Vietnam National University, Hanoi

International School

==========

Master Thesis

Vehicle Recognition in Regions of Interest within Surveillance Camera Systems

NGUYEN THI HOA AN

Field: **Informatics and Computer Engineering**

Code: **8480111.01QTD**

Hanoi - 2026

Vietnam National University, Hanoi

International School

==========

Master Thesis

Vehicle Recognition in Regions of Interest within Surveillance Camera Systems

NGUYEN THI HOA AN

Field: **Informatics and Computer Engineering**

Code: **8480111.01QTD**

Supervisor: **Dr. Pham Dinh Tan**

Hanoi - 2026

CERTIFICATE OF ORIGINALITY

I, the undersigned, hereby certify my authority of the study project report entitled Vehicle Recognition in Regions of Interest within Surveillance Camera Systems submitted in partial fulfillment of the requirements for the degree of Master of Informatics and Computer Engineering. Except where the reference is indicated, no other person's work has been used without due acknowledgement in the text of the thesis.

Hanoi, 2026

Approved by

SUPERVISOR

(Signature and full name)

Date:……

ABSTRACT

Today, with the rapid pace of urbanization, traffic monitoring and automated parking management have become an urgent need. However, traditional methods are often costly and difficult to maintain. Current object recognition models are primarily based on rectangular bounding boxes, leading to significant errors when vehicles are parked diagonally or when there is overlap within the area of interest (ROI). This study proposes a new vehicle recognition system by transforming the object recognition problem into pose estimation to accurately determine the vehicle's position based on its four wheels.

The system uses Deep Learning models (YOLOv8-Pose and YOLOv11-Pose) combined with a Polygon Intersection geometry algorithm to calculate the degree of vehicle overlap relative to the ROI. The experiment was conducted on a dataset of nearly 900 images (divided into training, validation, and test sets in an 80:10:10 ratio) collected from parking lots in South Korea in 2025, including various harsh weather conditions such as rain, snow, and different lighting conditions.

The experimental results showed that both YOLOv8 and YOLOv11 architectures achieved near-perfect mAP50 Box recognition accuracy at 99.5%. For the feature point recognition (Pose) task, YOLOv8-Pose achieved an mAP50 of 55.4%, slightly better than YOLOv11-Pose (52.7%). Notably, the developed logic algorithm is capable of recovering obscured points based on geometric properties, ensuring stable system operation even when vehicles are partially obscured or parked off-line. When the complete pipeline was evaluated end-to-end on the whole test set (225 image–ROI decision samples), the final ROI occupancy decision achieved an accuracy of 97.78%, with a false-occupancy rate of 2.76% and a missed-occupancy rate of 1.25%.

Despite limitations in inference speed (~96.3ms/image), the study demonstrated the superior effectiveness of using Keypoints compared to traditional bounding box methods in smart parking space management. The next steps in the project include optimizing speed using TensorRT and deploying it on Edge devices.

Keywords: *Vehicle Recognition, YOLOv8-Pose, Region of Interest (ROI), Keypoint Annotation, Polygon Intersection, Smart Parking.*

Table of Contents

FIGURES

1 INTRODUCTION

1.1 Background

In recent years, with the rapid development of technology and AI, and the increase in traffic, especially vehicles (e.g., cars, trucks), the problem of traffic monitoring and automated parking management has become more urgent than ever. Traditional methods such as using magnetic sensors or manual monitoring personnel are often costly to install, difficult to maintain, and have low scalability, requiring additional manpower.

To solve these difficulties, AI (Artificial Intelligence) is now being applied increasingly widely in all areas of life, including transportation. The field of computer vision has proven its effectiveness in combining AI technology with monitoring and management. More specifically, using surveillance cameras combined with Deep Learning to automatically recognize and analyze vehicle behavior. However, current conventional object detection models primarily return results in the form of rectangular bounding boxes. In practice, especially in parking lots or diagonal lanes, the use of bounding boxes often causes significant errors due to overlapping rectangular boxes or the inclusion of unwanted areas, leading to inaccurate determination of the vehicle's actual location within the region of interest (ROI).

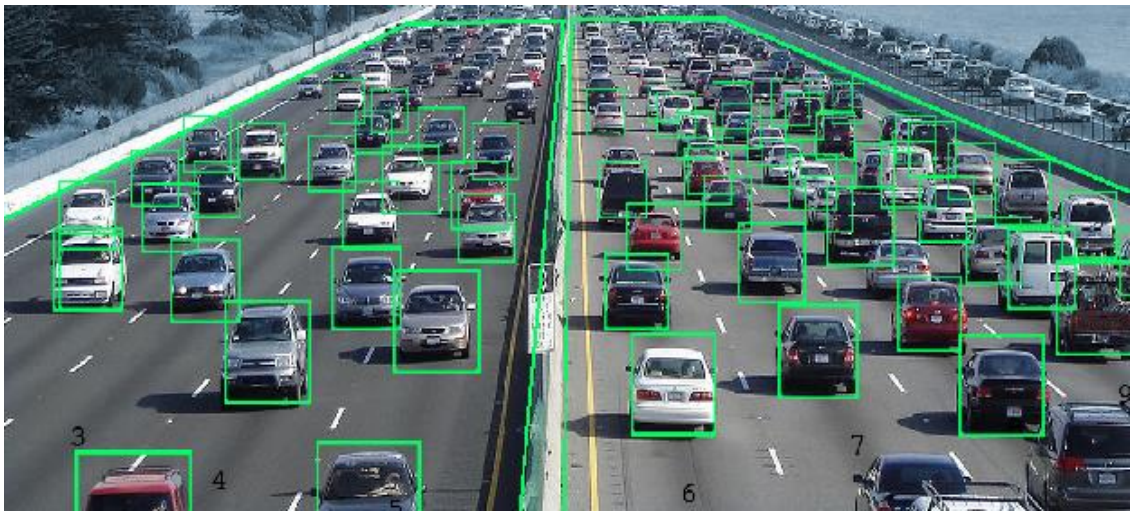

Figure 1: Traffic Monitoring using AI

Based on this real-world problem, the author decided to undertake the project "Building a Vehicle Monitoring and Detection System in a Defined Area Using YOLOv8 Keypoints and Geometric Algorithms". The project focuses on solving the problem of accurately determining vehicle

location by combining the recognition power of Deep Learning models (YOLOv8 Pose) with the accuracy of geometric mathematics (Polygon Intersection), aiming to create an automated, flexible, and highly reliable monitoring solution.

1.2 Research Objectives

This research focuses on addressing the following core objectives:

- **Optimizing the vehicle detection problem**: Converting from the object detection (bounding box) problem to the pose estimation (keypoints) problem. The goal is to eliminate the unnecessary background noise often found in rectangular bounding boxes, especially when the vehicle moves diagonally or turns around, and to accurately determine the vehicle's position based on its four wheels.
- **Building a training database**: Creating a standardized dataset with nearly 900 image samples labeled with four vehicle corner points, to train the YOLOv8 model to accurately recognize the geometric shape of the vehicle.
- **Solving the "In-ROI" problem using geometry**: Applying the Shapely library and vector mathematics to calculate the intersection area between the "Vehicle Polygon" and the "No Parking/Restricted Area Polygon". The goal is to make highly accurate logical decisions about whether a vehicle is in violation of parking rules or is parked correctly.

1.3 Research Subjects and Scope

- Research Subjects: Vehicles (cars, trucks, etc.) in a parking environment, YOLOv8 model (Ultralytics) and geometric processing libraries (Shapely, OpenCV).
- Research Scope: Focuses on image data collected from high-angle surveillance cameras (CCTV), under daytime and nighttime lighting conditions, and in various weather conditions such as rain, snow, fog, etc.
- Problem: Limited to detecting vehicles and determining the "in-ROI" (In-Region of Interest) status based on the IoU threshold (Intersection over Union threshold).
- Technique: Uses a hybrid approach: AI is used to extract coordinate features, and rule-based algorithms are used for logical decision-making.

1.4 Research Methods

To carry out the project, the author employed the following research methods:

- Data collection and processing method: Nearly 900 images were collected from real-world cameras, including images in all weather conditions such as rain, sun, snow, fog, etc. Images were also collected during the day and night. All collected data was manually labeled with Keypoints for the four corners of the vehicle to train the deep learning model.
- Empirical method:
  - Training the YOLOv8 model with hyperparameter tuning.
  - Evaluating the model through Precision, Recall, and mAP metrics.
  - Mathematical modeling method: Using the Shapely library to model the vehicle and ROI area as polygons. Applying the formula for calculating the intersection area to filter the results, addressing the shortcomings of the traditional Bounding Box method.

1.5 Research Gap and Contributions

Research gap: Most existing approaches to parking-space monitoring and vehicle detection within a Region of Interest (ROI) rely either on per-space image-patch classification or on axis-aligned rectangular bounding boxes produced by conventional object detectors. Patch-classification methods require the parking layout to be fixed and each space to be calibrated in advance, while axis-aligned bounding boxes inevitably include a considerable portion of surrounding background. As a result, these methods make inaccurate occupancy decisions when a vehicle is parked diagonally, crosses a space boundary, or is partially occluded by an adjacent vehicle. Representing a vehicle by its actual ground footprint – the polygon formed by its four wheel contact points – and combining a learned keypoint detector with interpretable, rule-based geometric reasoning for the final occupancy decision remains under-explored in the literature. This thesis addresses that gap.

The main contributions of this thesis are as follows: (i) it reformulates the problem of determining vehicle occupancy within an ROI from conventional object detection into a four-wheel keypoint (pose estimation) problem, so that each vehicle is represented by its ground footprint polygon

instead of an axis-aligned bounding box; (ii) it constructs a dataset of nearly 900 surveillance-camera images annotated with both vehicle bounding boxes and four wheel keypoints per vehicle, covering diverse lighting (day/night) and weather (sun, rain, fog, snow) conditions; (iii) it proposes a geometric post-processing pipeline consisting of a missing-keypoint reconstruction mechanism based on parallelogram properties and an Intersection-over-Vehicle-Area occupancy score computed by polygon intersection, which keeps the occupancy decision robust when a wheel is occluded; and (iv) it provides a comparative evaluation of YOLOv8-Pose and YOLOv11-Pose under an identical experimental setting, together with a functional software prototype and a web-based demonstration.

2 BACKGROUND KNOWLEDGE

2.1 Overview of Computer Vision

Computer vision is one of the most popular and widely applied subfields of artificial intelligence: it enables machines to process and analyze images and videos in order to extract meaningful information [1]. It comprises three complementary processes – recognition, reconstruction, and understanding – which together allow a system to identify entities such as people, text, or vehicles, extract their three-dimensional features, and deduce the relationships between them [1]. The explosion of machine learning, and especially of deep neural networks trained on large-scale datasets, has overcome traditional algorithmic barriers and made computer vision a cornerstone technology in fields such as autonomous vehicles, medical diagnostics, security surveillance, and smart-city development [2].

2.2 Fundamental Concepts and Techniques

A typical computer-vision pipeline consists of five main stages [3]:

1. **Image Acquisition and Representation:** raw images captured by sensors are enhanced, denoised, and normalized so that the data is optimized for subsequent analysis [3].
2. **Feature Detection and Extraction:** instead of processing millions of pixels, the system localizes and encodes distinctive features such as edges, corners, and contours, which remain recognizable under varying conditions [3].

3. **Image Segmentation and Object Detection:** the image is partitioned into homogeneous regions [4], and objects are simultaneously localized (bounding box) and classified – answering the two fundamental questions "what is it?" and "where is it?".
4. **Motion Analysis and Scene Understanding:** the analysis is extended from static images to the dynamics of the whole scene, allowing the system to interpret the ongoing context rather than isolated objects [3].

5. **Evolution from Traditional Methods to Deep Learning:** hand-crafted filters, rules, and pre-processing steps have been replaced by Convolutional Neural Networks (CNNs) that learn critical features autonomously from large datasets, enabling modern models such as YOLOv8 to combine high accuracy with real-time processing speed [5].

   Finally, there has been a paradigm shift from traditional methods—which required humans to manually design filters, mathematical rules, and pre-processing steps—to Deep Learning. With the advent of Convolutional Neural Networks (CNNs), computers can now autonomously process vast datasets to learn critical features. This transition

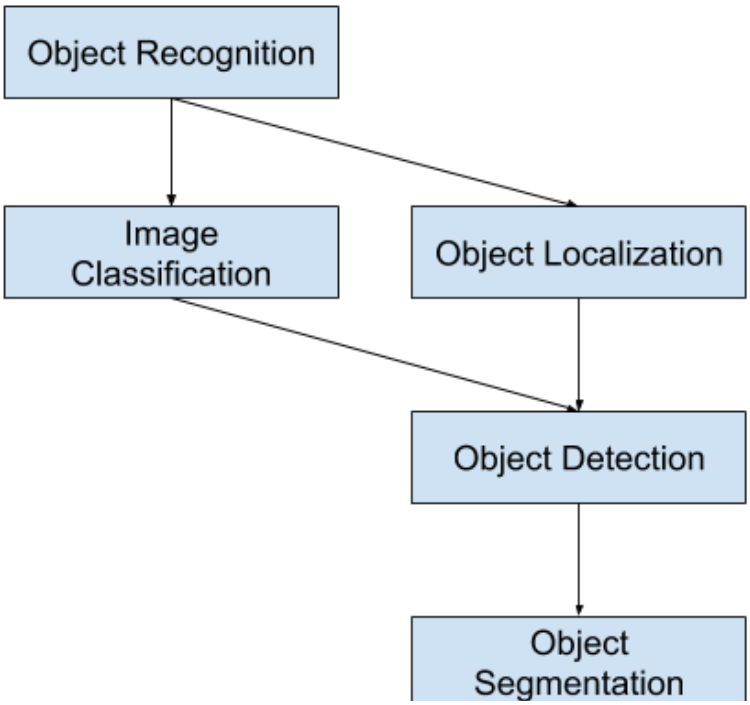

Figure 2: Computer vision tasks

has enabled modern models (such as YOLOv8) to achieve unprecedented levels of accuracy and real-time processing speed. [1]

2.3 Applications in Computer Vision

Computer vision is applied in a wide range of fields. In autonomous and assisted driving, it helps drivers and unmanned vehicles detect surrounding objects and people, making operation safer [6]; in medical diagnosis, it helps doctors analyze images more accurately and promptly; and in traffic monitoring – the domain most relevant to this thesis – it enables continuous, automated observation of road users to support security and order in smart cities.

2.4 Object Detection and Pose Estimation

Object detection is a problem that requires drawing a bounding box around each object of interest in an image and assigning them a label [2].

- The input is an image with one or more objects.
- Output: One or more bounding boxes and labels for each bounding box.

2.4.1 Class of R-CNN family models

The class of feature-determining models based on CNN networks is known as R-CNN (regions with CNN features), developed by Ross Girshick and colleagues. This class of models includes three main models: R-CNN, Fast R-CNN, and Faster-RCNN. They are designed to serve the tasks of object localization and object recognition. [3]

2.4.1.1 R-CNN (2014)

The R-CNN model was first introduced in 2014 by Ross Girshick and colleagues in the landmark work "Rich feature hierarchies for accurate object detection and semantic segmentation", marking a turning point in the application of convolutional neural networks (CNNs) to object localization, detection, and segmentation, with superior results on standard benchmarks such as VOC-2012 and ILSVRC-2013 [7].

Technically, the architecture of R-CNN consists of three independent modules [4]:

- Region Proposals: approximately 2,000 candidate regions that may enclose objects of interest are generated for each image by the Selective Search algorithm [8].
- Feature Extraction: a deep CNN (AlexNet) computes a 4096-dimensional feature vector from each candidate region.

- Classification: linear SVM models use the extracted features to determine the label of the object within each region.

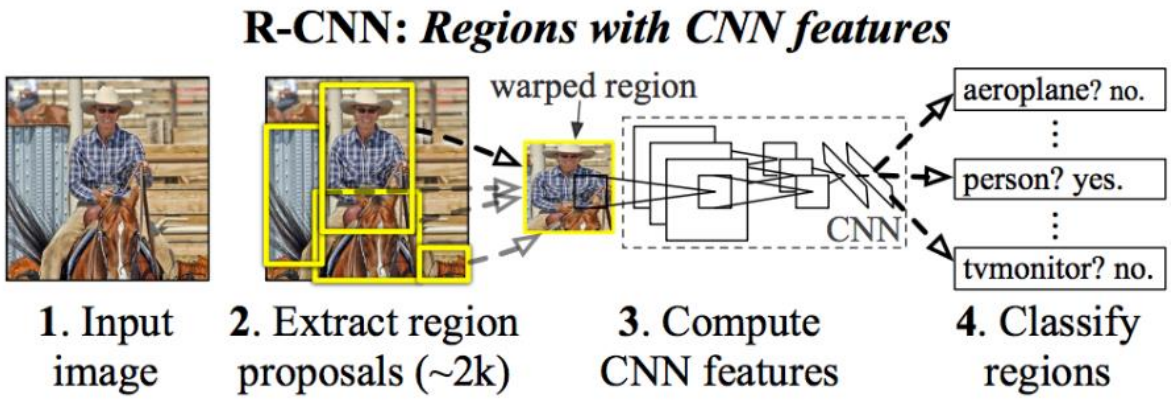

Figure 3: R-CNN

Advantages and Disadvantages

Although R-CNN is a significant advancement over traditional methods due to its simple and accessible structure, this model still has significant performance limitations:

- Slow processing speed: features must be extracted by the deep CNN separately for each of the 2,000 candidate regions of every image, resulting in an extremely large computational load.
- Multi-stage training: region extraction, CNN training, SVM training, and bounding-box regression are trained independently, making the system difficult to optimize comprehensively.

2.4.1.2 Fast R-CNN

To overcome these shortcomings, Fast R-CNN was developed as an improved version in 2015 [8]. Inspired by the SPPnet technique (2014) – which showed that features can be extracted from the entire image only once instead of separately for every cropped region – Fast R-CNN combines all steps into a single trainable model:

- Feature Map Extraction: the entire image is passed through a CNN (e.g., VGG-16) once to create an overall feature map.
- RoI Pooling: the region proposals are projected directly onto the feature map and converted into a uniform, fixed size.

- Multitask output: two parallel branches then perform Softmax classification and bounding-box regression.

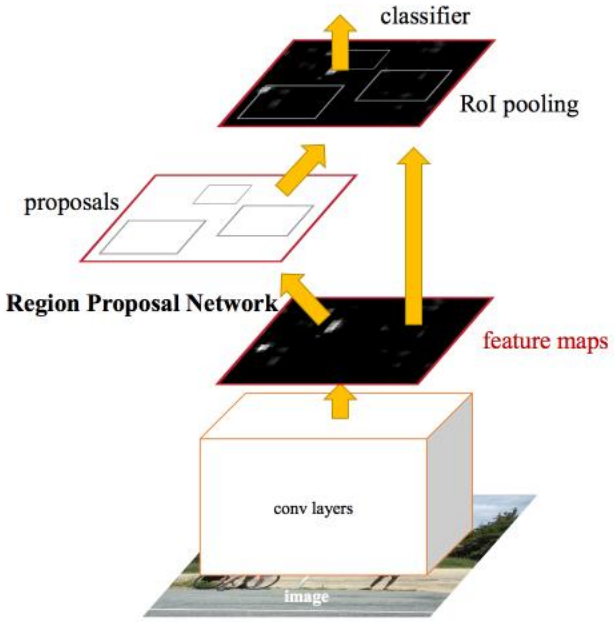

Figure 4: Fast R-CNN

Fast R-CNN is truly a leap forward: it's significantly faster in both training and prediction. However, despite its speed, the model still has a small bottleneck: it still relies on external algorithms (like Selective Search) to find questionable regions.

2.4.2 YOLO

Besides R-CNNs, YOLO is also a well-known family of object recognition models that is gradually proving its power with widespread use today. The biggest difference between YOLO and R-CNNs is speed. While R-CNN models often prioritize high accuracy, YOLO is designed to run at high speed, reaching real-time processing capabilities [5].

2.4.2.1 YOLO first version (2015)

The YOLO model was first introduced by Joseph Redmon and his colleagues in 2015 in the paper: "You Only Look Once: Unified, Real-Time Object Detection".

Mechanism of Operation

Instead of dividing the process into many complex steps (such as finding candidate regions and then classifying them), YOLO uses a single neural network trained end-to-end: the image is divided into a grid of cells, and each cell predicts bounding boxes for objects whose center falls within it, described by five key parameters – the center coordinates (x, y), the size (width, height), and a confidence score – together with the probabilities of the object classes [9].

Architecture

The YOLO architecture consists of a convolutional base network responsible for extracting features from the input image, followed by extra layers that predict the class label and the bounding-box coordinates. In the first version, the base network uses the Darknet architecture, whose output is a 7×7×1024 feature map; thanks to this flexible design, YOLO has many versions customized for different input image sizes [5][9].

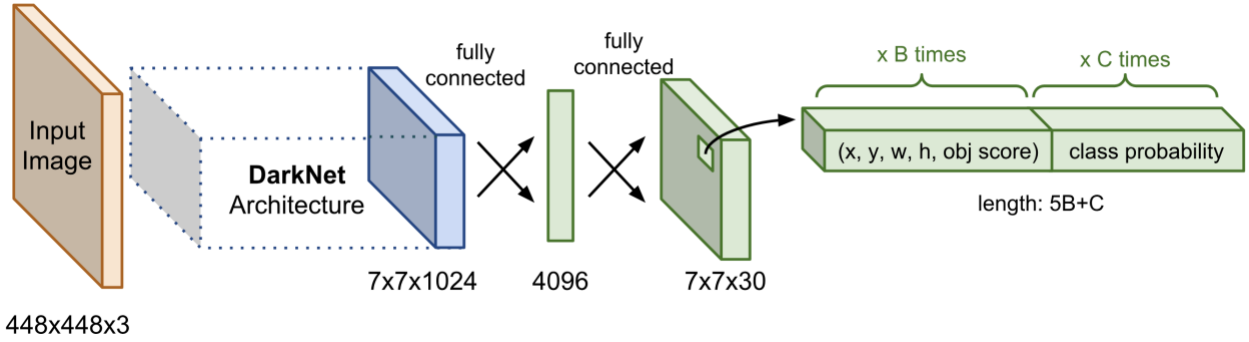

Figure 5: YOLOv1

In YOLOv3, the base network is upgraded to Darknet-53, a deeper backbone of 53 convolutional layers with Batch Normalization, Leaky ReLU activation, and stride-2 downsampling, which reduces the feature-space size and parameter count [6]. This design gives YOLOv3 a good balance between speed and accuracy in object detection [1].

Figure 6: YOLOv3

Output

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

$$y^T = [p_0, \underbrace{\langle t_x, t_y, t_w, t_h \rangle}_{\text{bounding box}}, \underbrace{\langle p_1, p_2, \ldots, p_c \rangle}_{\text{scores of c classes}}]$$

- p0 is the probability of predicting an object appearing within the bounding box.
- x, y, w, h are the coordinates for the center and the width and length dimensions of the bounding box.

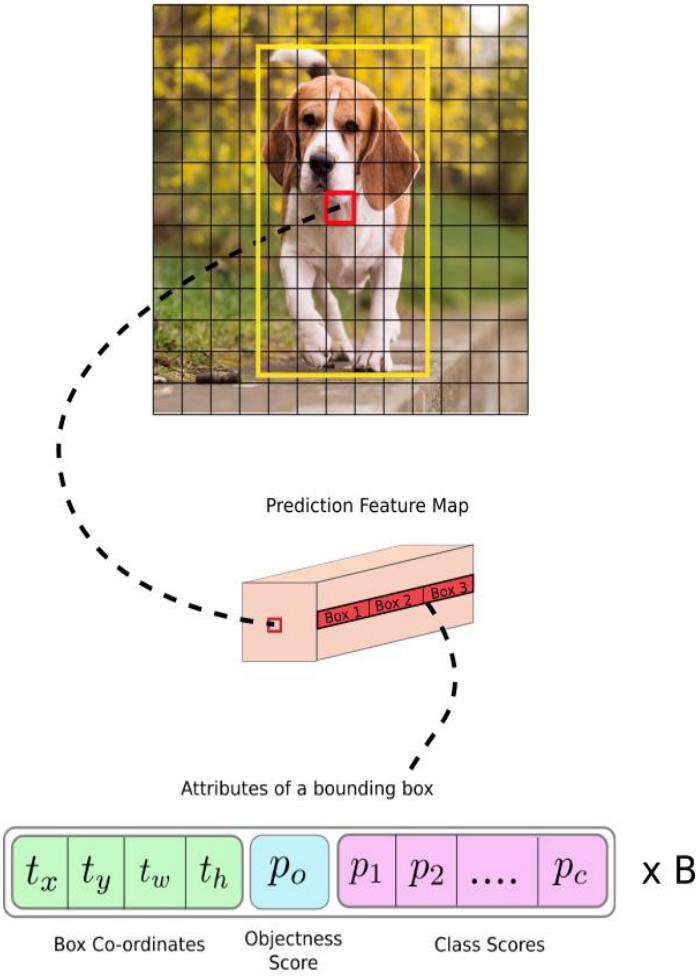

Figure 7: Example output YOLO

At each cell of the output feature map, the model selects three anchor boxes of different sizes (e.g., one tall, one square, one wide) whose centers coincide with the center of that cell, so that each box can specialize in objects of a particular shape; the output at each cell is the concatenated vector of the predictions of the three boxes. This structure allows the model to "specialize":

- Box 1: Can specialize in small/thin objects (e.g., lampposts).
- Box 2: Specializes in medium-sized objects (e.g., people).

Each predicted box is described by 4 coordinate parameters (x, y, w, h), 1 confidence score (conf), and the probabilities of the object classes; for example, when training with 80 classes and 3 anchors per cell, each feature-map location produces 85 × 3 = 255 output parameters, allowing multiple objects with different shapes and sizes to be predicted at the same spatial location on the image [7].

For cases where an object is surrounded by at least two anchor boxes, the anchor box to be identified will be the one with the highest IoU (Intersection over Union) relative to the ground truth bounding box [6].

Loss function

The YOLO loss function includes localization loss, used to measure the error of the bounding box, and confidence loss, used to measure the error of the probability distribution of the classes.

In the formula, simply put, the first sum is the loss of the prediction that the cell contains an object or not, and the second sum is the loss of the probability distribution if the cell contains an object.

Advantages and disadvantages

Amazing speed: YOLO can process at 45 frames per second (fps), and optimized versions can reach up to 155 fps. However, in early versions, YOLO often encountered localization errors and had lower accuracy compared to R-CNN. Nevertheless, for practical applications, YOLO remains a suitable and effective model.

2.4.2.2 YOLOv8

In 2023, YOLOv8 was officially released. It is one of the most popular and widely used versions of the YOLO family, boasting high accuracy and fast processing speed. This is due to the significant improvements in the network architecture (backbone, neck, head) and loss function compared to previous versions. YOLOv8 employs an intelligent combination strategy to simultaneously optimize label recognition (Classification) and object location (Regression) instead of simple loss functions like previous versions. [7]

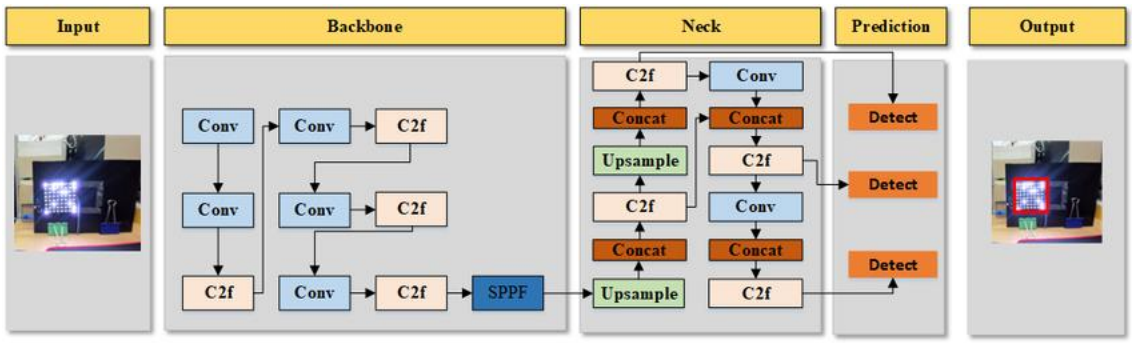

Figure 8: Architectural Footprint of YOLOv8

YOLOv8 retains the basic structure of YOLO, consisting of three parts (Backbone, Neck, Head), but significantly modifies each part.

- Backbone: YOLOv8 integrates an enhanced version of the Cross Stage Partial (CSP) node module to overcome gradient redundancy, simplifying computation and enhancing feature reuse. This Backbone efficiently extracts hierarchical feature maps representing both low-level textures and high-level semantic information, thereby optimizing speed and accuracy while maintaining representational capabilities [7].
- Neck: YOLOv8 uses an optimized version of PANet (Path Aggregation Network) combined with FPN (Feature Pyramid Network). This optimizes the flow of feature information from Backbone to Head, resulting in more efficient multi-scale feature synthesis. This improvement plays a crucial role in helping the model detect objects of different sizes and scales, especially in complex contexts with small or closely spaced objects. It is one of the reasons why YOLOv8 is so powerful in practical traffic control problems [7].
- Head: YOLOv8 switches from an anchor-box-based method to an anchor-free bounded box prediction method. This change simplifies the prediction process, reduces the number of hyperparameters, and decreases computational complexity. Furthermore, this method makes the model more flexible, increasing its adaptability to objects with diverse aspect ratios and sizes, instead of just the three boxes as before [7].

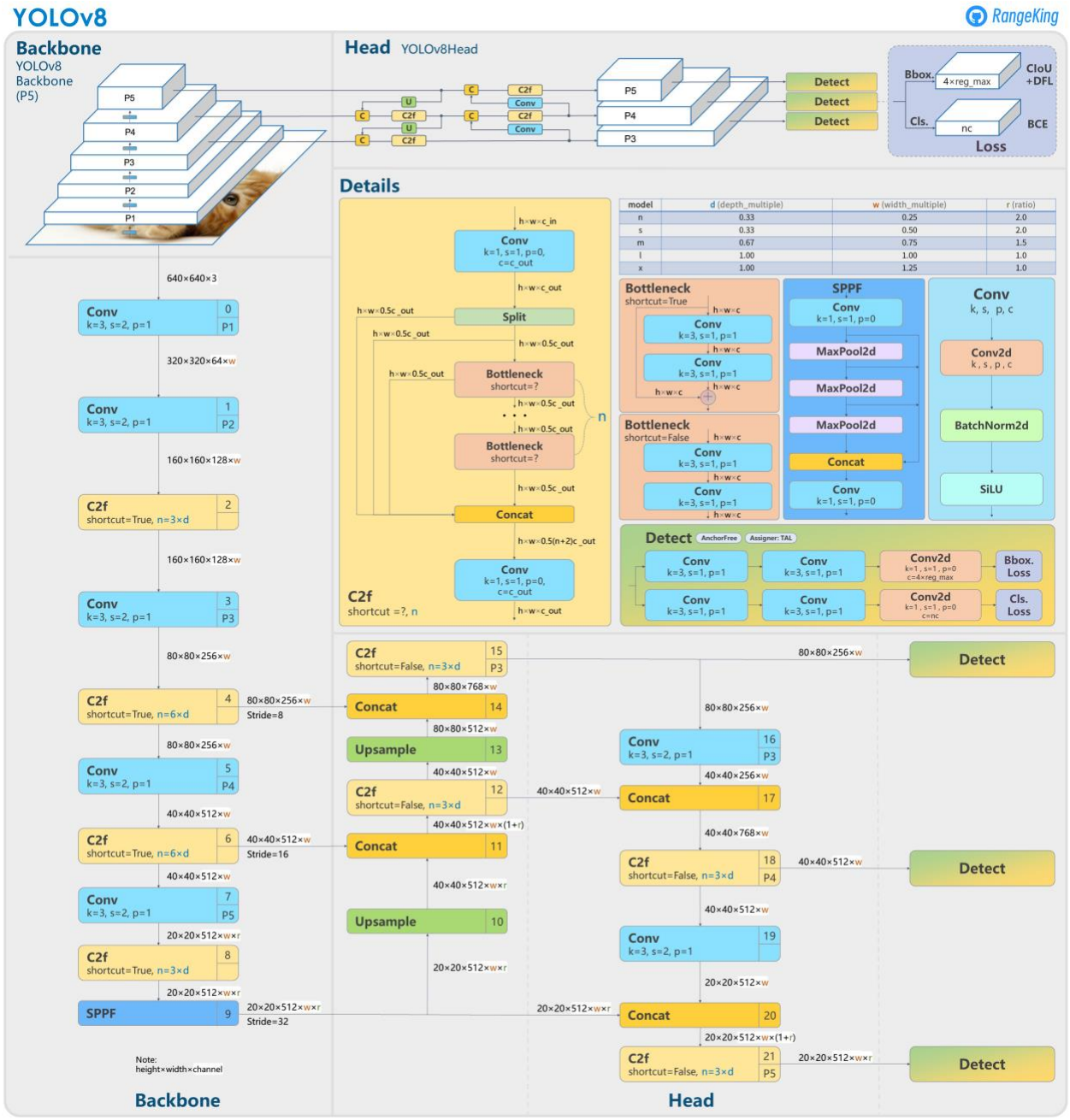

Figure 9: Model Structure of YOLOv8

Loss function

The loss function in YOLOv8 includes three core components [7]:

- Focal Loss: Used for classification tasks to assign greater weights to samples that are difficult to classify. This directly addresses the class imbalance problem commonly found in real-world datasets, while significantly enhancing the ability to detect small or hidden objects that are often in the minority.

- IoU Loss: The Intersection over Union (IoU) Loss function focuses on calculating the degree of bounding box deviation. It helps refine and improve the accuracy of bounding box predictions, allowing the model to pinpoint object locations in images as accurately as possible.
- Objectness Loss: Ensures that the model only focuses attention on areas of the image that have a high probability of containing objects, thereby improving the overall detection capability of the entire network.

Performance Comparison

Metric	YOLOv5	YOLOv8
mAP@0.5	50.5%	55.2%
Inference Time	30 ms/image	25 ms/image
Training Time	12 hours	10 hours
Model Size	14 MB	12 MB

2.5 Related Work

Parking-space occupancy detection. Early deep-learning approaches formulate parking occupancy as a binary classification problem on image patches cropped around each pre-defined parking space. Amato et al. [12] trained a compact CNN on the CNRPark-EXT and PKLot datasets for decentralized parking lot occupancy detection, and Acharya et al. [13] combined deep features with an SVM classifier for real-time image-based occupancy detection. These methods achieve high accuracy on regular, well-marked parking layouts, but they require every space to be delineated and calibrated in advance, and they degrade when a vehicle is parked diagonally, straddles two spaces, or when spaces are only partially visible – exactly the situations targeted by this thesis.

ROI-based vehicle localization with bounding boxes. A second family of methods runs a general-purpose object detector (e.g., Faster R-CNN [7] or YOLO [9]) over the whole frame and decides occupancy by intersecting the detected axis-aligned bounding box with the ROI. Because an axis-aligned box contains a considerable amount of background around the vehicle body, the overlap between the box and the ROI does not faithfully reflect the physical occupancy of the ground area. Oriented bounding-box detection, such as the RoI Transformer of Ding et al. [14], reduces this

background inclusion by predicting rotated rectangles; however, a rotated rectangle around the vehicle body still differs from the vehicle footprint on the ground plane, especially under the strong perspective distortion of oblique surveillance cameras.

Segmentation-based approaches. Instance segmentation methods such as Mask R-CNN [8] produce pixel-accurate vehicle masks and can, in principle, support precise geometric reasoning. Their main drawbacks in the surveillance setting are the higher computational cost, which conflicts with the latency requirements of continuous 24/7 monitoring, and the fact that the segmented mask covers the whole visible vehicle body (including parts that overhang the ground, such as the roof or hood seen from an oblique angle) rather than the ground contact region that actually determines whether the vehicle occupies a parking space.

Keypoint-based vehicle representation. Keypoint (pose) estimation has been integrated into single-stage detectors, notably YOLO-Pose [15], and is available in the pose variants of YOLOv8 [11] and YOLOv11 [16]. Most existing applications concern human pose estimation; applying the keypoint formulation to vehicles by detecting the four wheel-ground contact points, and then coupling the resulting footprint polygon with a rule-based polygon-intersection algorithm for the final ROI occupancy decision, is the approach adopted in this thesis. Compared with the three families above, this hybrid design retains the recognition power of deep learning while keeping the decision stage interpretable, calibration-free with respect to individual parking spaces, and robust to diagonal parking, boundary crossing, and partial occlusion.

3 SYSTEM ANALYSIS AND DESIGN

3.1 Problem Description and System Requirements

3.1.1 The Problem

In urban management practice, determining parking spaces, managing empty spaces in parking lots, or monitoring vehicles entering priority lanes are crucial problems. Traditional methods often rely on manual observation via CCTV screens or the use of on-site physical sensors, which are expensive and difficult to maintain, not to mention the cost and difficulty in maintaining the sensors themselves.

The system designed in this project focuses on automating the monitoring process using computer vision. The specific problem includes:

- Defining the Region of Interest (ROI): Instead of checking the entire frame, the system allows users to define specific spatial areas (e.g., a parking space, a bus lane segment) through a provided list of angular coordinates.

- Prediction and Analysis: The system uses a deep learning model to detect the presence of vehicles and calculate the geometric relationship between those vehicles and the defined ROI (Region of Interest) area.

System Input and Output Data:

- Input: Images or frames extracted from high-angle surveillance video/cameras. ROI configuration file (JSON format) containing the coordinates of 4 corner points for each monitored area (e.g., [(x1, y1), (x2, y2), (x3, y3), (x4, y4)]).
- Output: A data dictionary (JSON) reflecting the state of each ROI area. True if a vehicle is present in the area and False if the area is empty. Additional information includes the boundary box coordinates and the intersection area index (IoU/Coverage) for verification purposes.

3.1.2 System Requirements

For the system to be applicable for 24/7 monitoring, the following non-functional requirements must be ensured:

Processing Speed (Performance/FPS):

- Latency: The model inference time (from receiving an input image to returning the ROI status result) must be less than 100 ms per image. The end-to-end processing time, which additionally includes image loading, result visualization, and JSON export, may be higher depending on the deployment environment.

Accuracy:

- Correct Recognition Rate: Due to the problem related to management (potentially calculating parking fees), the accuracy (Precision) must be above 90% to avoid reporting a parking error when the space is empty (False Positive).
- Fault tolerance: The system must operate stably under changing lighting conditions (day/night), weather conditions (snow/rain), and different camera angles thanks to the characteristic extraction power of the YOLOv8 model.

3.2 System Flowchart

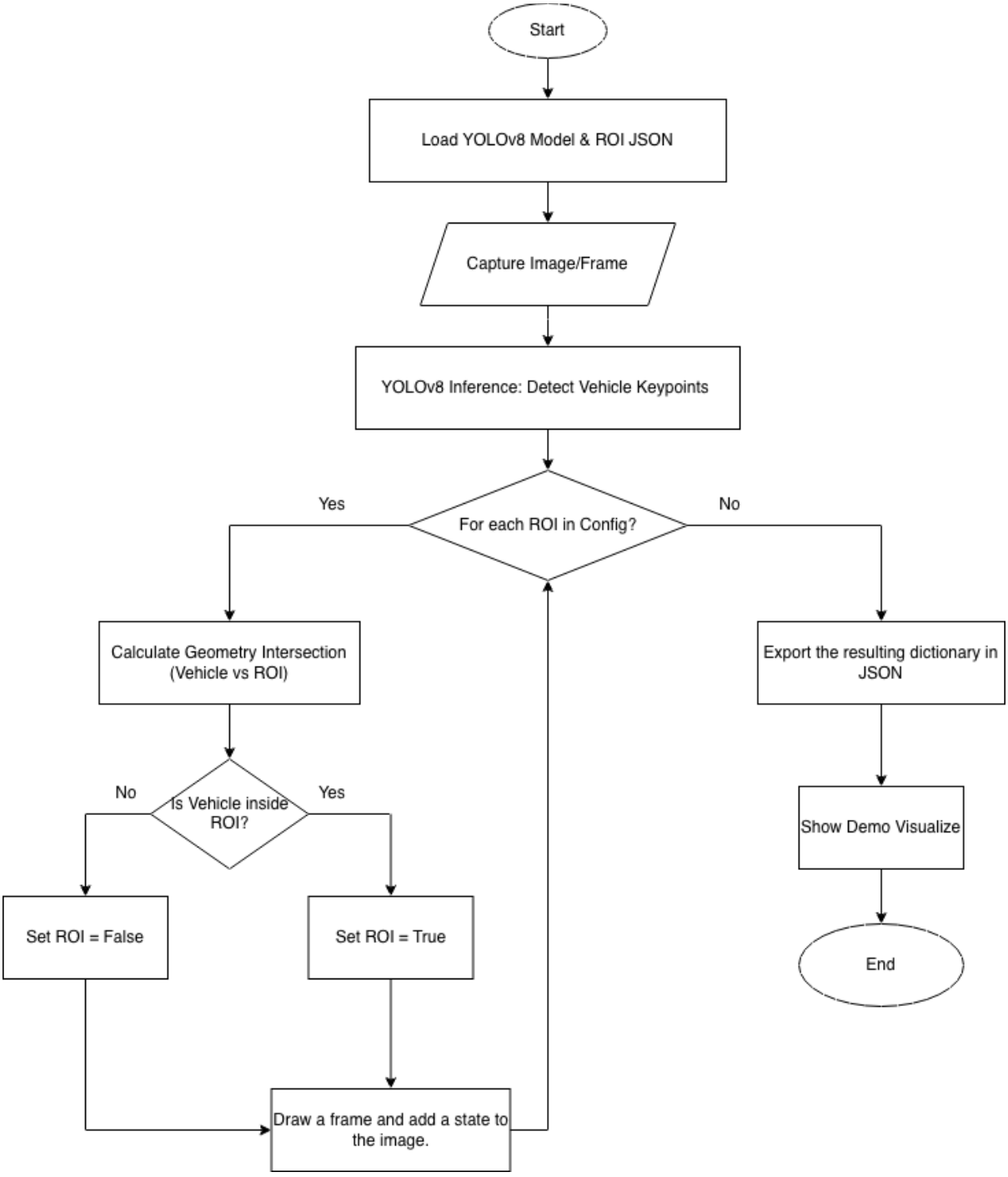

Figure 10: Flowchart

Step 1: System Initialization

First, the program loads the pre-trained YOLOv8 model from the .pt file. Simultaneously, the system reads the JSON ROI configuration file, which defines the regions of interest using the coordinates of points forming a polygon.

Step 2: Input Data Acquisition

The system receives images or frames from video as input data for processing.

Step 3: AI Inference Implementation

The frames are fed into the YOLOv8 model to detect vehicles. The model predicts the keypoints of the vehicle instead of just using a rectangular outline, helping to accurately determine the vehicle's position and orientation in the image.

Step 4: ROI Region Verification

The system sequentially verifies each ROI defined in the configuration file.

Step 5: Calculate Geometric Intersection

For each ROI, the system calculates the intersection between the vehicle polygon (created from keypoints) and the ROI region polygon.

Step 6: Determine Region Status

If the intersection is large enough, exceeding the allowed threshold, the vehicle is determined to be on the ROI (True).

If there is no intersection or the intersection is very small, the region is determined to be empty, meaning there are no vehicles on the ROI (False).

Step 7: Update and Display Results

The status of each region (color and label) is drawn directly onto the image frame for easy user observation.

Step 8: Export Results

All ROI statuses are aggregated and exported as JSON data. The system then proceeds to process the next image.

4 IMPLEMENTATION AND EXPERIMENTAL RESULTS

4.1 Dataset Construction

4.1.1 Data Collection

Data is an extremely important component, directly impacting the generalization capabilities of deep learning models. Therefore, data preparation is a crucial step in this project. For this project, the author constructed a dataset using real-world sources:

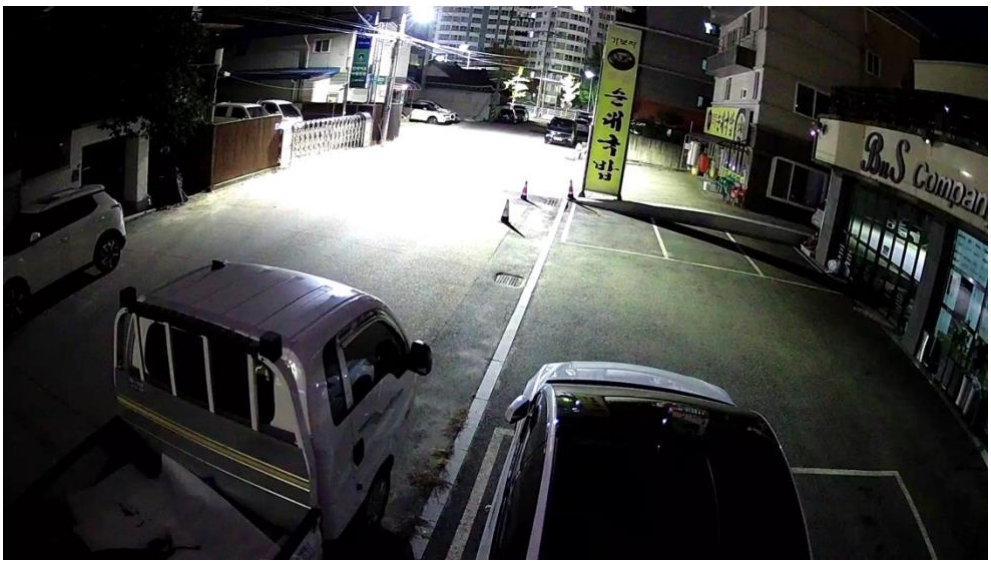

Figure 11: Image in the dataset

- Data Sources and Context: All image data was extracted from CCTV systems installed in outdoor parking lots in South Korea in 2025. The data is highly realistic and up-to-date, reflecting current traffic trends.
- Camera Characteristics: The cameras are mounted in fixed positions with diagonal viewing angles. The fixed camera angle facilitates the establishment of ROI regions and minimizes errors.
- Diversity of Environmental Conditions: The cameras operate 24/7, allowing the author to collect data in all weather conditions, including both good and bad weather. To ensure accuracy, the data also includes various environmental conditions such as rain, snow, sunshine, etc.
- Lighting conditions: Including daytime (direct sunlight, complex shadows) and nighttime (artificial light from high-pressure lamps, causing glare or noise).

- Severe weather conditions: The dataset contains images of vehicles in rain, fog, and especially snow (common in South Korea at the end of the year). Training on data with snow helps the model learn to recognize Keypoints even when the vehicle's surface is partially covered.
- Quantity and format: A total of nearly 900 images were selected and extracted. These images were then standardized to the appropriate format and resolution to prepare for the subsequent labeling process.

In total, the annotated images contain 4,800 labeled vehicle instances, corresponding to 19,200 annotated wheel keypoints (four per vehicle); 12,291 of these keypoints (64.0%) are directly visible wheel points with observed coordinates, while the remaining points correspond to wheels occluded by the vehicle body or by adjacent vehicles and were annotated at their estimated positions. The images originate from two main fixed camera sites, BNS_001 (51.8% of the images) and BNS_002 (43.6%), which together account for 95.4% of the dataset; the small remainder (4.6%) consists of auxiliary frames from a few other cameras and is included to increase scene diversity. In terms of lighting, 77.6% of the images are daytime scenes and 22.4% are nighttime scenes captured under infrared (near-monochromatic) illumination. Regarding the collection procedure, video streams were recorded from the surveillance cameras over a period of approximately 10 days and then extracted into individual frames; duplicate and near-duplicate frames were subsequently filtered out before the remaining images were labeled. This filtering step limits near-duplicate content between consecutive frames, so that the selected images differ substantially in scene content.

4.1.2 Data Labeling and Keypoint Annotation

My project aims to answer the question of whether the vehicle is within the defined ROI (Region of Interest). Therefore, labeling must not only identify the vehicle and its position but also be used for geometric calculations, specifically determining the vehicle's position within the ROI.

Tools Used: Labelme

The system uses Labelme, a popular open-source software that allows for various labeling methods, from rectangles (bounding boxes) and polygons to keypoints. Labelme supports exporting JSON format containing detailed coordinates of each point, which is very convenient for conversion to YOLOv8 format.

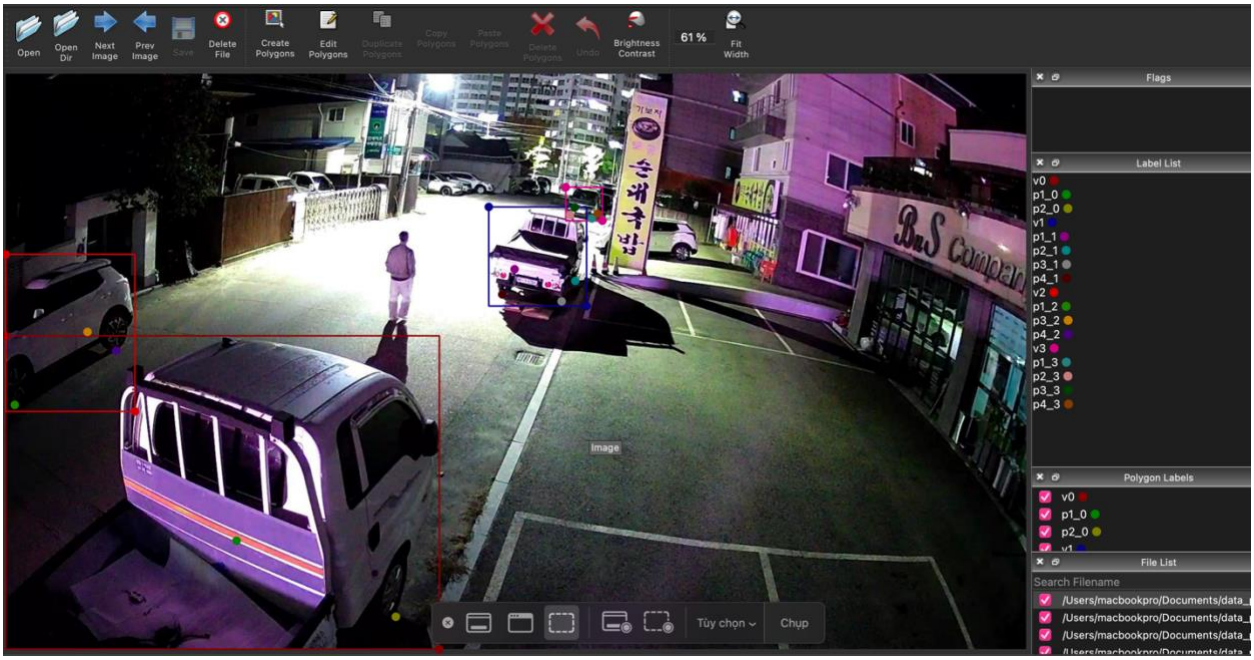

Figure 12: Labelme

Labeling Method

As explained above, labeling requires identifying the specific location of the vehicle. Therefore, to increase the accuracy of the question of whether the vehicle is within the ROI, each vehicle in the image is labeled in two parts:

- Bounding Box: A rectangular box is drawn around the entire vehicle. This helps the YOLOv8 model perform Object Detection, identify the spatial area containing the vehicle, and classify the object.
- Keypoint Marking: Four keypoints are marked at the four lower corners of the vehicle (corresponding to the positions of the four wheels on the ground). These four points form a polygon representing the vehicle's "footprint" on the road surface. This is the core database for the system to calculate the intersection area with the ROI in the utils.py file, helping to eliminate errors when only the front or rear of the vehicle protrudes into the designated area but the wheels remain outside.

Figure 13: Bounding box and Keypoints

Output Data Format

It requires two inputs:

- Image: The path to an image in which vehicles are to be detected.
- ROI Dictionary: A JSON file containing a dictionary of ROI coordinates. Each ROI is a key-value pair where the key is the name of the ROI, and the value is a list of four coordinates representing the bounding polygon (e.g., [(x1, y1), (x2, y2), (x3, y3), (x4, y4)]).

4.1.3 Data Partitioning

To ensure objectivity in evaluating the model's performance and avoid overfitting, the total dataset of nearly 900 images was divided into three independent sets according to the common ratio in Deep Learning problems: 80:10:10.

The specific division was performed as follows:

- Training Set (80%, equivalent to approximately 720 images): This is the largest dataset, used for the YOLOv8 model to update the weights during backpropagation. This set contains all variations in weather conditions (sun, rain, snow) and lighting in South Korea so that the model can learn the stable features of the vehicles.

- Validation Set (10%, equivalent to approximately 90 images): Used to adjust hyperparameters and monitor the model's convergence during training. This dataset helps the system decide when to stop training early if accuracy no longer improves, preventing the model from mechanically memorizing the training set.
- Test Set (10%, equivalent to 90 images): This is a completely new dataset that the model has never encountered during the learning process. The Test Set is used to provide final evaluation metrics such as Precision, Recall, and mAP, accurately reflecting the system's performance in real-world deployments at parking lots. To preserve the independence of the three subsets, duplicate and near-duplicate frames had been filtered out before partitioning (see Section 4.1.1), so that near-identical frames from the same camera and time window do not appear in different subsets.

4.2 Model Training

4.2.1 Experimental Environment

- Device: MacBook Pro (13-inch, 2019, Two Thunderbolt 3 ports)
- Processor (CPU): 1.4 GHz Quad-Core Intel Core i5
- Memory (RAM): 8 GB 2133 MHz LPDDR3
- Graphics (GPU): Intel Iris Plus Graphics 645 1536 MB

4.2.2 Hyperparameter Configuration

Parameters	**Setting Values**
Model Version	yolov8n-pose.pt, yolov11n.pt
Image Size	640×640
Epochs	100
Batch Size	16
Optimizer	AdamW
Augmentation	Enable

4.2.3 Training Results & Loss Analysis

4.2.3.1 YOLOv8

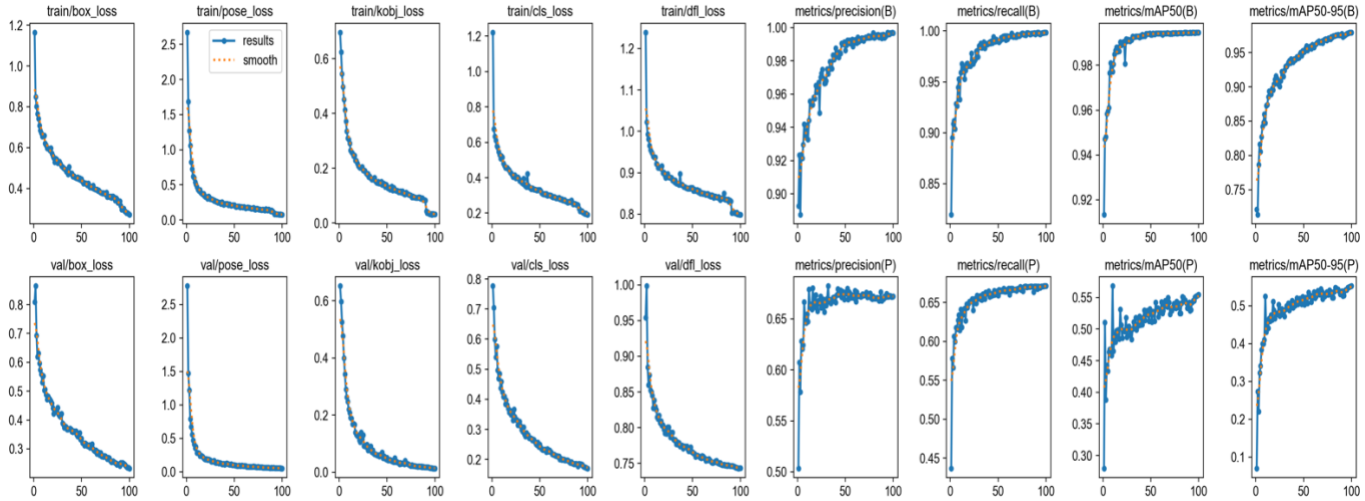

Figure 14: Result yolov8

Loss Analysis

The YOLOv8 training process over 100 iterations (epochs) showed that the model had extremely stable convergence.

Convergence and Overfitting: All loss functions on the validation set (val/box_loss, val/pose_loss, val/kobj_loss, val/cls_loss, val/dfl_loss) showed smooth decline trajectories that closely followed the decline of the training set (train/).

Notably, no validation loss graph showed any signs of increasing again at the end of the process, confirming that the model was not overfitted and maintained excellent data generalization capabilities.

Convergence Speed Specification: For the pose feature recognition task, pose_loss and kobj_loss recorded very steep declines in the first 20 epochs.

This demonstrates that the YOLOv8 architecture quickly grasps the spatial structure of keypoints in the early stages, before entering a phase of gradual refinement.

Performance Evaluation (Training Results)

Object Detection (Bounding Box - B): YOLOv8's performance on object localization is excellent. The metrics/precision(B) and metrics/recall(B) scores surged from the very first epochs.

Notably, the average accuracy metrics/mAP50(B) approached the perfect 1.0, and the strictness mAP50-95(B) also rose to extremely high levels, reaching over 0.95 in the final epochs.

Pose Estimation (P): The results for the pose estimation task were average to good. The metrics/precision(P) index rose from 0.50 in the early epochs and converged around ~0.68.

The metrics/mAP50(P) index reached approximately 0.55 at the end of the training period, while metrics/mAP50-95(P) was slightly above 0.50.

4.2.3.2 YOLOv11

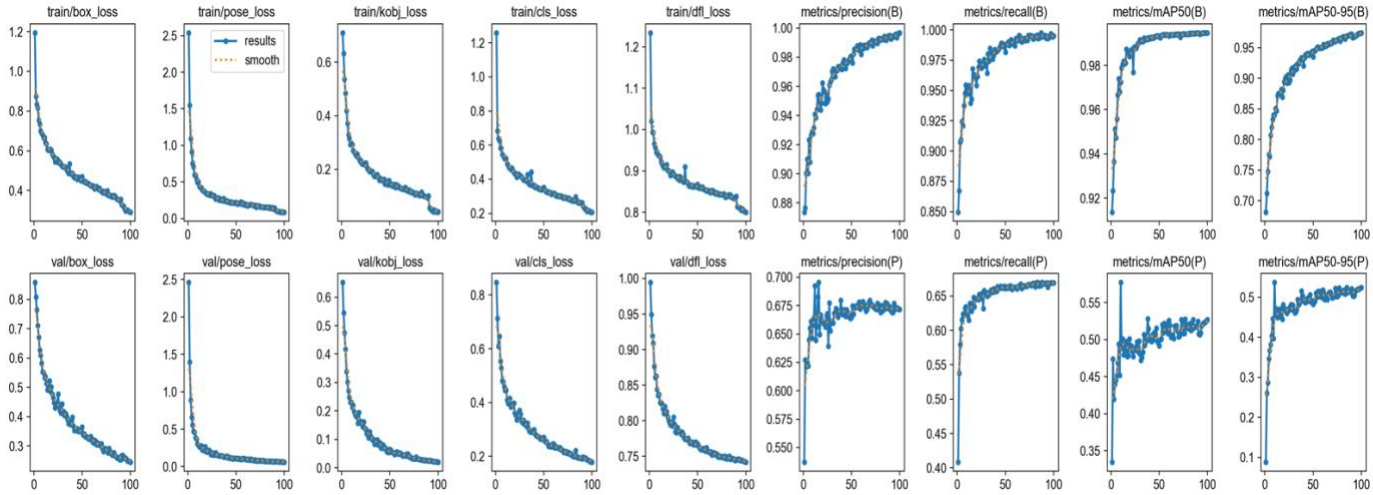

Figure 15: Result yolov11

Loss Analysis

The optimization process of the YOLOv11 model also proceeded smoothly and clearly demonstrated the accuracy of the new network architecture.

Similar to YOLOv8, the loss function curves on the val/ set of YOLOv11 continuously and steadily declined throughout 100 epochs without any upward curve.

The close alignment between the loss function of the train set and val once again confirms the model's good overfitting resistance.

The classification loss (val/cls_loss) and distribution loss (val/dfl_loss) indices decreased smoothly, showing that the model's reliability in object classification was continuously strengthened through each iteration.

Performance Evaluation (Training Results)

Object Detection (Bounding Box - B): The ability to locate and identify objects continues to be maintained at a very high level. The mAP50(B) graph forms a smooth converging curve and almost reaches 1.0.

The mAP50-95(B) index also closely follows the 0.95 mark, demonstrating the superiority of YOLOv11 in basic detection problems.

Pose estimation (Pose - P): For the keypoints problem, YOLOv11 shows some variation during the learning process. The metrics/precision(P) index converges clearly and fluctuates within a narrow range around 0.675 (clearly shown by the narrowed y-axis from 0.550 - 0.700).

However, the composite metrics/mAP50(P) index shows strong fluctuations in the early stages and finally settles at approximately 0.52 - 0.53, slightly lower than the ideal limit.

4.2.3.3 Comparison of YOLOv8 and YOLOv11

From experimental data extracted after 100 epochs, we can evaluate the performance correlation between the two architectures, YOLOv8 and YOLOv11, on the same problem:

Core Similarities:

- Optimization and Overfitting Resistance: Both YOLO versions demonstrate excellent training stability. The loss curve trajectories of both models are almost identical in terms of slope and convergence value, and both avoid overfitting.
- Bounding Box Task Performance Reaches Saturation: There is no significant difference in object detection. Both YOLOv8 and YOLOv11 solve the bounding box localization problem with near-perfect accuracy (mAP50(B) approximately 1.0).

This shows that for typical object detection tasks, both architectures have reached their highest performance limits.

Differences in Pose Estimation Task:

The difference between the two architectures only becomes apparent when faced with the more complex task of estimating the pose point (Pose):

- In terms of accuracy (mAP): YOLOv8 shows slightly better performance on this dataset, with the mAP50(P) reaching approximately 0.55, while YOLOv11 only reaches around 0.52 - 0.53.

- In terms of local stability: YOLOv11 shows slightly better stability in the precision(P) measure, with the graph oscillating with a narrower amplitude in the last epochs. However, this stability does not help YOLOv11 surpass YOLOv8 in overall mAP.

Conclusion and Research Directions:

Experiments show that upgrading the architecture from YOLOv8 to YOLOv11 does not provide a significant performance leap for this specific pose estimation task after 100 epochs (both only showed mAP50(P) fluctuation in the range of 0.52 - 0.55).

Therefore, instead of continuing to search for a solution by changing the network architecture version, a more feasible and effective optimization direction for further research includes: applying specific data enhancement techniques for keypoints (Pose Data Augmentation), addressing data imbalance issues, or increasing the number of epochs combined with a learning rate scheduling strategy to help the model break through its current limitations.

4.2.4 Confusion matrix analysis

4.2.4.1 YOLOv8

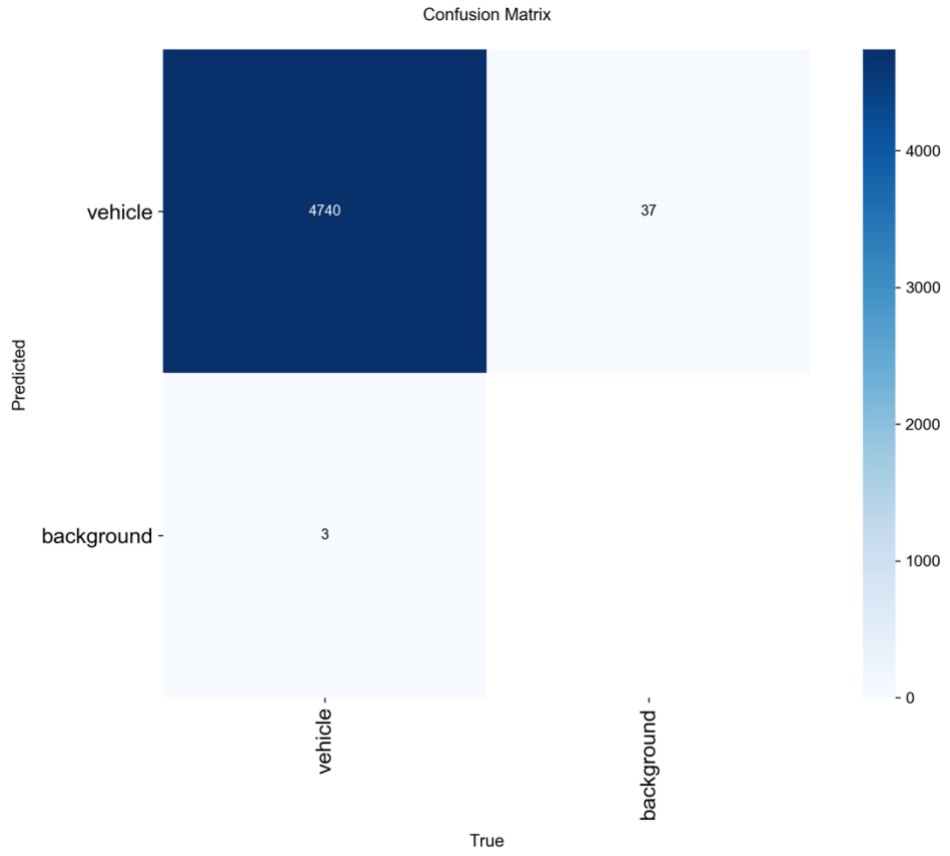

Figure 16: Confusion matrix YOLOv8

The confusion matrix provides a visual and detailed view of the model's classification performance for a specific object class (in this case, vehicle) and background. Based on the YOLOv8 results, we have the following analyses:

- True Positive (TP): The model demonstrated absolute excellence by correctly predicting 4740 cases as vehicles out of a total of 4743 (4740 + 3) actual vehicles. This demonstrates the model's extremely high recall sensitivity.
- False Negative (FN): Only 3 cases of actual vehicles were misidentified as backgrounds by the model. This number is extremely small compared to the dataset, indicating that the risk of missing objects in YOLOv8 is extremely low.

- False Positive (FP) Rate: The model had 37 instances of misidentifying background details as vehicles. Although this is the largest source of error for the model, at 37/4740, its impact on overall accuracy (Precision) is negligible.

4.2.4.2 YOLOv11

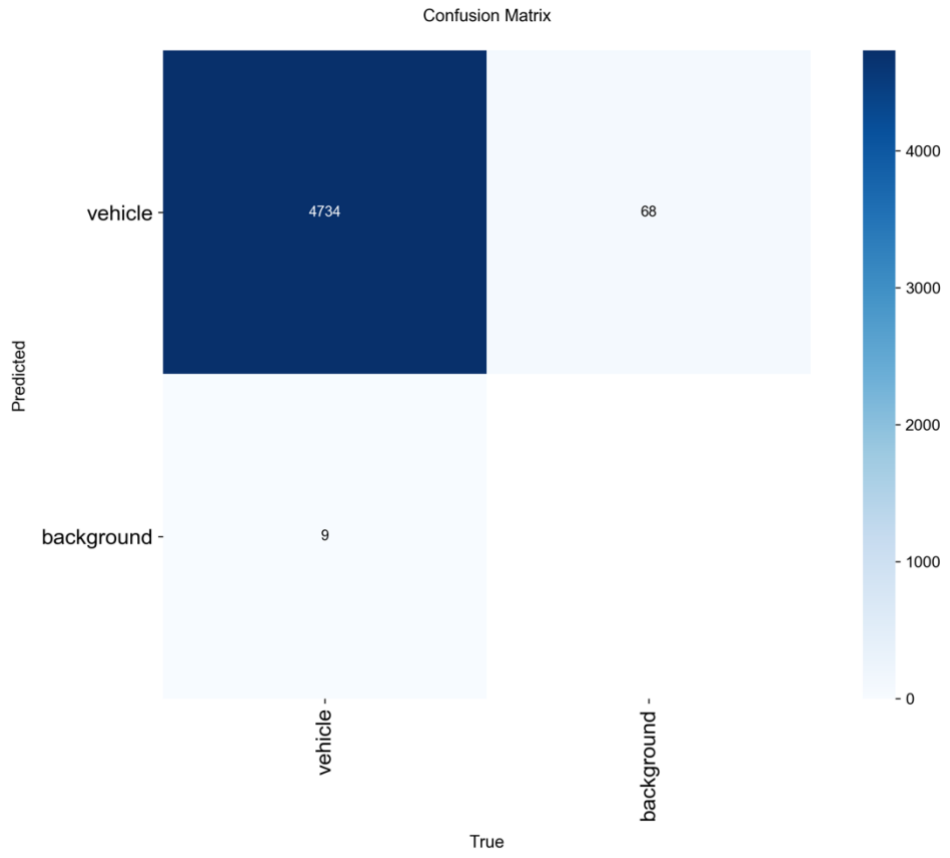

Figure 17: Confusion matrix YOLOv11

For the YOLOv11 model's confusion matrix, object classification results remain very high, however, there is a slight shift in the error distributions:

- True Positive (TP) Rate of Identification: The model accurately predicted 4734 cases as vehicles. A number that is still incredibly impressive and close to perfect.
- False Negative (FN): There were 9 instances where vehicles were misidentified as backgrounds. Although this increased compared to the previous model, the false negative rate is still within the acceptable limits for real-world recognition tasks.

- False Positive (FP): The model mispredicted backgrounds as vehicles in 68 cases. This is the most notable limitation of this model in distinguishing the boundary between objects and background noise.

4.2.4.3 Conclusion

Similarities: Both YOLOv8 and YOLOv11 excel at vehicle detection. The common point between the two is the much higher tendency to make False Positive errors (mistaking the background for a car) compared to False Negative errors (missing a car).

This suggests that the network architecture tends to be "sensitive" (leaning towards trying to catch the object at all costs) rather than "cautious" (only reporting when extremely certain).

The difference: Although the accuracy graph (mAP) in the previous analysis showed saturation and equivalence, when viewed through the Confusion Matrix, YOLOv8 shows a complete dominance in prediction purity:

- YOLOv8 missed fewer vehicles (3 cases compared to 9 cases for YOLOv11).
- YOLOv8 was less affected by background noise, with nearly half the number of false alarms (FP) compared to YOLOv11 (37 cases compared to 68 cases).

In conclusion, on this specific dataset, YOLOv8 performs more efficiently and smoothly than YOLOv11 in the classification task. YOLOv8 demonstrates better differentiation between vehicle features and background features. The fact that YOLOv11 (a newer architecture) generates more noise (FP) may be due to the overly complex feature extraction structure of the network, leading to oversensitivity to background textures in this dataset.

4.2.5 System Implementation

4.2.5.1 Project Structure

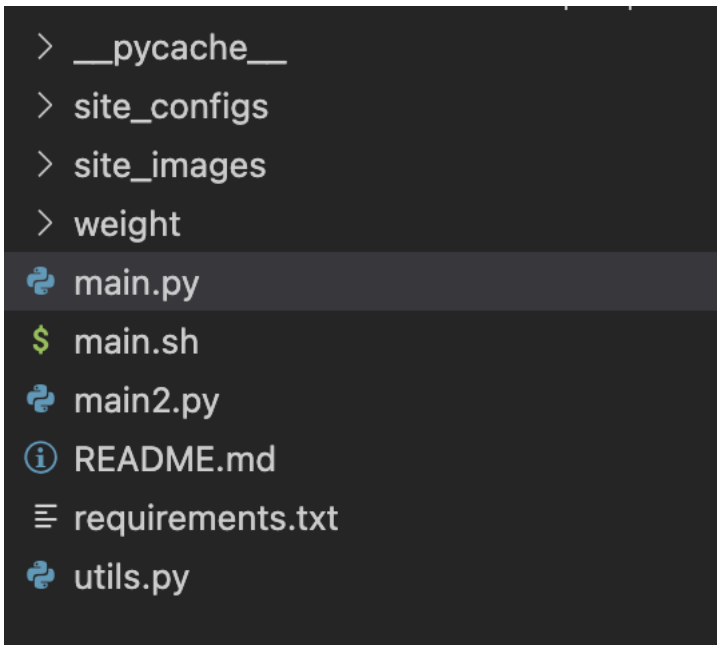

Figure 18: Project Structure

To ensure modularity and ease of maintenance, the system's source code is divided into files with specialized roles, with the two core files being main.py and utils.py:

- main.py (Main Coordination): This is the entry point of the program. This file is responsible for setting input parameters through the parse_arguments() function using the argparse library, including: configuration file path (--config_path), model weight path (--weight_path), confidence threshold (--conf_thresh), and intersection area threshold (--iou_thresh). main.py will initialize an object from the VehicleDetection class to load the YOLOv11/YOLOv8 model and start the entire recognition processing flow.
- Utils.py (Utility and Logic Processing): This file acts as a library containing core algorithm processing functions. This includes functions for extracting vehicle position features such as get_keypoints_list(), a function for handling missing points calculate_missing_point(), and spatial geometry mathematical functions (using the shapely library) such as calculate_polygon_iou() and check_vehicle_in_roi() to calculate vehicle position within the parking space.

- The weight/ and site_configs/ folder: Store the optimized model weight files and JSON files defining the ROI coordinates for each specific parking space.

4.2.5.2 ROI Configuration

The parking lot space management system uses a JSON configuration file (roi_dict). Each zone of interest is defined as a convex polygon based on the coordinates of the corner points:

- Loading the configuration: The load_config function in main.py reads the JSON file to determine the list of locations to be monitored.
- Polygon definition: These coordinates are then converted into Polygon objects using the shapely library for use in geometric interference operations.

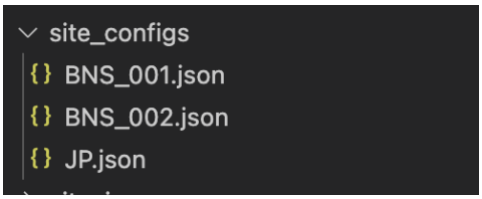

Figure 19: ROI config

Camera 1

Figure 20: Camera 1

Camera 2

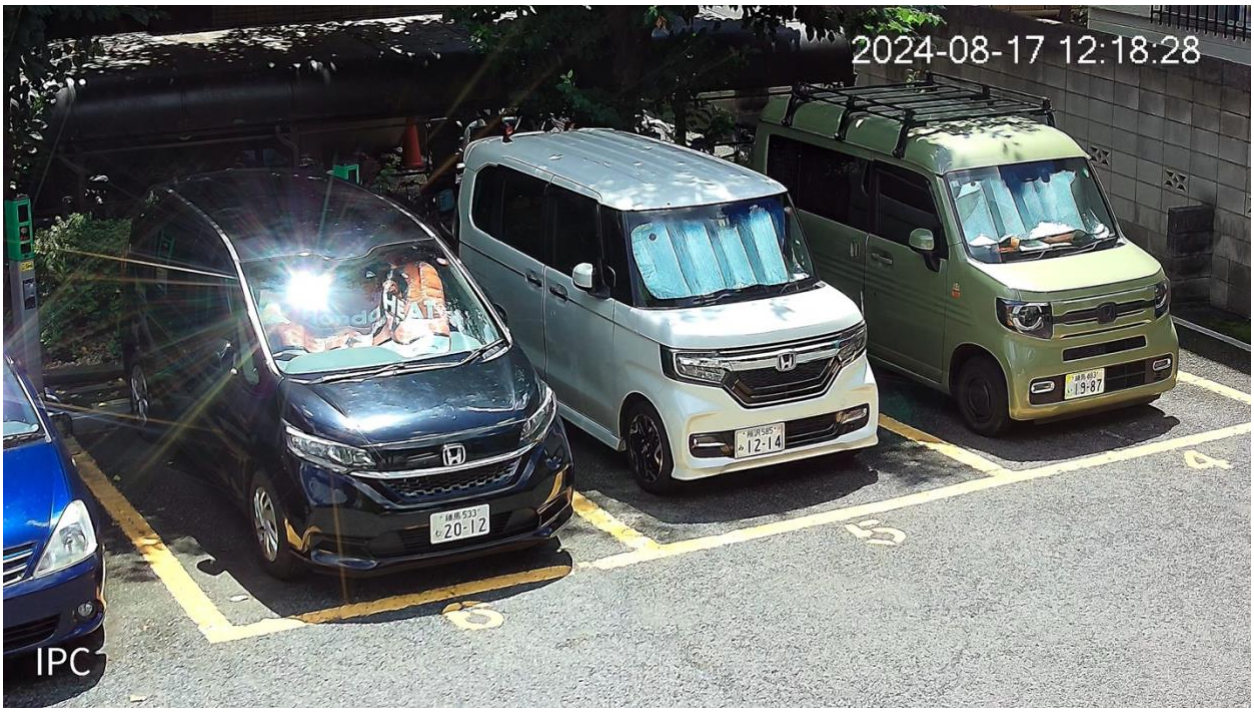

Figure 21: Camera 2

Camera 3

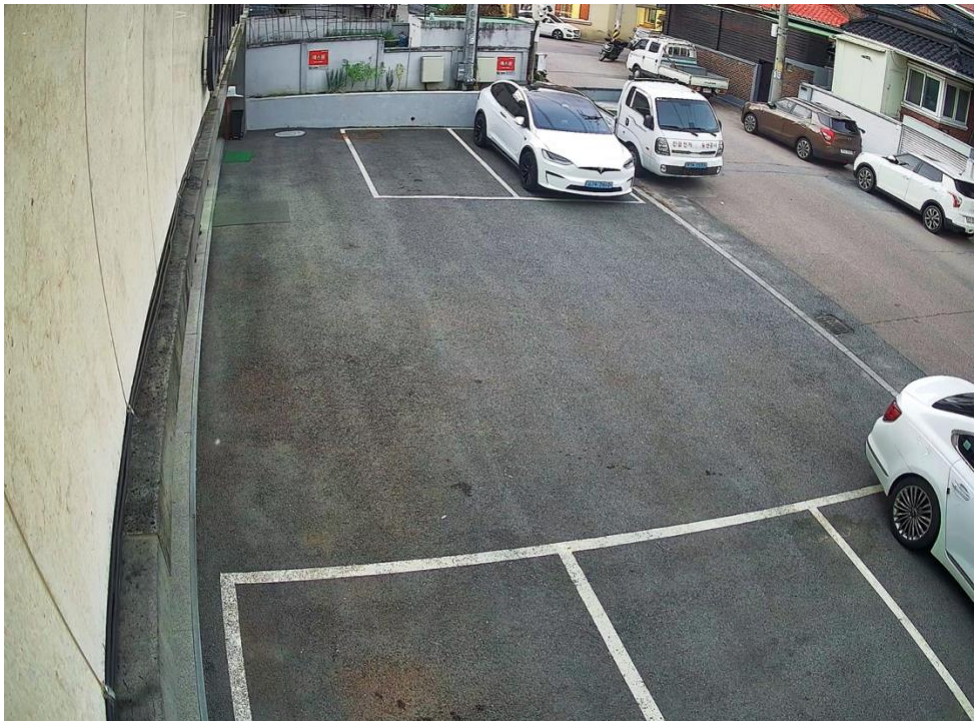

Figure 22: Camera 3

4.2.5.3 Geometric Logic Algorithm

Missing Point Reconstruction

In this project, this is considered a core component used to transform and calculate the model's results into parking state information (True/False). The algorithm is designed to handle real-world problems such as partially obscured objects or errors in keypoint prediction. This is what differentiates this project from other typical problems.

In real-world parking situations, a wheel or corner of a vehicle may be obscured by an adjacent vehicle or due to the vehicle being parked in a corner. Therefore, the `calculate_missing_point` function is implemented to increase the system's reliability.

- Logic: If the model predicts at least 3 out of 4 keypoints with high confidence (conf > conf_thresh), the system will automatically calculate the coordinates of the fourth point based on the geometric properties of a parallelogram (assuming the vehicle's chassis is missing).
- Vector formula: If point P_0 is missing, it is restored based on the vector of the remaining three points:

$$P\_0 = P\_3 + (P\_1 - P\_2)$$

This mechanism ensures that the vehicle's "footprint" polygon is always fully formed to perform area calculations, instead of completely discarding incomplete recognition results.

```
def calculate_missing_point(k_conf, k_xy, conf_thresh):
    # Find the index of the point with confidence below the threshold
    missing_idx = next((i for i, val in enumerate(k_conf) if val <
    conf_thresh), None)

    if missing_idx is None:
        return k_xy  # No missing points found

    # Calculate the missing point based on its index
    if missing_idx == 0:
        k_xy[0] = [k_xy[3][0] + (k_xy[1][0] - k_xy[2][0]), k_xy[3][1]
        + (k_xy[1][1] - k_xy[2][1])]
    elif missing_idx == 1:
        k_xy[1] = [k_xy[2][0] + (k_xy[0][0] - k_xy[3][0]), k_xy[2][1]
        + (k_xy[0][1] - k_xy[3][1])]
    elif missing_idx == 2:
        k_xy[2] = [k_xy[1][0] - (k_xy[0][0] - k_xy[3][0]), k_xy[1][1]
        - (k_xy[0][1] - k_xy[3][1])]
    elif missing_idx == 3:
        k_xy[3] = [k_xy[0][0] - (k_xy[1][0] - k_xy[2][0]), k_xy[0][1]
```

Figure 23: Missing Point Reconstruction

Occupancy Ratio

To calculate whether a vehicle is within the parking area, instead of using the traditional IoU (Intersection over Union), the author uses the Intersection over Vehicle Area variant to assess the extent to which the vehicle enters the ROI. That is, the system will have two polygons: one polygon of the parking area, and one polygon of the vehicle. The system will calculate what percentage of the vehicle's polygon intersects with the parking area.

- Create polygons: Use the shapely library to construct two polygons: P_{ROI} (parking area) and P_{vehicle} (area of contact surface of the vehicle from 4 keypoints).
- Calculate intersection area: Determine the area that the vehicle actually covers within the ROI. Formula for calculating Score:

$$Score = \frac{Area(P_{vehicle} \cap P_{ROI})}{Area(P_{vehicle})}$$

Figure 24: Calculate intersection area

This eliminates the situation where a vehicle passes by (only the frame of the vehicle goes over the ROI) but the wheels haven't actually entered the designated area.

4.2.5.4 Post-processing

Confidence Thresholding

Any prediction result with a confidence score below conf_thresh (default 0.5) is immediately discarded. This step helps eliminate "noise" or non-vehicle objects with similar shapes in low light or heavy snow conditions.

Non-Maximum Suppression (NMS)

This is a crucial technique for handling cases where a vehicle is represented by multiple overlapping bounding boxes in a model.

- Mechanism: NMS considers all bounding boxes with overlaps greater than a defined threshold. It only retains the box with the highest confidence score and discards the rest.
- Role: Ensures that each vehicle in the parking lot is represented by only one unique set of 4 Keypoints, avoiding duplicate calculations that cause the system to report an incorrect number of vehicles present.

ROI Filtering & Sorting

After calculating the IoU for each pair (Vehicle, ROI), the check_vehicle_in_roi function sorts the results in descending order of the Score. This allows the system to always prioritize identifying the vehicle with the greatest coverage of a parking space, increasing accuracy in situations where vehicles are parked across the line.

4.3 Evaluation and Discussion

After the experimental process and system deployment, the results obtained from the two models, YOLOv8-Pose and YOLOv11-Pose, provided valuable insights into the performance of the vehicle recognition problem using four feature points in a real-world context.

4.3.1 Quantitative Evaluation

Based on the indicators obtained after 100 training epochs, the system achieved very high accuracy in the box detection problem and fairly good stability in the feature point detection problem.

	Metrics	YOLOv8-Pose	YOLOv11-Pose	Difference
Box	Precision (P)	99.7	99.7	0
	Recall (R)	99.8	99.5	-0.3
	mAP50	99.5	99.5	0
	mAP50-95	97.9	97.4	-0.5
Pose(4 keypoints)	Precision (P)	67.2	67.2	0
	Recall (R)	67.1	66.9	-0.2
	mAP50	55.4	52.7	-2.7
	mAP50-95	55.2	52.4	-2.8

Analysis of Results:

Vehicle Detection Capability (Box): Both models achieved near-perfect accuracy with an mAP50 of 99.5%. This confirms the system's ability to reliably identify the presence of cars in parking lots without missing any objects.

Feature Point Recognition Capability (Pose): This is a more difficult task due to the very small pixel size of the four wheel corners. Experimental results show that YOLOv8-Pose is performing slightly better than YOLOv11-Pose by about 2.7% in mAP50 (55.4% vs. 52.7%).

4.3.2 Discussion of Experimental Performance

The fact that YOLOv8 achieved slightly better results than YOLOv11 in the Pose Estimation problem on this specific dataset can be explained by the following aspects:

- Architectural Stability: With a medium-scale dataset (nearly 900 images) and fixed camera angle, the YOLOv8-Pose architecture showed very high stability. The pre-trained weights of version v8 may have been more suitable to the geometric characteristics of vehicles in the context of Korean parking lots compared to version v11.
- Box Match: YOLOv8 achieved higher Recall (0.998) and mAP50-95 (0.979) scores, indicating a slightly better ability to draw a box that closely fits the vehicle body compared to v11. This indirectly supports more accurate positioning of the 4 Keypoints located inside that box.
- System Requirements: Although there are slight differences, both models achieve mAP for Pose above 50%. According to the processing logic in the utils.py file, this level of accuracy is entirely sufficient for the algorithm to recover missing points (calculate_missing_point) and calculate the area of the intersecting polygon (IoU) stably.

4.3.3 Demo

To run the program, the author will execute the command below.

```
python main.py --image_path 'site_images/JP_TEST.jpg' --config_path 'site_configs/BNS_002.json'
```

Figure 25: main.sh

Figure 26: Result demo

```
image 1/1 /Users/macbookpro/Downloads/RoI_checking/site_images/JP_TEST.jpg: 384x640 4 vehicles, 244.3ms
Speed: 11.4ms preprocess, 244.3ms inference, 12.5ms postprocess per image at shape (1, 3, 384, 640)
Keypoints list: [array([[     2614.4,      1343.6],
       [     1938.9,      1494.5],
       [     1440.3,      1129.5],
       [     2134.3,      1015.5]], dtype=float32), array([[       1328,      1551.3],
       [     675.62,      1674.2],
       [     316.85,      1335.9],
       [     963.48,      1222.7]], dtype=float32), array([[     3761.5,      1201.2],
       [     3097.6,      1298.3],
       [     2389.9,      955.35],
       [     2992.9,      880.79]], dtype=float32)]
iou:  4.9022109177871644e-05
iou:  0.0
iou:  1.0
iou:  0.9994880369239391
iou:  0.0
iou:  0.0
iou:  0.0
iou:  1.0
iou:  0.0
Ảnh đã lưu tại: output_debug.jpg
```

Figure 27: IoU

```
RETURN ROI value dictionary: {'04': [[array([      2361,      324.93,      3837.6,      1355.2], dtype=float32), array([[     3761.5,
  1201.2],
       [     3097.6,      1298.3],
       [     2389.9,      955.35],
       [     2992.9,      880.79]], dtype=float32), 1.0], [array([     1293.7,      282.1,      2751.5,      1539.2], dtype=float32), arr
ay([[     2614.4,      1343.6],
       [     1938.9,      1494.5],
       [     1440.3,      1129.5],
       [     2134.3,      1015.5]], dtype=float32), 4.9022109177871644e-05]], '05': [[array([     1293.7,      282.1,      2751.5,      1
539.2], dtype=float32), array([[     2614.4,      1343.6],
       [     1938.9,      1494.5],
       [     1440.3,      1129.5],
       [     2134.3,      1015.5]], dtype=float32), 0.9994880369239391]], '06': [[array([     260.03,      509.46,      1456.9,      1705.
7], dtype=float32), array([[       1328,      1551.3],
       [     675.62,      1674.2],
       [     316.85,      1335.9],
       [     963.48,      1222.7]], dtype=float32), 1.0]]}
```

Figure 28: ROI value dictionary

Results:

The program took approximately 0.24 seconds to process the image end-to-end. Note that this end-to-end time includes image loading, drawing the visualization, and exporting the JSON result; the pure model inference time, averaged over the test set, is approximately 96.3 ms per image, which is the quantity referred to by the latency requirement in Section 3.1.2. It should also be noted that these times were measured on a CPU-only MacBook Pro without a dedicated GPU (Section 4.2.1), which considerably limits the achievable speed; deployment on GPU hardware with ONNX/TensorRT optimization (Section 5) is expected to reduce the inference time substantially. The image was handled efficiently in 3 steps:

1.Vehicle Search in Image

The system detected a total of 4 vehicles. Three were clearly parked in the middle, and one was partially obscured in the left corner of the screen.

2. Wheel Marking and Noise Filtering

Instead of using a large frame around the vehicle, the system identified 4 wheel points to create a plane.

3. Parking Check

The system used the undercarriage planes of the 3 vehicles to calculate and compare the results with the 3 pre-drawn parking spaces (spaces 4, 5, and 6). The results were:

- Space 4: Black SUV parked perfectly (100%).
- Space 5: White van parked perfectly (99.94%).
- Space 6: Black car (on the left) parked perfectly (100%).

The algorithm is running very well and efficiently. Focusing only on the wheels completely eliminates vehicles that are passing the edge of the camera or are obscured, thus accurately identifying which vehicles are actually parked safely in the correct area.

Web Demo

On the initial web interface, users can select a camera in the left corner of the screen; each camera will have a different ROI file. Then, users select the "Upload" button to upload the image to be processed, select the “Nhận diện phương tiện trong ROI” (Vehicle detection in ROI) button, wait a few seconds, and the resulting image and results for each parking space will appear.

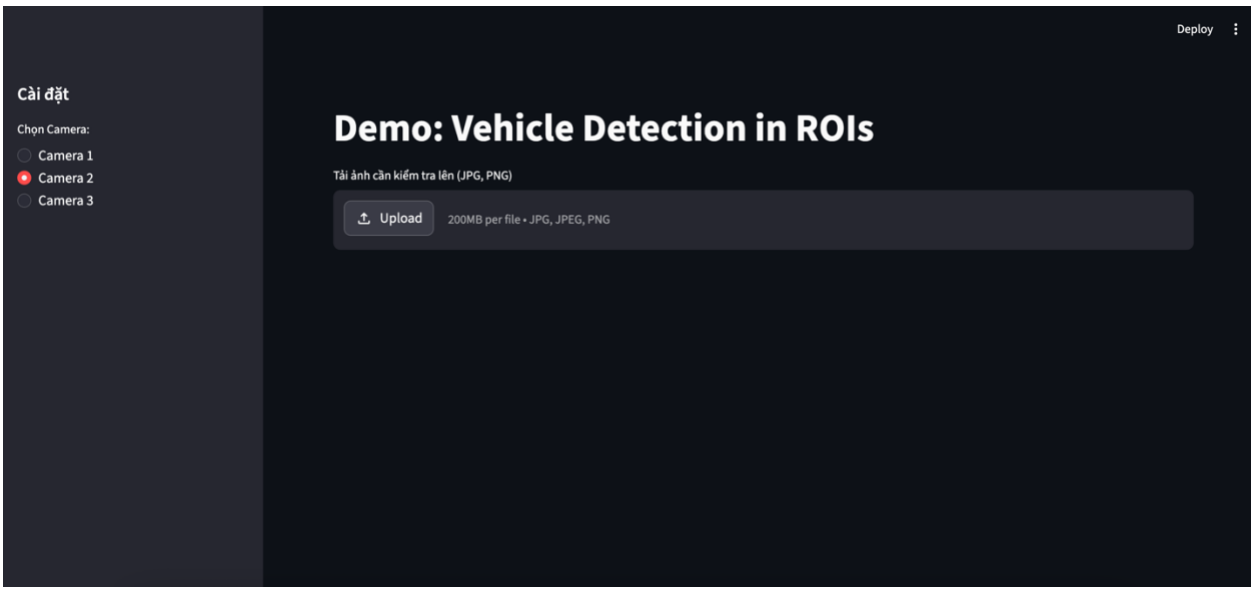

Figure 29: Web demo 1

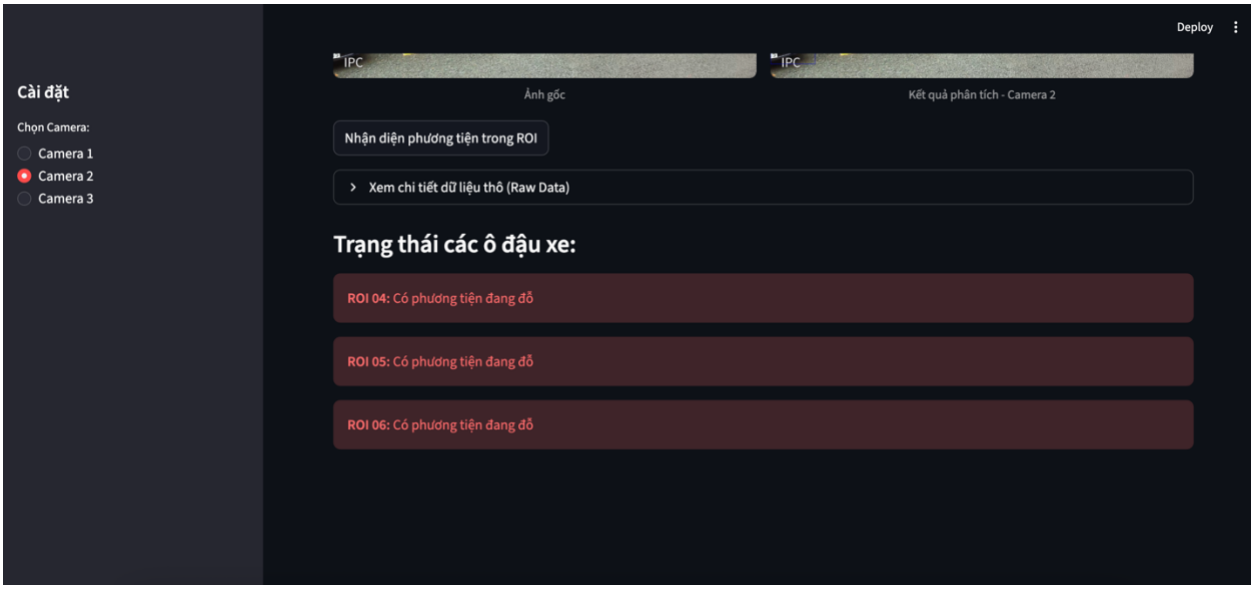

Figure 30: Web demo 2

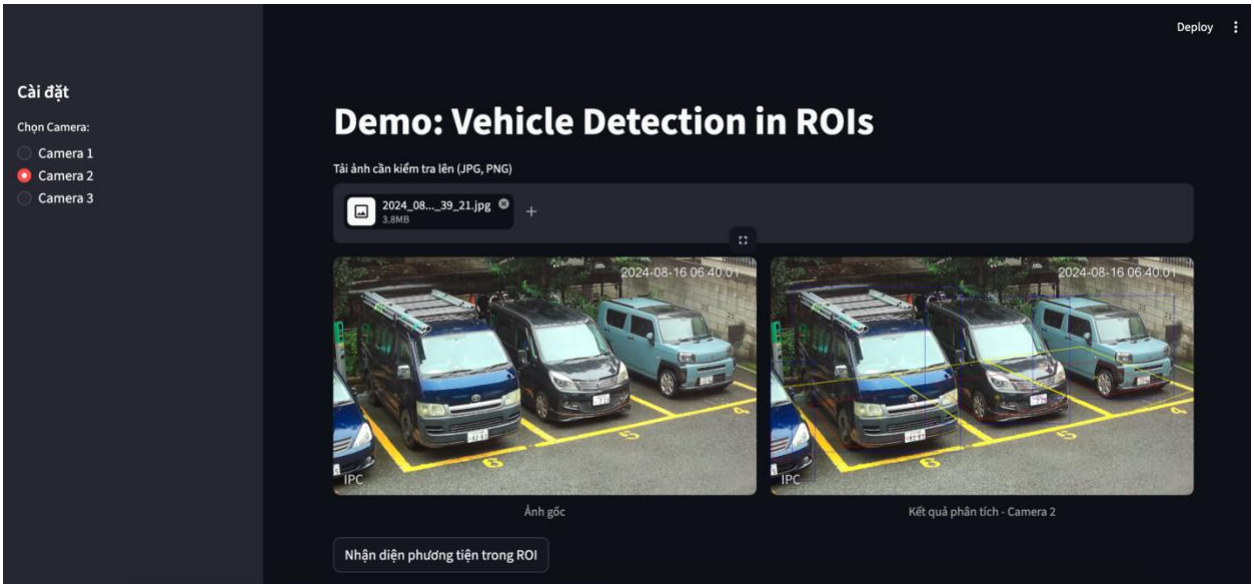

Figure 31: Web demo 3

In addition to the web-based demonstrations above, the figure below shows a detection result on a crowded scene at camera site BNS_001, in which more than ten adjacent vehicles – including partially occluded ones – are detected simultaneously, each with its bounding box and four wheel keypoints. This case illustrates the behaviour of the system in the adjacent-vehicle situations analyzed in Section 4.3.4.

Figure 32: Detection result with multiple adjacent vehicles

4.3.4 Evaluation of the Final ROI Occupancy Decision

While the metrics reported in Section 4.3.1 evaluate the intermediate stage of the pipeline – i.e., how accurately the model detects vehicles and their wheel keypoints – the final output of the system is a binary occupancy decision for each ROI: occupied (True) or empty (False). To evaluate this final decision stage comprehensively, the following protocol was applied over the whole 90-image test set, covering the two camera sites BNS_001 (3 ROIs per image) and BNS_002 (2 ROIs per image).

Each pair (test image, ROI) is treated as one evaluation sample, giving 135 samples for BNS_001, 90 samples for BNS_002, and 225 samples in total. For every sample, the ground-truth occupancy state was annotated, and the system prediction was obtained by running the complete pipeline (YOLOv8-Pose inference, missing-keypoint reconstruction, polygon intersection, and thresholding with conf_thresh = 0.5 and iou_thresh = 0.75). Four outcomes are distinguished: TP (the ROI is occupied and the system reports occupied), FP or false occupancy (the ROI is empty but the system reports occupied), FN or missed occupancy (the ROI is occupied but the system reports empty), and TN (the ROI is empty and the system reports empty). From these counts, Accuracy, Precision, Recall, F1-score, the false-occupancy rate FP/(FP+TN), and the missed-occupancy rate FN/(FN+TP) are computed. The results are summarized in the table below.

Metric	BNS_001	BNS_002	Whole test set
Number of images	45	45	90
Samples (image × ROI)	135	90	225
TP / FP / FN / TN	59 / 4 / 0 / 72	20 / 0 / 1 / 69	79 / 4 / 1 / 141
Accuracy	97.04%	98.89%	97.78%
Precision	93.65%	100.00%	95.18%
Recall	100.00%	95.24%	98.75%
F1-score	96.72%	97.56%	96.93%
False-occupancy rate	5.26%	0.00%	2.76%
Missed-occupancy rate	0.00%	4.76%	1.25%

Over the whole test set, the final occupancy decision achieved an Accuracy of 97.78% (95% Wilson confidence interval: 94.9%–99.0%), with a Precision of 95.18%, a Recall of 98.75%, and an F1-score of 96.93%. The false-occupancy rate – the critical error type for parking-fee applications identified in the system requirements (Section 3.1.2) – was 2.76%, and the missed-occupancy rate was 1.25%. All five errors originated from four distinct failure modes: (i) a vehicle parked across the boundary line between two adjacent spots, whose predicted wheel-keypoint quadrilateral spills into the neighboring ROI (1 case); (ii) an oversized vehicle (a truck) significantly larger than a standard parking-spot footprint, whose keypoints overlap two adjacent ROIs simultaneously (1 case, contributing 2 of the 4 false positives); (iii) a keypoint-recovery error at a low-confidence corner, causing a small spillover into an adjacent ROI (1 case); and (iv) a complete missed detection at night under infrared imaging with glare streaks from a nearby fence, where the parked vehicle produced no detection at all (1 case, the only false negative). Notably, none of the errors were random: each traces back to a specific geometric edge case (line-straddling, size mismatch) or a known model weakness (low keypoint confidence, low-light infrared degradation) rather than to noise in the occupancy decision logic itself.

Because the YOLO inference is deterministic for fixed weights, the variability of the final decision stems from the two decision thresholds. A sensitivity analysis was therefore conducted by sweeping conf_thresh and iou_thresh over the cached inference results, without re-running the detector. The final occupancy decision remained stable over the entire tested range of iou_thresh (0.10–0.95) and over conf_thresh in the range 0.30–0.65; performance degrades only slightly

(Accuracy 97.78% → 97.33%, Recall 98.75% → 97.50%) when conf_thresh ≥ 0.70, at which point the wheel keypoints of one valid vehicle no longer pass the confidence filter and the vehicle is removed from the detection list, indicating that the reported performance is not the product of a finely tuned operating point. The insensitivity to iou_thresh reflects the bimodal nature of the intersection ratio on this test set: for a genuinely parked vehicle the footprint–ROI overlap is close to complete, whereas for a vehicle outside the ROI it is close to zero, so the binary decision rarely depends on the exact threshold value – a stability property of the footprint-polygon design itself.

These results also clarify the relationship between the moderate keypoint-detection mAP50 (≈55%) and the quality of the final binary decision. The mAP metric penalizes any localization deviation of each individual keypoint, whereas the occupancy decision depends only on the intersection ratio between two polygons: a displacement of a wheel keypoint by several pixels changes the footprint polygon only marginally and rarely moves the intersection ratio across the decision threshold. In addition, the missing-keypoint reconstruction module recovers an occluded fourth point from the parallelogram property of the remaining three, so the most common failure mode of the pose model (one occluded wheel) does not propagate to the decision stage, and the vehicle-detection stage that gates the whole pipeline operates at near-saturated accuracy (mAP50 (Box) = 99.5%). Empirically, a keypoint mAP50 of approximately 55% is therefore sufficient for the binary ROI occupancy task, as confirmed by the 97.78% final-stage accuracy.

5 Conclusion

After the research, development, and experimentation of the intelligent parking monitoring system, the project has achieved the following important results:

- Regarding the AI model: Two model architectures, YOLOv8-Pose and YOLOv11-Pose, were successfully trained and evaluated on a specialized dataset of nearly 900 images extracted from real-world cameras in South Korea. Both models achieved extremely high accuracy in vehicle detection, with mAP50 (Box) accuracy reaching 99.5% and wheel keypoint detection mAP50 (Pose) of approximately 55%, successfully calculating vehicle detection within the parking area.
- Regarding feature point recognition technique: The system has demonstrated the effectiveness of using 4 Keypoints to identify vehicle footprints instead of using only the traditional outline. Although Pose Estimation is a challenging task, the mAP50 (Pose)

accuracy of approximately 52–55% ensures that the input coordinates are sufficiently reliable for subsequent calculations.

- Regarding the geometric logic algorithm: The utils.py module has been successfully developed with intelligent post-processing functions such as recovering missing points based on geometric properties and calculating the area of polygon intersections, thereby enabling the system to accurately calculate the vehicle's parking position even when the vehicle is parked across the lines or partially obscured. Evaluated end-to-end on the whole 90-image test set (225 image–ROI decision samples), the final ROI occupancy decision reached an Accuracy of 97.78%, a Precision of 95.18%, a Recall of 98.75%, and an F1-score of 96.93%, with a false-occupancy rate of 2.76% and a missed-occupancy rate of 1.25%.
- Data practicality: The project has successfully tested the recognition capability under all specific weather conditions (sun, rain, night, and heavy snowfall) in South Korea.

Limitations

Despite achieving high accuracy, the system still has some performance limitations that need to be considered. Firstly, it is slow processing time. The average pure model inference time, measured over the 90-image test set on the experimental hardware described in Section 4.2.1, is around 96.3 ms/image (equivalent to about 10 FPS); the end-to-end processing time of the demonstration pipeline (including image loading, visualization, and result export) is approximately 0.24 s/image. These figures are strongly affected by the experimental hardware: both training and inference were performed on a MacBook Pro with a CPU-only configuration (Intel Core i5, integrated Intel Iris Plus graphics, no dedicated GPU), as described in Section 4.2.1. This speed is not suitable for hard real-time systems as it can cause latency for the processing server. As proposed in the future development directions below, converting the model to the ONNX format and optimizing it with NVIDIA TensorRT on GPU hardware is expected to reduce the inference time to below 20–30 ms/image, satisfying the latency requirement with a large margin. Secondly, framework dependence is one of the limitations of the system. Running the model directly in the .pt weighted format requires a full PyTorch environment, making deployment on Edge devices with limited hardware resources difficult. Thirdly, although the final ROI occupancy decision was evaluated over the whole 90-image test set (Section 4.3.4) and achieved high accuracy, the evaluation covers

only two fixed camera sites; more challenging cases – such as adjacent vehicles, partially occupied spaces, severe occlusion, strong shadows, snow-covered markings, and changes in camera perspective – should be included to further establish the generalization capability of the system. Fourthly, only two closely related architectures (YOLOv8-Pose and YOLOv11-Pose) were compared; a stronger evaluation should additionally include at least one alternative approach, such as instance segmentation, oriented bounding-box detection, or another keypoint-estimation model.

Future Development Directions

To improve the system and expand its future application capabilities, the project proposes the following development directions:

- Integrating an Object Tracking module: Applying algorithms such as ByteTrack or BoT-SORT to link objects between frames. Tracking will help stabilize parking status and minimize instantaneous errors from the AI model.
- Deploying a Cloud/Edge system: Optimizing the model to run on Edge devices (such as NVIDIA Jetson) or deploying it to a Cloud platform for centralized management of multiple parking lots simultaneously via a Web/Mobile interface.
- Optimizing speed with ONNX and TensorRT: * Convert the model from .pt format to ONNX (Open Neural Network Exchange) to increase flexibility when deploying on various hardware platforms.
- Integrate NVIDIA's TensorRT (TRT) to optimize graph computation and fully utilize the power of Tensor cores on the GPU. This is expected to significantly reduce inference time (down to below 20-30ms/image).

Bibliography

[1] A. F. Joseph Redmon, "YOLOv3: An Incremental Improvement," 2018.

[2] J. D. T. D. J. M. Ross Girshick, "Rich feature hierarchies for accurate object detection and semantic segmentation," 2013.

[3] K. H. R. G. J. S. Shaoqing Ren, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," 2016.

[4] G. G. P. D. R. G. Kaiming He, "Mask R-CNN," 2018.

[5] S. D. R. G. ,. A. F. Joseph Redmon, "You Only Look Once: Unified, Real-Time Object Detection," 2016.

[6] U. Almog, "YOLO V3 Explained," 2020.

[7] M. Yaseen, "What is YOLOv8: An In-Depth Exploration of the Internal Features of the Next-Generation Object Detector," 2024.

[8] C. S. Rina Diane Caballar, "What is computer vision?," [Online]. Available: https://www.ibm.com/think/topics/computer-vision.

[9] G. M. F. P. S. C. J. T. Z. H. Kaifeng Gao, Computer Science Review, 2020.

[10] F. Z. Christopher Kirwan, Smart Cities and Artificial Intelligence, 2020.

[11] A. S. E.-C. I. A.-H. M. Alsakka F., Automation in Construction, 2023.