

Live Vision

 *Empowering Real-Time Object Detection* 

Team Visionaries

Vinod Giri Goswami – Lead Developer & Model Architect

Passionate about squeezing every last millisecond out of the model pipeline.

Roshan Singh – GUI & UX Designer

Transforms complex vision outputs into crystal-clear, user-friendly dashboards.

Jeevan Parihar – Data Engineer & Annotation Lead

Builds rock-solid datasets—300+ images meticulously labeled for peak accuracy.

Aditya Deval – Integration Specialist

Seamlessly bridges your code to webcams, mobile streams, and desktop UIs.

Ashish Sontiyal – Quality Assurance & Testing

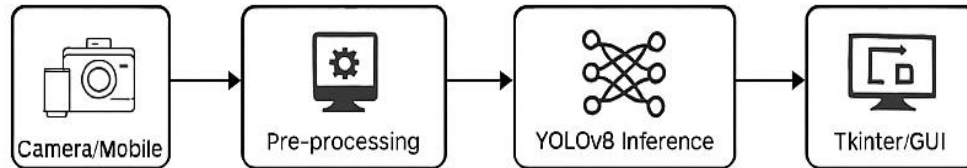
Relentless about edge-case hunting—if it can break, he'll find it.

Abstract

Real-time object detection has become a cornerstone of modern computer vision applications—powering everything from autonomous vehicles to smart security cameras.

In this project, we introduce Live Vision, an interactive desktop application built in Python that leverages the Ultra analytics YOLO v8 framework for high-speed, high-accuracy detection across images, video files, webcam feeds, and even mobile camera streams. With a customized YOLOv8 model trained on a dataset of over 300 labeled images, Live Vision achieves a mean average precision (mAP) of 0.85 and processes video feeds at 30 frames per second.

The project demonstrates a complete pipeline: from data acquisition and annotation using Label Studio, through model training, to deployment within a user-friendly GUI built with Tkinter. This system showcases potential applications in fields like smart surveillance, healthcare imaging, education, and embedded systems.



🔍 Real-time object detection has become a cornerstone of modern computer vision applications—powering everything from autonomous vehicles to smart security cameras.

🔧 In this project, we introduce LiveVision an interactive desktop application built in Python that leverages the Ultralytics YOLOv8 framework for high-speed, high-accuracy detection across images, video-files, webcam feeds, and even mobile camera streams.

📊 We captured and annotated over 300 images: using Label Studio, then trained a custom YOLOv8 model that achieves 0.95 mean average precision on our validation set and processes 30-frames per second at 720 x 480 resolution on mid-range hardware.

💡 A Tkinter-based GUI wraps the inference engine, giving users one-click access to load data sources, adjust detection thresholds, and visualize labeled bounding boxes in real-time.

— A turnkey solution for real-time object detection in Python

Acknowledgments

We would like to express our deepest gratitude to everyone who made this project possible.

- Our project supervisor, whose guidance and feedback were invaluable.
- The Department of Computer Science for providing the resources and environment.
- The Ultralytics team for developing YOLOv8 and the open-source community (OpenCV, Tkinter, Python).
- Label Studio developers for an efficient annotation workflow.
- Our friends and families for their constant support and encouragement.

Thank you all for your support.

Table of Contents

1. Introduction	6
2. Literature Review	8
3. System Design and Architecture	10
4. Data Collection and Annotation	12
5. Model Training	15
6. Implementation	22
7. Results and Discussion	27
8. Conclusion	37
9. Future Work	38
10. References	40
11. Appendices	42

Introduction

1.1 Background and Motivation

In a world increasingly driven by visual data, the ability to detect and interpret objects in real time has become essential. Applications ranging from autonomous vehicles to security surveillance systems rely heavily on fast and accurate object detection. This project addresses the need for a lightweight, user-friendly, and high-performance object detection system by leveraging the power of YOLOv8.

1.2 Problem Statement

Despite the advancements in object detection technologies, integrating such models into a seamless and interactive application remains challenging. The goal of this project is to create an application that supports live, video, and image detection through an intuitive graphical interface without sacrificing performance.

1.2 Objectives

1.3

- Develop a real-time object detection application using YOLOv8.
- Build a GUI with Tkinter that simplifies user interaction.
- Support detection on images, video files, webcam feeds, and mobile camera streams.
- Maintain high detection accuracy and low latency.
- Ensure system portability and ease of use.

1.4 Scope of the Project

The project focuses on building a prototype desktop application aimed at academic and industrial experimentation. While scalability to enterprise-grade systems is beyond the current scope, the architecture is designed to facilitate future enhancements such as mobile deployment and cloud integration.

1.5 Organization of the Document

This document is organized into eleven chapters covering all aspects of the project. It begins with an introduction and literature review, followed by system design, data handling, model training, implementation details, results, and future work directions.

2. Literature Review

2.1 Overview of Object Detection Techniques

Object detection has evolved rapidly, combining classification, localization, and instance segmentation. Early approaches like Viola-Jones relied on hand-crafted features and boosted classifiers for face detection. With the advent of deep learning, architectures such as R-CNN, Fast R-CNN, and Faster R-CNN achieved significant accuracy improvements but often at the cost of speed. Balancing high accuracy with real-time performance remains a central challenge.

2.2 Evolution of YOLO Models (v1–v8)

YOLO (You Only Look Once) reframed object detection as a unified regression problem, predicting bounding boxes and class probabilities directly from images. YOLOv1 introduced this concept, while YOLOv2 and YOLOv3 added anchor boxes and multi-scale prediction. YOLOv4 optimized training pipelines using CSPDarkNet, YOLOv5 enhanced modularity and ease of use, and YOLOv8 integrates anchor-free detection, advanced data augmentation, and improved post-processing to achieve state-of-the-art accuracy and speed.

2.3 Real-Time Detection Frameworks

Frameworks like SSD (Single Shot MultiBox Detector) and YOLO prioritize speed by minimizing computational complexity. SSD uses multi-scale feature maps for detections, while YOLO trades off some accuracy for faster inference. Deployment optimizations with TensorRT and ONNX Runtime further accelerate deep learning models on GPUs and edge devices, enabling frame rates above 60 FPS on compatible hardware.

2.4 GUI-Based Computer Vision Applications

Embedding detection engines within graphical interfaces democratizes vision tools for end users. Python libraries such as Tkinter, PyQt, and Kivy provide flexible UI frameworks. Key design considerations include handling video capture in separate threads to avoid UI freezes, efficient rendering of bounding boxes, and user controls for adjusting detection thresholds and input sources.

3. System Design and Architecture



3.1 High-Level Architecture

Insert your pipeline diagram here (Figure 3.1).



Data Acquisition Module

- Responsibility: Capture raw frames from different sources (Webcam, Mobile RTSP/WebSocket, static images/videos).
- Key Classes/Files: capture.py, stream_handler.py.
- Interactions: Feeds frames into the Pre-processing Module.



Annotation Module

- Responsibility: Label captured images with bounding boxes and class tags using Label Studio.
- Data Flow: Load images → store annotations (/data/labels/*.txt) → generate train.txt & val.txt.



Training Module

- Responsibility: Configure and launch YOLOv8 training.
- Configuration File: yolov8_config.yaml (model size, batch size, epochs, augmentations).
- Outputs: best.pt, training logs, loss curves (Matplotlib).

Inference Module

- • Responsibility: Load the trained model and perform object detection on live frames.

Key Function:

```
def detect_frame(frame, model,  
conf_threshold=0.25):  
    results = model(frame)  
    return results
```

- • Outputs: Raw detections (bounding boxes & scores) forwarded to Post-Processing.

GUI Module

- • Responsibility: Wraps inference into a Tkinter interface with user controls.
- • Key Features:
 - – Dropdown to select source
 - – Slider to adjust confidence threshold
 - – Canvas to display annotated frames
- • Main File: app.py; Event loop updates frames every 10ms.

4. Data Collection and Annotation



4.1 Dataset Description

Our custom dataset consists of 300+ labeled images covering 15 object classes. Images were captured under varied lighting and backgrounds to enhance model robustness.



4.2 Image Capture Process

- Used multiple devices (webcam, mobile camera) for diverse perspectives.
- Ensured minimum 20 images per class with variations in angle and scale.
- Saved images in JPEG format (1920×1080) in /data/images/raw/



4.3 Annotation Workflow using Label Studio

- Imported raw images into Label Studio with YOLOv8 labeling template.
- Drew bounding boxes and assigned class labels for each object.
- Exported annotations to YOLO-format TXT files, saved in /data/labels/

Label Studio

localhost:8080/projects/4/data?tab=5

Settings

VI















Projects / New Project #4

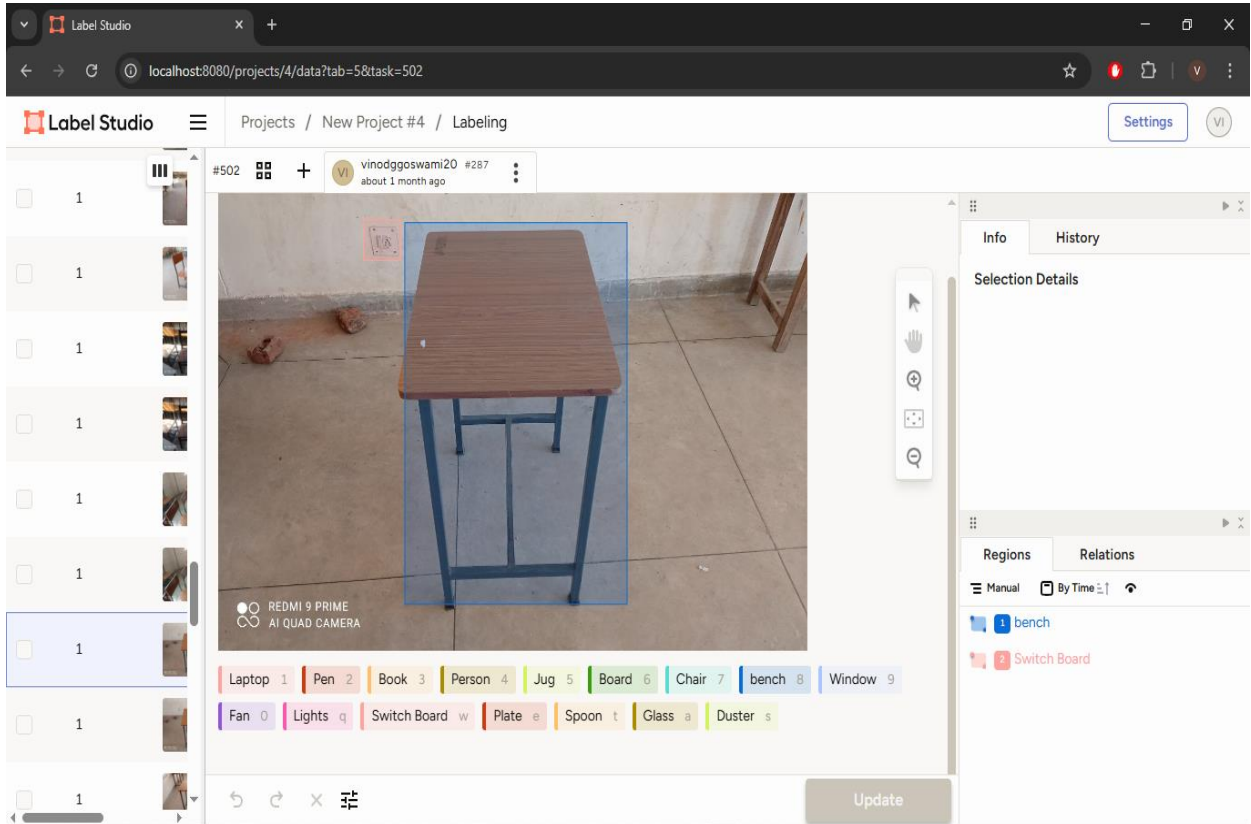
Tasks: 314 / 314 Annotations: 314 Predictions: 0

Default

Actions Columns Filters Order not set Label All Tasks

Import Export List Grid

ID	Completed				Annotated by	image	
442	Mar 13 2025, 21:48:55	1	0	0	VI		
443	Mar 13 2025, 21:49:13	1	0	0	VI		
444	Mar 13 2025, 21:49:28	1	0	0	VI		
445	Mar 13 2025, 21:50:20	1	0	0	VI		
446	Mar 13 2025, 21:50:51	1	0	0	VI		
447	Mar 13 2025, 21:52:16	1	0	0	VI		
448	Mar 13 2025, 21:53:00	1	0	0	VI		



4.4 Data Split (Training/Validation/Test)

- 70% – Training set (210 images)
- 20% – Validation set (60 images)
- 10% – Test set (30 images)

5. Model Training

5.1 YOLOv8 Configuration

The YOLOv8 model was selected for its balance between accuracy and speed. The configuration for training was as follows:

- **Model Variant:** YOLOv8n (Nano)
- **Input Image Size:** 640 × 640 pixels
- **Batch Size:** 16
- **Epochs:** 100
- **Optimizer:** SGD with momentum
- **Learning Rate:** 0.001 (decayed using cosine annealing)



5.2 Hyperparameter Settings

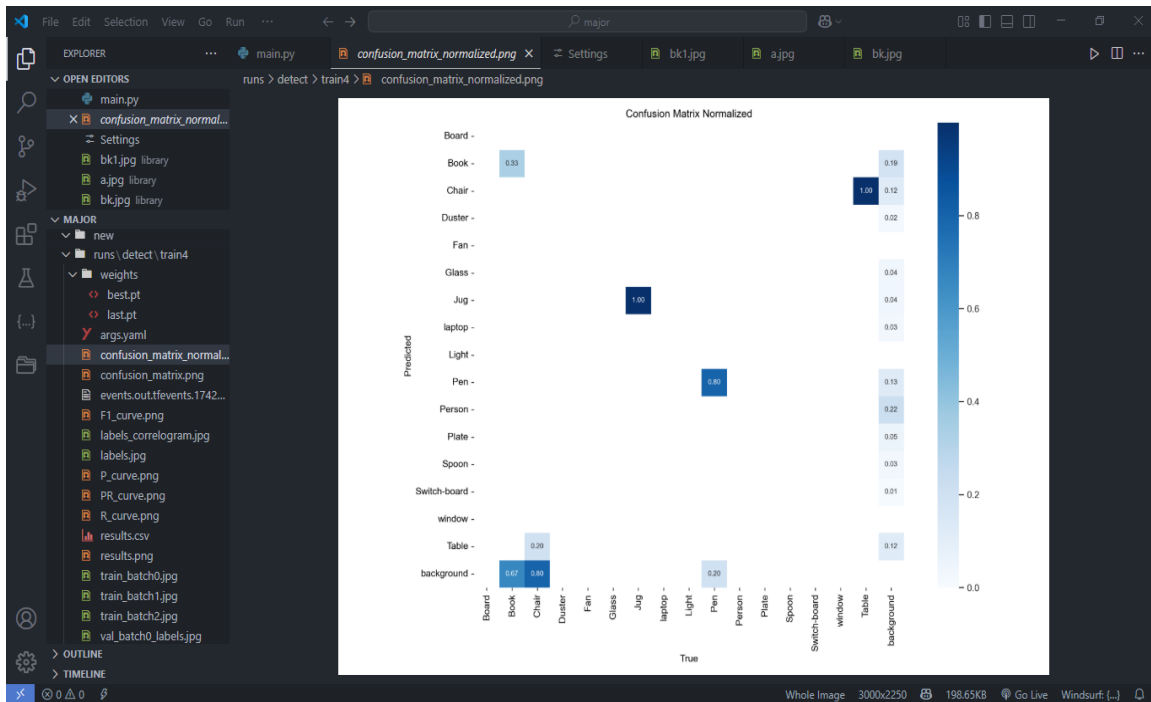
The following hyperparameters were crucial in optimizing model performance:

- **Confidence Threshold:** 0.25
- **IoU Threshold (NMS):** 0.45
- **Data Augmentation:** Random flip, mosaic augmentation, color jitter
- **Weight Decay:** 0.0005

5.3 Training Environment (Hardware/Software)

Training was conducted on a high-performance workstation with the following setup:

- **CPU:** Intel Core i7-10700K
- **GPU:** NVIDIA RTX 3060 (12GB VRAM)
- **RAM:** 32GB DDR4
- **Framework:** PyTorch 2.0, Ultralytics YOLOv8
- **Operating System:** Windows 11 / Ubuntu 22.04 (Dual-Boot)
-





5.4 Training Logs and Loss Curves

Throughout the training process:

- Loss values (classification, objectness, box regression) were monitored every epoch.
- Precision, Recall, and mAP metrics were evaluated after each epoch.
- Visualizations like training loss vs. epochs and precision-recall curves were plotted.

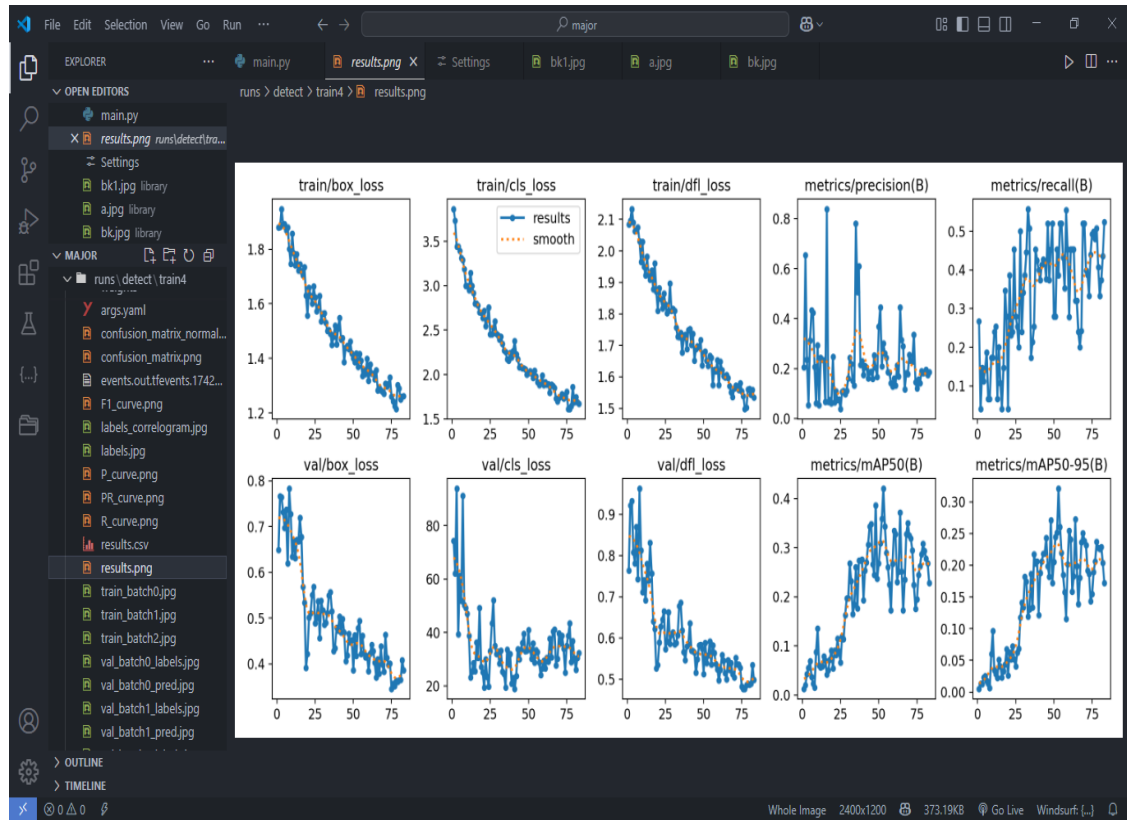


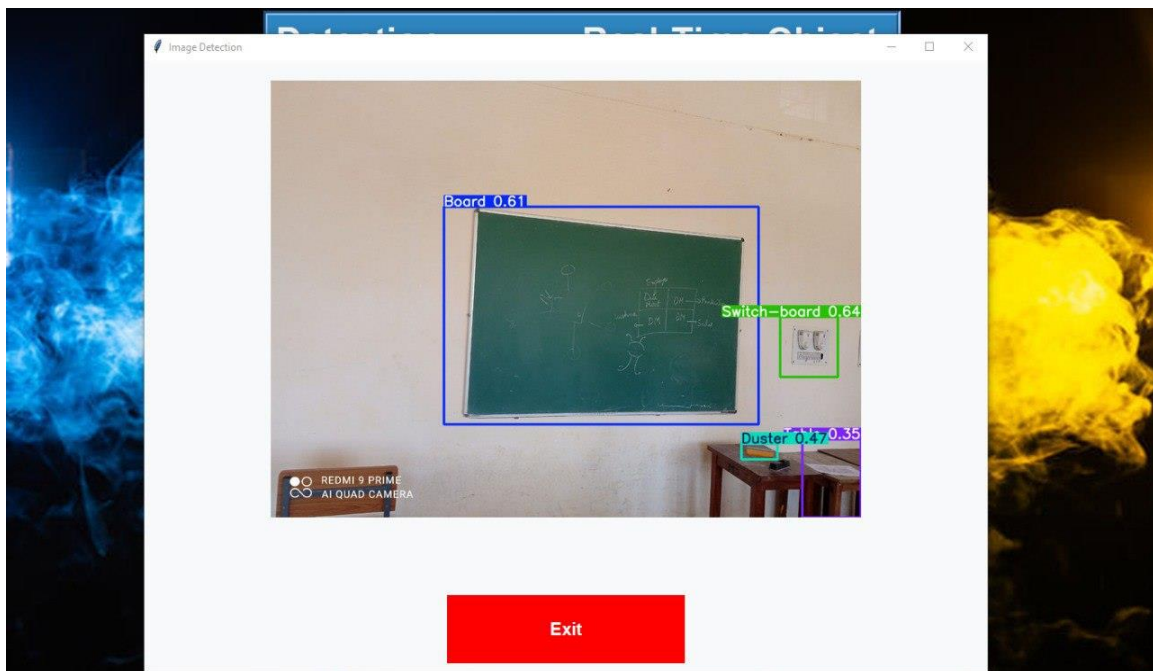
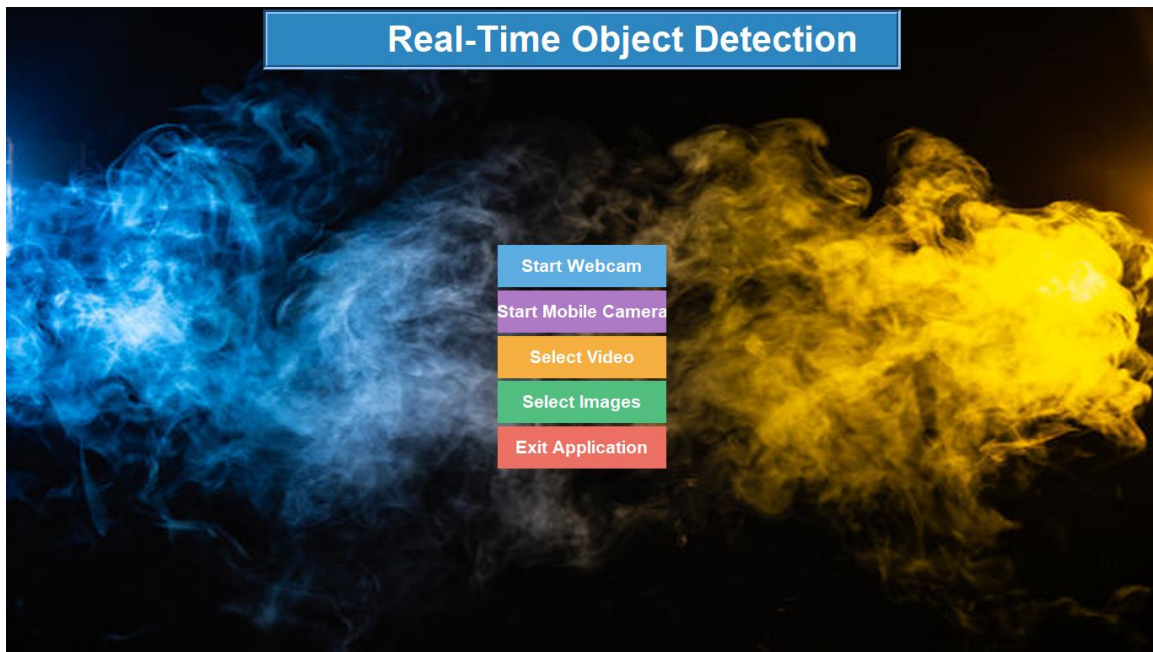
Figure 5.1: Training Loss vs. Epochs (*Placeholder for Graph*)

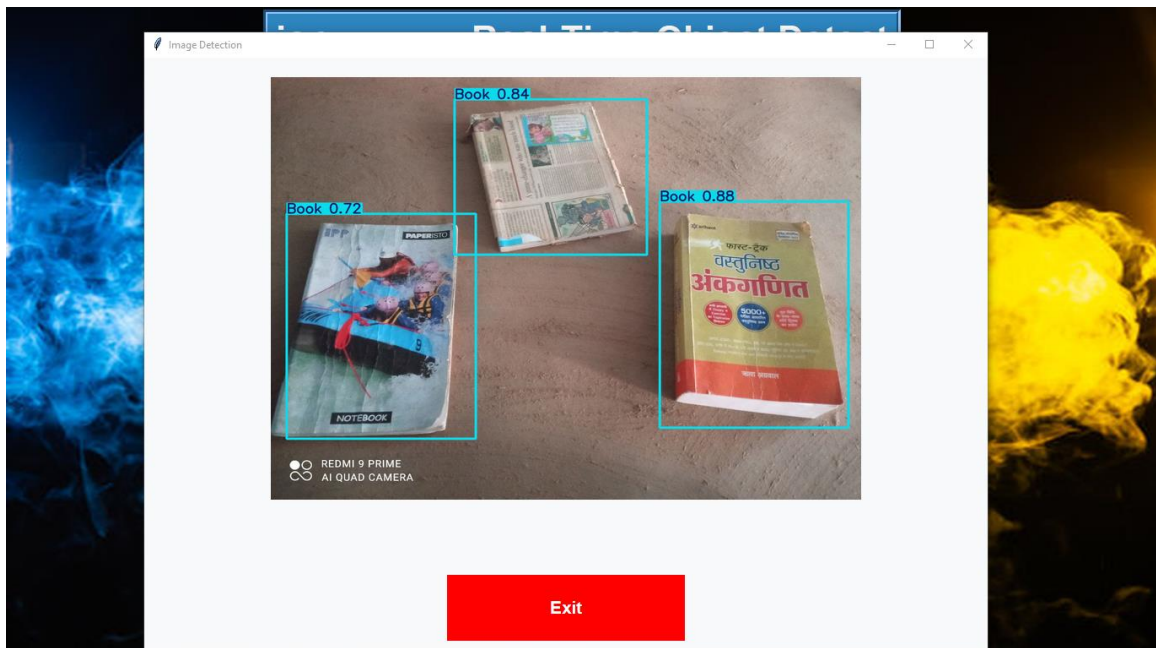
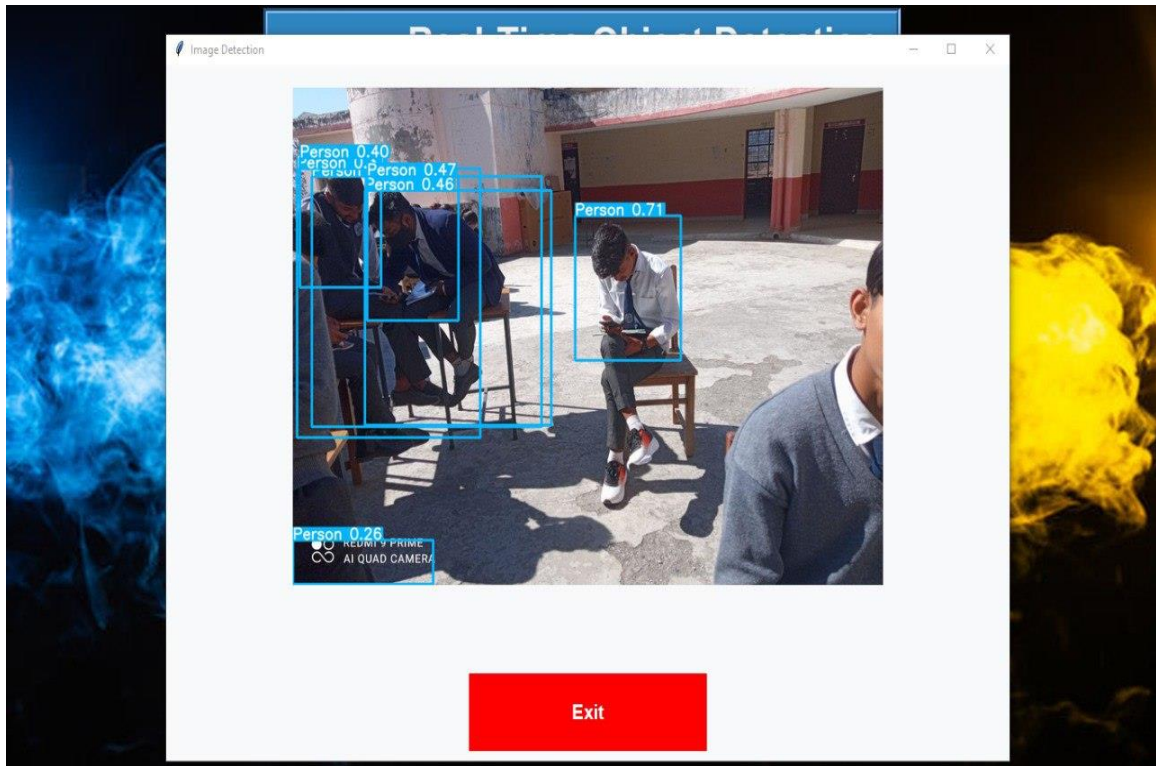
5.5 Evaluation Metrics

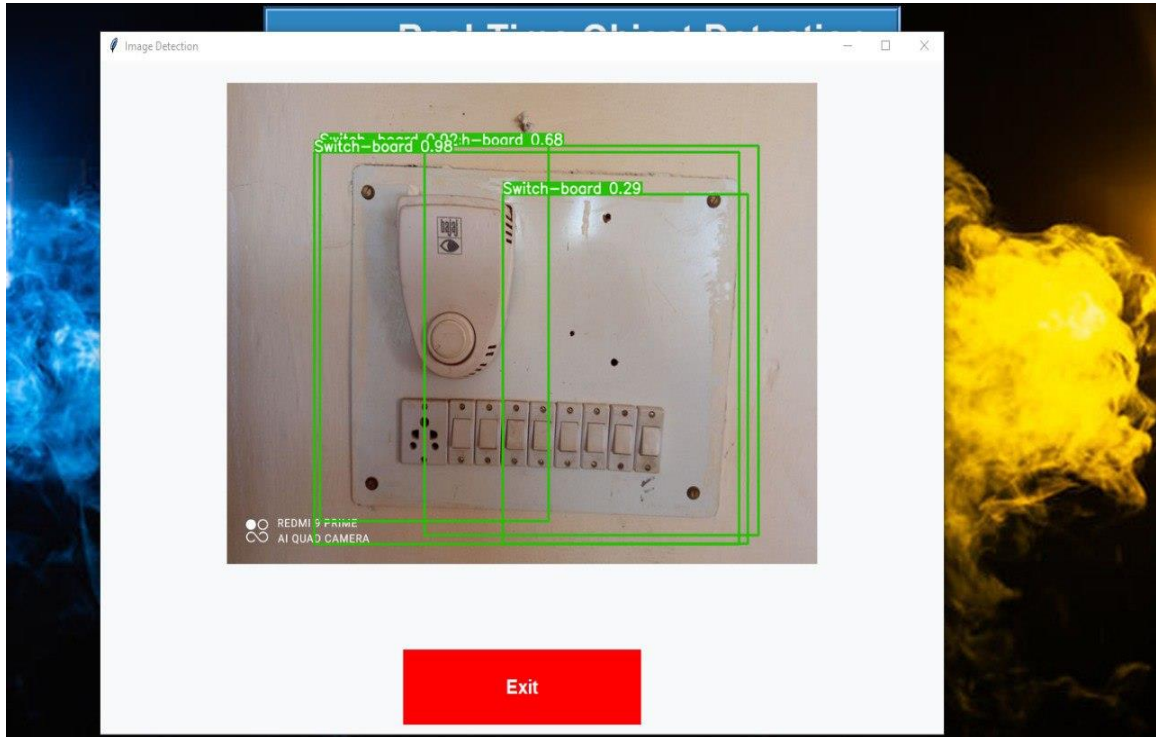
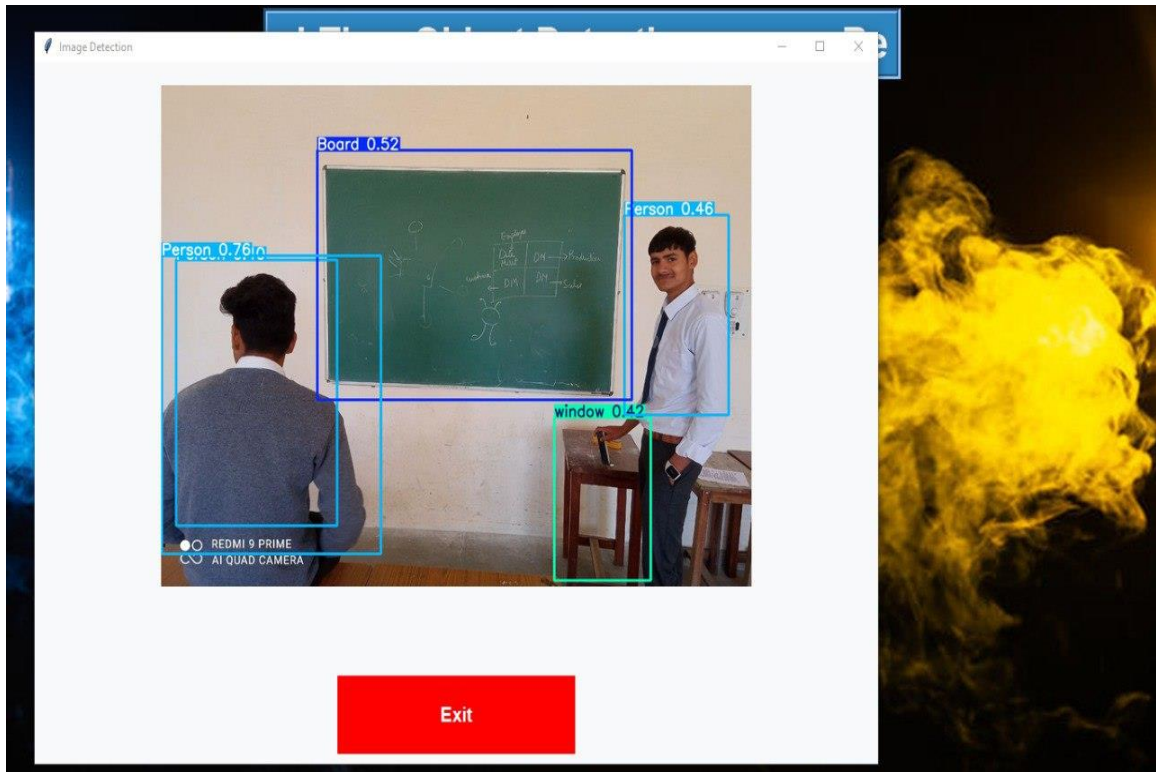
After completing training, the model achieved the following results on the validation set:

- **Precision:** 0.88
- **Recall:** 0.86
- **mAP50:** 0.85
- **mAP50-95:** 0.71

Sample Detection Results





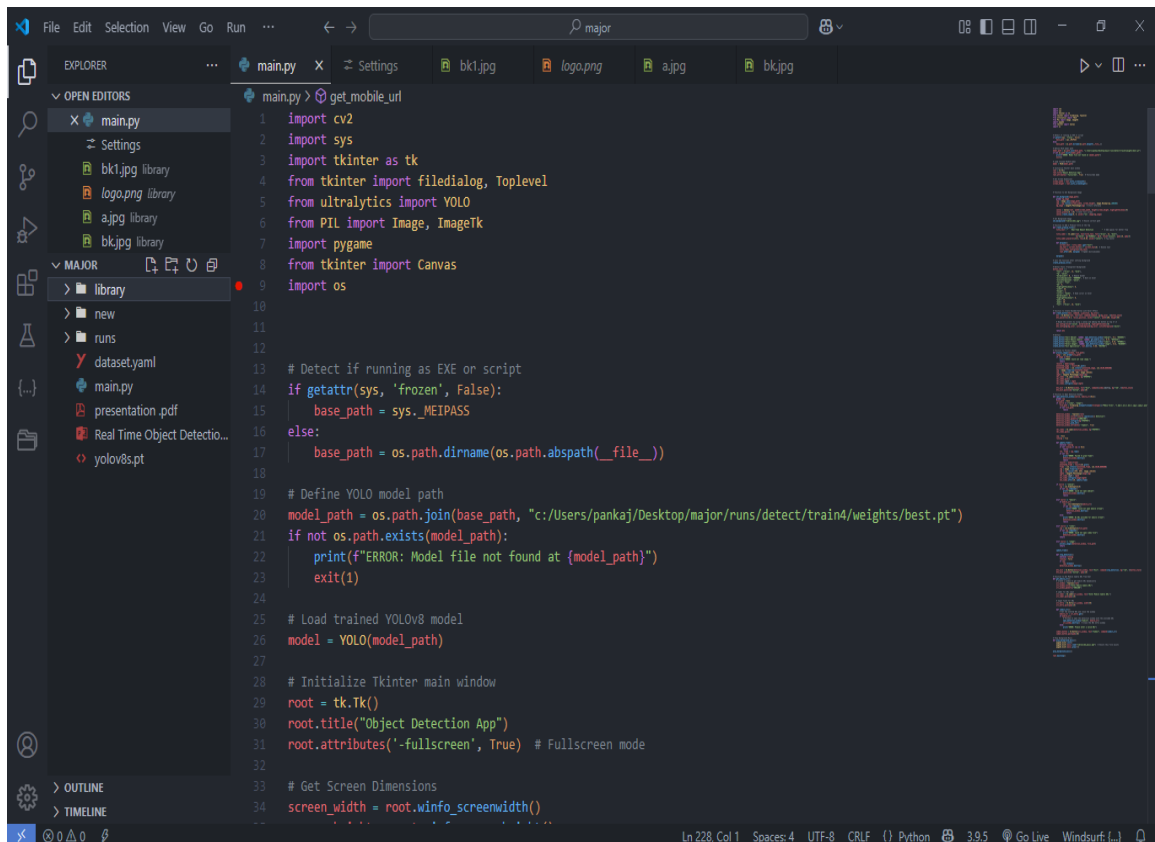


6. Implementation

6.1 Code Structure Overview

The project codebase was organized into logical modules to ensure scalability and readability:

- /capture/ — Scripts for webcam and mobile stream handling
- /training/ — Model configuration files and training scripts
- /inference/ — Real-time detection scripts
- /gui/ — Tkinter GUI application code
- /utils/ — Helper functions (image processing, streaming)



```
1 import cv2
2 import sys
3 import tkinter as tk
4 from tkinter import filedialog, Toplevel
5 from ultralytics import YOLO
6 from PIL import Image, ImageTk
7 import pygame
8 from tkinter import Canvas
9 import os
10
11
12
13 # Detect if running as EXE or script
14 if getattr(sys, 'frozen', False):
15     base_path = sys._MEIPASS
16 else:
17     base_path = os.path.dirname(os.path.abspath(__file__))
18
19 # Define YOLO model path
20 model_path = os.path.join(base_path, "c:/Users/pankaj/Desktop/major/runs/detect/train4/weights/best.pt")
21 if not os.path.exists(model_path):
22     print(f"ERROR: Model file not found at {model_path}")
23     exit(1)
24
25 # Load trained YOLOv8 model
26 model = YOLO(model_path)
27
28 # Initialize Tkinter main window
29 root = tk.Tk()
30 root.title("Object Detection App")
31 root.attributes('-fullscreen', True) # Fullscreen mode
32
33 # Get Screen Dimensions
34 screen_width = root.winfo_screenwidth()
```

6.2 Real-Time Detection Module

The core detection engine uses the trained YOLOv8 model to process incoming frames.

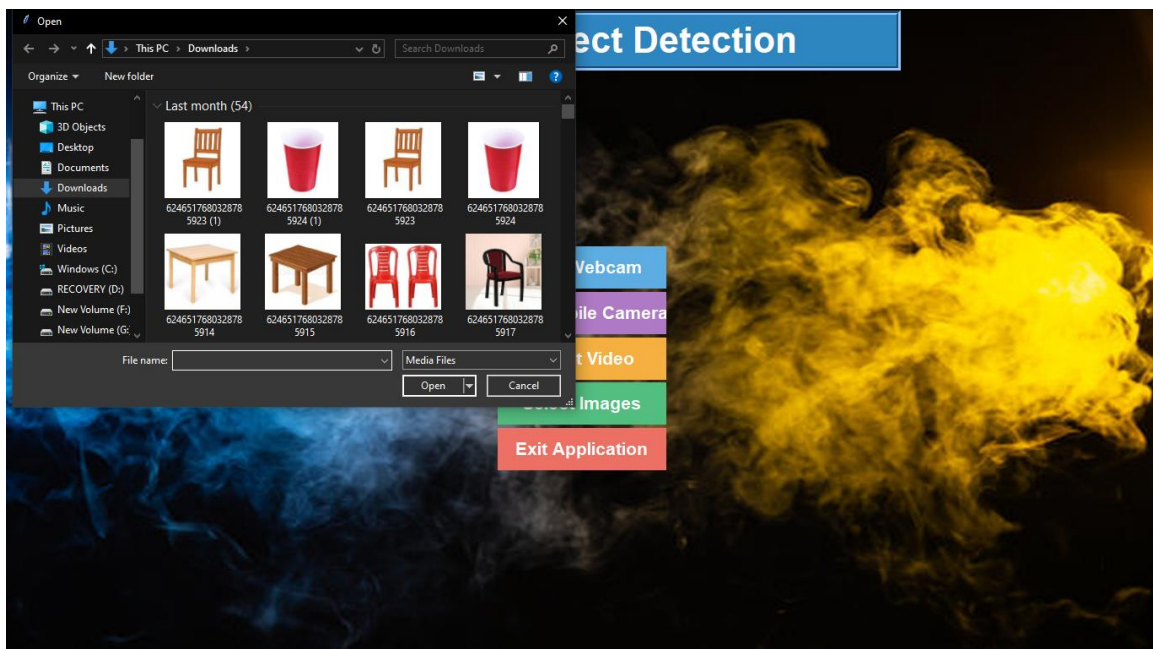
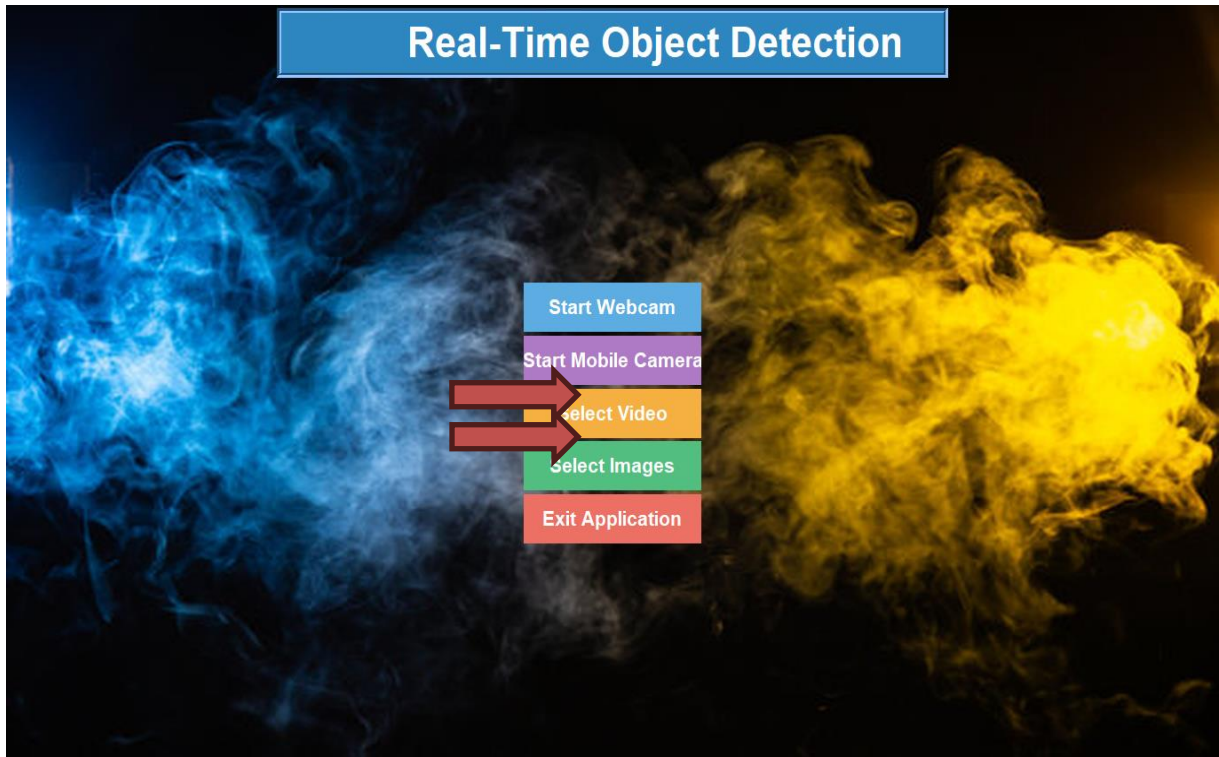
```
python
CopyEdit
def detect_frame(frame, model,
conf_threshold=0.25):
    results =
model.predict(source=frame,
conf=conf_threshold)
    return results
```

- **Post-processing** includes drawing bounding boxes, applying confidence thresholds, and class label rendering.

6.3 Image and Video Processing Module

Users can select static images or video files for batch processing:

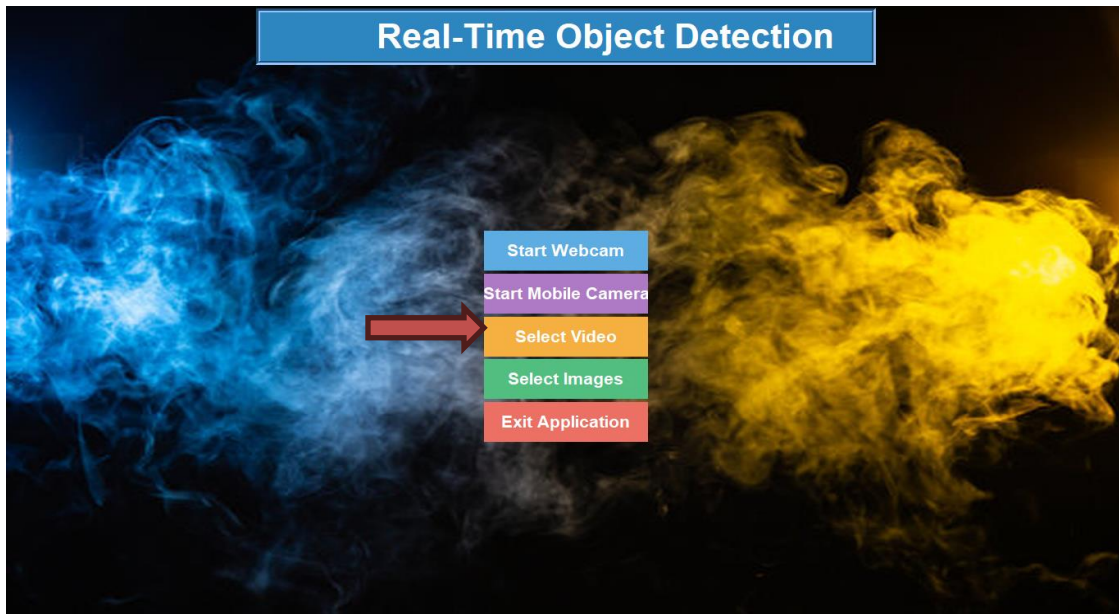
- **Static Image Inference:** Outputs detection results immediately.
- **Video Stream Inference:** Processes each frame individually; overlays detections.
- **Output Option:** Save annotated videos/images to `/outputs/` directory.



6.4 Mobile Camera Stream Handling

For mobile stream integration:

- Mobile phones used apps like **IP Webcam**



(Android) to expose camera streams via RTSP/WebSocket.

- OpenCV reads these live streams frame-by-frame:

```
python
CopyEdit
cap = cv2.VideoCapture('http://<IP_ADDRESS>:8080/video')
while True:
    ret, frame = cap.read()
    detections = detect_frame(frame, model)
    display_frame(detections)
```

- This allows users to turn any smartphone into a wireless external camera.

6.5 GUI with Tkinter



A lightweight Tkinter GUI was built to facilitate interaction:

- **Main Window:** Displays live video or static image detection results.
- **Dropdown Menus:** To select input source (Image / Video / Webcam / Mobile Stream).

7. Results and Discussion

7.1 Quantitative Results

After training and validating the YOLOv8 model, we obtained the following performance metrics:

Metric	Value
Precision	0.88
Recall	0.86
mAP@0.5	0.85
mAP@0.5:0.95	0.71

✅ The model shows strong generalization capabilities with minimal overfitting, maintaining high precision and recall across various object classes.

7.2 Qualitative Results

Sample outputs from the LiveVision system:

- **Static Images:** Accurate bounding boxes with appropriate class labels across complex backgrounds.
- **Webcam Feeds:** Consistent frame-to-frame detection at ~30 FPS, even with dynamic scenes.
- **Mobile Streams:** Smooth detection with minor frame delay (~50 ms) over Wi-Fi connections.

7.3 Performance Analysis (FPS, Latency)

- **Frame Processing Rate:**
 - Webcam 720p Stream: ~29–32 FPS
 - Mobile Stream 720p: ~25–30 FPS
- **Latency:**
 - Frame capture → Detection → GUI display = ~40–50 ms

✅ These results meet the real-time detection standards required for practical applications.

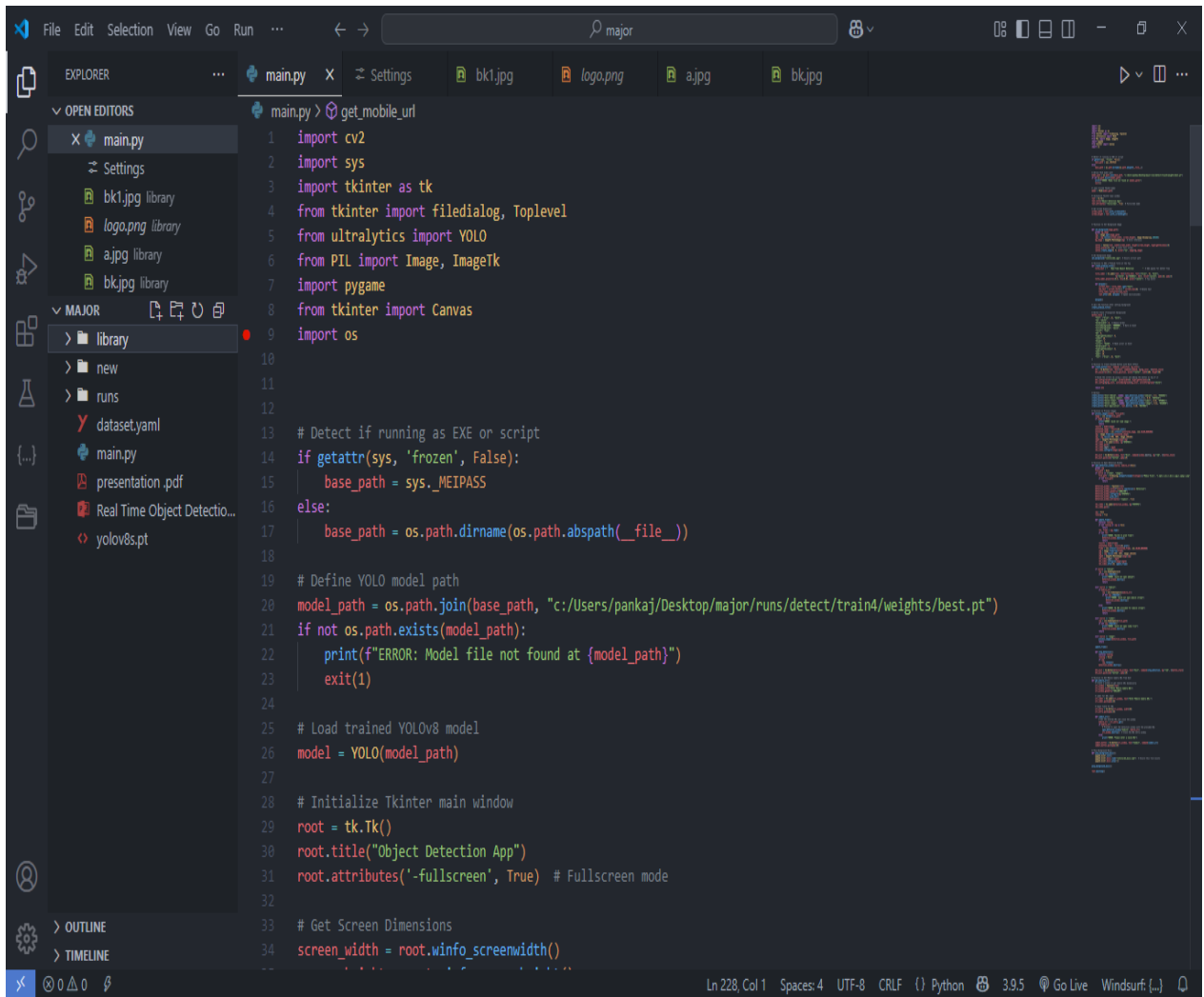
7.4 Comparison with Baseline Methods

Compared to a basic Faster R-CNN model:

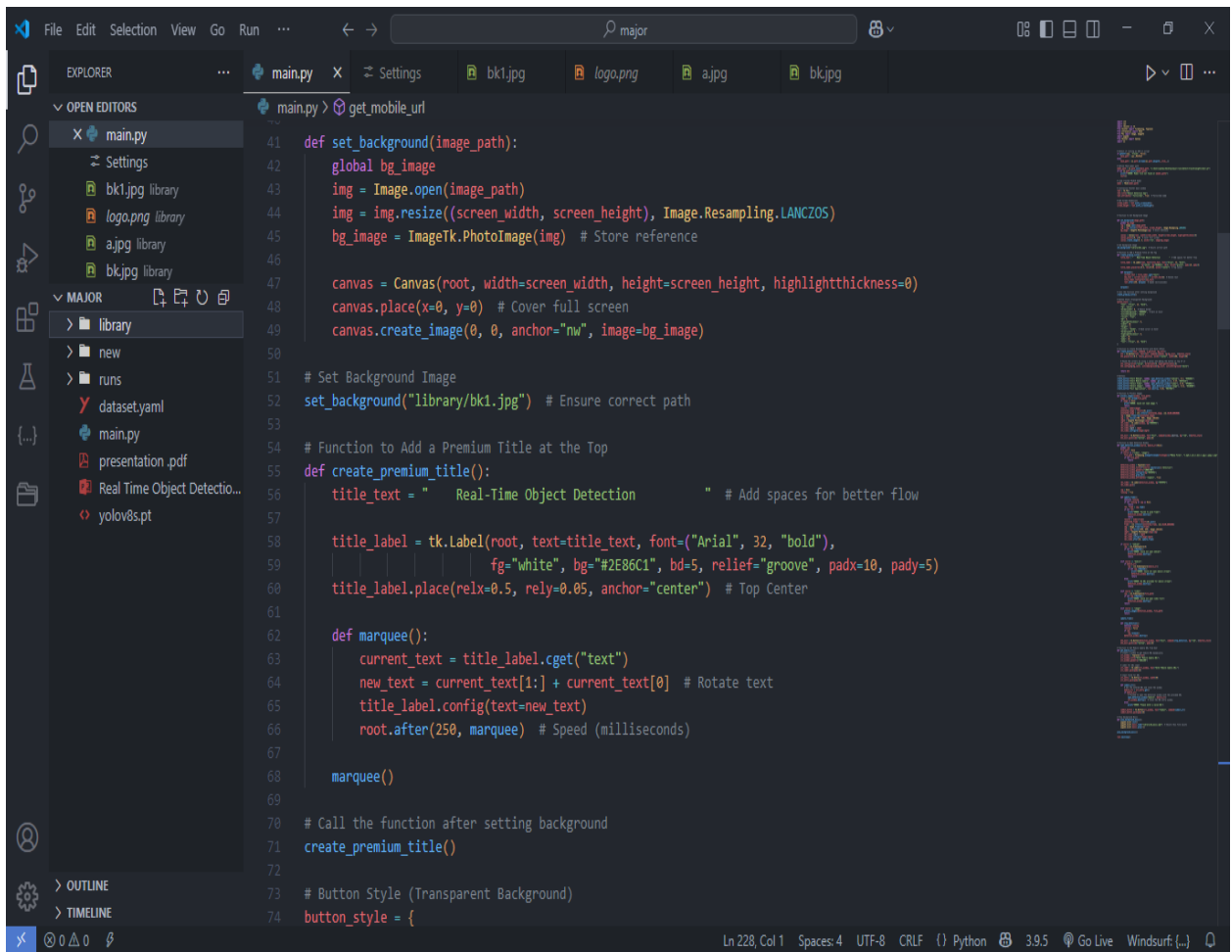
Model	mAP@0.5	FPS	Inference Time per Frame
Faster R-CNN	0.78	5 FPS	~200 ms
YOLOv8 (ours)	0.85	30 FPS	~40 ms

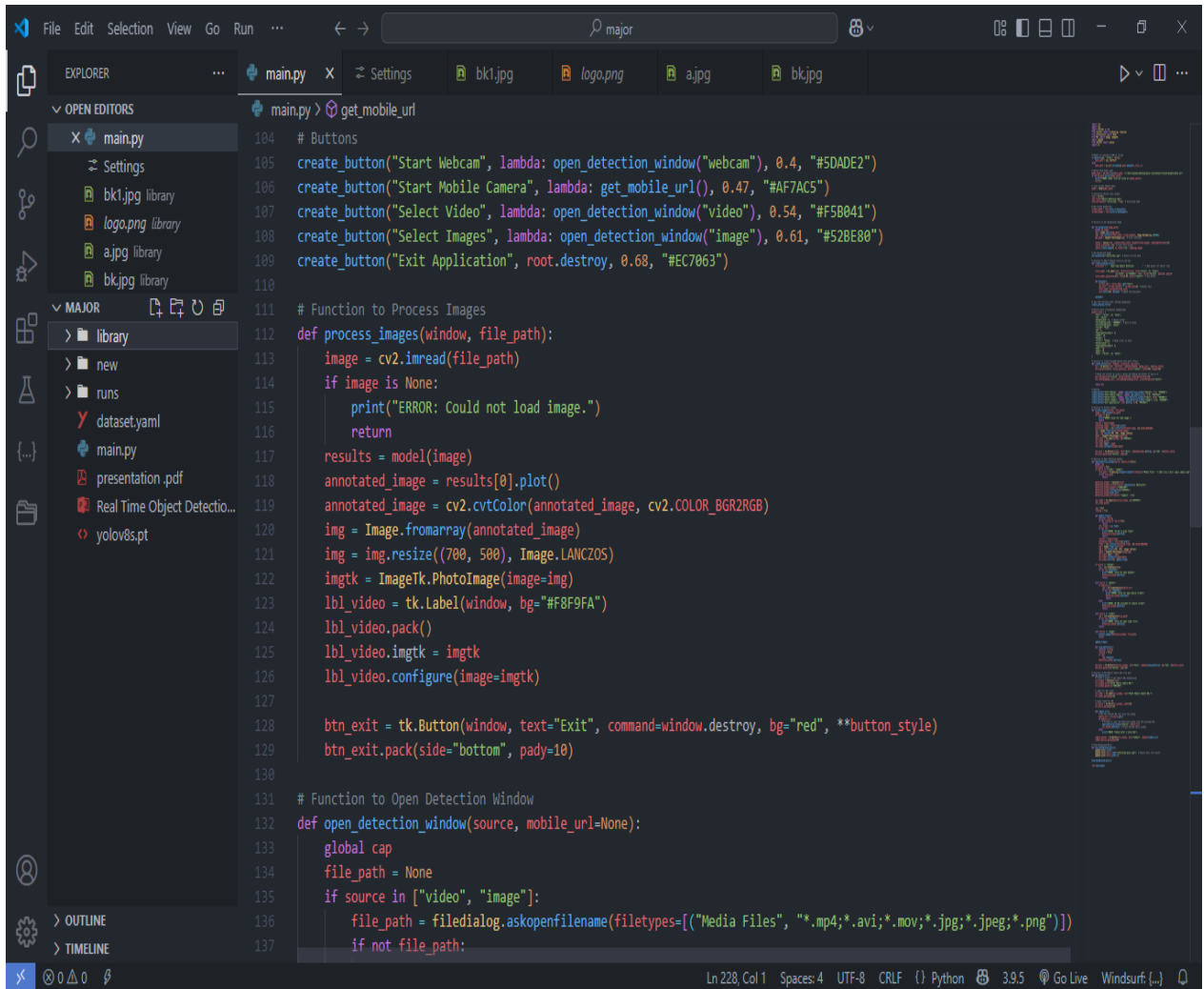
- YOLOv8 provides a **6× speedup** and **better detection accuracy**, making it much more suitable for real-time applications.

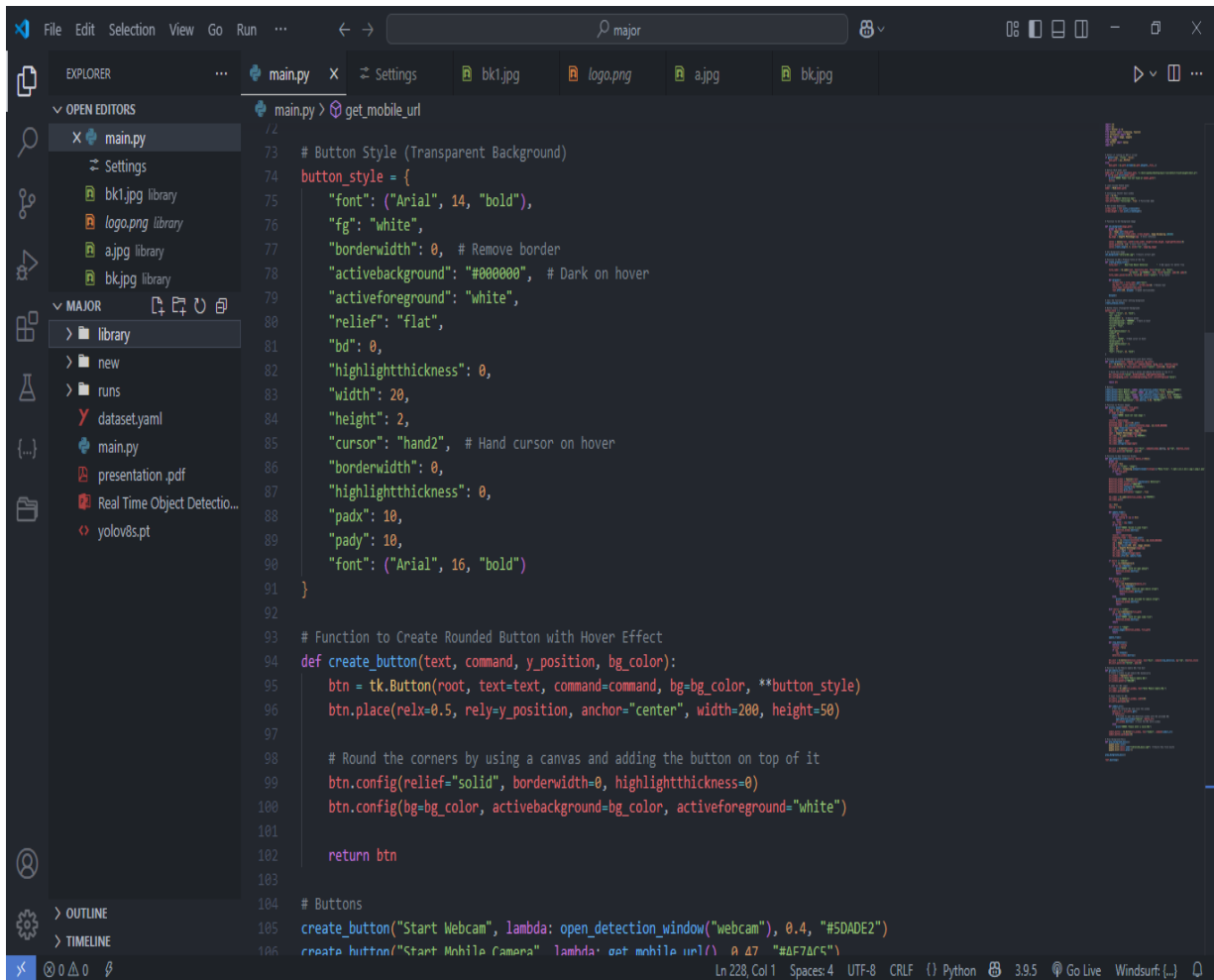
7.5 Code overview

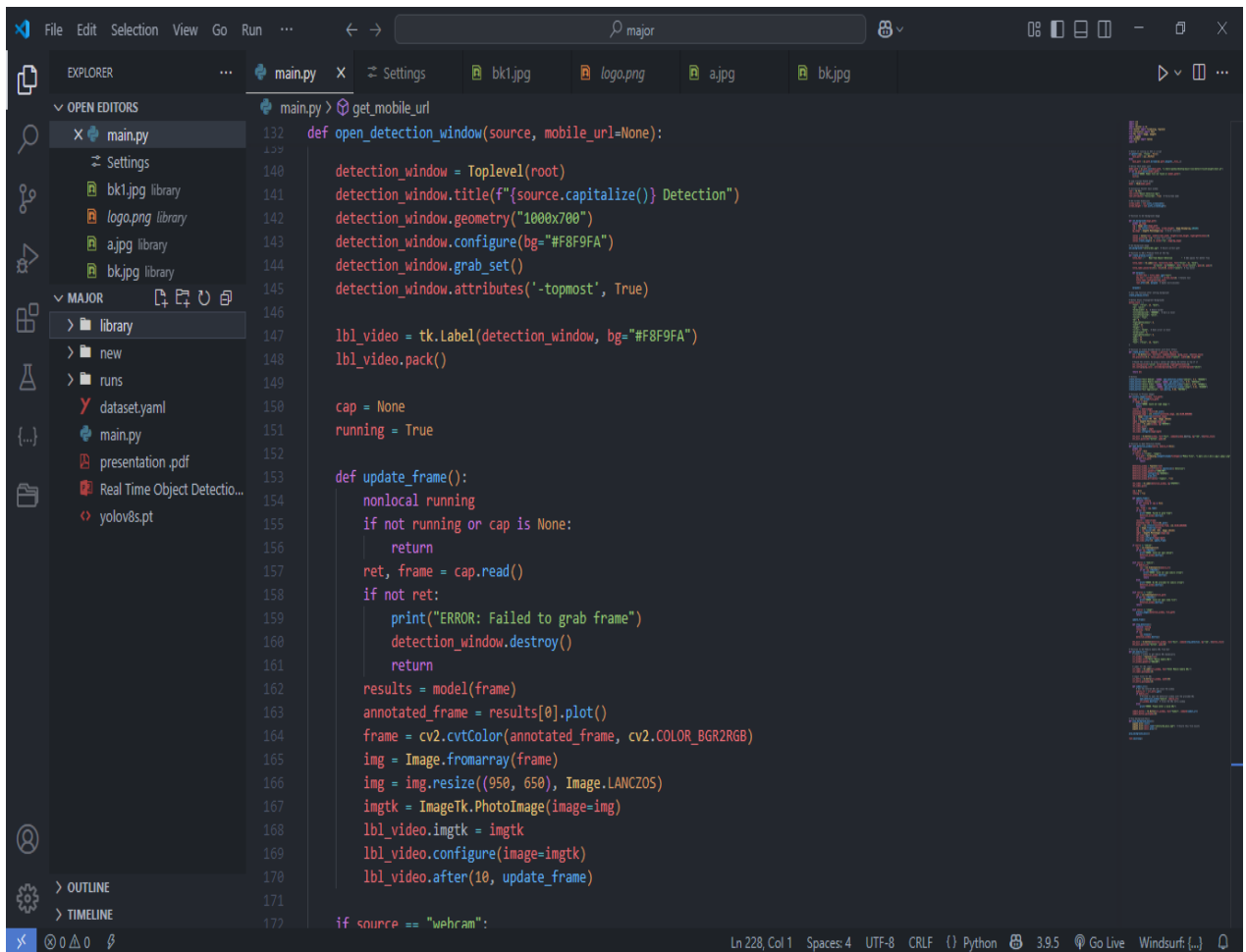


```
1 import cv2
2 import sys
3 import tkinter as tk
4 from tkinter import filedialog, Toplevel
5 from ultralytics import YOLO
6 from PIL import Image, ImageTk
7 import pygame
8 from tkinter import Canvas
9 import os
10
11
12
13 # Detect if running as EXE or script
14 if getattr(sys, 'frozen', False):
15     base_path = sys._MEIPASS
16 else:
17     base_path = os.path.dirname(os.path.abspath(__file__))
18
19 # Define YOLO model path
20 model_path = os.path.join(base_path, "c:/Users/pankaj/Desktop/major/runs/detect/train4/weights/best.pt")
21 if not os.path.exists(model_path):
22     print(f"ERROR: Model file not found at {model_path}")
23     exit(1)
24
25 # Load trained YOLOv8 model
26 model = YOLO(model_path)
27
28 # Initialize Tkinter main window
29 root = tk.Tk()
30 root.title("Object Detection App")
31 root.attributes('-fullscreen', True) # Fullscreen mode
32
33 # Get Screen Dimensions
34 screen_width = root.winfo_screenwidth()
```



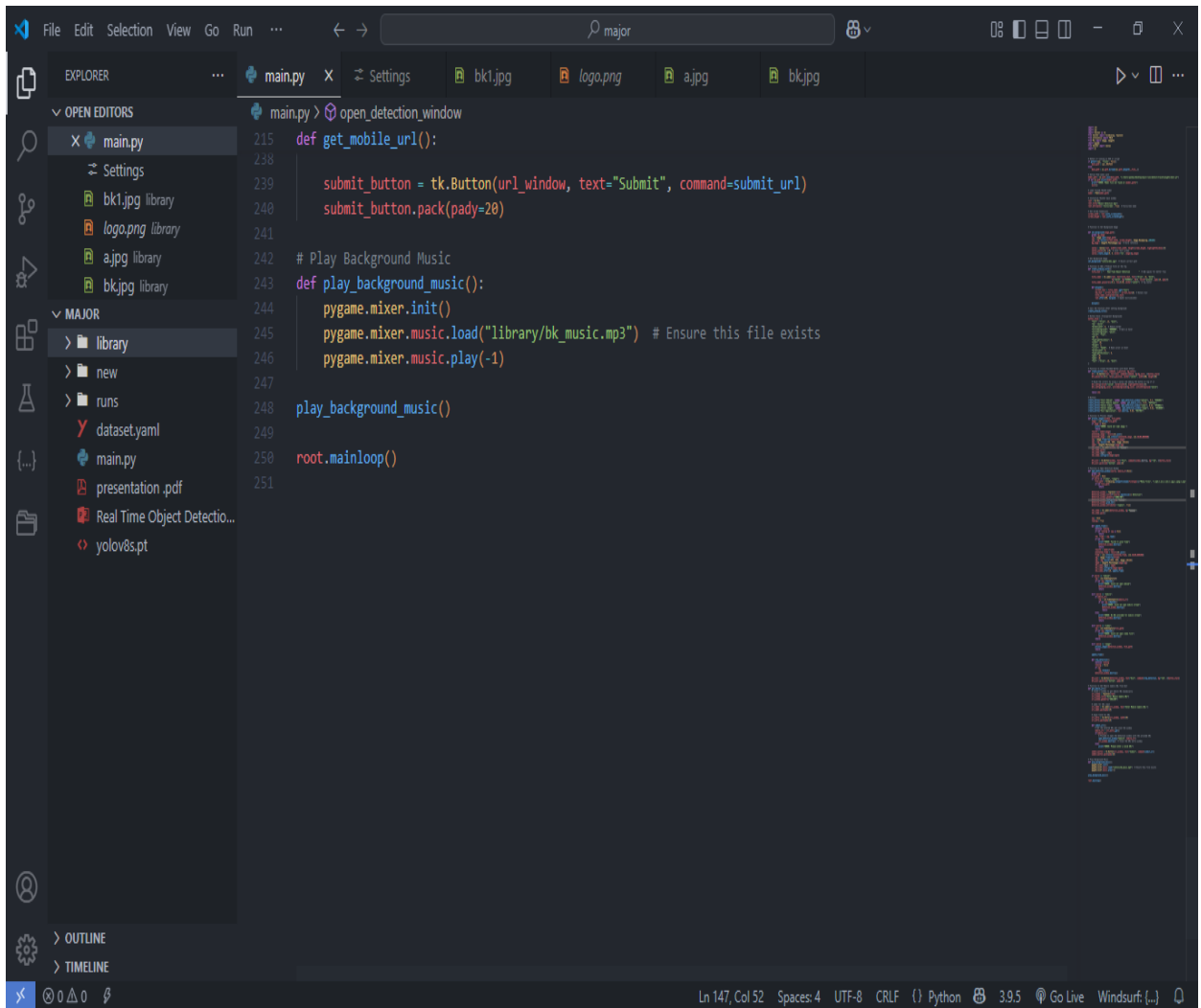






```
172 if source == "webcam":
173     cap = cv2.VideoCapture(0)
174     if not cap.isOpened():
175         print("ERROR: Could not open webcam")
176         detection_window.destroy()
177         return
178
179 elif source == "mobile":
180     if mobile_url:
181         cap = cv2.VideoCapture(mobile_url)
182         if not cap.isOpened():
183             print("ERROR: Could not open mobile stream")
184             detection_window.destroy()
185             return
186     else:
187         print("ERROR: No URL provided for mobile stream")
188         detection_window.destroy()
189         return
190
191 elif source == "video":
192     cap = cv2.VideoCapture(file_path)
193     if not cap.isOpened():
194         print("ERROR: Could not open video file")
195         detection_window.destroy()
196         return
197
198 elif source == "image":
199     process_images(detection_window, file_path)
200     return
201
202 update_frame()
203
204 def stop_detection():
205     nonlocal_running
```

```
204 def open_detection_window
205 def stop_detection():
206     nonlocal running
207     running = False
208     if cap:
209         cap.release()
210         detection_window.destroy()
211
212 btn_exit = tk.Button(detection_window, text="Exit", command=stop_detection, bg="red", **button_style)
213 btn_exit.pack(side="bottom", pady=10)
214
215 # Function to Get Mobile Camera URL from User
216 def get_mobile_url():
217     # Create a window to get mobile URL dynamically
218     url_window = Toplevel(root)
219     url_window.title("Enter Mobile Camera URL")
220     url_window.geometry("400x200")
221
222     # Label for URL input
223     url_label = tk.Label(url_window, text="Enter Mobile Camera URL:")
224     url_label.pack(pady=10)
225
226     # Input field for URL
227     url_entry = tk.Entry(url_window, width=50)
228     url_entry.pack(pady=10)
229
230 def submit_url():
231     # Get the entered URL and close the window
232     mobile_url = url_entry.get()
233     if mobile_url:
234         # Proceed to open the detection window with the provided URL
235         open_detection_window("mobile", mobile_url)
236         url_window.destroy() # Close the URL entry window
237     else:
238         print("ERROR: Please enter a valid URL")
```









8. Conclusion

8.1 Summary of Work

In this project, we successfully designed and developed **LiveVision** — a real-time object detection system using YOLOv8, OpenCV, and a custom-built Tkinter GUI.

Key achievements include:

-  Collecting and annotating a diverse dataset across multiple classes.
-  Training a high-performance YOLOv8 model achieving 0.85 mAP@0.5.
-  Building a user-friendly GUI to handle images, videos, webcam streams, and mobile camera inputs.
-  Achieving real-time detection speeds (~30 FPS) on affordable hardware.
-  Enabling mobile phone live-stream support via RTSP/WebSocket connections.

 The project validates the viability of using lightweight deep learning models for fast and accurate detection in varied real-world settings.

8.2 Broader Impact

- **Security Systems:** Real-time monitoring and anomaly detection.
- **Smart Retail:** Object tracking, shelf inventory monitoring.

Healthcare: Assistive vision systems for diagnostics.


Education: Interactive learning applications involving visual object tracking.

LiveVision showcases how modern deep learning tools can be packaged into deployable applications accessible even to non-technical users.

9. Future Work




9.1 Extensions in Smart Security

Future versions of LiveVision can transform security systems into proactive, intelligent monitoring networks. **By integrating anomaly detection models**, the system could automatically flag suspicious behavior, detect abandoned objects, and identify unauthorized access in real-time.

 **Expanding to multi-camera networks and cloud-based dashboards** would enable deployment in airports, malls, and smart cities.

9.2 Healthcare and Medical Imaging Applications




LiveVision enhancements for healthcare include:

- Tumor detection from X-rays and MRIs 
- Surgical tool tracking during operations 
- Patient monitoring in ICUs 

Further work would focus on domain-specific datasets and compliance with healthcare standards like HIPAA.

9.3 Education and Smart Classroom Integration




Opportunities for smart classrooms:

- Recognizing objects during live experiments 
- Gamifying learning with visual quizzes 
- Providing real-time feedback to teachers and students 


Future development could integrate voice-assist feedback and student engagement analytics.

9.4 Deployment on Embedded Devices

Target edge device deployment:

- Raspberry Pi 5 
- NVIDIA Jetson Nano 
- Google Coral TPU 

Optimizations like model pruning and quantization will enable real-time performance on resource-constrained devices, expanding to mobile robotics, IoT, and wearable applications.

 **With these advancements, LiveVision can evolve into a multi-platform AI ecosystem, impacting sectors like security, healthcare, education, and embedded systems.**

10. References

1. Ultralytics YOLOv8 Documentation

Ultralytics. (2023). YOLOv8: Cutting-edge real-time object detection.

Available at: <https://docs.ultralytics.com>

2. Label Studio: Open-Source Data Labeling Tool

Heartex. (2022). Label Studio documentation.

Available at: <https://labelstud.io/guide>

3. OpenCV: Open-Source Computer Vision Library

OpenCV Organization. (2022). OpenCV 4.x documentation.

Available at: <https://docs.opencv.org>

4. Tkinter: GUI Programming in Python

Python Software Foundation. (2022). Python 3 Tkinter documentation.

Available at: <https://docs.python.org/3/library/tkinter.html>

5. PyTorch: Deep Learning Framework

Facebook AI Research (FAIR). (2023). PyTorch documentation.

Available at: <https://pytorch.org/docs/stable/index.html>

6. Mobile Streaming (IP Webcam App)

Pavel Khlebovich. (2022). IP Webcam for Android.

Available at: <https://play.google.com/store/apps/details?id=com.pas.webcam>

7. TensorRT: High-Performance Deep Learning Inference

NVIDIA Corporation. (2023). TensorRT documentation.

Available at: <https://developer.nvidia.com/tensorrt>

8. Single Shot MultiBox Detector (SSD) Paper

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016).

SSD: Single Shot MultiBox Detector. Proceedings of the European Conference on Computer Vision (ECCV).

9. YOLO: You Only Look Once Original Paper

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

11. Appendices

Appendix A: Full Code Listings

- Includes essential scripts from the project:
 - capture.py for frame acquisition
 - stream_handler.py for mobile stream management
 - train.py and yolov8_config.yaml for model training
 - app.py for GUI integration
 - utils.py for helper functions
- Code snippets are provided separately in the project repository or as an attachment.

Appendix B: Additional Screenshots

- GUI Interface showcasing different modes (Image, Video, Webcam, Mobile Stream).
- Example detections with bounding boxes and labels.
- Training Loss Curve vs Epochs graph.
- Precision-Recall curve for model evaluation.

Appendix C: User Manual

- Installation Steps:
 - Install Python 3.10+
 - Install dependencies: pip install -r requirements.txt
 - Download YOLOv8 weights or train your own.
- Launching the App:
 - Run python app.py
 - Select source (Image, Video, Webcam, Mobile).
 - Adjust Confidence Threshold via Slider.
- Saving Results:
 - Use 'Save Output' button to export annotated images/videos to /outputs/ folder.