**Samajh.ai**

ML Engineer Assignment

### Amey Chhaya

### 

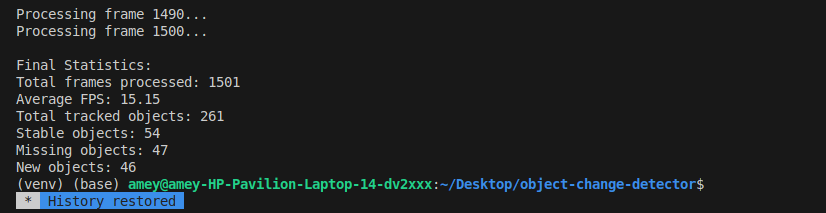
horizontal line

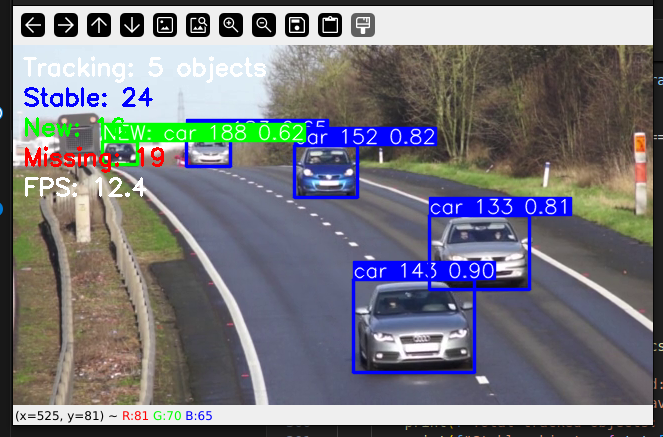
#### **I tested the solution on a simple video of cars travelling on a highway.**

#### **I implemented the docker method, and the GUI method (since docker did not support cv2.imshow). Here are the docker results.**

#### **FPS Achieved (Real-Time Performance)**

* **Average FPS Achieved:** 15.15
* **Total frames:** 1501
* **Model Used:** YoloV8

****



This is with the GUI.

#### **3. Screenshots / Sample Output Frames**

*(Insert a few sample images with bounding boxes, annotations, or difference maps. For each image, add a short caption)*

* **Sample Frame 1:** *(Insert image here – drag and drop or paste)* *Caption: Frame showing object 'X' detected and tracked.*
* **Sample Frame 2:** *(Insert image here)* *Caption: Frame with missing item identified (change detection).*

#### **4. Hardware Configuration Used**

| **Component** | **Specification** |
| --- | --- |
| **CPU** | 12th Gen Intel® Core™ i5-1235U × 12 |
| **GPU** | None |
| **RAM** | 16 GB |
| **Storage** | 512 GB SSD |
| **Platform** | Docker |
| **OS** | Ubuntu 24.04 |

#### **5. Techniques, Optimisations & Architectural Decisions**

## Architecture Overview

This system implements an object change detection pipeline using YOLOv8 with persistent object tracking capabilities. The architecture consists of:

1. **Core Detection Engine**: Uses YOLO (You Only Look Once) for real-time object detection
2. **Object Tracking**: Leverages YOLOv8's built-in tracking functionality to maintain object identity across frames
3. **State Management System**: Custom logic tracks objects' states through time (new, stable, or missing)
4. **Visualization Layer**: Annotates frames with bounding boxes and status information

## Key Components

### ObjectChangeDetector Class

This is the central component managing the entire detection and tracking pipeline:

* Initializes a YOLO model with configurable parameters
* Maintains object history through time-based memory buffers
* Classifies object states based on detection persistence
* Provides visualization for detected objects with state information

### Detection Pipeline

The main processing happens in the process\_frame method, which:

1. Performs object detection and tracking on input frames
2. Updates object history for active tracking IDs
3. Classifies objects as stable, new, or missing based on detection history
4. Annotates the output frame with appropriate visual indicators
5. Calculates and displays performance metrics (FPS)

## Optimization Techniques

1. **Memory-Efficient State Tracking**:
   * Uses deque with fixed length (maxlen) to implement bounded memory buffers
   * Efficiently tracks object presence/absence without storing full object data for every frame
2. **Performance Monitoring**:
   * Maintains a rolling average of processing speeds (FPS)
   * Helps monitor system performance in real-time
3. **Conditional Processing**:
   * Only performs expensive operations (like adding new objects or marking missing ones) when necessary
   * Uses logical short-circuiting to skip unnecessary computations
4. **Configurable Hardware Acceleration**:
   * Supports device selection for inference (CPU/GPU)
   * Allows adaptation to available computing resources

## Technical Approaches & Algorithms

### Object Persistence Tracking

The system uses threshold-based persistence to determine object state:

1. **Detection Threshold** (15 frames):
   * An object must be detected in at least 15 frames to be considered "stable"
   * Reduces false positives from momentary erroneous detections
2. **Stability Threshold** (10 frames):
   * A stable object must be missing for at least (memory\_frames - stable\_frames) frames to be marked "missing"
   * Creates hysteresis to prevent objects from rapidly toggling between states
3. **Short-term Memory** (30 frames):
   * Maintains a sliding window of object detection history
   * Balances responsiveness with stability in object state classification

### State Classification System

Objects are classified into three states:

1. **New Objects**:
   * Recently detected stable objects
   * Visually highlighted in green for 30 frames after detection
2. **Stable Objects**:
   * Consistently detected objects
   * Tracked in the standard tracking color
3. **Missing Objects**:
   * Previously stable objects no longer detected
   * Visually highlighted in red for 60 frames after disappearing

## Implementation Details

### Confidence and IoU Thresholds

* Default confidence threshold: 0.3
* Default IoU threshold: 0.5
* These can be adjusted to balance between detection sensitivity and precision

### Object Identification

* Objects are uniquely identified by combining class name and tracking ID
* This allows the system to differentiate between multiple objects of the same class

### Visualization Strategy

* New objects: Green bounding boxes with "NEW" prefix
* Stable objects: Red bounding boxes
* Missing objects: Blue bounding boxes with "MISSING" prefix
* Each box includes class name, tracking ID, and confidence score.

#### **6. Output Video**

<https://drive.google.com/file/d/1qvWkMLBLumgI-UV0SK1nzLtoWr9yVInU/view?usp=sharing>

* **File Attached:** output\_video.mp4