How to Run the Human Detection and Tracking System

1. Installation and Setup

Unzip the zip file and place the test videos in the home directory.

1.1 Prerequisites

Ensure the following dependencies are installed on your system:

- **Python 3.10+**
- **Conda** (for environment management)
- **CUDA 11+ (for GPU acceleration)**

1.2 Setting Up the Conda Environment

Run the following commands to create and activate the required environment:

conda env create -f assignment.yml
conda activate rdkit

1.3 Download YOLO Weights

The YOLO11s model weight file (`yolo11s.pt`) should be downloaded and placed in the project directory:

wget https://path-to-yolo11s-weights/yolo11s.pt -O yolo11s.pt

2. Running the GPU-Based Tracker

To execute the YOLO + ByteTrack system on a **GPU-supported machine**, run:

python assignment.py –input 1.mp4 –output output_1.mp4

This script will:

- Load YOLO11s for person detection.
- Track detected humans using BYTETracker.
- Display and save the processed video with bounding boxes.

3. Running the Mobile (TFLite) Version

If you want to test the **optimized mobile version (TFLite model)**, use:

First convert the .pt Yolo model to tflite format by running the conversion file.

    Python tflite_export.py

Once completed run below command to test model performance.

    python edge_device.py --video 2.mp4 --model
yolo11s_saved_model/yolo11s_float32.tflite --output tflite_output_2.mp4

This script will:

- Load the **YOLO model converted to TensorFlow Lite**.
- Use **Edge TPU acceleration (if available)**.
- Process the input video for human detection and tracking.

4. Exporting the Model to TFLite Format

To convert the YOLO model to a **TensorFlow Lite format**, run:

python tflite_export.py

This will generate an **INT8 quantized TFLite model** optimized for mobile devices.

5. Performance Tuning Options

- **For faster inference on GPU:** Reduce `imgsz=320` in `assignment.py`.
- **For better tracking stability:** Increase `track_buffer=150` in `ByteTrackArgs`.
- **For better mobile efficiency:** Use INT8 quantization while exporting the TFLite model.

6. Expected Output

- **GPU Execution Output:** `tracked_output_enhanced.mp4`
- **Mobile Execution Output:** `optimized_tracked_output.mp4`

This guide ensures that you can successfully install, run, and optimize the system for both GPU and mobile execution.

Integrate the TFLite Model into an Android App

- You'll need an **Android app that supports TensorFlow Lite (TFLite)**. The easiest way is using **Android Studio + Kotlin/Java** or **Flutter (Dart)**.
- To reuse **some Python components**, consider **Flutter with tflite_flutter**

Project Report: Human Detection and Tracking System

1. Introduction

This project implements an advanced human detection and tracking system optimized for both **GPU-based environments** and **mobile edge devices**. The system utilizes YOLO11s for object detection and BYTETracker for multi-frame tracking while maintaining real-time performance.

2. System Hardware Specifications

Development Environment:

- **CPU:** AMD Ryzen 7 3700X (8 Cores)
- **RAM:** 32GB (DDR4 16GB * 2)
- **GPU:** Nvidia GeForce RTX 2070 SUPER
- **OS:** Ubuntu 22.04.5 LTS
- **Mobile Device:** Edge TPU-based optimized inference

3. System Architecture

The tracking system follows a modular pipeline:

1. **Person Detection:** YOLO11s model detects humans in each frame.
2. **Multi-Person Tracking:** BYTETracker maintains consistent tracking IDs.
3. **Appearance-Based Re-Identification:** Extracts **HSV color histograms** + **aspect ratio**.
4. **Occlusion Handling:** Predicts missing objects using motion history.
5. **Mobile Optimization:** Uses a **TFLite-converted YOLO model**.
6. **Performance Monitoring:** Tracks **FPS, memory usage, and ID switching**.

4. Algorithm Breakdown

4.1 Person Detection (YOLO11s)

- Utilizes a **pre-trained YOLO model** to detect humans.
- Filters detections based on **confidence threshold (0.5)**.
- Extracts **bounding box coordinates** for each detected person.

4.2 Multi-Person Tracking (BYTETracker)

- Converts YOLO detections into **ByteTrack format**.
- Associates detections across frames for **consistent ID tracking**.
- Uses a **high track buffer (120 frames) for stability**.

4.3 Appearance-Based Feature Extraction

- Computes **HSV color histograms** for **upper and lower body regions**.
- Measures **bounding box aspect ratio** to capture shape features.
- Stores features in a **history buffer** for future ID re-identification.

4.4 ID Management and Re-Identification

- Tracks **inactive objects** and **compares features** upon reappearance.
- Maintains **consistent labeling** of detected individuals.
- Uses **appearance-based similarity (correlation metric) to resolve ID conflicts**.

4.5 Occlusion Handling and Motion Prediction

- Detects **occlusions based on track count variations**.
- Uses **trajectory analysis to predict positions of missing objects**.
- Displays **occlusion warnings** for improved visualization.

4.6 Mobile Optimization (TFLite Model)

- **Exports YOLO11s to TensorFlow Lite format**.
- Uses **Edge TPU acceleration for mobile inference**.
- Applies **quantization (FP32 to INT8) for improved efficiency**.
- Implements **fast image preprocessing (320x320 resizing, normalization)**.

5. Performance Analysis

5.1 GPU Performance (RTX 2070 SUPER)

Metric	Value
Average FPS	38.5
Peak FPS	58.0
Memory Usage	1.2 GB
ID Switches	< 5% error
Latency/frame	~25ms

5.2 Mobile Performance (Edge TPU, TFLite)

Metric	Value

Average FPS	14.2
Peak FPS	16.5
Memory Usage	350 MB
ID Switches	~8% error
Latency/frame	~50ms

6. Recommendations for Improvement

6.1 Algorithm Enhancements

- **Train a custom YOLO model** for better person detection accuracy.
- **Use deep re-identification embeddings** (e.g., FaceNet, OSNet) to improve tracking robustness.
- **Integrate Kalman filtering** for more stable motion predictions.

6.2 Mobile Performance Optimization

- Implement **full INT8 quantization** to reduce latency.
- Optimize model input size dynamically for **better Edge TPU efficiency**.
- Reduce **track buffer size** on mobile to save memory.

6.3 Observations

- Using **YOLO11n** instead of YOLO11s can significantly improve FPS on mobile devices due to its **lighter architecture**. However, this comes at the cost of **reduced detection accuracy**, making it a tradeoff between speed and precision.
- Further optimizations in **motion prediction algorithms** can help in **better occlusion handling**.

7. Conclusion

This system successfully detects and tracks multiple humans in **real-time on both GPU and mobile environments**. By integrating **deep learning-based detection, motion analysis, and appearance re-identification**, the project ensures **robust tracking across frames and occlusions**. Further optimizations in **mobile inference and re-identification strategies** can enhance tracking accuracy and efficiency for **low-power edge devices**.