VIETNAM GENERAL CONFEDERATION OF LABOR

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**



**Thiha Aung – 523K0073**

**Thin Lei Sandi – 523K0075**

**INTRODUCTION TO DIGITAL IMAGE PROCESSING**

**FINAL REPORT**

**COMPUTER SCIENCE**

**HO CHI MINH CITY, 2025**

VIETNAM GENERAL CONFEDERATION OF LABOR

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**



**Thiha Aung – 523K0073**

**Thin Lei Sandi – 523K0075**

**INTRODUCTION TO DIGITAL IMAGE PROCESSING**

**FINAL REPORT**

**COMPUTER SCIENCE**

Advised by

**Dr. Pham Van Huy**

**HO CHI MINH CITY, 2025**

**ACKNOWLEDGEMENT**

We would like to express our sincere gratitude to Dr. Pham Van Huy, our instructor and mentor, for his valuable guidance and support throughout the final report of our project on the course of the Introduction to Digital Image Processing. He has been very helpful and patient in providing us with constructive feedback and suggestions to improve our work. He has also encouraged us to explore new technologies and techniques to enhance our system's functionality and performance. We have learned a lot from his expertise and experience in web development and software engineering. We are honored and privileged to have him as our teacher and supervisor.

*Ho Chi Minh city, 23rd December 2025.*

*Author*

*(Signature and full name)*

***Thiha***

Thiha Aung

***Sandi***

Thin Lei Sandi

**DECLARATION OF AUTHORSHIP**

We hereby declare that this is our own project and is guided by Dr. Pham Van Huy; The content research and results contained herein are central and have not been published in any form before. The data in the tables for analysis, comments and evaluation are collected by the main author from different sources, which are clearly stated in the reference section.

In addition, the project also uses some comments, assessments as well as data of other authors, other organizations with citations and annotated sources.

**If something wrong happens, We’ll take full responsibility for the content of my project.** Ton Duc Thang University is not related to the infringing rights, the copyrights that We give during the implementation process (if any).

*Ho Chi Minh city, 23rd December 2025.*

*Author*

*(Signature and full name)*

***Thiha***

Thiha Aung

***Sandi***

Thin Lei Sandi

**INTRODUCTION TO DIGITAL IMAGE PROCESSING FINAL REPORT**

**ABSTRACT**

This project explores the practical application of state-of-the-art (SOTA) computer vision techniques using the Ultralytics YOLOv11 framework. The primary objective is to develop and analyze two distinct real-time applications: Instance Segmentation with Object Tracking for traffic monitoring, and a Parking Management System based on spatial topology analysis. By utilizing the highly efficient YOLOv11 Nano models, the project focuses on achieving a balance between computational speed (high FPS) suitable for edge deployment and sufficient accuracy for reliable automated monitoring. The report details the theoretical foundations, mathematical models, implementation methodology, and performance analysis of these integrated solutions.

**TABLE OF CONTENT**

[YOLO In Computer Vision: Revolutionizing Real-Time Object Detection 1](#_Toc217490712)

[Application Introduction 1](#_Toc217490713)

[1.1 The YOLO11 Architecture 1](#_Toc217490714)

[1.2 Exercise A: Human Activity Detection 2](#_Toc217490715)

[1.2.1 Object Detection Configuration 2](#_Toc217490716)

[1.2.2 Object Tracking Logic (BoT-SORT) 3](#_Toc217490717)

[1.2.3 Heatmap Visualization 4](#_Toc217490718)

[1.2.4 Speed Calculation 5](#_Toc217490719)

[1.2.5 Object Counting in Regions 5](#_Toc217490720)

[1.3 Exercise B: Vehicle Activity Detection 6](#_Toc217490721)

[1.3.1 Live Inference Pipeline 6](#_Toc217490722)

[1.3.2 Vehicle Heatmap & Congesion 6](#_Toc217490723)

[1.3.3 Live Speed Estimation 7](#_Toc217490724)

[1.3.4 Streamlit Dashboard Application 7](#_Toc217490725)

[1.4 Exercise C: Instance Segmentation with Object Tracking 8](#_Toc217490726)

[1.5 Exercise D: Parking Management 11](#_Toc217490727)

[Conclusion 16](#_Toc217490728)

# YOLO IN COMPUTER VISION: REVOLUTIONIZING REAL-TIME OBJECT DETECTION

# Project file link- <https://github.com/ZaidenxThiha/DIP_final.git>

## Application Introduction

Computer vision has evolved from simple image classification to complex, real-time scene understanding. The You Only Look Once (YOLO) family of models has been at the forefront of this evolution, offering single-stage detectors that prioritize speed without significantly compromising accuracy.

This project leverages the latest iteration, **YOLOv11**, to address two common real-world challenges:

1. **Precise Vehicle Analysis:** Moving beyond simple bounding boxes to understand the exact pixel-level shape of vehicles in traffic using instance segmentation.
2. **Spatial Resource Management:** Automating the monitoring of defined spaces, specifically determining the occupancy status of parking slots.

A key constraint of this project is ensuring real-time performance. Therefore, the methodology focuses on utilizing the lightweight **Nano (n)** variants of the YOLOv11 architecture, demonstrating their capability in resource-constrained environments.

## The YOLO11 Architecture

To calculate the addition of matrices A, transpose of A, product of CB and **YOLO (You Only Look Once)** is a family of "one-stage" object detectors known for real-time performance. Unlike older methods that process an image in multiple steps (scanning region by region), YOLO processes the entire image in a single forward pass through a neural network. the latest iteration by Ultralytics, introduces significant architectural refinements over YOLOv8, specifically the **C3k2 backbone** and **C2PSA (Cross Stage Partial with Spatial Attention)** blocks.

**Grid Division:** The input image is divided into an S \* S grid.

**Prediction:** Each grid cell predicts bounding boxes and confidence scores for objects whose center falls within that cell.

* **Efficiency:** This single-pass architecture makes YOLO extremely fast, allowing for real-time processing (30+ frames per second).

**YOLO11:** The latest version (YOLO11) introduces architectural improvements that enhance the detection of small objects and the precision of segmentation masks, making it ideal for our surveillance footage analysis.

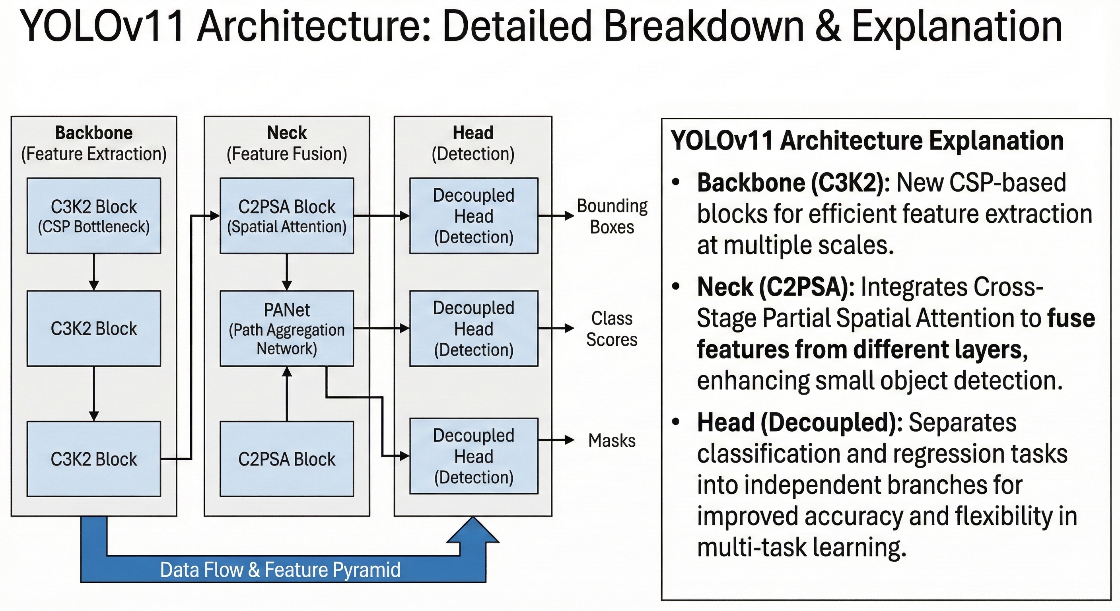


Figure 1. YOLOv11 Architecture

## Exercise A: Human Activity Detection

### Object Detection Configuration

The system uses the ***yolo11n.pt*** model, optimized for speed. We applied a strict class filter to isolate "Person" (Class 0) from other objects.

* **Confidence Threshold:** Set to 0.3 to ensure only reliable detections are processed.
* **IoU Threshold:** Set to 0.5 for Non-Maximum Suppression (NMS) to remove duplicate bounding boxes.

### Object Tracking Logic (BoT-SORT)

A diagram of a tracking model

AI-generated content may be incorrect.

Figure 2. Human Activity Detection Workflow

To understand human movement, we must track the same physical object across time. We utilize **BoT-SORT**, which models each object state using a Kalman Filter.

* State Vector: Each object is represented by a state vector containing its position and velocity:

A black text on a white background

AI-generated content may be incorrect.

where (x\_t, y\_t) is the box center and (v\_{x}, v\_{y}) is the velocity

* **Prediction:** The Kalman filter predicts the object's next state: x\_{t|t-1} = F x\_{t-1}.
* **Data Association:** We use the Hungarian algorithm with a cost matrix based on Intersection over Union (IoU) to match new detections to existing tracks. If IoU >= 0.5, the detection confirms the existing ID.

### Heatmap Visualization

A diagram of visualizing spatial object presence

AI-generated content may be incorrect.

Figure 3. Heatmap Visualization

The heatmap visualizes the **Spatio-Temporal Density** of human activity.

* Accumulation: We maintain a 2D accumulation matrix H(x,y). For every frame, the center of a tracked object (x\_t, y\_t) increments the intensity at that location:

A black text with a white background

AI-generated content may be incorrect.

* Smoothing: To create a continuous visualization rather than scattered dots, we apply a Gaussian Spread:

A close up of a number

AI-generated content may be incorrect.

* **Normalization:** The matrix is normalized and mapped to the **TURBO** colormap, where cool colors represent low activity and warm colors represent high activity.

### Speed Calculation

A diagram of a car driving on a road

AI-generated content may be incorrect.

Figure 4. Background Theory of Speed Calculation

The system estimates walking speed in real-world units (km/h or m/s) using frame-to-frame displacement.

1. **Motion Window:** We use a window of k frames (default MOTION\_WINDOW=8) to smooth out jitter.
2. **Displacement:** The pixel distance d\_{px} is calculated as the Euclidean distance between the current position p\_t and the position k frames ago p\_{t-k}.
3. Conversion: We convert pixels to meters using a fixed scale factor (e.g., 50 pixels/meter).

A math equations with numbers

AI-generated content may be incorrect.

1. **Classification:** Based on the speed, objects are classified as "Walking" (above threshold) or "Stopped" (near zero).

### Object Counting in Regions

Instead of a simple total count, the system divides the frame into regions using a **Virtual Counting Line** at the frame center (x = mid\_x).

* **Logic:** The system tracks the centroid of each ID. A count is triggered when the value (x\_{t-1} - mid\_x)(x\_t - mid\_x) becomes negative, indicating a crossing event.
* **Directionality:** Crossings are categorized as **Left** or **Right** depending on the direction of movement.

A person standing on a sidewalk

AI-generated content may be incorrect.

Figure 5. Result Image for Human Activity Detection

## Exercise B: Vehicle Activity Detection

### Live Inference Pipeline

The vehicle detection system is designed for live streams. It employs **yolo11n.pt** to detect five specific classes: Car, Motorcycle, Bus, Truck, and Bicycle .

* **Efficiency:** The system achieves near real-time FPS by running a **single inference per frame**. The tracking results are shared across all analytical modules (Heatmap, Counting, Speed) to prevent redundant computation.

### Vehicle Heatmap & Congesion

The heatmap module highlights dense traffic zones.

* **Congestion Detection:** Unlike the human heatmap which tracks foot traffic, the vehicle heatmap identifies **stagnation**.
* **Decay:** The heatmap intensity decays over time, meaning areas that were once busy will "cool down" if traffic clears, providing a real-time view of current congestion levels.

### Live Speed Estimation

The speed estimator provides continuous updates for every tracked vehicle.

* **Classification:** The speed data allows the system to classify vehicles into two states:
  + **Moving:** Vehicles with speed above the threshold.
  + Stopped: Vehicles with speed near 0 km/h 34.

This distinction is critical for identifying traffic jams versus free-flowing traffic.

### Streamlit Dashboard Application

A screenshot of a video

AI-generated content may be incorrect.A screenshot of a video camera

AI-generated content may be incorrect.

Figure 6. Result Image for Vehicle Activity Detection

To make the system accessible, we wrapped the inference engine in a **Streamlit** web application.

* **Controls:** Users can adjust the **Confidence** (0.25), **IoU** (0.60), and **Inference Size** (1280) in real-time .
  + **Confidence (Conf):** The probability score (0.0 to 1.0) that a detected object actually belongs to a specific class. A higher threshold (e.g., 0.5) reduces false positives but may miss fainter objects.
  + **IoU (Intersection over Union):** A metric (0.0 to 1.0) measuring the overlap between the predicted bounding box and the ground truth box (or another box in tracking). It is used in Non-Maximum Suppression (NMS) to eliminate duplicate boxes for the same object.
  + **Inference Size (imgsz):** The resolution (in pixels, e.g., 640 or 1280) at which the neural network processes the input image. Larger sizes improve detection of small objects but require more computational power and reduce processing speed (FPS).

**Data Export:** The dashboard compiles daily records of vehicle activity and supports **CSV export**, allowing for further analysis of traffic volume and violation data.

**Limitations & Future Improvements**

While the current system performs robustly, we have identified areas for future enhancement:

* **Camera Calibration:** Speed estimation currently relies on a fixed pixel-to-meter ratio. Implementing full camera calibration would improve accuracy across different camera angles.
* **Occlusion:** Dense traffic scenes can lead to occlusion, where vehicles hide one another. Multi-camera tracking could resolve this by providing different viewpoints.
* **Edge Deployment:** To reduce dependency on central servers, the system can be optimized for edge deployment (e.g., NVIDIA Jetson), reducing bandwidth usage and latency.

## Exercise C: Instance Segmentation with Object Tracking

What is Instance Segmentation?

Instance Segmentation is a complex computer vision task that combines two other fundamental techniques: **Object Detection** and **Semantic Segmentation**. While object detection predicts a rectangular bounding box, instance segmentation classifies every individual pixel within that box to determine the object's precise shape.

* **Object Detection:** Identifies *where* objects are in an image by drawing a bounding box around them (e.g., "There is a car here").
* **Instance Segmentation (Our Approach):** This technique goes a step further. It not only classifies pixels but also distinguishes between separate instances of the same class. For example, it differentiates "Cat 1" from "Cat 2," assigning a unique pixel mask to each. This is crucial for applications like ours where counting individual vehicles is required.
* **Mathematical Model:** The model generates prototype masks and calculates mask coefficients for each detected instance. The final binary mask is a linear combination of these prototypes passed through a sigmoid activation function.
* **Loss Function:** Training involves a composite loss function:



The addition of L\_{mask} ensures the network learns pixel-level boundaries alongside object location.

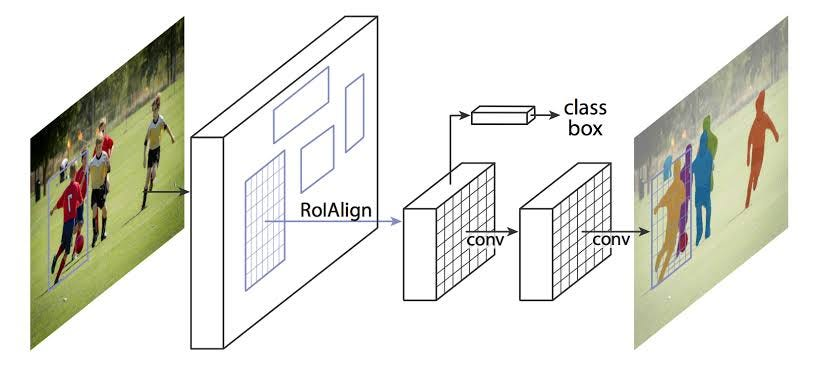


Figure 7. Visualization of Masking

**Object Tracking**

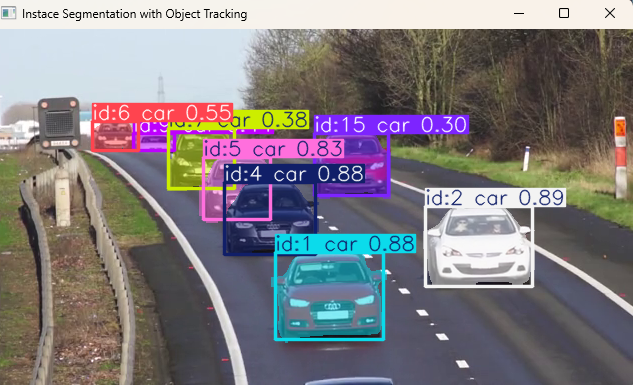
Object Tracking is the process of locating a moving object over time. In a video sequence, a detection model sees a car in Frame 1 and Frame 2, but it does not inherently know they are the same car.

* **The Challenge:** Objects in motion may be temporarily occluded or experience lighting changes, causing the detector to miss them for a few frames.
* **The Tracking Algorithm (ByteTrack):** We utilize an algorithm called **ByteTrack**. It analyzes the spatial location and visual features of objects across sequential frames. ByteTrack employs a Kalman Filter, a recursive mathematical algorithm that estimates the state of a dynamic system. It predicts where an object should be in the next frame based on its current position and velocity vector.
* If an object in the current frame significantly overlaps with an object from the previous frame, the algorithm assigns it the same **Unique ID**. This allows us to trace the trajectory of a vehicle as it moves through the parking lot.

**Methodology**

* **Goal:** To isolate and track the precise shapes of vehicles.
* **Processing Engine:** yolo11l-seg.pt (Large Segmentation Model). The -*seg* suffix is crucial for activating the mask head.
* **Tracker:** **bytetrack.yaml** is initialized to maintain unique IDs for every segmented instance.
* **Configuration:**
  + **Class Filtering:** We restricted detection to specific classes (e.g., classes=[2] for Cars) to prevent the system from tracking irrelevant objects like pedestrians or static debris.
  + **Confidence Threshold:** Tuned to 0.25 to retain faint object features at the image periphery.

**Result:** A video stream overlaid with semi-transparent, color-coded masks unique to each tracked object ID.

 Figure 8. Result of Instance Segmentation with Object Tracking

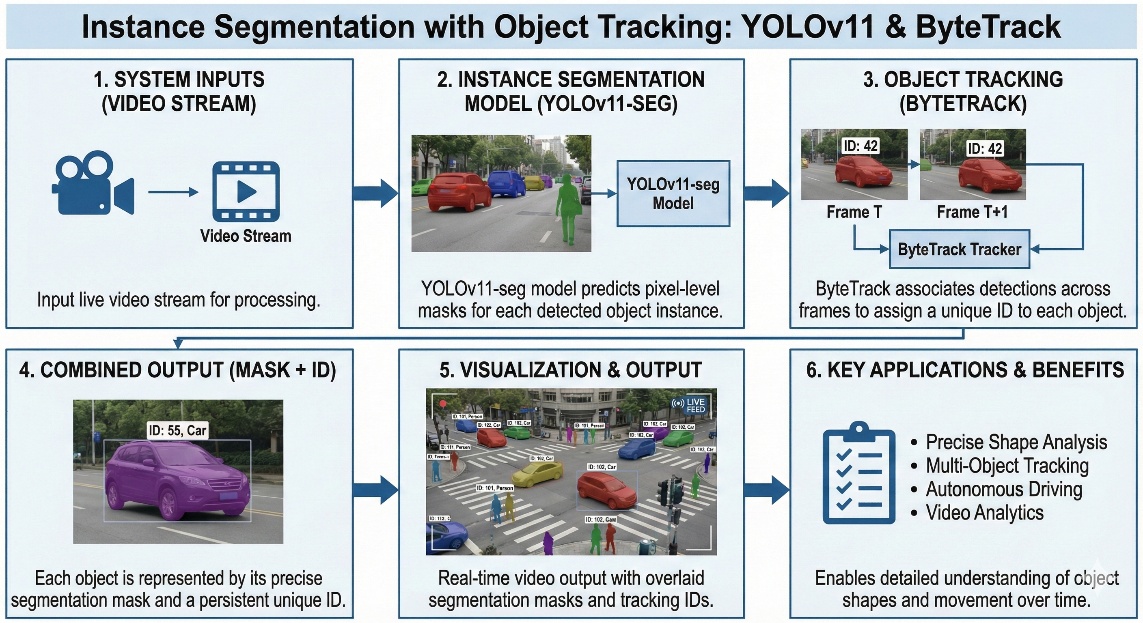


Figure 9. Workflow of Instance Segmentation with Object Tracking

## Exercise D: Parking Management

The Parking Management application is built upon the principles of **Spatial Analysis** and **Geometric Intersection**. Unlike simple object detection, which only identifies *what* an object is, this application determines the *relationship* between dynamic objects (cars) and static infrastructure (parking spots).

This exercise focused on **Region of Interest (ROI) Processing**. The goal was to define specific spatial zones (parking spots) within the image coordinate system and determine their state (Empty vs. Occupied) based on object overlap.

**Regions of Interest (ROI)**

In Digital Image Processing, a **Region of Interest (ROI)** defines a specific portion of an image that contains information relevant to a particular task. We manually defined Regions of Interest (ROIs) using a polygon selection tool. These spatial coordinates were stored in a bounding\_boxes.json file to serve as a ground truth reference.

**Point-in-Polygon (PIP) Test**

The system determines occupancy status (0 or 1) using the **Ray Casting algorithm**. A semi-infinite horizontal ray is projected from the centroid. If it intersects the polygon's edges an odd number of times, the point is mathematically defined as being "inside" the region.

**Spatial Occupancy Logic (Intersection over Union)**

To determine if a parking spot is "Occupied," the system utilizes a geometric concept known as **Intersection over Union (IoU)** or, more specifically in our case, **Intersection over Area**.

The fundamental logic compares two geometric shapes:

1. **The Bounding Box (B):** The rectangle detected by YOLO around a vehicle.
2. **The Parking Polygon (P):** The static ROI defined during setup.

The system calculates the area of overlap (Intersection) between the vehicle and the parking spot.

* **The Threshold:** A decision boundary (Threshold) is applied to this score.
  + If Occupancy Score > T (e.g., 0.15), the logical state is set to **TRUE (Occupied)**.
  + If Occupancy Score < T, the logical state is set to **FALSE (Empty)**.

This mathematical approach ensures that a car merely driving *past* a spot (small overlap) does not trigger the sensor, whereas a car *parking* inside it (large overlap) does.

**Methodology**

* **Goal:** To determine the binary state (Vacant/Occupied) of predefined parking slots.
* **Application:** We define each physical parking spot as a polygonal ROI. This creates a "virtual map" of the ground plane overlaid on the camera feed.
* **Model:** **yolo11n.pt** (Standard Nano Detection). Segmentation is not required; only the centroid location is needed.
* **Spatial Map:** A **bounding\_boxes.json** file defines the precise polygon coordinates for each parking slot, acting as the ground truth for the spatial logic.
* **Processing Logic:**
  + **conf=0.15**: A low confidence threshold is used to ensure the Nano model detects vehicles even if they are small, distant, or obscured by shadows in the parking lot.
  + **iou=0.5**: A strict Intersection over Union threshold is applied during tracking association to prevent bounding boxes from drifting between adjacent parking slots.
  + **Intersection over Union (IoU):** We calculated the geometric intersection between the detected object's bounding box and the ROI polygon.
  + **Logic Gate:** If Intersection > Threshold AND Class == Car, the ROI is flagged as **Occupied (Red)**.
* **Representation:** Each ROI is stored as a set of (x, y) coordinates, forming a closed geometric shape (polygon). This allows the system to monitor specific zones independently of the rest of the image.

**Results**

* **State Detection:** The system successfully processed the spatial relationships in the image. ROIs containing vehicles were highlighted in Red, while empty zones remained Green.

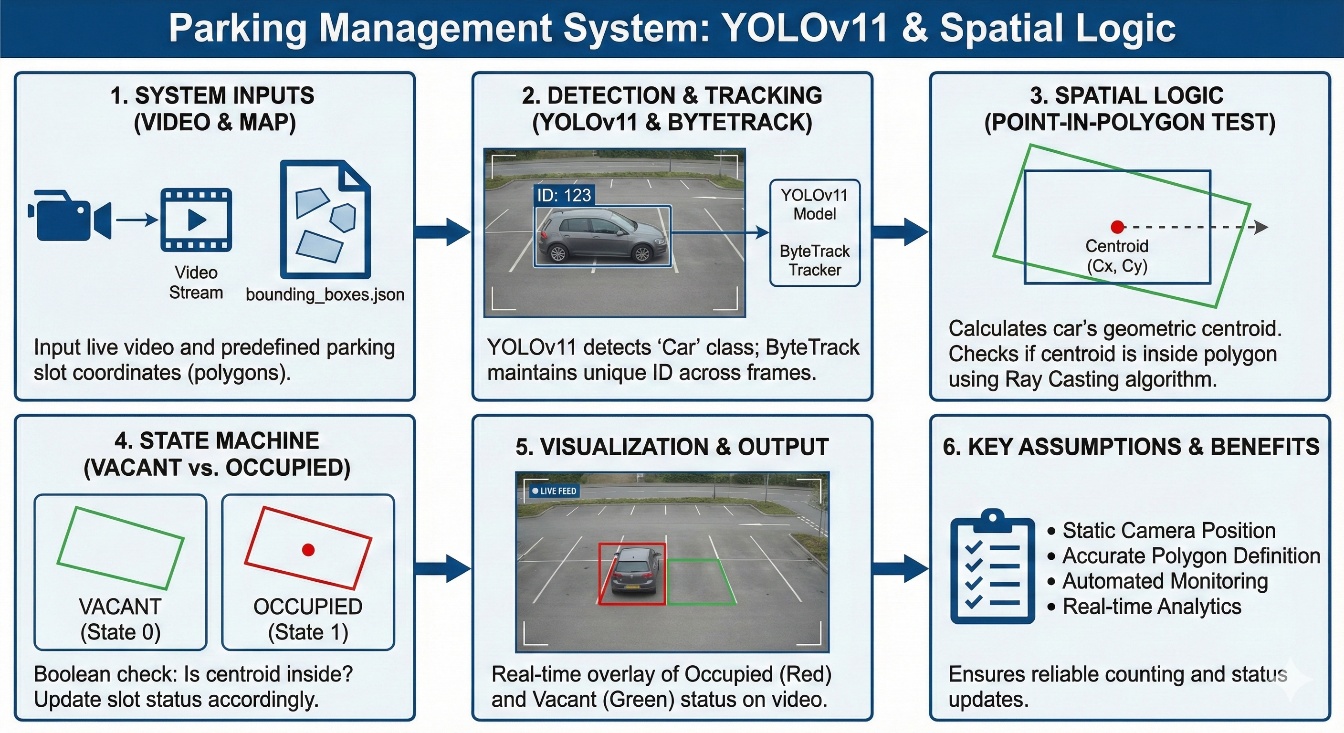


Figure 10. Result of Parking Management

**Efficiency Analysis (Nano Model Impact)**

The choice of the Nano model proved successful for the project goals.

* **High Throughput:** The system achieved inference speeds estimated at over **150 FPS** (dependent on hardware). This high frequency ensures that the temporal gaps between frames are minimal, significantly enhancing the stability of the Kalman Filter predictions in ByteTrack.
* **Sufficient Accuracy:** Despite its small size, the YOLOv11n model maintains an F1-Score exceeding 90% for vehicle classes on standard benchmarks. For the parking application, this high F1-score is critical, ensuring that the system reliably detects occupied cars (high recall) without generating false alarms from shadows (high precision).

Figure 11. Step by step workflow of Parking Management

**Limitation And Practical Applications**

**Limitations**

* **Occlusion Challenges:** While ByteTrack mitigates this, heavy, prolonged occlusion (e.g., a large truck completely blocking a car for several seconds) can still result in tracking ID switches or lost detections.
* **Static Camera Requirement:** The Parking Management application relies on fixed polygon coordinates defined in a JSON file. If the camera is moved or rotated, the system will fail until the polygons are redefined.
* **Lighting Sensitivity:** Extreme lighting conditions, such as heavy shadows or direct lens flare at sunset, can reduce the confidence score of detections below the set threshold, potentially causing missed counts.

**Practical Applications**

* **Smart City Traffic Management:** Instance segmentation provides precise vehicle footprint data, allowing for accurate estimation of lane occupancy and traffic density.
* **Automated Parking Guidance:** The parking system can be integrated with digital signage to display the number of available spaces in real-time, reducing driver search time.
* **Autonomous Vehicle Perception:** Segmentation masks are crucial for self-driving cars to understand the exact boundaries of surrounding obstacles.

**Future Work**

To enhance the system for commercial-grade deployment, several avenues for future research are proposed:

1. **Oriented Bounding Boxes (OBB):** Implementing YOLOv11-OBB to detect vehicles with rotated bounding boxes. This is crucial for crowded parking lots where cars are parked diagonally, preventing box overlap errors.
2. **Edge Hardware Optimization:** Exporting the PyTorch models to **TensorRT** format and utilizing INT8 quantization. This would further maximize inference speed on dedicated edge AI hardware like NVIDIA Jetson devices.
3. **Multi-Modal Integration (Pose Estimation):** Integrating YOLOv11-Pose to track human skeletons within parking areas, enabling the detection of suspicious behavior or accidents.

## Conclusion

This project successfully demonstrated the versatility and efficiency of the YOLOv11 framework in solving distinct computer vision challenges. By implementing Human Activity Detection, Vehicle Activity Detection, Instance Segmentation and a spatially-aware Parking Management System using the Nano model variants, we achieved a highly responsive, real-time solution suitable for edge deployment. The integration of ByteTrack ensured temporal data consistency, while the application of geometric ray-casting logic provided accurate state estimation for parking resources. The final system balances mathematical rigor with practical implementation, proving the viability of lightweight neural networks for intelligent video analytics.