sensor-triggered stop-and-go smart parking platform, targeting deployment on a resource-constrained edge device. The pipeline combines YOLOv8 for license plate detection, PaddleOCR for text recognition, and a rule-based normalization stage to reduce Optical Character Recognition (OCR) errors caused by spacing inconsistencies and plate-format variations. In the OCR-only ablation study conducted on cropped plate images, PaddleOCR outperformed the other OCR options evaluated, achieving up to 96.0% exact-match accuracy and 98.78% character-level accuracy, with an average OCR-only processing time of 52.55 ms per image. When evaluated as a complete end-to-end pipeline on a Raspberry Pi with ONNX Runtime, the system achieved 83.5% exact-match accuracy and 94.83% character-level accuracy, with an average end-to-end latency of 1713 ms per image, indicating that edge-side operation is feasible for sensor-triggered parking entry and exit events despite CPU-only hardware constraints. In addition to the ANPR module, the proposed platform connects edge devices with Firebase services and Flutter-based user applications for parking status updates, user interaction, reservation matching, and access logs. These results show that a low-cost edge-based ANPR architecture can support practical sensor-triggered smart parking operations without depending on continuous cloud-side inference.

KEYWORDS: Automatic number plate recognition (ANPR); edge computing; Internet of Things (IoT); PaddleOCR; Raspberry Pi; smart parking systems; YOLOv8

1 Introduction

In smart cities, parking management remains a practical challenge in dense urban environments, especially in places such as universities, hospitals, and shopping centers where demand changes throughout the day. In such settings, drivers may spend several minutes searching for a free parking space, which increases local congestion, wastes fuel, and contributes to emissions. The broader role of the Internet of Things (IoT) in smart-city infrastructure and the growing importance of parking services in urban mobility have been widely discussed in recent studies [1–3]. One practical way to improve parking operations is to integrate smart parking software with Automatic Number Plate Recognition (ANPR). In such a setting, entry and exit events can be logged automatically, parking duration can be tracked, and access decisions can be made without manual intervention. Smart parking platforms have also been investigated not only for availability monitoring, but also for reservation management and compliance improvement [4,5]. However,

such automation requires recognition pipelines that are both accurate and efficient on limited parking-side hardware. Most ANPR systems still follow a two-stage pipeline: first, a detector localizes the plate, and then an Optical Character Recognition (OCR) module reads the characters. This design is practical and widely used in ITS, but its deployment cost depends heavily on where inference is executed. Recent deep learning-based vehicle and plate recognition systems have demonstrated strong performance under real traffic conditions, yet they are often developed for more capable platforms or broader vehicle-analysis tasks rather than lightweight parking-side deployment [6]. This trade-off changes when inference is moved to the edge. Instead of sending frames continuously to a server, the device can process data locally and transmit only the required events or records. This approach reduces bandwidth usage, improves deployment flexibility, and is especially useful in places where connectivity is unstable, such as underground parking areas or enclosed facilities. However, deploying ANPR on small edge devices remains difficult. Many studies report good recognition performance, but often on hardware that is more powerful than what would be acceptable for a low-cost parking installation. Raspberry Pi-class devices remain attractive in practice, yet high-frame-rate continuous inference on such platforms is still challenging in IoT environments. Another limitation is that many ANPR studies stop at detection and recognition, without showing how the module fits into a working parking system or how it handles country-specific plate formats such as those used in Turkey. Taken together, these limitations point to the need for an ANPR framework that is not only accurate, but also deployable on low-cost CPU-only edge hardware and integrable with a complete smart parking workflow. In this study, we propose an edge-based ANPR pipeline for low-cost devices operating in IoT-based smart parking environments. The proposed pipeline combines YOLOv8 for plate detection, PaddleOCR for text recognition, and a rule-based normalization stage that enforces Turkish plate structure. To support CPU-only edge deployment, the detector is executed through ONNX Runtime. In our implementation, the ANPR pipeline runs locally on Raspberry Pi devices, while Firebase services are used for synchronization, logging, and user-side interaction. Multiple edge nodes and ESP32-based sensor units can operate independently while exchanging events through the backend, enabling scalable deployment without centralized continuous image processing.

The main contributions of this paper are summarized as follows:

- We propose an event-driven ANPR pipeline for low-cost edge devices that reduces dependence on cloud-side inference during normal operation in an IoT-based parking platform.
- We develop a Turkish plate-oriented recognition pipeline that combines YOLOv8 detection, PaddleOCR, and rule-based normalization.
- We present a comparative ablation study of OCR engines and processing strategies, including normalization and retry mechanisms, and analyze their effects on accuracy and latency.
- We demonstrate end-to-end deployment of the ANPR pipeline on Raspberry Pi hardware using ONNX Runtime, showing that model-format optimization is an important practical factor for CPU-only edge inference.
- We integrate the ANPR module into an IoT-based smart parking workflow with Firebase backend services and Flutter-based user applications.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 presents the system architecture. Section 4 describes the methodology and ANPR pipeline. Section 5 reports the experimental results. Section 6 explains the smart parking system in the IoT context. Finally, Section 7 concludes the paper.

2 Related Work

Recent ANPR research has increasingly combined deep detection, text recognition, and embedded AI to support intelligent transportation and smart parking applications. This section reviews prior work from four perspectives: license plate detection and recognition methods, edge-based ANPR systems, smart parking systems using computer vision and IoT, and the remaining research gap.

2.1 License Plate Detection and Recognition Methods

License plate detection has evolved from handcrafted image processing pipelines to deep one-stage detectors that are robust under viewpoint, illumination, and background variation. The YOLO family played a major role in this transition by enabling fast end-to-end object detection suitable for transportation applications [7–9]. For example, Laroca et al. showed that YOLO-based detection can provide robust real-time license plate localization in unconstrained environments, while later work extended this idea toward layout-independent ANPR by detecting both plates and characters [10,11]. More recent studies also confirm the effectiveness of modern YOLO variants when combined with OCR-based recognition modules [12,13]. After localization, the recognition stage must decode the plate reliably despite blur, skew, low resolution, and regional formatting differences. Earlier deep learning-based pipelines often retained explicit character segmentation, which preserved interpretability but remained sensitive to segmentation errors [14,15]. To address this limitation, segmentation-free recognition models such as LPRNet and Fast-LPRNet were introduced to predict the full plate string directly from the cropped plate image [16,17]. In parallel, practical OCR engines have also become attractive in modular ANPR pipelines. PP-OCR, for example, provides a lightweight OCR framework suitable for embedded applications [18]. Compared with end-to-end recognition models, OCR-based pipelines are often easier to adapt to new plate formats and easier to debug in deployment, although they may require stronger preprocessing and post-processing. Thus, prior work indicates a trade-off between end-to-end efficiency and the adaptability of detector-plus-OCR pipelines [19,20].

2.2 Edge-Based ANPR Systems

Although high ANPR accuracy has been widely reported, many systems are still evaluated on desktop GPUs or cloud-connected servers rather than low-power embedded devices. This has motivated growing interest in edge-based ANPR, where inference is performed closer to the camera to reduce latency, bandwidth use, and privacy risks. Early low-cost efforts, such as the Raspberry Pi-based system of Fakhar et al., demonstrated that embedded ANPR is feasible, but also highlighted the limitations of classical OpenCV-plus-OCR pipelines under constrained hardware conditions [21]. More recent edge-oriented systems rely on lightweight deep models. Lin et al. deployed an edge-AI ANPR system on Jetson AGX Xavier using YOLOv4-Tiny and a modified recognition network, while Sonnara et al. proposed Light-Edge for Jetson Nano using a unified detection-and-recognition architecture [22,23]. These studies show that embedded ANPR is increasingly practical, but many reported systems still rely on GPU-equipped boards rather than truly CPU-constrained IoT-class devices. Efficient OCR frameworks further show that edge deployment depends on computational cost as much as recognition accuracy [18]. However, relatively fewer studies present complete ANPR deployment on low-cost devices while also integrating backend verification and application-level system functions [24].

2.3 Smart Parking Systems Using Computer Vision and IoT

Smart parking research has traditionally focused more on occupancy estimation than on vehicle identification. Vision-based parking systems often detect vehicles or classify parking slots to estimate availability,

then integrate those results into mobile or web services through IoT infrastructure. Qin et al. proposed an edge-AI parking surveillance system for real-time occupancy monitoring, while Nithya et al. used computer vision with Faster R-CNN and YOLOv3 for parking-space detection in an IoT-based setting [25,26]. These works demonstrate the value of combining vision and IoT for parking management, but they mainly address whether a space is occupied rather than who occupies it. Other studies extend parking systems toward guidance, environmental robustness, or access control. Rajesh et al. combined computer vision, IoT, and edge computing for intelligent parking guidance, while Satyanath et al. highlighted the effect of degraded visibility on parking-space detection [27,28]. Neupane et al. further expanded the scope toward policy-aware parking management by detecting vehicles, license plates, and disability badges in an accessible parking system [29]. These studies show a shift from passive availability sensing toward more policy-aware parking systems. Even so, comparatively fewer works combine real-time ANPR, backend verification, and parking management in a unified edge-oriented workflow.

2.4 Research Gap and Motivation

The reviewed literature reveals several consistent gaps. First, although modern YOLO-based detectors provide strong plate localization, reliable end-to-end ANPR still depends heavily on the robustness of the recognition stage and on practical deployment conditions [10,12]. Second, existing recognition approaches involve a trade-off between efficient end-to-end models and more flexible OCR-based pipelines, but few studies analyze this trade-off in a parking-oriented edge deployment scenario [16,18,19]. Third, much of the edge ANPR literature still targets comparatively powerful embedded GPUs such as Jetson platforms, whereas low-cost IoT-class devices impose stricter CPU, memory, and thermal limits [22,23]. Finally, the smart parking literature has largely emphasized occupancy detection, guidance, or isolated IoT integration rather than tightly combining ANPR, backend verification, and parking management in one cohesive edge-ready system [25,27]. These limitations motivate the need for a system that is not only accurate in laboratory settings but also deployable as a complete operational pipeline. Accordingly, our approach combines event-driven license plate detection, lightweight text recognition, and smart parking integration within an edge-based architecture tailored to practical deployment constraints. The contribution therefore lies not only in the use of modern deep learning models, but also in the full-system integration of ANPR with parking-management functions such as entry logging, reservation matching, and IoT-connected decision support.

2.5 Comparison with Representative Prior Systems

Table 1 compares the proposed system with representative prior works in terms of edge hardware, CPU-only deployment, parking integration, and backend support.

Table 1: Comparison of the proposed system with representative prior works.

System	Hardware	CPU-Only	Parking Integration	Backend
Lin et al. [22]	Jetson AGX Xavier	No	No	No
Sonnara et al. [23]	Jetson Nano	No	No	No
Rajesh et al. [27]	Edge + IoT	N/A	Yes (guidance)	Cloud
Fakhar et al. [21]	Raspberry Pi	Yes	No	No
Proposed	Raspberry Pi 4	Yes	Yes: ANPR + access	Firebase

As shown in Table 1, prior edge-ANPR studies mainly use GPU-equipped Jetson boards or Raspberry Pi-based ANPR without full parking-backend integration. In contrast, the proposed system combines CPU-only Raspberry Pi deployment, a YOLOv8–PaddleOCR ANPR pipeline, Turkish plate normalization, Firebase-based verification, access logging, and Flutter-based user applications.

3 Architecture

This section outlines the architecture of the proposed edge-based ANPR system and its integration into the smart parking platform.

3.1 Overall System Architecture

The proposed system is composed of four layers: the edge device layer, the cloud backend layer, the application layer, and the communication layer. At the edge, a Raspberry Pi performs local ANPR using camera input. In the event-driven configuration, an infrared sensor triggers processing only when a vehicle is present. In extended setups, an ESP32 can support sensor and barrier control. The backend relies on Firebase for authentication, storage, event logging, synchronization, and access validation. After recognition, plate data are transmitted to the backend and checked against registered vehicles, reservations, and parking rules. The application layer provides Flutter-based mobile and web interfaces for vehicle registration, reservation management, availability checking, and administrative monitoring. Edge-to-cloud communication is achieved through lightweight protocols such as HTTP or MQTT. As illustrated in Fig. 1, the architecture places latency-sensitive ANPR tasks at the edge and uses the cloud mainly for coordination and service management.

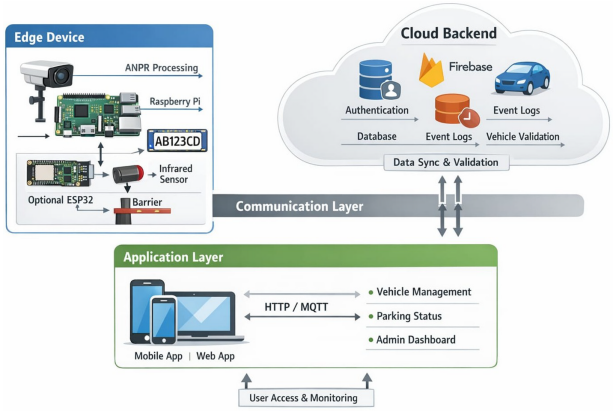

Figure 1: Overall architecture of the proposed smart parking system.

3.2 ANPR Pipeline within the Architecture

The ANPR pipeline is event-driven. Instead of analyzing frames continuously, the system starts processing only after a vehicle trigger is received. This reduces unnecessary computation on embedded hardware. Once triggered, the camera captures an image and the YOLOv8 detector localizes the license plate region. The cropped plate is then passed to PaddleOCR for text recognition. A rule-based normalization step is applied afterward to correct common OCR inconsistencies according to Turkish plate structure. The recognized plate is then checked against Firestore records to determine whether the vehicle is registered, associated with an active reservation, or should simply be logged as an external entry. Based on this result, the system can update parking records, generate notifications, or trigger access-control actions.

3.3 Design Considerations

Since the target hardware is a Raspberry Pi, the design prioritizes low-latency inference and controlled CPU usage. For this reason, the YOLOv8 detector is exported to ONNX and executed with ONNX Runtime [30], which is more suitable than standard PyTorch inference on CPU-only edge devices. The pipeline is intentionally divided into detection, OCR, and normalization stages instead of using a single end-to-end model. This makes the system easier to debug, easier to adapt to Turkish license plate rules, and easier to update when one stage changes. A second practical design choice is sensor-based triggering. By avoiding continuous processing, the system reduces both computation and energy use, which is important for long-running embedded deployments.

3.4 Modularity and Deployment

Each module in the system has a separate role, including plate detection, text recognition, normalization, backend communication, and user interaction. This separation makes the system easier to maintain and allows individual modules to be replaced without redesigning the full platform. The same structure also supports distributed deployment. Multiple Raspberry Pi nodes can run locally in different parking areas, while ESP32-based sensor units provide auxiliary trigger or control signals. All nodes remain synchronized through the backend, which makes the platform easier to expand across larger parking sites.

4 Methodology

4.1 Overview of the Method

The ANPR pipeline is organized as an event-driven sequence for execution on a resource-constrained edge device. After a hardware trigger is received, the system performs plate detection, region extraction, OCR, rule-based normalization, and database verification. The implementation runs on a Raspberry Pi and is designed to limit computation while remaining usable under common field conditions such as lighting variation, slight motion blur, and partial occlusion.

As shown in Fig. 2, each triggered image passes through detection, OCR, and normalization before the final plate string is generated.

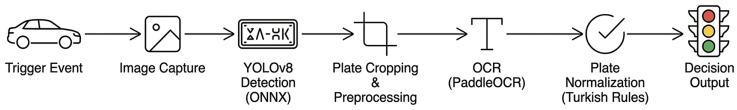

Figure 2: Overview of the proposed ANPR pipeline.

4.2 License Plate Detection

License plate detection is performed using the YOLOv8 object detection framework [31]. A custom model was trained on a Turkish license plate dataset consisting of 3413 training images, 975 validation images, and 1554 test images, annotated in YOLO format [32]. The YOLOv8n variant was selected because it offered a more practical speed–accuracy trade-off for Raspberry Pi deployment than heavier detector variants. The model contains approximately 3 million parameters and requires 8.2 GFLOPs. Training was performed with an input resolution of 640 × 640 pixels for 100 epochs using stochastic gradient descent. The trained detector achieved 94.7% precision, 91.5% recall, and an mAP of 96.8% at IoU = 0.5 on the validation set during model development, indicating strong localization performance on Turkish plate images. These validation metrics were used for training-stage assessment and model selection, while the final held-out test performance is reported separately in Section 5.3.

4.3 OCR-Based Text Recognition

Text recognition is performed with PaddleOCR [33]. In our tests, it handled noisy crops, low-resolution plate regions, and uneven lighting more consistently than the alternative OCR options we evaluated. Even when detection is correct, OCR quality still depends heavily on the crop. Small shifts in the bounding box, reflections, or tilted plates can make recognition unstable. For that reason, the cropped plate is preprocessed using grayscale conversion, contrast enhancement, and conditional upsampling before OCR is applied. As illustrated in Fig. 3, the proposed ANPR pipeline operates in two sequential stages. In the first stage, YOLOv8n predicts the bounding box of the license plate region and extracts the corresponding crop. In the second stage, the cropped and preprocessed plate is passed to PaddleOCR, which produces a raw character string. This output is then forwarded to the rule-based normalization module described in Section 4.4, which enforces Turkish plate structure and reduces common character ambiguities before the result is sent to Firebase for verification.

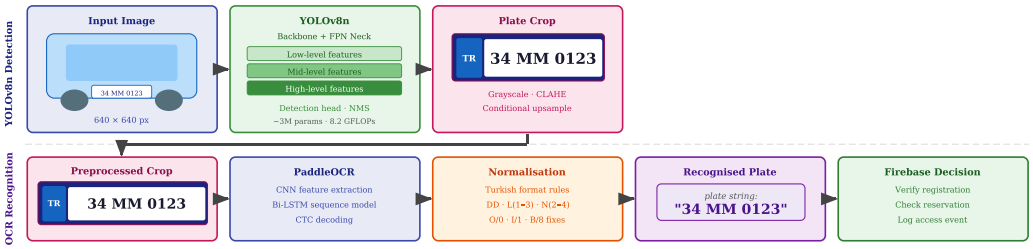

Figure 3: Overview of the proposed two-stage ANPR pipeline. YOLOv8n performs license plate detection and crop extraction, while PaddleOCR and rule-based normalization generate the final plate string before backend verification.

4.4 Plate Normalization

To improve recognition reliability, a rule-based normalization module is applied to the raw OCR output. This module encodes the structural constraints of Turkish license plates, which generally follow the format

$$DD\ L(1-3)\ N(2-4)$$

where DD denotes the two-digit province code (01–81), L denotes one to three alphabetic characters, and N denotes two to four numeric digits. Candidate plate strings are generated from the raw OCR output in three steps. First, spaces and non-alphanumeric noise characters are removed to produce a compact string. Second, multiple letter–digit split boundaries are tested to account for segmentation ambiguity in the OCR output. Third, position-aware character substitutions are applied separately to the letter block and the digit block. In the letter block, digit-like characters are corrected to visually similar letters, such as 0→O, 1→I, 5→S, 8→B, and 6→G. In the digit block, letter-like characters are corrected to visually similar digits, such as O→0, I→1, S→5, B→8, and L→1. Letters that are not used in Turkish license plates, such as Q, W, and X, are excluded from valid letter-block candidates.

The normalization stage evaluates the generated candidate plate strings and retains the one with the highest structural compatibility score:

$$S(c) = S_{\text{letter}}(c) + S_{\text{tail}}(c) + S_{\text{prior}}(c) + S_{\text{penalty}}(c) \tag{1}$$

where c denotes a candidate plate string, $S_{\text{letter}}(c)$ rewards valid letters and safe digit-to-letter corrections in the letter block, $S_{\text{tail}}(c)$ rewards valid numeric endings, especially three- and four-digit tails, $S_{\text{prior}}(c)$ gives a small preference to common Turkish plate layouts such as three-letter and two-letter middle blocks, and $S_{\text{penalty}}(c)$ penalizes suspicious split boundaries and risky O/0-like transitions near the letter–digit boundary.

In practice, the module resolves common OCR confusions such as O→0, I→1, and B→8 when characters appear in numeric positions. It also checks multiple segmentation boundaries and retains the candidate that best matches Turkish plate structure. When the blue "TR" band on the left side of the plate is likely to interfere with recognition, the retry logic can repeat OCR on a left-trimmed crop and compare the resulting candidate using the same normalization rules. This normalization stage improves the consistency of the final plate string and reduces common OCR errors in Turkish license plates.

4.5 Edge Inference Optimization

To run the detector efficiently on Raspberry Pi hardware, the YOLOv8 model is exported from PyTorch to ONNX and executed with ONNX Runtime [30], which is more suitable for CPU-based inference. Several practical optimizations are used to reduce overhead. First, the system runs in a persistent loop so that the models are loaded only once. Second, detection is executed at configurable intervals rather than on every frame. If detection is performed once every k frames, the effective detector invocation rate becomes

$$f_{\text{det}} = \frac{f_{\text{in}}}{k} \tag{2}$$

where f_{in} is the input frame rate and f_{det} is the detector execution rate. Third, image upsampling is applied only when the plate crop is too small for stable OCR. The average per-trigger processing time of the edge pipeline can therefore be expressed as

$$T_{\text{edge}} = T_{\text{cap}} + T_{\text{det}} + T_{\text{pre}} + T_{\text{ocr}} + T_{\text{norm}} + T_{\text{db}} \tag{3}$$

where T_{cap}, T_{det}, T_{pre}, T_{ocr}, T_{norm}, and T_{db} denote the average time spent in image capture, detection, crop extraction and preprocessing, OCR, normalization, and database verification, respectively. Since model loading is performed only once in the persistent loop, T_{load} is treated as an initialization cost and is not included in the recurring per-trigger latency. Table 2 provides the measured per-stage latency breakdown of the Raspberry Pi pipeline. The values are reported as mean latency per triggered image and correspond to the same edge configuration summarized in Table 3. Image capture latency was excluded from the profiling because the evaluation script operated on stored test images rather than a continuously sampled live camera stream. Crop extraction and rule-based normalization contributed less than 1 ms of overhead compared with detection and OCR, while retry OCR was not triggered on this evaluation set.

Table 2: Per-stage latency breakdown of the ANPR pipeline on Raspberry Pi (mean ms per triggered image).

Pipeline Stage	Mean Latency (ms)
Image capture	Excluded
YOLOv8 detection (ONNX)	604
Crop extraction and preprocessing	<1
PaddleOCR recognition	997
Retry OCR, if triggered	<1
Rule-based normalization	<1
Firebase verification/logging	157
Total pipeline	1758

Table 3: Hardware and software configuration for edge evaluation.

Component	Configuration
Edge device	Raspberry Pi 4 Model B
RAM	8 GB
Operating system	Debian GNU/Linux 12 (bookworm)
Python version	3.11.2
ONNX Runtime	1.24.4
PaddleOCR	2.7.0.3
OpenCV	4.10.0.84
Camera module	Logitech 720p webcam
Camera resolution	1280 × 720
YOLO input size	640 × 640
CPU threads	Default ONNX Runtime CPU threading
Backend service	Firebase Firestore

The latency breakdown shows that PaddleOCR recognition is the dominant computational cost in the pipeline, followed by YOLOv8 ONNX detection. Firebase verification and access-event logging contributed a smaller but measurable network overhead. The measured total latency of 1758 ms is consistent with the previously reported end-to-end latency of approximately 1713 ms, with the small difference likely caused by runtime variability, profiling overhead, and environment-dependent OCR execution characteristics.

Together, these changes reduced the processing overhead enough to make the pipeline usable on a CPU-only Raspberry Pi without moving inference to a GPU server.

4.6 System Workflow

The workflow starts with an external sensor trigger. After a vehicle is detected, the camera captures an image and passes it to the processing module. The YOLOv8 detector localizes the plate, and the cropped region is then processed by PaddleOCR. The recognized text is normalized according to Turkish plate rules and then verified against the Firebase database for authorization or event logging. The resulting workflow remains modular, so later changes—such as multi-camera support or a character-level detection stage—can be added without rewriting the full pipeline. To further examine robustness, additional OCR engines and processing strategies, including EasyOCR- and Tesseract-based alternatives, were also evaluated in the ablation study presented in the Experimental Results section.

5 Experimental Results and Evaluation

Detection performance was first measured on a high-performance PC to assess the detector under less restrictive hardware conditions. End-to-end performance was then evaluated on a Raspberry Pi to reflect the actual deployment setting.

5.1 Experimental Setup

All edge-side experiments were conducted on a Raspberry Pi 4 Model B with 8 GB RAM running Debian GNU/Linux 12 (bookworm). The system used Python 3.11.2, ONNX Runtime 1.24.4 for YOLO inference, PaddleOCR 2.7.0.3 for optical character recognition, and OpenCV 4.10.0.84 for image processing. The camera stream was captured at 1280 × 720 resolution using a Logitech 720 p webcam, while YOLO inference

was performed with an input size of 640 × 640 on the CPU using the default ONNX Runtime threading configuration. Recognized plates were verified against a Firebase Firestore backend. Table 3 summarizes the hardware and software configuration used for the Raspberry Pi edge evaluation.

5.2 Dataset and Annotation

The OCR and end-to-end recognition evaluation was conducted on a manually verified subset of 200 real-world Turkish license plate images. This subset was used for plate-string evaluation, while the YOLOv8 detector was trained and evaluated separately on the larger bounding-box dataset described in Section 4.2. Since the available dataset only included bounding box annotations for detection, a semi-automatic annotation process was employed to generate ground-truth labels for text recognition. Initially, an offline inference pipeline was used to produce preliminary plate predictions for each image. These predictions were then manually reviewed and corrected to ensure high labeling accuracy. The final ground-truth file contains image-level plate-string annotations, providing a controlled benchmark for evaluating recognition performance. This procedure reduced the amount of manual labeling required while still producing a usable ground-truth set for recognition evaluation [32]. The 200-image set includes variation in lighting, viewing angle, and plate quality, making it suitable for examining system behavior outside ideal capture conditions. All OCR configurations were evaluated on the same dataset and with the same preprocessing pipeline to ensure fair comparison. We acknowledge that the 200-image set is limited in size and not systematically balanced across environmental factors such as time of day, weather, or occlusion. Therefore, the results should be interpreted as indicative of pipeline behavior under the captured conditions rather than as a comprehensive benchmark. A larger and more diverse evaluation set is needed for stronger generalization claims and is considered future work.

The evaluation metrics used in this study are summarized in Table 4.

Table 4: Evaluation metrics used in this study.

Metric	Definition
Exact-match accuracy	$\mathrm{Acc}_{\mathrm{exact}} = \frac{N_{\mathrm{correct}}}{N_{\mathrm{total}}}$
Character-level accuracy	$\mathrm{Acc}_{\mathrm{char}} = \frac{\sum_{i=1}^{N} n_i^{\mathrm{correct}}}{\sum_{i=1}^{N} n_i^{\mathrm{total}}}$
Average latency	$\mathrm{Latency}_{\mathrm{avg}} = \frac{1}{N}\sum_{i=1}^{N} t_i$
FPS	$\mathrm{FPS} = \frac{1000}{\mathrm{Latency}_{\mathrm{avg}}}$
IoU	$\mathrm{IoU}(A,B) = \frac{\lvert A \cap B \rvert}{\lvert A \cup B \rvert}$
Detection filtering	$\hat{B} = \{b_i \mid s_i > \tau\}$

5.3 Detection Performance

The detection results reported in this section were measured on the held-out test split and therefore represent the final detector evaluation on unseen data. These results should be distinguished from the validation-set metrics reported in Section 4.2, which were used for training-stage assessment and model

selection. The license plate detection model was evaluated independently on a PC using a held-out test split. The model achieved a precision of 91.20% and a recall of 90.14%, indicating that most license plates were correctly detected while keeping false positives low. The detector obtained an mAP@0.5 of 87.30% and an mAP@0.5:0.95 of 74.97%, demonstrating reliable localization performance under both standard and stricter evaluation criteria. The lower held-out test metrics compared with the validation results are expected because the test split provides a more conservative estimate of detector performance on unseen images. As summarized in Table 5, these results indicate that plate localization is sufficiently reliable for the downstream recognition stage in most test cases.

Table 5: Detection performance on the held-out test split (PC evaluation).

Metric	Value
Precision	91.20%
Recall	90.14%
mAP@0.5	87.30%
mAP@0.5:0.95	74.97%

5.4 OCR Module Evaluation

The OCR module was first evaluated independently using cropped license plate regions. Using EasyOCR, the system achieved a character-level accuracy of 87.42%, while the exact-match accuracy remained limited at 50.5%. This result shows that even a small number of character-level mistakes can noticeably reduce full-plate accuracy. To further investigate OCR performance, an ablation study was conducted comparing multiple OCR engines and configurations.

5.5 Ablation Study

An ablation study was conducted on the same 200-image dataset to examine the effects of normalization, retry logic, crop handling, and OCR engine choice. The goal was to estimate how much each design decision contributed to the final recognition performance. This experiment was conducted on a PC/GPU workstation using cropped plate images prepared for OCR evaluation, and was intended to select the OCR configuration for the final system rather than to report Raspberry Pi runtime performance. The full end-to-end Raspberry Pi deployment latency is reported separately in Section 5.7.

The results show that OCR engine choice affects final system performance more strongly than the other tested modifications. PaddleOCR [33] significantly outperformed both EasyOCR and Tesseract in terms of accuracy and computational efficiency. Normalization provided a substantial improvement for EasyOCR (+6.5% exact-match), while retry-based processing offered a smaller but consistent gain. In contrast, crop refinement strategies did not improve performance on this dataset. PaddleOCR improved both accuracy and latency relative to EasyOCR, making it the most suitable OCR module to carry forward into the Raspberry Pi end-to-end deployment. Regarding the latency values for the two PaddleOCR configurations, the slightly lower average latency observed for PaddleOCR + Retry compared with PaddleOCR alone is attributed to run-to-run variability and input-dependent OCR behavior in the workstation evaluation environment. The retry step is conditional: it is triggered only when the initial OCR output fails the Turkish plate-format check or produces fewer than five alphanumeric characters, and is therefore not executed for every image. The latency values in Table 6 should consequently be interpreted as average comparative measurements across the evaluated image set rather than as deterministic per-image costs. Latency values for Raspberry Pi deployment are reported separately in Section 5.7 and Table 2.

Table 6: OCR ablation results on cropped plate images in the PC/GPU evaluation environment.

Configuration	Exact Acc.	Char. Acc.	Latency (ms)	FPS
EasyOCR (baseline)	59.0%	90.71%	462.77	2.16
+ Normalization	65.5%	91.24%	462.04	2.16
+ Retry + Norm.	66.0%	91.94%	497.41	2.01
PaddleOCR	95.5%	98.65%	63.60	15.72
PaddleOCR + Retry	96.0%	98.78%	52.55	19.03
Tesseract	0.0%	14.63%	217.87	4.59

5.6 Deployment-Oriented Edge Trade-off Analysis

To further analyze deployment-oriented optimization choices on Raspberry Pi hardware, an additional trade-off analysis was conducted. The analysis examined FP32 and INT8 ONNX inference, ONNX Runtime thread settings, input resolution, and OCR crop expansion. This experiment was added to clarify that the term edge-optimized refers to practical deployment choices for CPU-only edge execution rather than to a full hardware-optimization benchmark. Table 7 summarizes the YOLO inference results at 640 px. Increasing the number of ONNX Runtime threads reduced inference latency for both FP32 and INT8 models. At four CPU threads, the INT8 model reduced mean YOLO inference latency from 539.5 to 240.8 ms, corresponding to a 2.24× speedup over FP32, while maintaining the same detection rate on the evaluated 200-image subset.

Table 7: YOLO inference latency and detection-rate trade-off on Raspberry Pi at 640 px.

Format	**Threads**	**Mean ± SD (ms)**	**P50 (ms)**	**P95 (ms)**	**Det. Rate**
FP32	1	1129.7 ± 22.7	1127.3	1130.4	100.0%
FP32	2	674.9 ± 2.6	674.7	677.6	100.0%
FP32	4	539.5 ± 106.4	503.5	854.9	100.0%
INT8	1	579.3 ± 1.0	579.2	580.8	100.0%
INT8	2	329.8 ± 0.9	329.9	331.5	100.0%
INT8	4	240.8 ± 50.1	223.9	378.5	100.0%

The effect of crop expansion on OCR performance was also evaluated while keeping the detector fixed at INT8, 640 px, and four CPU threads. The crop-expansion experiment was conducted as a relative trade-off analysis for OCR-region handling and should not be interpreted as a replacement for the primary end-to-end evaluation reported in Section 5.7. Differences in detector configuration, quantization setting, and evaluation flow may therefore produce different absolute recognition values. As shown in Table 8, no expansion produced the lowest OCR latency but lower recognition accuracy, likely because some plate characters were partially clipped. Moderate expansion values of 0.12–0.18 provided the best accuracy range. However, increasing the expansion to 0.25 added more background content and reduced exact-match accuracy.

These results show that the final deployment configuration involves a trade-off between latency and recognition robustness. Therefore, the system should be interpreted as deployment-oriented and edge-optimized for sensor-triggered stop-and-go parking, rather than as a continuous high-FPS ANPR system.

Table 8: Relative OCR trade-off analysis under different bounding-box expansion values.

Expand Frac.	Mean ± SD (ms)	P50 (ms)	P95 (ms)	Exact Match	Char. Acc.
0.00	955.1 ± 587.8	865.9	2035.9	66.50%	80.18%
0.06	1028.3 ± 712.0	629.0	2525.1	71.50%	86.01%
0.12	1166.5 ± 890.1	872.8	3107.4	73.00%	90.30%
0.18	1239.8 ± 1035.6	653.3	3340.6	73.00%	91.77%
0.25	1360.3 ± 1245.4	836.1	3953.3	67.50%	90.31%

5.7 End-to-End System Performance

The complete ANPR pipeline was evaluated on a Raspberry Pi using ONNX Runtime-based inference [30]. A Logitech C505 HD webcam was used for image capture in this prototype evaluation. Therefore, the results represent a practical low-cost edge deployment setting rather than an optimized industrial-camera setup. It is important to distinguish this evaluation from the OCR-only ablation in Table 6: the ablation study measures recognition on pre-cropped plate images, whereas Table 9 reports the complete image-to-result pipeline, including detection, crop extraction, OCR, normalization, and CPU-only edge execution. The system achieved an end-to-end exact-match accuracy of 83.5% and a character-level accuracy of 94.83%. The average processing latency was 1713 ms per image, corresponding to approximately 0.58 FPS. This throughput is not intended for free-flow or high-speed traffic scenarios. However, it is acceptable for the proposed sensor-triggered stop-and-go parking use case, where vehicles slow down or stop near the gate and only one or a few images are required for access verification. As expected, the end-to-end exact-match accuracy is lower than the 96.0% OCR-only result in Table 6, since the full pipeline introduces additional sources of error such as missed detections, imperfect bounding boxes, partial crops, motion blur, illumination variation, and camera-quality limitations. Nevertheless, the 94.83% character-level accuracy shows that most remaining errors are localized to individual characters rather than complete plate-string failures.

Table 9: End-to-end performance on Raspberry Pi.

Metric	Value
Exact Match Accuracy	83.5%
Character Accuracy	94.83%
Avg. Latency	1713 ms
FPS	0.58

5.8 Qualitative Results

To complement the quantitative evaluation, qualitative examples from the evaluated dataset are shown in Fig. 4. To avoid directly reproducing fully visible plate numbers in the paper, the examples are partially anonymized while still preserving the intermediate outputs of the pipeline. Each sample illustrates the main processing stages, including the input image with detection result, the cropped plate region, the OCR output, and the final normalized plate string.

These examples show that the proposed pipeline remains effective under both clear and more challenging imaging conditions. In particular, the combination of YOLO-based localization, PaddleOCR-based recognition, and Turkish rule-based normalization helps reduce common OCR ambiguities and improves final plate-string consistency.

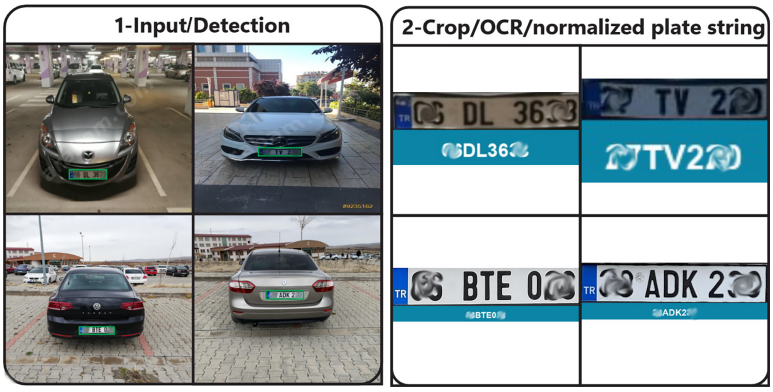

Figure 4: Qualitative examples of the proposed ANPR pipeline with partially anonymized plate numbers.

5.9 Error Analysis

An error analysis was conducted to identify the main sources of failure in the system. The majority of errors were attributed to OCR character confusion, particularly between visually similar characters such as O/0, B/8, S/5, and Z/2. Additional errors were caused by imperfect bounding box localization, leading to partial cropping of characters, as well as challenging environmental conditions such as glare, motion blur, and low illumination. Despite these limitations, most errors were minor and localized at the character level, as reflected by the high character-level accuracy compared with exact-match accuracy. This suggests that further gains are still possible, mainly through better OCR stability and slightly tighter plate localization.

5.10 Discussion

The results indicate that the proposed pipeline provides a usable trade-off between recognition accuracy and computational cost for edge deployment. The combination of YOLOv8 detection, PaddleOCR recognition, and rule-based normalization produced the most reliable performance among the tested configurations. The deployment-oriented trade-off analysis further showed that INT8 quantization and ONNX Runtime threading can reduce YOLO inference latency on Raspberry Pi hardware, while OCR crop expansion affects the balance between recognition robustness and processing cost. The findings suggest that system-level optimization, rather than isolated improvement of individual modules, is critical for ANPR on edge devices. In particular, the integration of detection, OCR, and normalization enables a complete deployable ANPR workflow on edge hardware, where the final performance depends on the interaction between all pipeline stages rather than on OCR accuracy alone. Among the evaluated OCR engines, PaddleOCR provided the best overall balance of accuracy and latency for the target application.

6 Smart Parking System Integration

The proposed ANPR pipeline is integrated into an IoT-based smart parking system that combines edge devices, backend services, and user applications. The recognized plate is used directly for access control, parking records, and user interaction.

As shown in Fig. 5, vehicle identification is performed at the edge, while verification and user-side functions are handled through the backend and application layers.

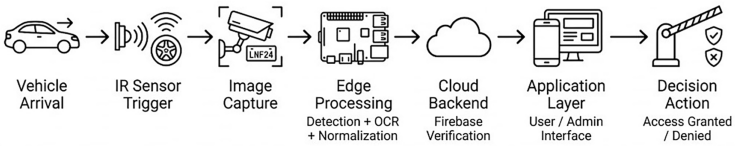

Figure 5: End-to-end workflow of the proposed smart parking system.

6.1 System Integration

Each parking entry point includes a Raspberry Pi, a camera module, and an infrared (IR) sensor. The IR sensor triggers processing only when a vehicle is present, which reduces both computation and power usage on the edge device. Each edge node performs detection and recognition locally, while communication with the backend is handled through lightweight protocols suitable for embedded deployment. Firebase is used as the backend infrastructure for storage, synchronization, and user-related services. Firestore stores user records, registered vehicles, reservations, and entry/exit logs, while authentication services manage user access. After plate recognition, the extracted plate number is sent to the backend for verification against registered vehicles and active reservations, and the corresponding event is then recorded. A Flutter-based mobile application serves as the main interface for end users, allowing vehicle registration, reservation management, and parking-related notifications. A separate web-based dashboard supports administrative monitoring, including user management, parking activity inspection, and access-log review. Recognition results and parking status are synchronized with the user interfaces in near real time.

6.2 Operational and Deployment Aspects

The workflow begins when a vehicle is detected by the sensor. The edge device captures an image, runs the ANPR pipeline locally, and generates a plate candidate. The result is then checked against backend records so that the system can log the event, support access decisions, and trigger notifications when needed. This design keeps time-critical processing at the edge while maintaining synchronized backend records. The overall design also supports scalability. Since each entry point performs recognition locally, the system can be extended to additional parking nodes without moving the full image-processing workload to a central server [27]. Because continuous image transmission is avoided, the architecture is more practical in environments with limited or unstable connectivity, such as enclosed or underground parking areas. The modular architecture also simplifies future updates to the detector, OCR stage, or backend services. To illustrate the communication advantage of edge-side inference, a simple bandwidth estimate can be considered. In a cloud-based design, the device may upload one compressed camera image for each vehicle event; for example, a JPEG frame captured at parking-gate distance may require approximately 200 KB after compression. In the proposed edge-based design, the Raspberry Pi transmits only the recognized plate string, timestamp, gate or device identifier, confidence score, and access decision as a structured event record, which can typically be represented in less than 1 KB. This can reduce the per-event upstream payload by approximately two orders of magnitude compared with a cloud-upload approach. Although the exact reduction depends on image resolution and event formatting, the estimate highlights the practical communication advantage of local ANPR inference in bandwidth-constrained parking environments.

6.3 Privacy and Data Handling

Because license plate information can be considered sensitive personal data in some jurisdictions, the proposed system follows a privacy-aware data-handling approach. The ANPR pipeline runs locally on the

edge device, so continuous video streams do not need to be transmitted to the cloud for recognition. In normal operation, the backend receives only the recognized plate string, timestamp, gate or device identifier, and access or logging decision. Raw images or cropped plate snapshots should be stored only when required for debugging, audit, or security review, and their retention period should be limited according to institutional or organizational policy. Access to stored plate records and logs should be restricted through authenticated roles, such as administrator and security staff, supported by the Firebase Authentication and backend access-control mechanisms used in the proposed architecture. These measures support GDPR-aware deployment principles such as data minimization, access control, and retention management.

7 Conclusion

In this paper, we implemented an edge-based ANPR pipeline and integrated it into a smart parking system built around IoT components, backend services, and user applications. The pipeline combines YOLOv8 for plate detection, PaddleOCR for text recognition, and a rule-based normalization stage to reduce common OCR errors in Turkish plate images. Deployment on a Raspberry Pi with ONNX Runtime showed that deep learning-based ANPR is feasible on low-cost edge hardware, although throughput remains lower than in server-based systems. The OCR-only ablation study showed that PaddleOCR performed better than the other OCR engines tested in both recognition accuracy and processing time on cropped plate images. The ablation study further showed that normalization and retry logic provided measurable gains, while the end-to-end Raspberry Pi evaluation confirmed that the pipeline can operate in a practical sensor-triggered stop-and-go parking setting despite CPU-only edge hardware constraints. Beyond recognition accuracy, the study demonstrates how edge-side plate reading can be connected to a broader parking workflow, where Firebase services and Flutter applications support synchronization, logging, and user interaction. The current system still has limitations. Throughput is constrained by the computational capacity of the Raspberry Pi, and further optimization is possible. Performance could improve with a larger and more diverse dataset and with newer edge hardware such as Raspberry Pi 5. Nevertheless, the results suggest that the proposed pipeline is a practical low-cost alternative to GPU-based or fully cloud-based ANPR deployments in smart parking scenarios, especially when the recognition module is integrated with backend functions such as vehicle registration, reservation matching, and access logging.

Acknowledgement: Not applicable.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: The authors confirm contribution to the paper as follows: conceptualization and supervision, Alireza Souri; methodology, software, validation, investigation, writing—original draft preparation, and writing—review and editing, Mohammad Ebrahimishadman and Alireza Souri. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The data supporting the findings of this study are available from the corresponding author upon reasonable request.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Vivek K, Swetha G, Kumar MVTRP, Veni AK, Praveen N, Kale MR. Role of IoT in smart cities: a review, applications, open challenges and solutions. In: 2025 International Conference on Electronics and Renewable Systems (ICEARS). Piscataway, NJ, USA: IEEE; 2025. p. 596–601.
2. Lin T, Rivano H, Le Mouel F. A survey of smart parking solutions. IEEE Trans Intell Transp Syst. 2017;18(12):3229–53. doi:10.1109/tits.2017.2685143.
3. Khanna A, Anand R. IoT based smart parking system. In: 2016 International Conference on Internet of Things and Applications (IOTA). Piscataway, NJ, USA: IEEE; 2016. p. 266–70.
4. Javaheri A, Channamallu SS, Kermanshachi S, Rosenberger JM, Pamidimukkala A, Kan C, et al. Evaluating the impact of smart parking systems on parking violations. IEEE Access. 2024;12(1):175585–96. doi:10.1109/access.2024.3503513.
5. Hu Z, Cao Y, Huang J, Wu L, Xu L, Zhang R. CRPark: a smart parking system with competition and reservation services. IEEE Trans Veh Technol. 2025;74(8):12981–96.
6. Mustafa T, Karabatak M. Real time car model and plate detection system by using deep learning architectures. IEEE Access. 2024;12(9):107616–30. doi:10.1109/access.2024.3430857.
7. Redmon J, Divvala S, Girshick R, Farhadi A. You only look once: unified, real-time object detection. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Piscataway, NJ, USA: IEEE; 2016. p. 779–88.
8. Redmon J, Farhadi A. YOLO9000: better, faster, stronger. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Piscataway, NJ, USA: IEEE; 2017. p. 6517–25.
9. Bochkovskiy A, Wang CY, Liao HYM. YOLOv4: optimal speed and accuracy of object detection. arXiv:2004.10934. 2020.
10. Laroca R, Severo E, Zanlorensi LA, Oliveira LS, Goncalves GR, Schwartz WR, et al. A robust real-time automatic license plate recognition based on the YOLO detector. In: 2018 International Joint Conference on Neural Networks (IJCNN). Piscataway, NJ, USA: IEEE; 2018. p. 1–10.
11. Laroca R, Zanlorensi LA, Gonçalves GR, Todt E, Schwartz WR, Menotti D. An efficient and layout-independent automatic license plate recognition system based on the YOLO detector. IET Intell Transp Syst. 2021;15(4):483–503. doi:10.5753/sibgrapi.est.2020.12978.
12. Moussaoui H, Akkad NE, Benslimane M, El-Shafai W, Baihan A, Hewage C, et al. Enhancing automated vehicle identification by integrating YOLO v8 and OCR techniques for high-precision license plate detection and recognition. Sci Rep. 2024;14(1):14389. doi:10.1038/s41598-024-65272-1.
13. Sarhan A, Abdel-Rahem R, Darwish B, Abou-Attia A, Sneed A, Hatem S, et al. Egyptian car plate recognition based on YOLOv8, Easy-OCR, and CNN. J Electr Sys Inform Technol. 2024;11(1):32. doi:10.1186/s43067-024-00156-y.
14. Montazzolli S, Jung C. Real-time brazilian license plate detection and recognition using deep convolutional neural networks. In: 2017 30th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI). Piscataway, NJ, USA: IEEE; 2017. p. 55–62.
15. Xu C, Zhang H, Wang W, Qiu J. License plate recognition system based on deep learning. In: 2020 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA). Piscataway, NJ, USA: IEEE; 2020. p. 1300–3.
16. Zherzdev S, Gruzdev A. LPRNet: license plate recognition via deep neural networks. arXiv:1806.10447. 2018.
17. Wang Z, Jiang Y, Liu J, Gong S, Yao J, Jiang F. Research and implementation of Fast-LPRNet algorithm for license plate recognition. J Electr Comput Eng. 2021;2021(4):8592216. doi:10.1155/2021/8592216.
18. Du Y, Li C, Guo R, Yin X, Liu W, Zhou J, et al. PP-OCR: a practical ultra lightweight OCR system. arXiv:2009.09941. 2020.
19. Li H, Wang P, Shen C. Toward end-to-end car license plate detection and recognition with deep neural networks. IEEE Trans Intell Transp Syst. 2019;20(3):1126–36. doi:10.1109/tits.2018.2847291.
20. Shashirangana J, Padmasiri H, Meedeniya D, Perera C. Automated license plate recognition: a survey on methods and techniques. IEEE Access. 2021;9:11203–25.

21. Fakhar SAG, Saad HM, Fauzan KA, Affendi HR, Aidil AM. Development of portable automatic number plate recognition (ANPR) system on Raspberry Pi. Int J Electr Comput Eng. 2019;9(3):1805. doi:10.11591/ijece.v9i3.pp1805-1813.
22. Lin CJ, Chuang CC, Lin HY. Edge-AI-based real-time automated license plate recognition system. Appl Sci. 2022;12(3):1445. doi:10.3390/app12031445.
23. Sonnara F, Chihaoui H, Filali F. Efficient real-time license plate recognition using deep learning on edge devices. J Real Time Image Process. 2025;22(4):159. doi:10.1007/s11554-025-01738-3.
24. Glasenapp LA, Hoppe AF, Wisintainer MA, Sartori A, Stefenon SF. OCR applied for identification of vehicles with irregular documentation using IoT. Electronics. 2023;12(5):1083. doi:10.3390/electronics12051083.
25. Ke R, Zhuang Y, Pu Z, Wang Y. A smart, efficient, and reliable parking surveillance system with edge artificial intelligence on IoT devices. IEEE Trans Intell Transp Syst. 2021;22(8):4962–74. doi:10.1109/tits.2020.2984197.
26. Nithya R, Priya V, Sathiya Kumar C, Dheeba J, Chandraprabha K. A smart parking system: an IoT based computer vision approach for free parking spot detection using faster R-CNN with YOLOv3 method. Wirel Personal Commun. 2022;125(4):3205–25.
27. Rajesh P, Kallakuri D, Chagarlamudi S, Vissavajhula SM, Meng J, Zhao J. Intelligent parking guidance system using computer vision, IoT, and edge computing. IEEE Open J Intell Transp Syst. 2025;6(7):1185–99. doi:10.1109/ojits.2025.3605318.
28. Satyanath G, Sahoo JK, Roul RK. Smart parking space detection under hazy conditions using convolutional neural networks: a novel approach. Multimed Tools Appl. 2023;82(10):15415–38. doi:10.1007/s11042-022-13958-x.
29. Neupane D, Bhattarai A, Aryal S, Bouadjenek MR, Seok U, Seok J. Shine: a deep learning-based accessible parking management system. Expert Syst Appl. 2024;238(9):122205. doi:10.1016/j.eswa.2023.122205.
30. Microsoft. ONNX runtime. GitHub repository. [cited 2026 May 23]. Available from: https://github.com/microsoft/onnxruntime.
31. Ultralytics. YOLOv8 documentation. 2023 [cited 2026 May 23]. Available from: https://docs.ultralytics.com.
32. Roboflow. Turkish number plates dataset [Dataset]. 2025 [cited 2026 May 23]. Available from: https://universe.roboflow.com/m-szsa4/turkish-number-plates-u5mza/model/1.
33. PaddlePaddle. PaddleOCR. GitHub repository. 2021 [cited 2026 May 23]. Available from: https://github.com/PaddlePaddle/PaddleOCR.