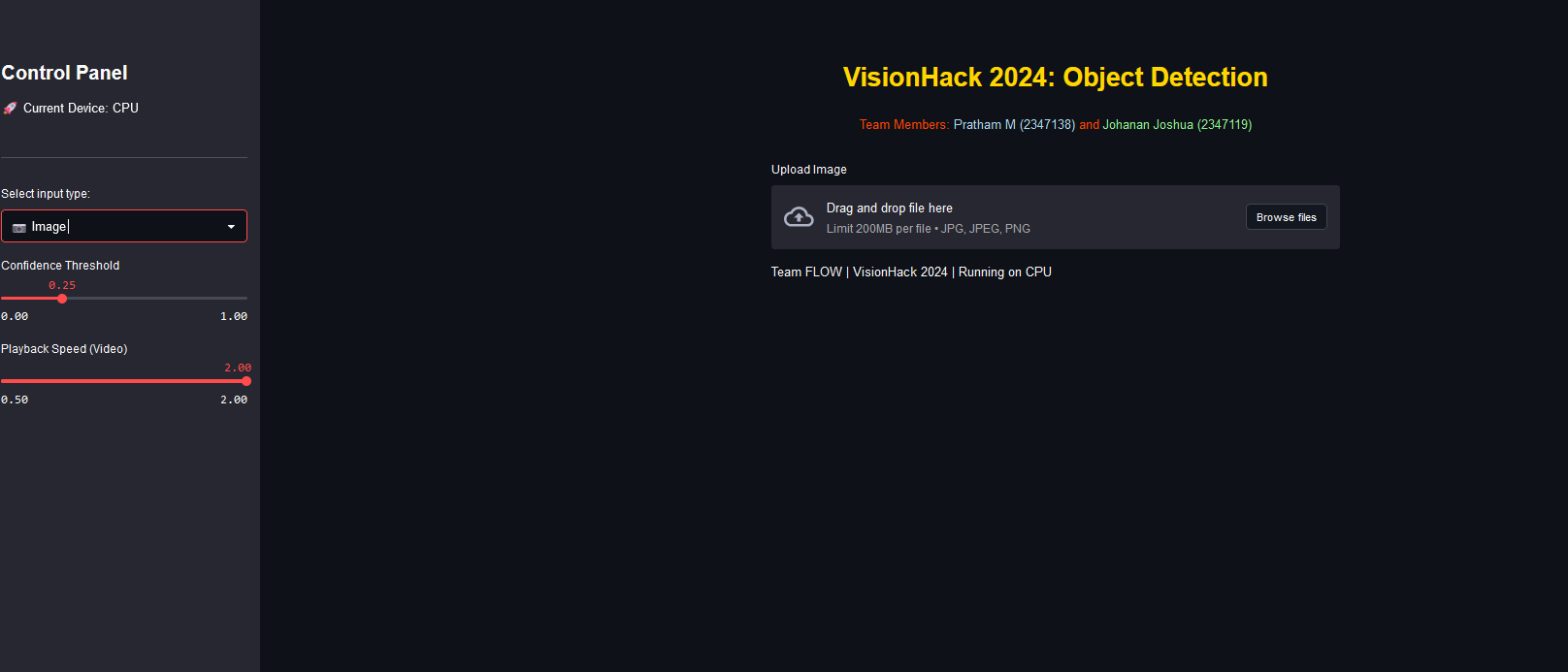
**OBJECT DETECTION**

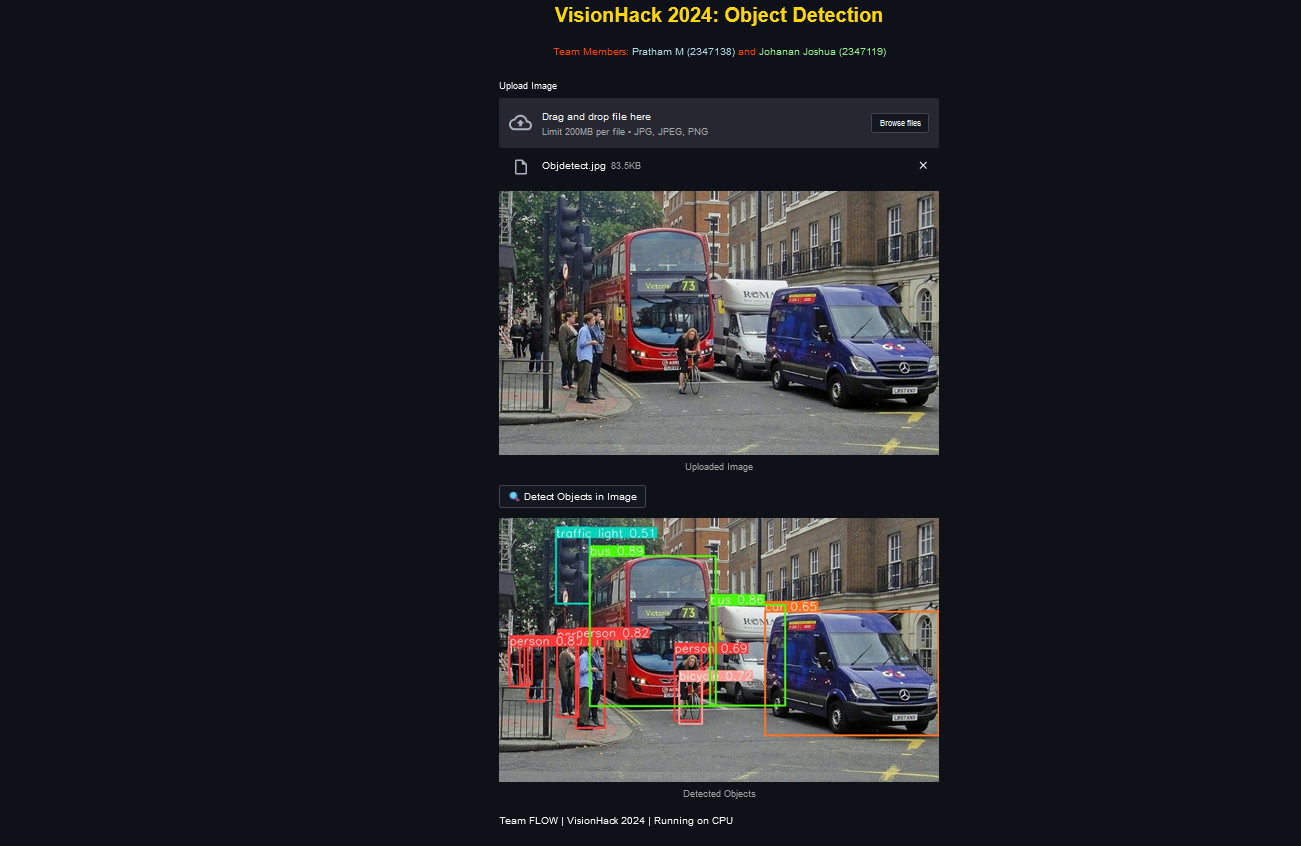
By Pratham M (2347138)

UI :

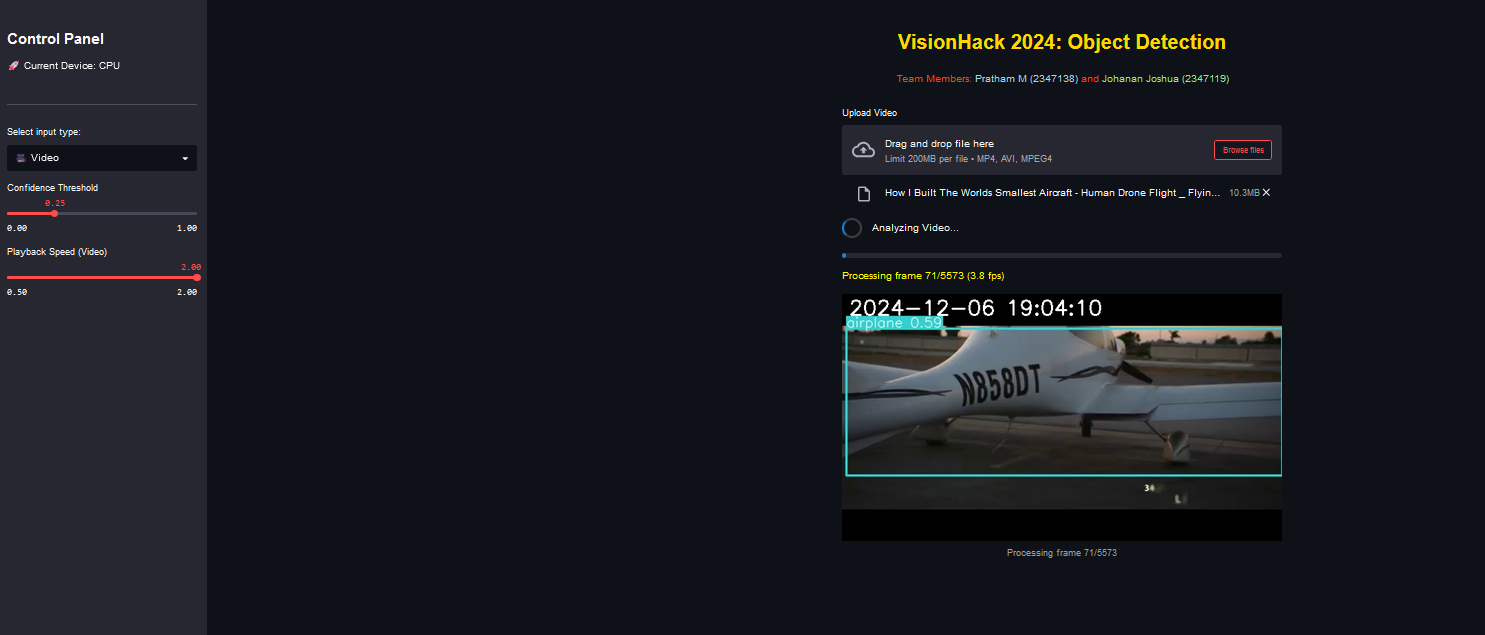


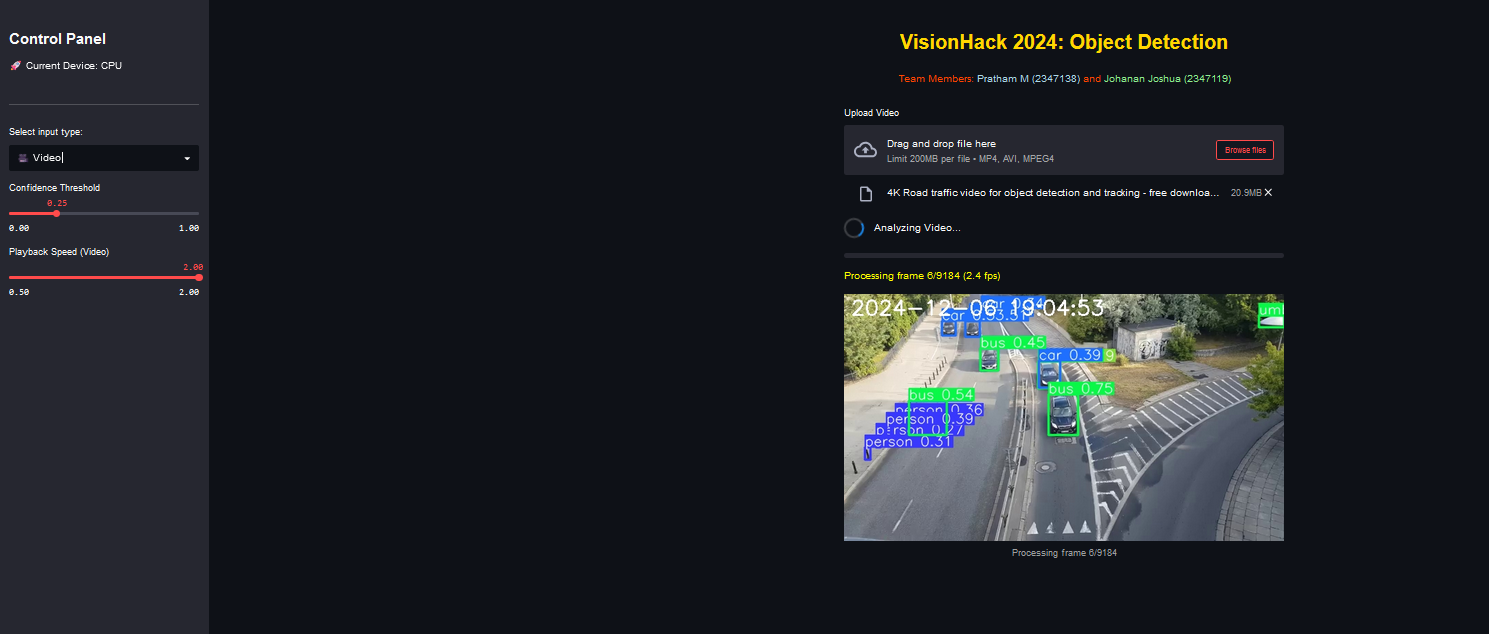
Output :

Object detection using Image

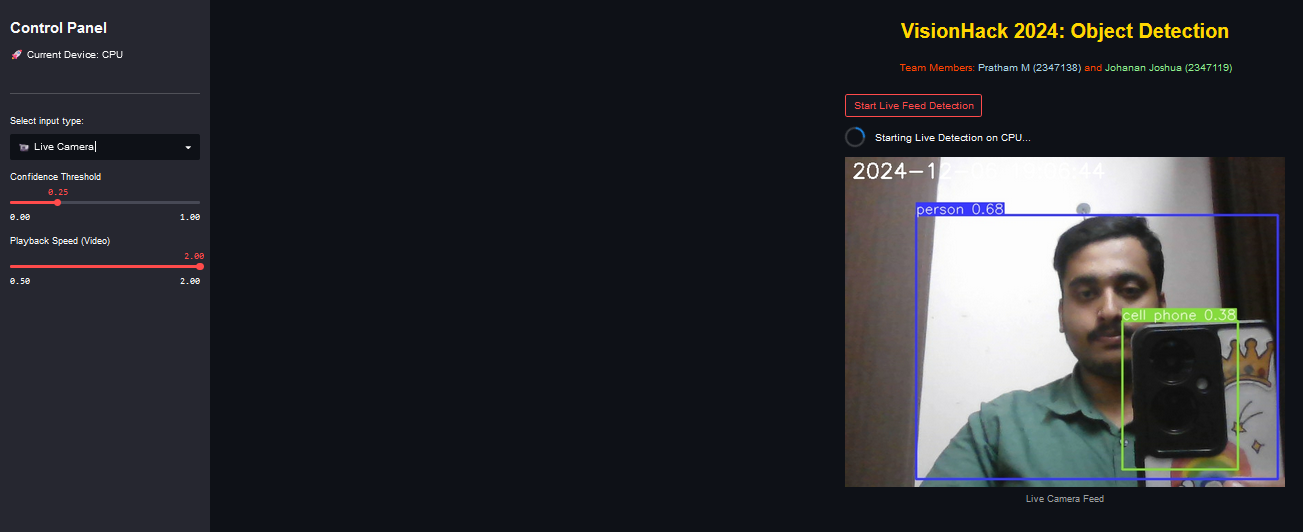


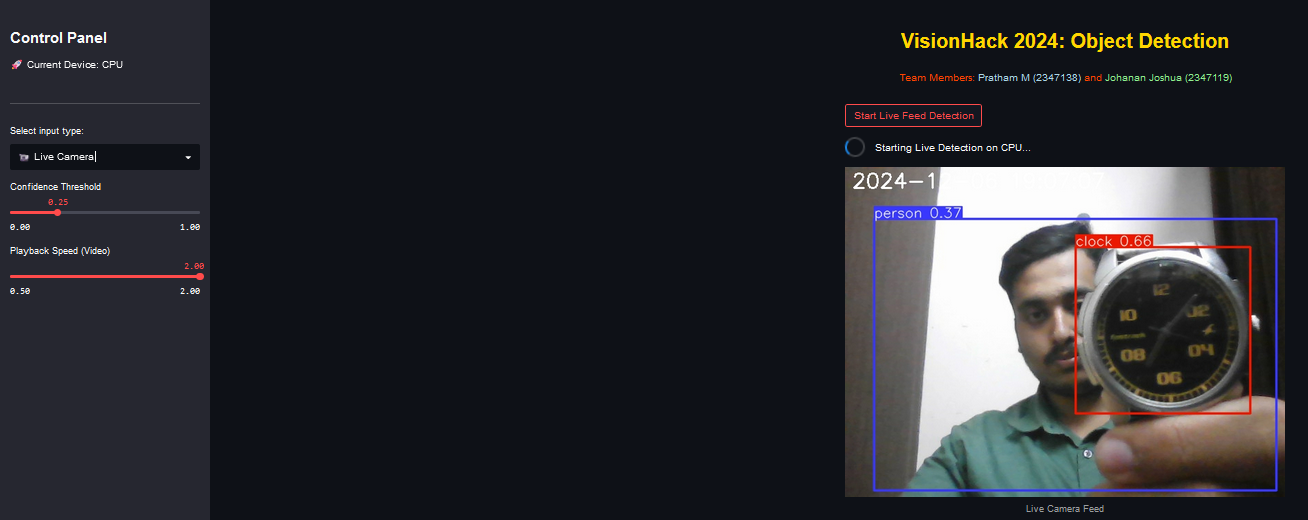
Object detection using video





Object detection using Live camera





I have used YOLOv5’s Pretrained models which are trained on the **COCO dataset (Common Objects in Context)**.

The **COCO dataset** contains over 330,000 labeled images across 80 object classes, such as:

* Person
* Vehicle (car, truck, bicycle, etc.)
* Animals (dog, cat, bird, etc.)
* Household items (chair, sofa, bed, etc.)
* Outdoor objects (traffic light, stop sign, etc.)

**1. Problem Being Solved**

* Detect objects in images and videos using a real-time object detection system.
* Identify and classify objects such as people, vehicles, and animals from predefined categories.

**2. Overview of the Approach**

* Use YOLOv5 (You Only Look Once Version 5), a state-of-the-art object detection algorithm.
* Leverage pretrained weights trained on the COCO dataset for high-accuracy object detection without requiring custom dataset training.

**3. How the System Works**

* **Input**:
  + Image or video file.
  + Alternatively, real-time video feed from a webcam.
* **Process**:
  + Load the pretrained YOLOv5 model.
  + Pass the input through the model to generate bounding boxes, class labels, and confidence scores.
* **Output**:
  + Annotated image/video with bounding boxes around detected objects, their labels, and confidence scores.

**4. Challenges Faced and Solutions**

* **Challenge**: Low accuracy on certain object classes.
  + **Solution**: Fine-tune the YOLOv5 model on a custom dataset specific to the target domain.
* **Challenge**: High latency in processing video streams.
  + **Solution**: Optimize inference speed by using smaller YOLO models (e.g., yolov5s) and leveraging GPU acceleration.
* **Challenge**: Misclassification in cluttered environments.
  + **Solution**: Apply post-processing techniques, such as non-maximum suppression (NMS), to refine predictions.

**5. System Architecture**

* **Model**: YOLOv5 (e.g., yolov5s, yolov5m, yolov5l).
* **Framework**: PyTorch for model loading and inference.
* **Input Pipeline**:
  + Load input images/videos using OpenCV or direct webcam streams.
* **Output Pipeline**:
  + Visualize results using OpenCV.
  + Save annotated outputs to disk.
* **Hardware**: GPU acceleration with CUDA for fast inference.

**6. Dataset Used**

* **COCO Dataset (Common Objects in Context)**:
  + Contains 330,000 labeled images.
  + Covers 80 object categories.
  + Includes bounding boxes, segmentation masks, and captions for labeled objects.

**Code:**

**App.py**

import torch

import cv2

import numpy as np

import tempfile

import os

import shutil

import time

import streamlit as st

from datetime import datetime

from PIL import Image

# Performance Optimization: Use torch.hub.load with device selection

def load\_yolo\_model(device='auto'):

    """Load YOLOv5 model with device optimization."""

    if device == 'auto':

        device = 'cuda' if torch.cuda.is\_available() else 'cpu'

    try:

        model = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True, device=device)

        return model, device

    except Exception as e:

        st.error(f"Model loading failed: {e}. Falling back to CPU.")

        model = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True, device='cpu')

        return model, 'cpu'

# Global model initialization

model, DEVICE = load\_yolo\_model()

def detect\_objects\_in\_image(image):

    """Detect objects in the image using YOLOv5 with performance optimizations."""

    try:

        # Ensure image is converted to correct format

        if isinstance(image, Image.Image):

            image = np.array(image)

        # Performance: Add inference mode and disable gradient computation

        with torch.no\_grad():

            results = model(image)

        return results

    except Exception as e:

        st.error(f"Image detection error: {e}")

        return None

def get\_compatible\_codec():

    """Try different codecs to find a compatible one with error handling."""

    codecs = [

        ('mp4v', '.mp4'),

        ('XVID', '.avi'),

        ('MJPG', '.avi'),

        ('WMV1', '.wmv')

    ]

    temp\_filename = tempfile.mktemp(suffix='.avi')

    test\_size = (640, 480)

    for codec, ext in codecs:

        try:

            fourcc = cv2.VideoWriter\_fourcc(\*codec)

            out = cv2.VideoWriter(temp\_filename, fourcc, 20, test\_size)

            if out.isOpened():

                out.release()

                os.remove(temp\_filename)

                return codec, ext

        except Exception as e:

            print(f"Codec {codec} not compatible: {e}")

    return 'XVID', '.avi'

def detect\_objects\_in\_video(uploaded\_file, progress\_bar, status\_text, frame\_placeholder, conf\_threshold=0.25, skip\_frames=1):

    """Optimized video object detection with improved performance."""

    with tempfile.NamedTemporaryFile(delete=False, suffix=".mp4") as temp\_file:

        shutil.copyfileobj(uploaded\_file, temp\_file)

        temp\_video\_path = temp\_file.name

    cap = cv2.VideoCapture(temp\_video\_path)

    if not cap.isOpened():

        st.error(f"Error opening video file at {temp\_video\_path}.")

        return None

    total\_frames = int(cap.get(cv2.CAP\_PROP\_FRAME\_COUNT))

    fps = int(cap.get(cv2.CAP\_PROP\_FPS))

    width = int(cap.get(cv2.CAP\_PROP\_FRAME\_WIDTH))

    height = int(cap.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))

    codec, ext = get\_compatible\_codec()

    output\_path = tempfile.mktemp(suffix=ext)

    out = cv2.VideoWriter(output\_path, cv2.VideoWriter\_fourcc(\*codec), fps // max(1, skip\_frames), (width, height))

    # Performance: Pre-configure model settings

    model.conf = conf\_threshold

    model.iou = 0.45  # Intersection over Union threshold

    frame\_count = 0

    processed\_count = 0

    start\_time = time.time()

    try:

        while cap.isOpened():

            ret, frame = cap.read()

            if not ret:

                break

            frame\_count += 1

            if frame\_count % skip\_frames != 0:

                continue

            # Performance: Use no\_grad context

            with torch.no\_grad():

                results = model(frame)

                annotated\_frame = np.array(results.render()[0])

            timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

            cv2.putText(annotated\_frame, timestamp, (10, 30),

                       cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 255, 255), 2)

            frame\_placeholder.image(cv2.cvtColor(annotated\_frame, cv2.COLOR\_BGR2RGB),

                                 caption=f'Processing frame {frame\_count}/{total\_frames}',

                                 use\_column\_width=True)

            out.write(annotated\_frame)

            progress = min(float(frame\_count) / total\_frames, 1.0)

            progress\_bar.progress(progress)

            elapsed\_time = time.time() - start\_time

            fps\_rate = processed\_count / elapsed\_time if elapsed\_time > 0 else 0

            status\_text.markdown(f'<span style="color: yellow;">Processing frame {frame\_count}/{total\_frames} ({fps\_rate:.1f} fps)</span>', unsafe\_allow\_html=True)

            processed\_count += 1

    finally:

        cap.release()

        out.release()

        cv2.destroyAllWindows()

        os.remove(temp\_video\_path)

    return output\_path

def detect\_objects\_in\_live\_camera(conf\_threshold=0.25):

    """Optimized live camera detection with performance improvements."""

    cap = cv2.VideoCapture(0, cv2.CAP\_DSHOW)  # Use DirectShow on Windows for better performance

    cap.set(cv2.CAP\_PROP\_FRAME\_WIDTH, 640)    # Set lower resolution for faster processing

    cap.set(cv2.CAP\_PROP\_FRAME\_HEIGHT, 480)

    if not cap.isOpened():

        st.error("Error accessing the camera.")

        return

    st\_frame = st.empty()

    model.conf = conf\_threshold

    model.iou = 0.45

    try:

        while True:

            ret, frame = cap.read()

            if not ret:

                st.warning("Failed to read from camera.")

                break

            # Performance: Use no\_grad context

            with torch.no\_grad():

                results = model(frame)

                annotated\_frame = np.array(results.render()[0])

            timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

            cv2.putText(annotated\_frame, timestamp, (10, 30),

                       cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 255, 255), 2)

            st\_frame.image(cv2.cvtColor(annotated\_frame, cv2.COLOR\_BGR2RGB), caption="Live Camera Feed", use\_column\_width=True)

    finally:

        cap.release()

        cv2.destroyAllWindows()

def main():

    st.set\_page\_config(

        page\_title="VisionHack 2024 - Object Detection",

        page\_icon="🎥",

        layout="centered",  # Reduced screen size

        initial\_sidebar\_state="collapsed"  # Sidebar starts collapsed

    )

    # Caching the CSS for faster loading

    st.markdown("""

        <style>

            body {

                margin: 0 auto;

                max-width: 900px; /\* Limit the content width \*/

            }

            .title {

                color: #FFD700;

                text-align: center;

                font-size: 2rem; /\* Adjusted font size \*/

                font-weight: bold;

                margin-bottom: 20px;

            }

            .subtitle {

                color: #FF4500;

                text-align: center;

                font-size: 1rem;

                margin-bottom: 30px;

            }

        </style>

    """, unsafe\_allow\_html=True)

    st.markdown(f"<div class='title'>VisionHack 2024: Object Detection </div>", unsafe\_allow\_html=True)

    st.markdown("<div class='subtitle'>Team Members: <span style='color: lightblue;'>Pratham M (2347138)</span> and <span style='color: lightgreen;'>Johanan Joshua (2347119)</span></div>", unsafe\_allow\_html=True)

    st.sidebar.title("Control Panel")

    st.sidebar.markdown(f"🚀 Current Device: {DEVICE.upper()}")

    st.sidebar.markdown("---")

    file\_type = st.sidebar.selectbox("Select input type:", ("📷 Image", "🎥 Video", "📹 Live Camera"))

    conf\_threshold = st.sidebar.slider("Confidence Threshold", 0.0, 1.0, 0.25)

    playback\_speed = st.sidebar.slider("Playback Speed (Video)", 0.5, 2.0, 1.0, step=0.1)

    if file\_type == "📷 Image":

        uploaded\_file = st.file\_uploader("Upload Image", type=["jpg", "jpeg", "png"])

        if uploaded\_file:

            st.image(Image.open(uploaded\_file), caption="Uploaded Image", use\_column\_width=True)

            if st.button("🔍 Detect Objects in Image"):

                with st.spinner("Analyzing Image..."):

                    results = detect\_objects\_in\_image(Image.open(uploaded\_file))

                    if results:

                        st.image(np.array(results.render()[0]), caption="Detected Objects", use\_column\_width=True)

    elif file\_type == "🎥 Video":

        uploaded\_file = st.file\_uploader("Upload Video", type=["mp4", "avi"])

        if uploaded\_file:

            with st.spinner("Analyzing Video..."):

                progress\_bar = st.progress(0)

                status\_text = st.empty()

                frame\_placeholder = st.empty()

                output\_video\_path = detect\_objects\_in\_video(uploaded\_file, progress\_bar, status\_text, frame\_placeholder, conf\_threshold)

                if output\_video\_path:

                    st.video(output\_video\_path)

    elif file\_type == "📹 Live Camera":

        if st.button("Start Live Feed Detection"):

            with st.spinner(f"Starting Live Detection on {DEVICE.upper()}..."):

                detect\_objects\_in\_live\_camera(conf\_threshold)

    st.markdown(f"<div class='team'>Team FLOW | VisionHack 2024 | Running on {DEVICE.upper()}</div>", unsafe\_allow\_html=True)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**object\_detection.py**

import torch

import cv2

import numpy as np

import tempfile

import os

import time

# Load YOLOv5 model

model = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True)

def detect\_objects\_in\_image(image):

    results = model(image)

    return results

def get\_compatible\_codec():

    """Try different codecs to find a compatible one"""

    codecs = [

        ('mp4v', '.mp4'),

        ('XVID', '.avi'),

        ('MJPG', '.avi'),

        ('WMV1', '.wmv')

    ]

    temp\_filename = tempfile.mktemp(suffix='.avi')

    test\_size = (640, 480)

    for codec, ext in codecs:

        try:

            fourcc = cv2.VideoWriter\_fourcc(\*codec)

            out = cv2.VideoWriter(temp\_filename, fourcc, 20, test\_size)

            if out.isOpened():

                out.release()

                os.remove(temp\_filename)

                return codec, ext

        except:

            continue

    return 'XVID', '.avi'  # Default fallback

def detect\_objects\_in\_video(uploaded\_file, progress\_bar, status\_text, frame\_placeholder, conf\_threshold=0.25, skip\_frames=2):

    model = load\_model()

    # Create a temporary file for the uploaded video

    with tempfile.NamedTemporaryFile(delete=False, suffix=".mp4") as temp\_file:

        shutil.copyfileobj(uploaded\_file, temp\_file)  # Save the uploaded file to the temporary file

        temp\_video\_path = temp\_file.name

    # Verify the video file path

    st.write(f"Temporary video file path: {temp\_video\_path}")  # Debugging

    cap = cv2.VideoCapture(temp\_video\_path)

    if not cap.isOpened():

        raise Exception("Error opening video file")

    total\_frames = int(cap.get(cv2.CAP\_PROP\_FRAME\_COUNT))

    fps = int(cap.get(cv2.CAP\_PROP\_FPS))

    width = int(cap.get(cv2.CAP\_PROP\_FRAME\_WIDTH))

    height = int(cap.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))

    codec = cv2.VideoWriter\_fourcc(\*'mp4v')

    output\_path = tempfile.mktemp(suffix='.mp4')

    out = cv2.VideoWriter(output\_path, codec, fps // skip\_frames, (width, height))

    frame\_count = 0

    processed\_count = 0

    start\_time = time.time()

    model.conf = conf\_threshold

    try:

        while cap.isOpened():

            ret, frame = cap.read()

            if not ret:

                break

            frame\_count += 1

            if frame\_count % skip\_frames != 0:

                continue

            results = model(frame)

            annotated\_frame = np.array(results.render()[0])

            # Add timestamp to frame

            timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

            cv2.putText(annotated\_frame, timestamp, (10, 30),

                       cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 255, 255), 2)

            frame\_placeholder.image(cv2.cvtColor(annotated\_frame, cv2.COLOR\_BGR2RGB),

                                 caption=f'Processing frame {frame\_count}/{total\_frames}',

                                 use\_column\_width=True)

            out.write(annotated\_frame)

            progress = min(float(frame\_count) / total\_frames, 1.0)

            progress\_bar.progress(progress)

            elapsed\_time = time.time() - start\_time

            fps\_rate = processed\_count / elapsed\_time if elapsed\_time > 0 else 0

            status\_text.text(f'Processing frame {frame\_count}/{total\_frames} ({fps\_rate:.1f} fps)')

            processed\_count += 1

    finally:

        cap.release()

        out.release()

        cv2.destroyAllWindows()

        os.remove(temp\_video\_path)  # Clean up temporary video file

    return output\_path

**Instructions how to run**

**1.System Requirements**

* **Operating System**: Windows/Linux/macOS.
* **Python**: Version 3.7 or later.

**2. Install Required Software**

* **Python**: Install Python from the [official website](https://www.python.org/downloads/).
* **Git**: Install Git for cloning repositories

**3. Set Up the Environment**

1. Open a terminal or command prompt.
2. Create a virtual environment (optional but recommended):

Run this in terminal

**python -m venv yolov5-env**

**source yolov5-env/bin/activate # For Linux/Mac**

**yolov5-env\Scripts\activate # For Windows**

**4. Clone the YOLOv5 Repository**

**git clone https://github.com/ultralytics/yolov5.git**

**cd yolov5**

**5. Install Required Dependencies**

**pip install -r requirements.txt**

**6. Download Pretrained YOLOv5 Weights**

* Download the YOLOv5 weights (e.g., yolov5s.pt) directly:

python detect.py --weights yolov5s.pt --source 0 # To download weights during execution

**7. Running Object Detection**

* **On Images**:

python detect.py --weights yolov5s.pt --source path/to/image.jpg

* **On a Video**:

python detect.py --weights yolov5s.pt --source path/to/video.mp4

* **Real-Time from Webcam**:

python detect.py --weights yolov5s.pt --source 0

**8. Customize the Run**

To save results in a specific directory:

python detect.py --weights yolov5s.pt --source path/to/image.jpg --project results --name run1

**9.Adjust confidence threshold:**

python detect.py --weights yolov5s.pt --source path/to/image.jpg --conf-thres 0.4

#### 10 Troubleshooting

* **Issue**: CUDA not found or GPU unavailable.
  + Ensure CUDA and cuDNN are correctly installed.
  + Use CPU by adding --device cpu in the command:

python detect.py --weights yolov5s.pt --source path/to/image.jpg --device cpu

**Missing Dependencies**:

* Re-run pip install -r requirements.txt.

**Cleanup (Optional)**

* Deactivate the virtual environment:

deactivate