Performance Benchmarking of YOLO Architectures for Real-Time Object Detection

A Comparative Study of **YOLOv5**, **YOLOv8**, and **YOLOv11**

Team Members

Beyza GÜLER

Ramazan YILDIZ

Abdelrahman MOHAMED

Department

Computer Engineering

Ataturk University

October 2025

Agenda

1. **Introduction & Problem Statement**
   Motivation and objectives
2. **Background Concepts**
   Object detection fundamentals
3. **YOLO Architecture & Selection**
   Model variants and advantages
4. **Dataset & Annotation Format**
   Roboflow fruits detection dataset
5. **Methodology & Training Pipeline**
   Implementation and configuration
6. **Experimental Results**
   YOLOv5, YOLOv8, YOLOv11 performance
7. **Model Comparison & Analysis**
   Benchmarking and insights
8. **Deployment & Web Application**
   Streamlit-based interface
9. **Challenges & Future Work**
   Limitations and recommendations

Comprehensive Analysis

Abstract

This project presents a comprehensive **real-time object detection system** based on YOLO architectures, with focus on performance benchmarking across multiple model generations.

Three Model Architectures

Benchmarking YOLOv5, YOLOv8, and YOLOv11 variants

Custom Fine-Tuning

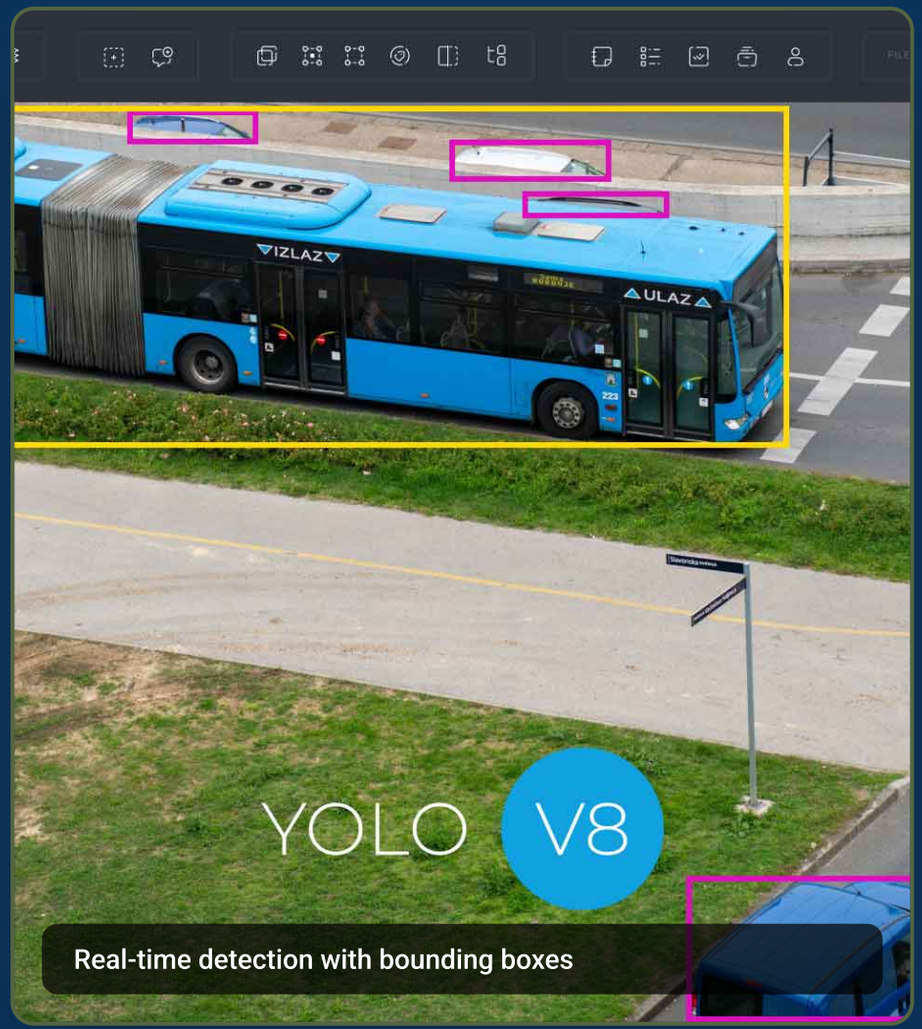
Pre-trained weights adapted to Roboflow fruit dataset

Streamlit Deployment

Interactive web interface for live detection

Key Metrics

Accuracy (mAP), Processing Speed (FPS), Real-time Performance



Real-time detection with bounding boxes

Problem Statement

The Challenge of Real-Time Object Detection

Speed vs Accuracy Tradeoff

High accuracy models often sacrifice real-time performance

Limited Accessibility

Complex deployment pipelines hinder adoption

Hardware Constraints

GPU availability and computational cost barriers

> *"Our goal: Build a balanced system that achieves **~30 FPS** while maintaining high detection accuracy"*

Key Application Areas

- Autonomous Driving
- Security Systems
- Smart Cities
- Industrial Automation

Real-time detection critical for safety & efficiency

Goals & Success Criteria

Project Objectives

Real-Time Performance

Process video at minimum 30 FPS

30+ FPS

High Detection Accuracy

Achieve competitive mAP scores

mAP > 0.75

User-Friendly Interface

Accessible web-based detection

Easy to Use

Scalable Architecture

Extensible for future enhancements

Modular Design

Success Metrics

- Training Stability
- Model Convergence
- Real-Time Inference
- Web Deployment

Continuous Improvement

Optimizing accuracy & speed

Background: Object Detection vs Classification

Image Classification

Single Label Assignment

Assigns ONE category to the entire image

Example: "cat" for an image containing a cat

- Simpler task
- Single prediction
- High computational efficiency

Object Detection

Multiple Objects + Localization

Detects, classifies, and locates ALL objects

Example: "cat (box 1), dog (box 2), person (box 3)"

- Bounding box regression
- Multiple predictions
- Real-time requirement

Why YOLO?

YOLO: You Only Look Once

Single-Stage Detector

Detection in ONE forward pass through the network

Real-Time Performance

Achieves high FPS rates suitable for video streams

Multiple Model Sizes

Nano, Small, Medium, Large variants for different needs

Strong Community Support

Extensive documentation, pre-trained weights, and tutorials

Proven Track Record

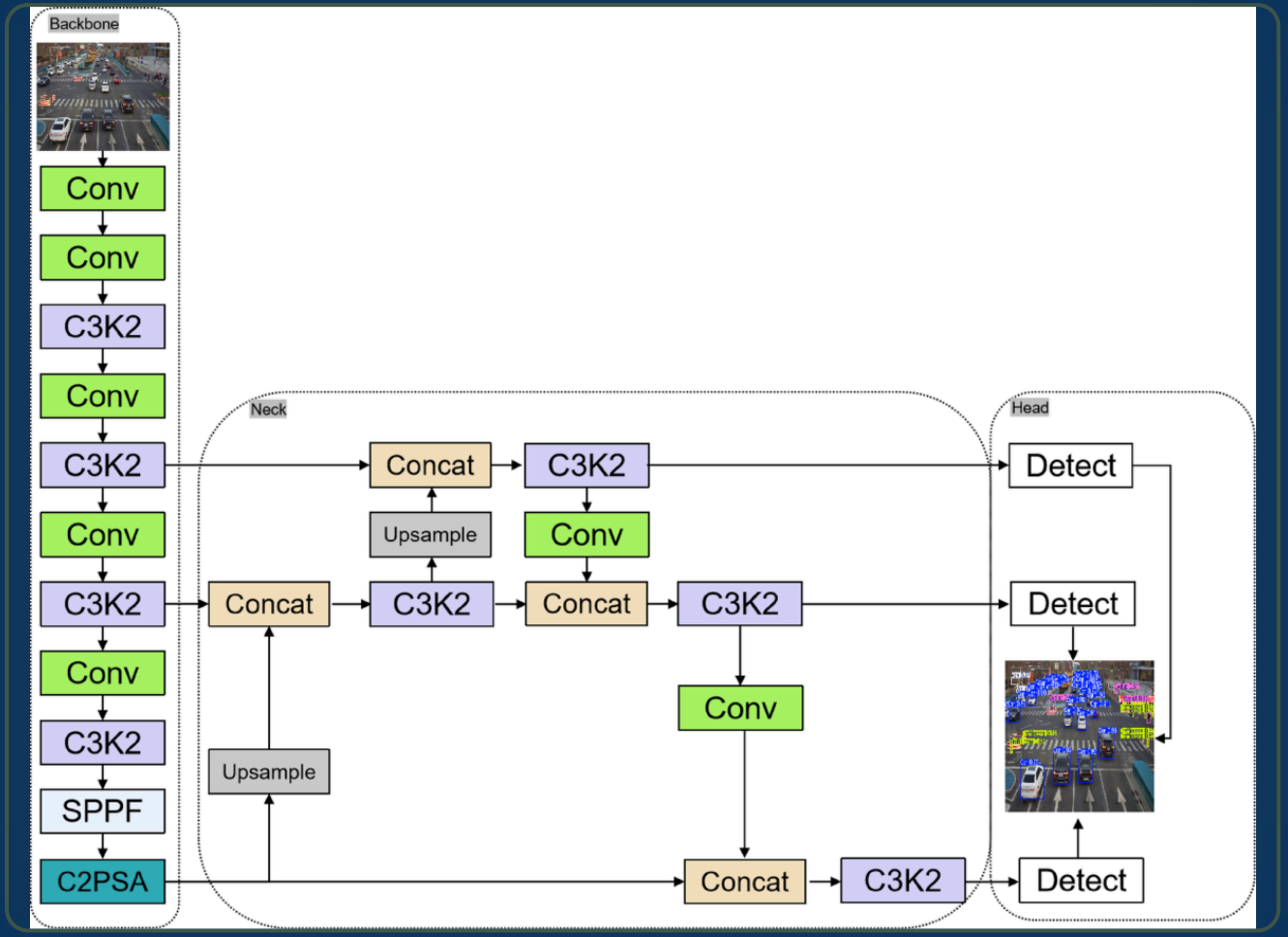
Widely Adopted
Industry standard for real-time detection

Accuracy-Speed Balance
Optimal tradeoff for most applications

Easy Integration
Simple deployment with PyTorch

Battle-Tested Architecture
YOLOv5 → YOLOv8 → YOLOv11 evolution

YOLO Architecture



Backbone

Feature extraction from input images

Neck

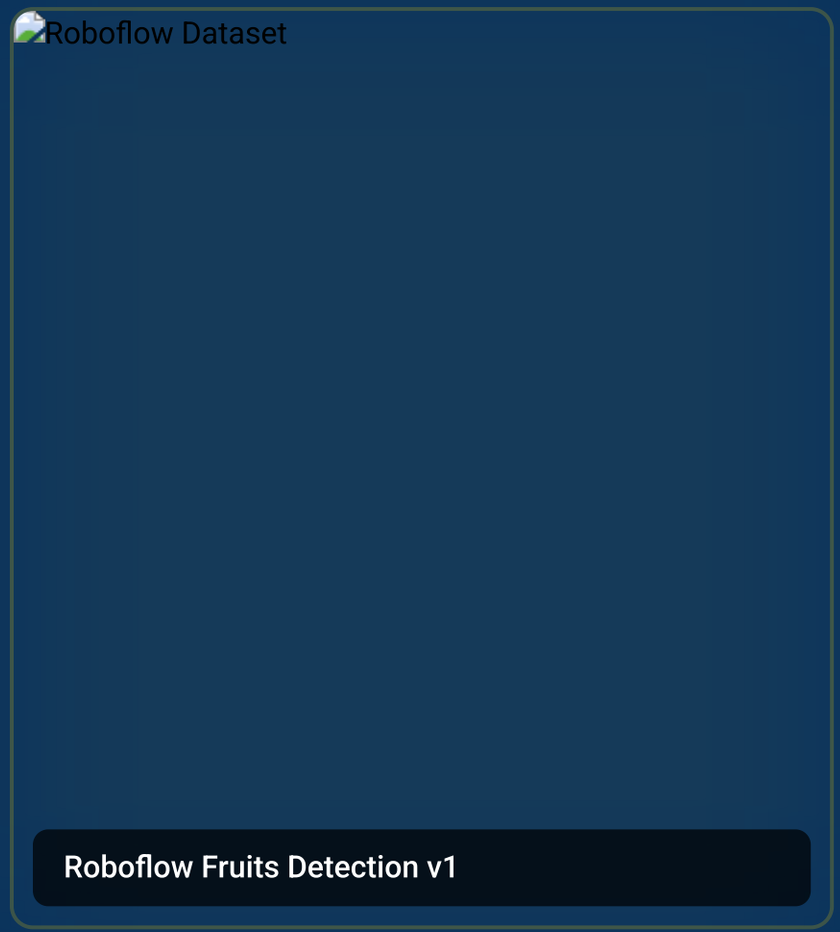
Multi-scale feature aggregation

Head

Bounding box & class prediction

→ Input → Backbone → Neck → Head → Detection

Dataset Overview



Roboflow Fruits Detection Dataset

Total Images	Fruit Classes	Format
2,974	9	YOLO

Data Distribution

Training Set	2,697 (91%)
Validation Set	187 (6%)
Test Set	90 (3%)

Pre-labeled dataset for fast model fine-tuning

Annotation Format

YOLO Format

Label File Format

Each line represents one object:

```
class_id x_center y_center
width height
```

All values normalized to [0, 1]

Example Annotation

```
0 0.71 0.84 0.15 0.23
```

Class ID	Center X	Center Y	Width	Height
0	0.71	0.84	0.15	0.23
Apple	Horizontal	Vertical	Box Width	Box Height

Coordinates are relative to image dimensions

Team & Roles

Beyza GÜLER
YOLOv11 Specialist & Reporting

- Trained YOLOv11 model
- Analyzed YOLOv11 metric results
- Technical documentation
- Reporting and analysis

Focus: Model evaluation & metrics analysis

Ramazan YILDIZ
AI Project Planning & Research

- Project initialization & preparation
- Research & dataset preparation
- Trained YOLOv8m & YOLOv8n models
- Model comparison analysis

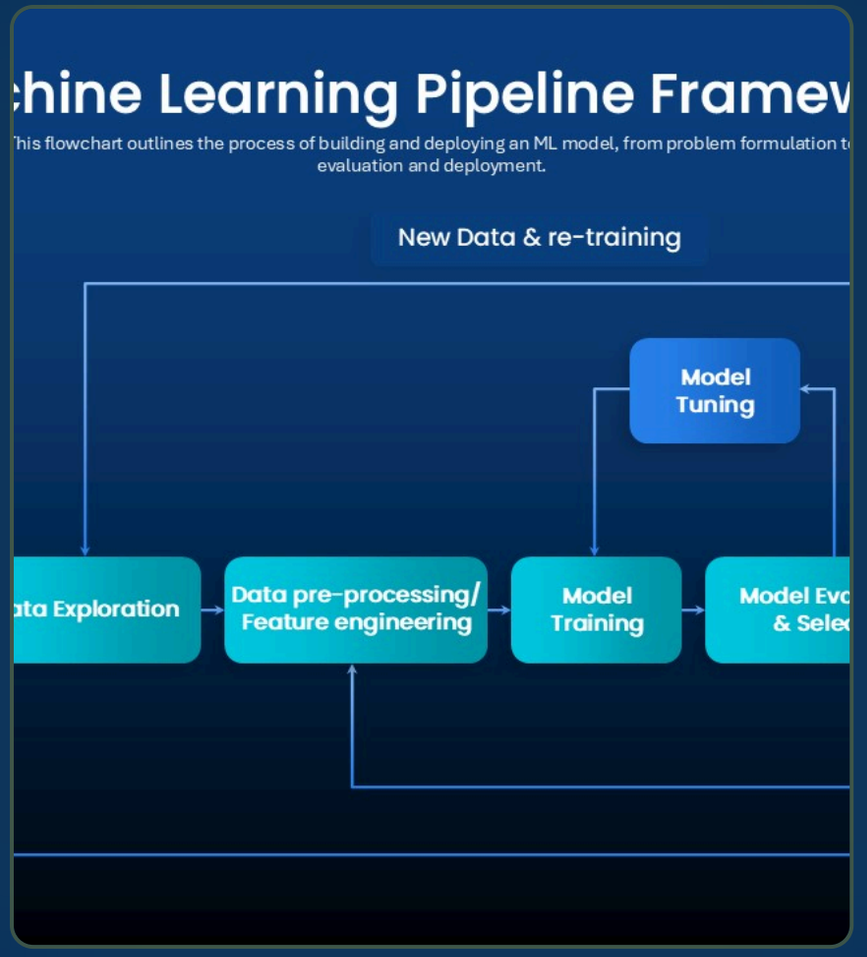
Focus: Project coordination & model benchmarking

Abdelrahman MOHAMED
YOLOv5 Specialist & Web Developer

- Trained YOLOv5 model
- Analyzed YOLOv5 metrics
- Developed web interface
- OpenCV preprocessing & visualization

Focus: Web deployment & system integration

Methodology Pipeline



End-to-End Workflow

Data Preparation

Roboflow dataset & splits

Preprocessing

OpenCV & resize operations

Model Training

YOLO fine-tuning

Evaluation

Precision, Recall, mAP, IoU

Deployment

Streamlit web app

Real-Time Detection

~30 FPS performance

→ Iterative refinement for optimal performance

Training Setup

Environment & Configuration

Platform

- Google Colab
- GPU Acceleration

Hyperparameters

Epochs: **50**

Img Size: **640**

Batch: **16**

Device: **0 (GPU)**

Consistent across all models for fair comparison

Training Code Example

```
from ultralytics import YOLO

model = YOLO('yolov8m.pt')

results = model.train(
  data='data.yaml',
  epochs=50,
  imgsz=640,
  batch=16,
  device=0
)
```

Best model saved as **best.pt** for deployment

YOLOv5 Results

Training Performance

Box Loss	Cls Loss	DFL Loss
Decreasing	Converged	Stable

Confusion Matrix

- Strong diagonal concentration
- High class separation accuracy
- Minimal confusion between classes

Key Metrics

mAP@0.5: **(See plot)**

Precision: **(See plot)**

Recall: **(See plot)**

mAP@0.5:0.95: **(See plot)**

YOLOv8 Results

Training Progress

Train Loss	Val Loss	mAP
Decreasing	Decreasing	Increasing

Confusion Matrix Analysis

- Strong diagonal concentration confirmed
- High accuracy across all fruit classes
- Minimal confusion between similar fruits

Performance Summary

Precision: **(See plot)**

Recall: **(See plot)**

mAP@0.5: **(See plot)**

mAP@0.5:0.95: **(See plot)**

YOLOv11 Results

Training Progress

Train Loss	Val Loss	mAP
Rapid Drop	**Stable**	**Highest**

Confusion Matrix Analysis

- Excellent class discrimination
- Strongest diagonal concentration
- Minimal false positives

Performance Highlights

Precision: **(See plot)**

Recall: **(See plot)**

mAP@0.5: **(See plot)**

mAP@0.5:0.95: **(See plot)**

Model Comparison

Model	Accuracy	Speed	Strength	Weakness
YOLOv5m	(From results)	Fastest	Fastest inference	Moderate accuracy
YOLOv8m	(From results)	Balanced	Good performance	Moderate speed
YOLOv11m	Highest	Moderate	Best accuracy	Slower than v5

Key Findings

- YOLOv11m achieved **highest detection accuracy**
- YOLOv5m provided **fastest inference time**

Real-Time Performance

Achieved FPS: **~30**

Measured on development machine

Models tested under **identical conditions**

Deployment Architecture

System Pipeline

1. **Input Sources**
   Webcam, Video, Images
2. **Preprocessing**
   OpenCV capture & resize
3. **Inference**
   YOLO model detection
4. **Postprocessing**
   Draw boxes & labels
5. **UI Visualization**
   Streamlit web app
6. **Analytics Logging**
   Detection summary

End-to-end latency: ~32ms/frame

Technology Stack

- Python
- YOLO
- PyTorch
- OpenCV
- Streamlit
- WebRTC

User Interface

- Overview Page
- Live Detection
- Analytics Dashboard
- Model Selection

Real-time performance: **~30 FPS**

Browser-based deployment

Web Application - Overview & Live Detection

Streamlit Web Interface

Overview & Live Detection Pages

Overview Page Features

Project Introduction
YOLOv8 architecture overview

Model Details
YOLOv8n, 80+ classes, ~30 FPS

Technology Stack
Python, YOLO, OpenCV, Streamlit

System Status
Active & operational

Live Detection Capabilities

- Bounding Boxes
- Class Labels
- Confidence

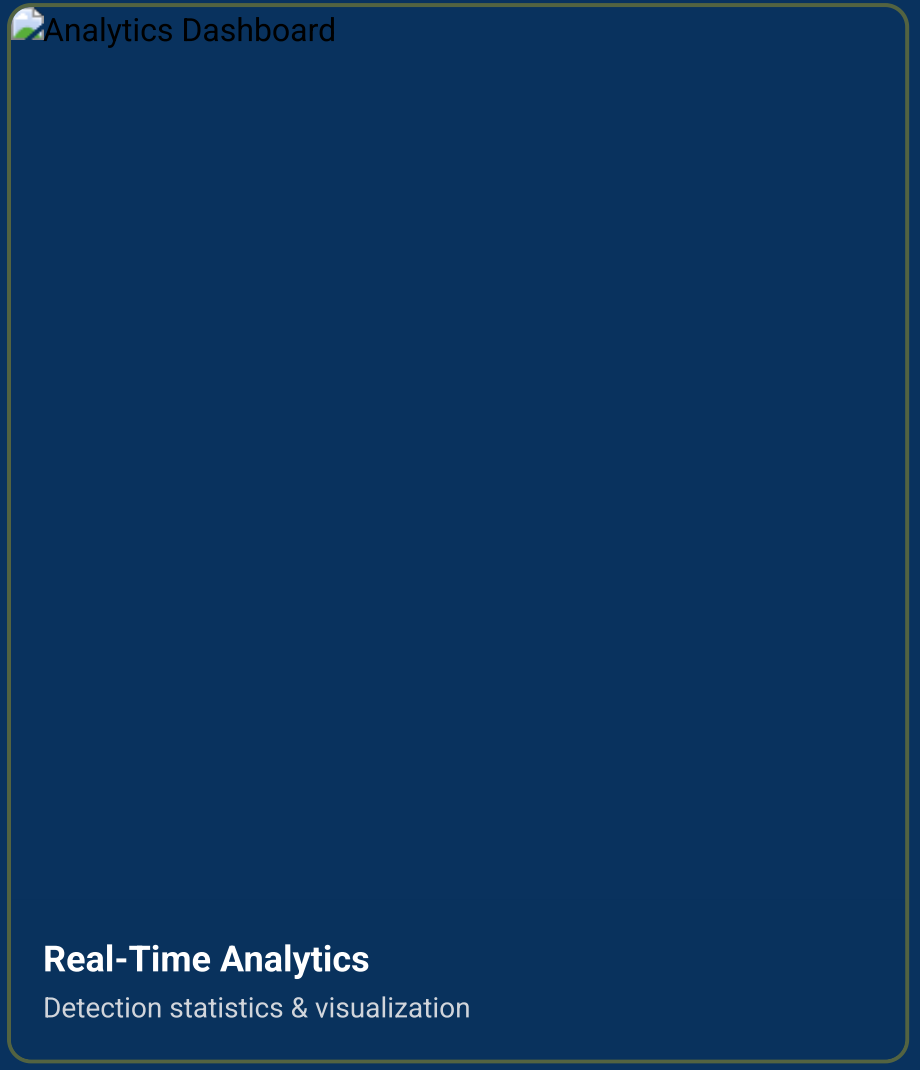
Real-Time Performance

~30 FPS

Browser-Based

Easy access

Analytics Dashboard



Real-Time Analytics

Detection statistics & visualization

Detection Statistics

Total Detections	Avg Confidence	Unique Classes	Processing Time
1,250	84.5%	18	32ms

Detection Frequency

- Real-time object distribution
- Bar chart visualization
- Most frequent objects

Top Detections

Person	High
Car	Medium
Fruits	Variable

Detection Details — Per-class statistics & logs

Challenges & Limitations

Key Challenges

Accuracy vs Speed Tradeoff
Larger models more accurate but slower

Lighting & Occlusion
Detection quality varies with conditions

Small Objects & Crowds
Harder to detect in complex scenes

Current Limitations

Limited Dataset
Only 9 fruit classes, need diversity

Hardware Constraints
GPU availability & compute resources

Generalization Issues
Performance drops in unseen environments

Lab Constraints
Webcam permissions & driver compatibility

Backup Plan & Risk Mitigation

Model Issues

- Use stable YOLOv8 or switch to YOLOv5
- Alternative frameworks available

Training Instability

- Reduce classes or data subset
- Adjust hyperparameters (LR, batch, epochs)

Data Problems

- Use additional public datasets
- Collect new training samples

Compute Load

- Migrate to cloud GPU platform
- Use lighter nano model variant

Web App Issues

- Alternative UI frameworks (Gradio, Dash)
- Core detection remains functional

Timeline Delays

- Overlap less critical project steps
- Extend schedule maintaining objectives

Mitigation Strategy

Proactive backup measures ensure project continuity

Project Roadmap

Development Timeline

Current & Next Phases

Phase 1: Completed

Current Status ✓

- ✓ Dataset sampling completed
- ✓ Prototype system deployed
- ✓ Real-time detection working
- ✓ Web interface established

Phase 2: Next Steps

Planned Development

- Image annotation & labeling
- Model fine-tuning (bigger dataset)
- Quantitative evaluation
- Performance optimization

Ready for Phase 2 Development — **Continuous Improvement**

Conclusion

Working Prototype

Real-time object detection successfully implemented and deployed

System Integration

YOLO + OpenCV + Streamlit web application fully functional

Performance Achievement

Stable real-time performance at **~30 FPS** achieved

Comprehensive Evaluation

- ✓ Benchmarked 3 YOLO architectures
- ✓ Produced training artifacts
- ✓ Generated evaluation metrics

Browser Deployment

- Accessible via web browser
- Real-time webcam detection
- Analytics dashboard included

Strong foundation for future enhancements

Future Work & Thank You

Enhanced Evaluation

- Per-class AP analysis
- Latency profiling
- Model quantization
- Rigorous benchmarking

Deployment Expansion

- Mobile app deployment
- On-device inference
- Edge computing optimization

UI Improvements

- Model switcher
- Threshold controls
- Save results feature
- Session logs

Dataset Expansion

- Larger custom dataset training
- More object classes
- Varied environmental conditions

Advanced Features

- 3D localization & depth estimation
- Multi-object tracking
- Instance segmentation

Thank You!

We appreciate your attention and feedback

Questions & Answers