VIETNAM GENERAL CONFEDERATION OF LABOR

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**



**TRAN THI THE NHAN - 523K0047**

**Final Project**

**Digital Image Processing**

**HO CHI MINH CITY, 2025**

VIETNAM GENERAL CONFEDERATION OF LABOR

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**



**TRAN THI THE NHAN - 523K0047**

**Final Project**

**Digital Image Processing**

Advised by

**Dr. Pham Van Huy**

**HO CHI MINH CITY, 2025**

**ACKNOWLEDGEMENT**

We would like to express our sincere gratitude to Mr. Pham Van Huy, our instructor and mentor, for his valuable guidance and support throughout the mid-term report of our project on Building a book management and ordering system on the MERN stack platform. 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 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, 14rd December 2025.*

*Author*

*(Signature and full name)*

**Nhan**

Tran Thi The Nhan

**DECLARATION OF AUTHORSHIP**

We hereby declare that this is our own project and is guided by Mr. 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, 14rd December 2025*

*Author*

*(Signature and full name)*

**Nhan**

Tran Thi The Nhan

**TABLE OF CONTENT**

[Chapter 1: YOLO Definitions 1](#_Toc152077847)

[1.1. YOLO Definitions 3](#_Toc152077857)

[1.2 YOLO Architecture 3](#_Toc152077857)

[Chapter 2: Methods for Specific Applications 2](#_Toc152077850)

[2.1 Live Inference with Streamlit Application 3](#_Toc152077857)

[2.2 Distance Calculation 3](#_Toc152077857)

[2.3 TrackZone 3](#_Toc152077857)

[Chapter 3: Future Work 3](#_Toc152077853)

[REFERENCES 6](#_Toc152077862)

# **Chapter 1: YOLO Definitions**

# Computer vision has revolutionized how machines interpret and understand the visual world. At the forefront of this revolution is YOLO (You Only Look Once), a family of real-time object detection models known for their speed and accuracy. This report details specific applications of the latest iteration, Ultralytics YOLO11, highlighting its versatility in live inference, spatial analysis, and targeted monitoring.

# **1.1. YOLO Definitions**

# **YOLO (You Only Look Once)** is a state-of-the-art, real-time object detection architecture. Unlike traditional two-stage detectors that apply a classifier to various parts of an image multiple times, *YOLO applies a single neural network to the full image*. It divides the image into a grid and predicts bounding boxes and probabilities for each grid cell simultaneously. This "single-shot" approach allows for extremely fast inference speeds, making it ideal for real-time applications.

# **1.2 YOLO Architecture**

# **image_recognition**

### 1.2.1 Input & Preprocessing

* **Standardization:** The model accepts images resized to **448×448 pixels**.
* **Padding:** Aspect ratio is preserved via padding to ensure uniform input dimensions for batch processing.

### 1.2.2 Feature Extraction (Backbone)

* **Deep Architecture:** The core is a CNN comprising **24 convolutional layers** and **4 max-pooling layers**.
* **Efficiency Strategy:** It utilizes alternating **1×1 reduction convolutions** (to compress channels/parameters) and **3×3 convolutions** (to capture spatial patterns), optimizing computational efficiency without losing expressive power.

### 1.2.3 Prediction Head (Output)

* **Fully Connected Layers:** The CNN features are flattened and passed through **2 fully connected layers**.
* **Final Tensor:** The raw output (vector of 1,470 values) is reshaped into a **7×7×30 3D tensor**.
  + **7×7:** Represents the grid cells dividing the image.
  + **30:** Represents the data per cell (2 bounding boxes × 5 parameters + 20 class probabilities).

### 1.2.4 Activation & Optimization

* **Hidden Layers:** Uses **Leaky ReLU** (slope 0.01 for negative values) to prevent "dead neurons" and maintain gradient flow.
* **Output Layer:** Uses a **Linear activation** function to predict raw bounding box coordinates and scores.
* **Regularization:** Incorporates **Batch Normalization** (for stability/speed) and **Dropout** (to prevent overfitting).

# **1.3 Ultralytics YOLO11**

# Ultralytics YOLO11 is the latest advancement in the YOLO series (following YOLOv8). It creates a unified framework for multiple computer vision tasks, including:

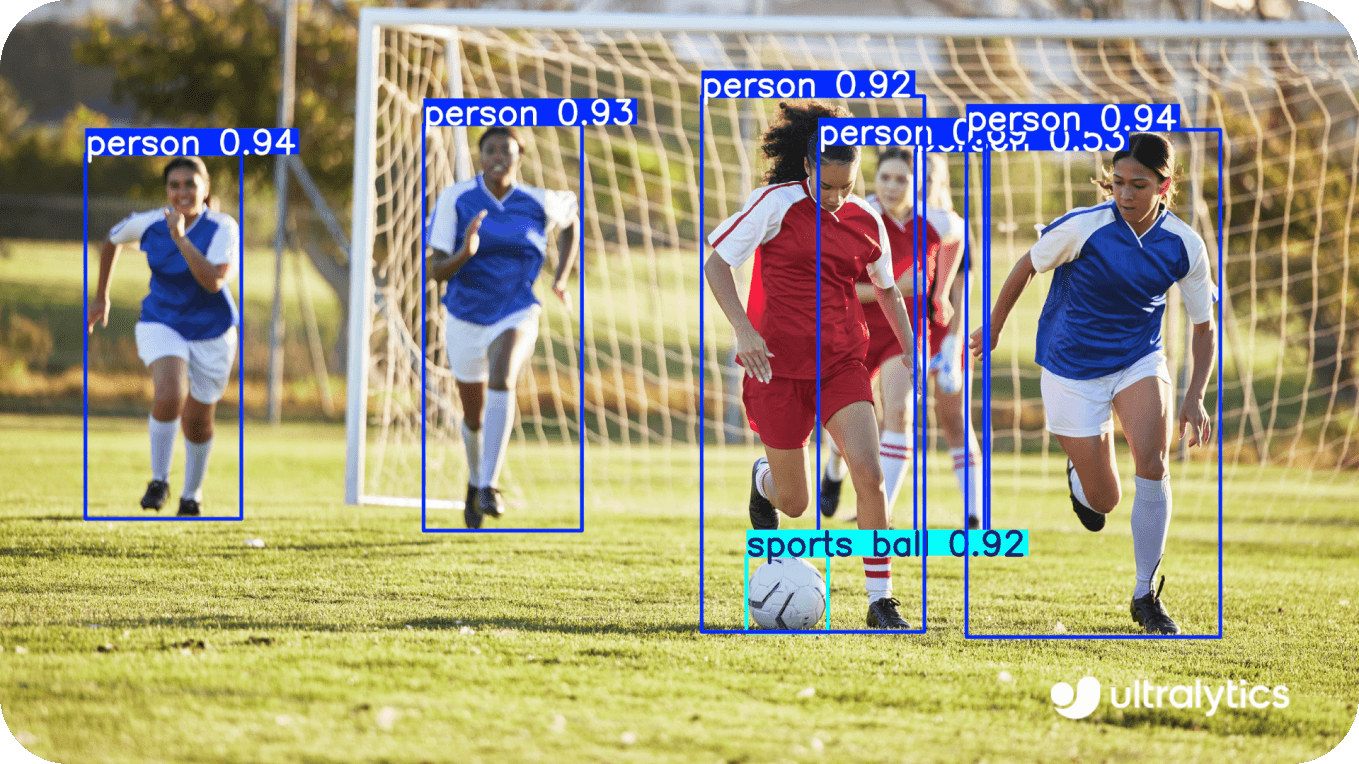
# Object Detection: Identifying and locating objects.

# Instance Segmentation: Detecting objects and delineating their exact shapes.

# Pose Estimation: Tracking keypoints (e.g., joints on a human body).

# Classification: Categorizing whole images.

# Oriented Object Detection (OBB): Detecting objects with rotation (e.g., ships in satellite imagery).



# *1. Using YOLO11 for object detection.*

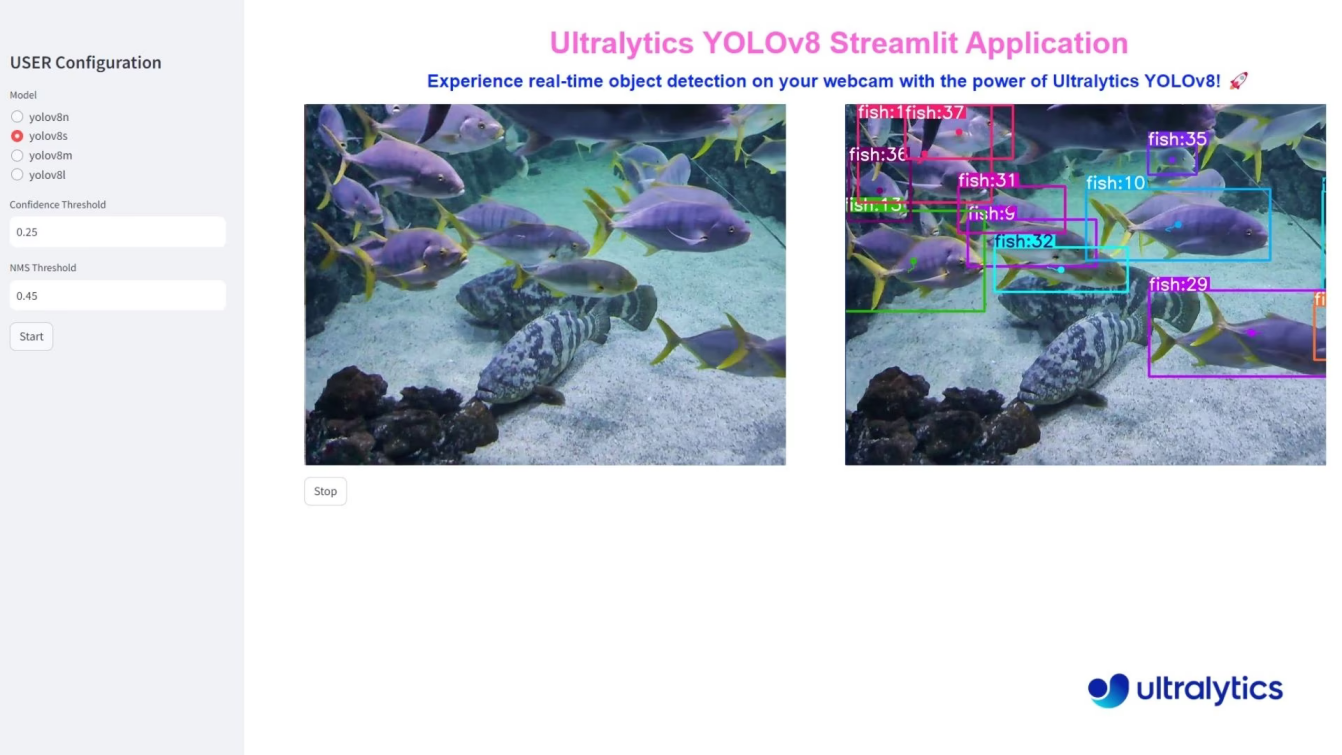
# YOLO11 is optimized for efficiency and ease of use, providing a Python package (*ultralytics*) and Command Line Interface (CLI) that simplifies the deployment of complex solutions.

# performance-comparison

# **Chapter 2: Methods for Specific Applications**

# **2.1 Live Inference with Streamlit Application**

# This application provides a user-friendly, web-based interface for real-time object detection.



# *2. Fish Detection using Ultralytics YOLO11*

# **Methodology:**

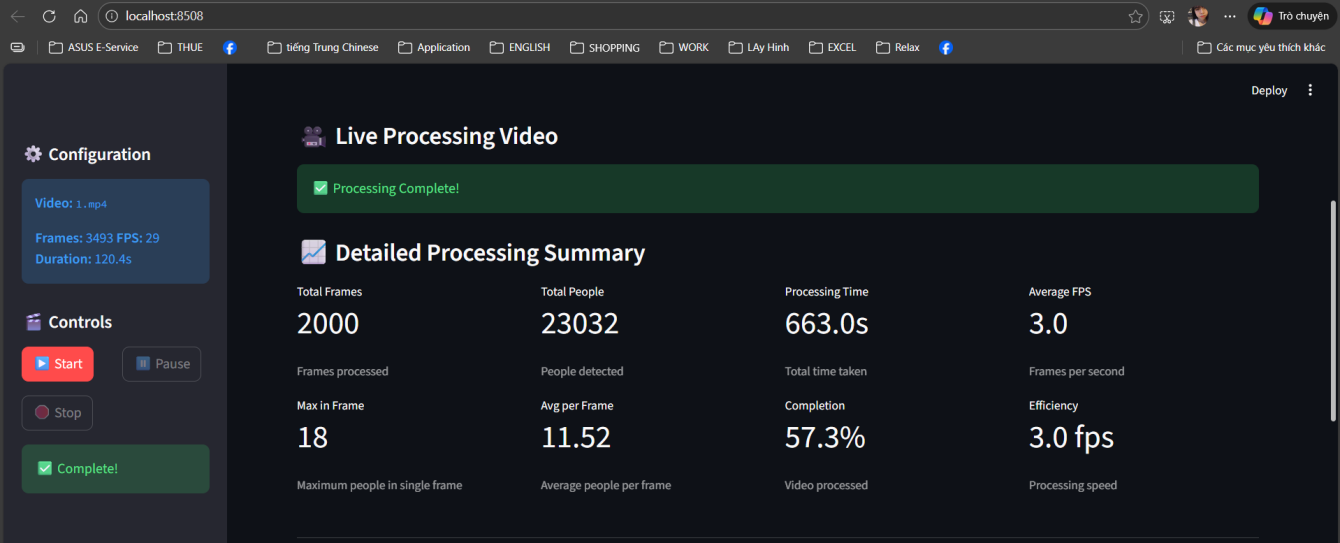
# ***Framework Integration:*** The application combines Streamlit, an open-source Python library for creating custom web apps, with the Ultralytics YOLO11 model.

# ***Execution Flow:*** The system captures a video feed (webcam or file). Each frame is processed by the YOLO11 model to detect objects.

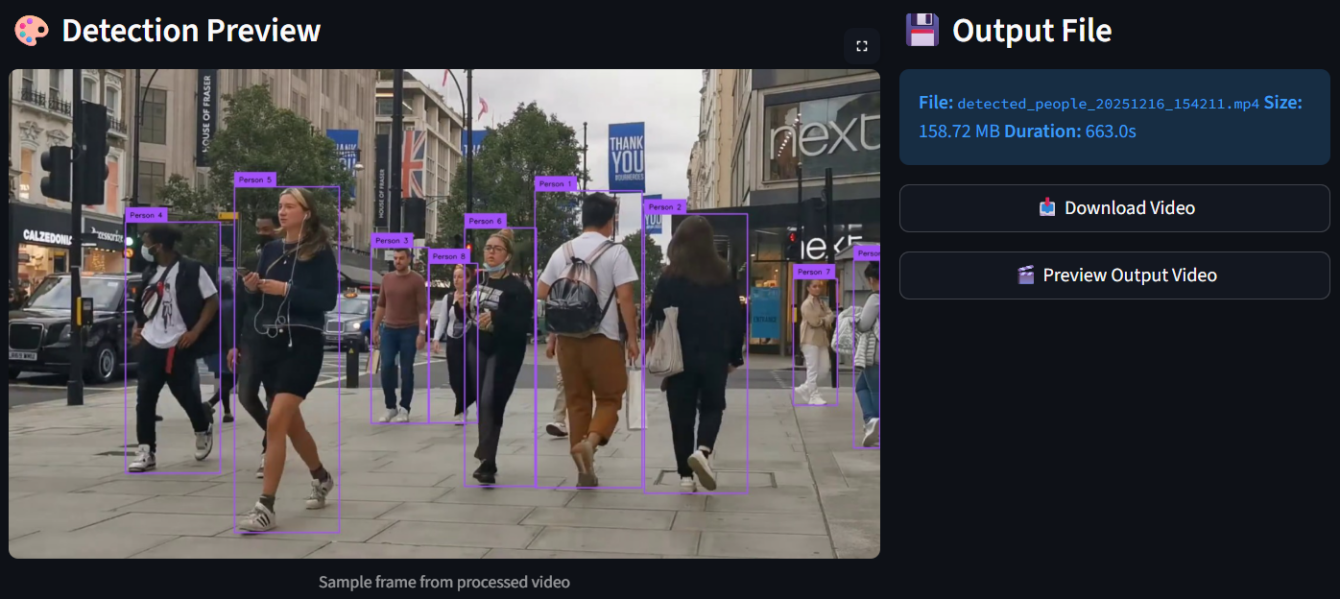
# ***Visualization:*** Results (bounding boxes, labels, confidence scores) are annotated onto the frame and rendered immediately in the web browser. The interface allows users to adjust parameters like confidence threshold and model type dynamically.

# **Code Implementation:**

# This code is a Streamlit web application designed to perform real-time people detection on n input video file using the YOLO11 object detection model. It includes advanced features like image enhancement (CLAHE) and object tracking via the supervision library.



# 3. The application interface after processing for video diagnosis, displaying the analysis results



# 4. Detected output video interface

# **Here is the step-by-step workflow of the application:**

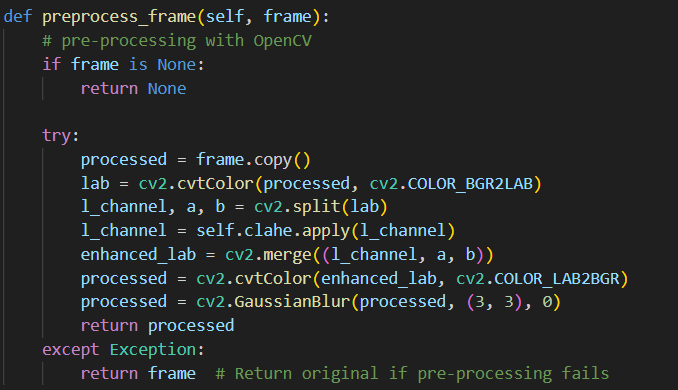
# ***Initialization:***

# The app sets the page layout and initializes the detection class.

# It checks for the video file (1.mp4) and displays its metadata (FPS, total frames) in the sidebar.

# ***Processing Loop (process\_video):***

# Input: Reads the video frame-by-frame using OpenCV.

* Enhancement: Applies Gaussian blur and CLAHE to the frame to help the model see better.

# 5. Preprocess each frame function

# Inference: YOLO11 scans the frame. If people are found, the supervision library tracks them (assigning IDs like "Person 1") and annotates the image.

# Output: The annotated frame is written to a new MP4 file (detected\_people\_[timestamp].mp4).

# Display: The app updates the live video feed on the web interface. Note: To maintain performance, the UI display update is throttled, separate from the actual processing FPS.

# ***Live Statistics:***

# While running, the app updates metrics for Current FPS, Total People Detected, and Processing Time every 0.5 seconds.

# ***Post-Processing:***

# Once processing finishes (or is stopped), the app releases system resources (garbage collection).

# It generates a Detailed Summary including average FPS and efficiency metrics.

# It provides a Download Button for the final annotated video and a preview player.

# **Key Features**

# Robust Error Handling: The code uses try/except blocks extensively to handle missing libraries (falling back to default YOLO plotting if supervision is missing) or corrupt video frames.

# Safety Limits: It sets a max\_frames limit (default 3000) to prevent the app from crashing due to memory overflow on long videos.

# Auto-Start: It contains logic to automatically start processing when the app loads if specific conditions are met.

# **2.2 Distance Calculation**

# This application estimates the physical distance between two objects detected in a 2D plane.

# **Methodology:**

# ***Centroid Calculation:*** The model detects objects and generates bounding boxes. The system calculates the centroid (center point) of each bounding box.

# ***Pixel Distance:*** When a user selects two objects (typically via mouse interaction), the Euclidean distance between their centroids is calculated in pixels.

# **Distance** =

# ***Real-World Mapping:*** To get a physical measurement (e.g., meters), the system multiplies the pixel distance by a pre-configured pixel-to-meter ratio (calibration factor).

# **Code Implementation:**

# This code creates an interactive video player that detects objects using the YOLOv11 model and allows you to measure the distance (in pixels) between any two detected objects by clicking on them.

### Code Overview

# The code is structured around a class named ***DistanceCalculator*** that handles three main tasks:

# Detection: It uses ultralytics to run object detection on every video frame.

# Interaction: It uses ***cv2.setMouseCallback*** to listen for user clicks. When you click near an object, it "snaps" to the center of that object's bounding box.

# Calculation: Once two objects are selected, it calculates the straight-line (Euclidean) distance between their center points.



# 6. A frame of the detection process when you selected 2 objects to calculate their distance

# As you can see in the image above is a frame of the detection process when you selected 2 objects to calculate their distance, details are:

# Selected object at (1190, 326)

# Distance calculated: 947.12 pixels

# Between: (244, 372) and (1190, 326)

# ***Main Steps for Implementation***

#### 1. Initialization: the script starts by loading the YOLO model (yolo11n.pt) and opening the video file using OpenCV.

#### 2. Frame Processing & Inference: in the process\_video loop, the code reads the video frame by frame. For each frame, it runs the YOLO model.

#### 3. Handling User Selection:

#### When the user clicks the mouse (*EVENT\_LBUTTONDOWN*):

#### The code takes the (x, y) coordinates of the mouse click.

#### It runs find\_closest\_detection to compare the click location against all current object centers.

#### If the click is close enough (within threshold=30 pixels), the valid object center is added to self.selected\_points.

4. Distance Calculation: once the list ***selected\_points*** contains exactly two points, the code triggers ***calculate\_distance***. It uses the Pythagorean theorem (Euclidean distance) to find the distance in pixels.

#### 5. Visualization (Overlay): finally, OpenCV is used to draw the visual elements onto the frame.

The processed frame is then displayed in a window and written to an output video file (.mp4).

# **2.3 TrackZone**

# TrackZone is a targeted monitoring solution that tracks objects only within specific, user-defined regions of a video frame.

# **Methodology:**

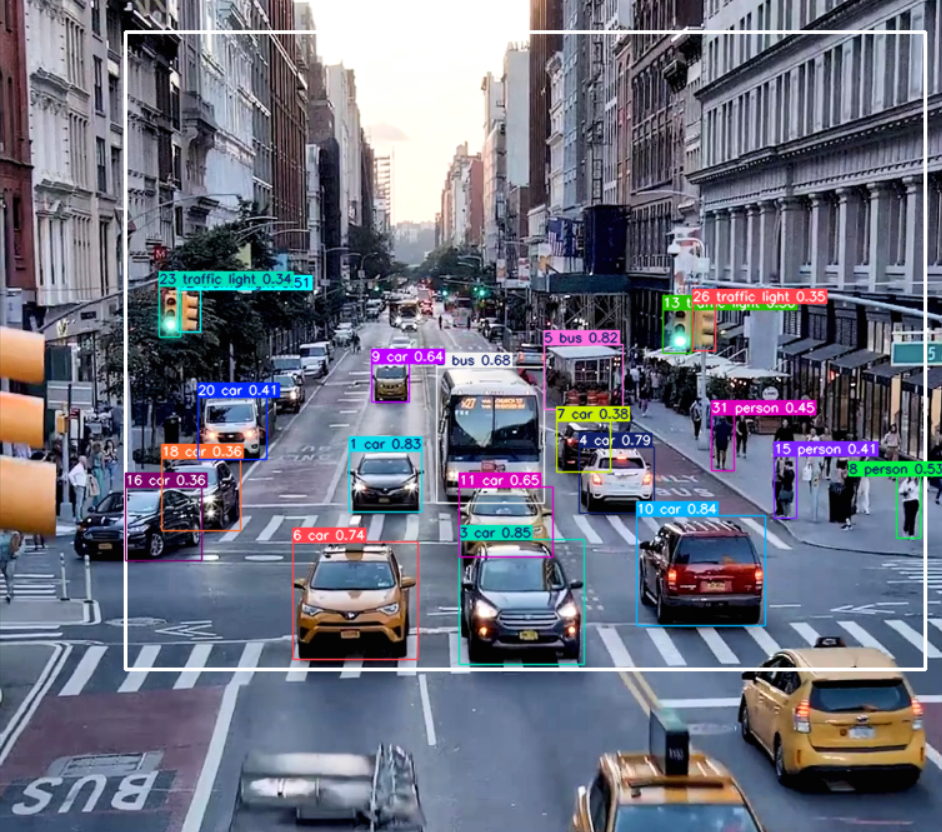
# ***Region Definition:*** Users define a specific polygon or zone (e.g., a specific aisle in a store or a safety zone in a factory) using coordinates.

# ***Selective Tracking:*** The logic filters tracks based on whether the object's centroid falls within the defined region. This reduces visual clutter and focuses analysis on relevant areas.

# ***Optimization:*** By focusing on specific zones, the system provides targeted analytics (e.g., "how long did a person stay in this specific zone?").

# **Code Implementation:**

# This program is an automated object tracking system that counts and tracks objects moving within a specific "zone" of a video. It uses the YOLO11 AI model to detect objects and computer vision techniques to enhance the video quality before processing.



# 7. A detected frame in processing of objects detetion in designated

# **The main steps of the workflow:**

# 1. Initialization of TrackZone

# It initializes the solutions.TrackZone tool from the Ultralytics library.

# Model: Loads yolo11m.pt (a medium-sized AI model) to recognize objects.

# Settings: Configures settings like line thickness for drawing boxes and sets the device to cuda (GPU) for faster processing.

# 2. Image Enhancement (Preprocessing)

# Before the AI looks at a frame, the preprocess\_aerial\_frame function improves it:

# Color Conversion: Changes the image from standard BGR color mode to LAB color mode.

# Contrast Enhancement (CLAHE): Applies "Contrast Limited Adaptive Histogram Equalization" to the Lightness (L) channel. This makes objects stand out more clearly against the background (useful for aerial views where lighting is often flat).

# Reconstitution: Merges the enhanced Lightness channel back with the color channels to create a sharper image.

# 3. Frame-by-Frame Processing Loop

# The code runs a loop for every frame in the video:

# Read: Grabs a frame from the video.

# Enhance: Runs the preprocessing function described above.

# Track & Detect: Passes the enhanced frame to trackzone. The AI detects objects and checks if they are inside the defined region points.

# 4. Cleanup

# Once the video ends, it releases the video file handles and closes all windows to free up computer memory.

# **2.4 Source files**

# All files of this project are in this Google Drive link: https://drive.google.com/drive/folders/1k6pTyimmCA-bn7SfIGRqRZfFWLJbuzPJ?usp=sharing

# **Chapter 3: Future Work**

# The field of computer vision and YOLO models is rapidly evolving. Future directions for these applications include:

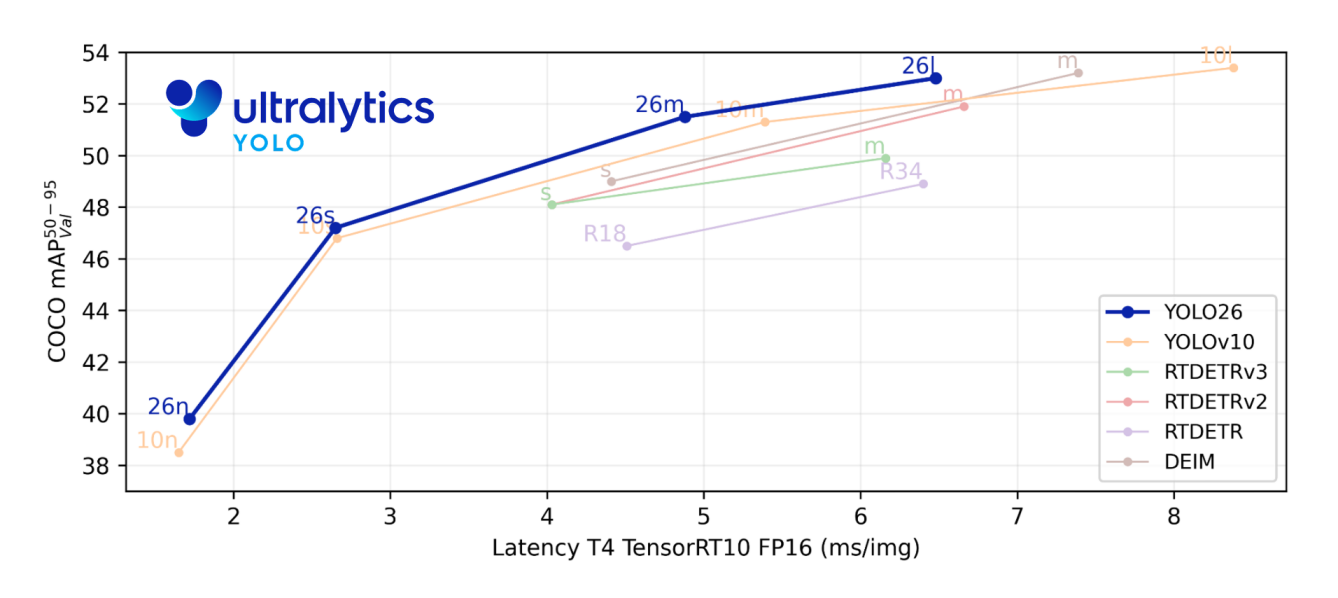
# Segment Anything Integration (SAM): Moving beyond bounding boxes to pixel-perfect segmentation (using models like SAM 3) to improve the accuracy of distance calculations and zone triggers.

# 3D Depth Estimation: Current distance calculation relies on 2D pixel logic. Future work involves integrating Depth Estimation models (monocular depth) to calculate true physical distance without manual calibration.

# Edge Optimization: deploying these solutions on low-power edge devices (like NVIDIA Jetson or Raspberry Pi) using techniques like INT8 quantization and TensorRT export for faster, decentralized processing.

# Multi-Model Logic: Combining YOLO with Large Vision-Language Models (LVLMs) to allow users to query video feeds with natural language (e.g., "Show me the red car in the TrackZone").

# **Ultralytics YOLO26** is the latest evolution in the YOLO series of real-time object detectors, engineered from the ground up for edge and low-power devices.



# It introduces a streamlined design that removes unnecessary complexity while integrating targeted innovations to deliver faster, lighter, and more accessible deployment. YOLO26 models are still under development and not yet released, it will coming soon.

# REFERENCES

Live Inference with Streamlit Application using Ultralytics YOLO11.https://docs.ultralytics.com/guides/streamlit-live-inference/

Distance Calculation using Ultralytics YOLO11.https://docs.ultralytics.com/guides/distance-calculation/

TrackZone using Ultralytics YOLO11.https://docs.ultralytics.com/guides/trackzone/