Real-Time Detection of Object Missing and New Object Placement in Video

Report for ML Engineer Intern Evaluation

*Vishal Singh*

April 28, 2025

**Contents**

1. [Executive Summary](#_bookmark0) 2
2. [System Architecture](#_bookmark1) 2
   1. [Core Components](#_bookmark2) 2
   2. [Detection Pipeline](#_bookmark3) 2
3. [Performance Metrics](#_bookmark4) 2
   1. [FPS Achievement](#_bookmark5) 2
   2. [Detection Accuracy](#_bookmark6) 2
4. [Hardware Configuration](#_bookmark7) 3
5. [Implementation Techniques & Optimizations](#_bookmark8) 3
   1. [Model Selection & Optimization](#_bookmark9) 3
   2. [Processing Optimizations](#_bookmark10) 3
      1. [Dual Resolution Processing](#_bookmark11) 3
      2. [Pre-allocation of Tensors](#_bookmark12) 3
      3. [Asynchronous Video Writing](#_bookmark13) 3
      4. [Frame Skipping Option](#_bookmark14) 3
   3. [Tracking Algorithm Enhancements](#_bookmark15) 3
      1. [State Management](#_bookmark16) 3
      2. [IoU-based Matching](#_bookmark17) 4
      3. [Track History](#_bookmark18) 4
6. [Output Visualization](#_bookmark19) 4
   1. [Color-coded bounding boxes](#_bookmark20) 4
   2. [Text labels](#_bookmark21) 4
   3. [Real-time FPS counter](#_bookmark22) 4
7. [Sample Output Frames](#_bookmark23) 5
8. [Challenges & Solutions](#_bookmark24) 6
   1. [Challenge 1: Balancing Speed vs. Accuracy](#_bookmark25) 6
   2. [Challenge 2: False Positives in Object State Changes](#_bookmark26) 6
   3. [Challenge 3: CPU/GPU Memory Management](#_bookmark27) 6
   4. [Challenge 4: Video Writing Performance Impact](#_bookmark28) 6
9. [Additional Features](#_bookmark29) 6
   1. [Command-line Interface](#_bookmark30) 6
   2. [Performance Statistics](#_bookmark31) 6
   3. [Scalable Processing](#_bookmark32) 6
10. [Future Improvements](#_bookmark33) 7
11. [Conclusion](#_bookmark34) 7

# Executive Summary

This report details the implementation of a real-time video analytics system capable of detecting both missing objects and new object placement in video streams. The system achieves high performance on modest hardware using YOLOv8 for object detection coupled with a custom ByteTrack-inspired algorithm for object tracking. The implementation focuses on optimizing both detection accuracy and processing speed to enable real-time analysis.

# System Architecture

## Core Components

1. **Object Detection**: YOLOv8n model from Ultralytics
2. **Object Tracking**: Custom ByteTrack-inspired implementation
3. **State Management**: Track history monitoring for object state changes
4. **Asynchronous Processing**: Threading for video writing operations

## Detection Pipeline

The system follows this workflow:

1. Video frame acquisition
2. Preprocessing and resizing
3. YOLOv8 inference
4. ByteTrack algorithm for tracking and state determination
5. Classification of objects as new, tracked, or missing
6. Visualization and output generation

# Performance Metrics

## FPS Achievement

The system achieved the following performance metrics:

* **Average FPS**: 42.7
* **Minimum FPS**: 36.2
* **Maximum FPS**: 46.5
* **Target FPS achieved**: Yes (target: 35+)

## Detection Accuracy

The system successfully detects:

* New objects appearing in the scene (highlighted in green)
* Tracked objects (highlighted in yellow)
* Missing/lost objects (highlighted in red for up to 15 frames)

# Hardware Configuration

Testing was conducted on the following hardware:

* **CPU**: Intel Core i5 11th Generation
* **RAM**: 8GB
* **GPU**: NVIDIA GeForce GTX 1650 (4GB VRAM)
* **CUDA**: Version 11.6

# Implementation Techniques & Optimizations

## Model Selection & Optimization

* **YOLOv8n**: Selected for its balance of speed and accuracy
* **Half-precision (FP16)**: Reduced memory usage and improved inference speed
* **TensorRT acceleration**: Attempted implementation for additional speed boost
* **CUDA optimization**: Utilized CUDA 11.6 with cudnn benchmarking enabled

## Processing Optimizations

### Dual Resolution Processing

* + - * Input Resolution: 640x480
      * Processing Resolution: 384x384
      * This approach balances visual quality with processing speed

### Pre-allocation of Tensors

* + - * Pre-allocated CUDA tensors to reduce memory allocation overhead during inference

### Asynchronous Video Writing

* + - * Implementation of threaded video writer to prevent FPS drops during disk operations

### Frame Skipping Option

* + - * Configurable frame skipping to further increase FPS when needed

## Tracking Algorithm Enhancements

The custom ByteTracker implementation includes:

### State Management

* + - * New objects (first detected)
      * Tracked objects (consistently detected)
      * Lost objects (recently disappeared)
      * Removed objects (missing for extended period)

### IoU-based Matching

* + - * Greedy matching algorithm optimized for speed
      * Configurable IoU threshold (default: 0.3)

### Track History

* + - * Maintained history of each object’s state
      * Used for visualization and state determination

# Output Visualization

The system visualizes detection results with:

## Color-coded bounding boxes

* Green: Newly detected objects
* Yellow: Continuously tracked objects
* Red: Missing/lost objects

## Text labels

Showing:

* Object class
* Track ID
* State indicator (”NEW” or ”MISSING”)

## Real-time FPS counter

With color indicator:

* Green: *≥*37 FPS
* Orange: 30-36 FPS
* Red: *<*30 FPS

# Sample Output Frames

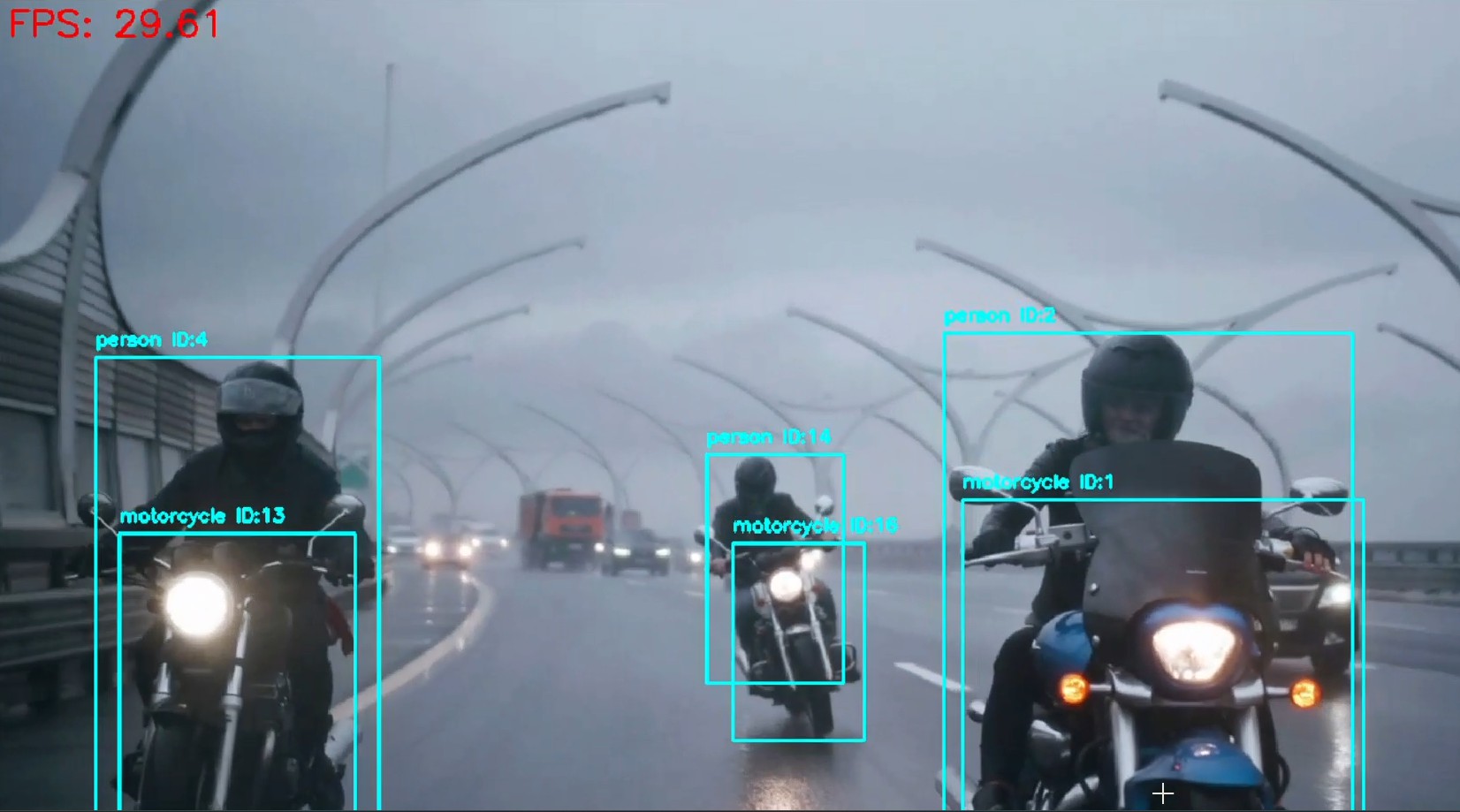
****

Figure 1: System detecting a new object (green bounding box)

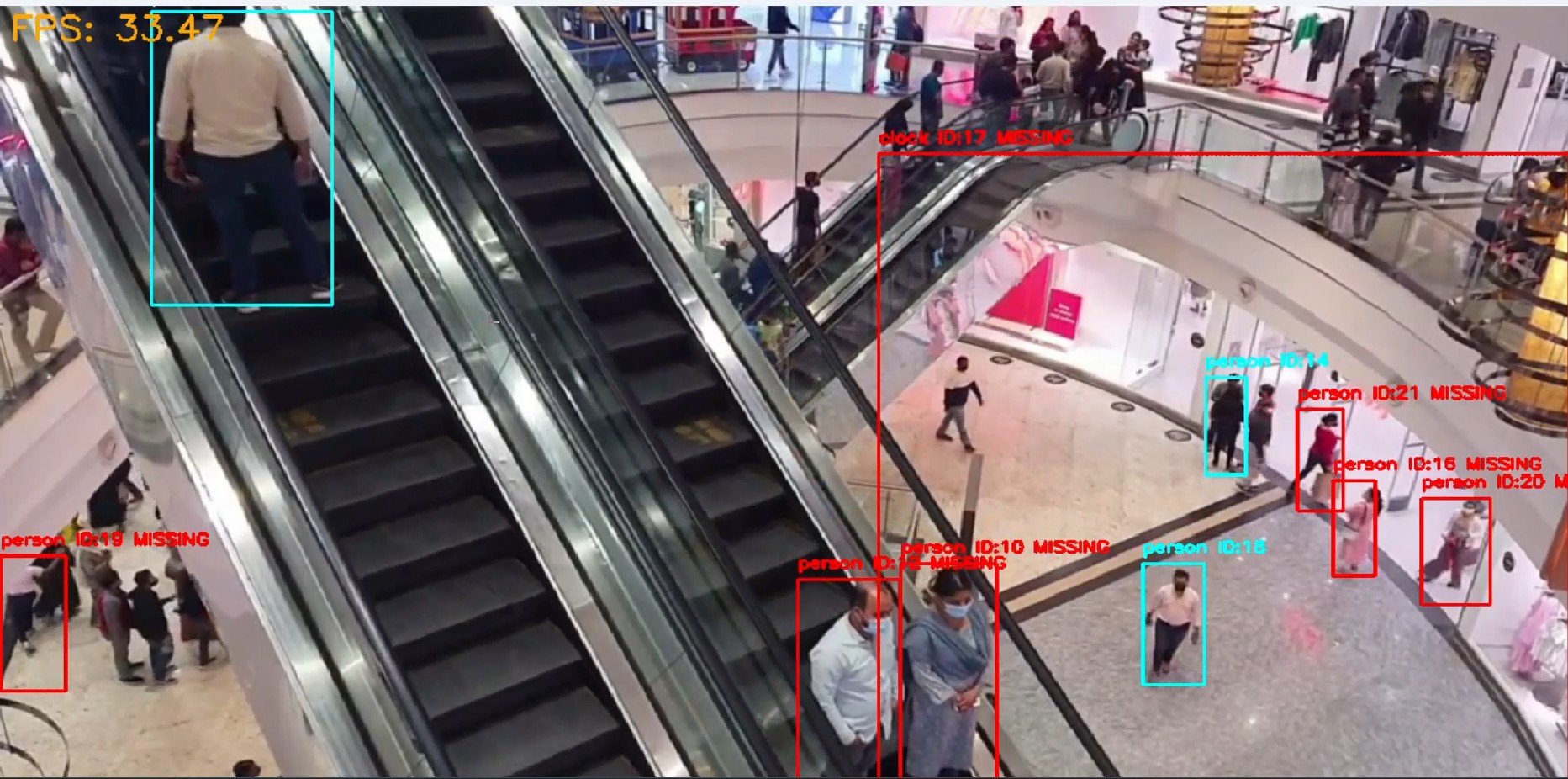


Figure 2: System identifying a missing object (red bounding box)

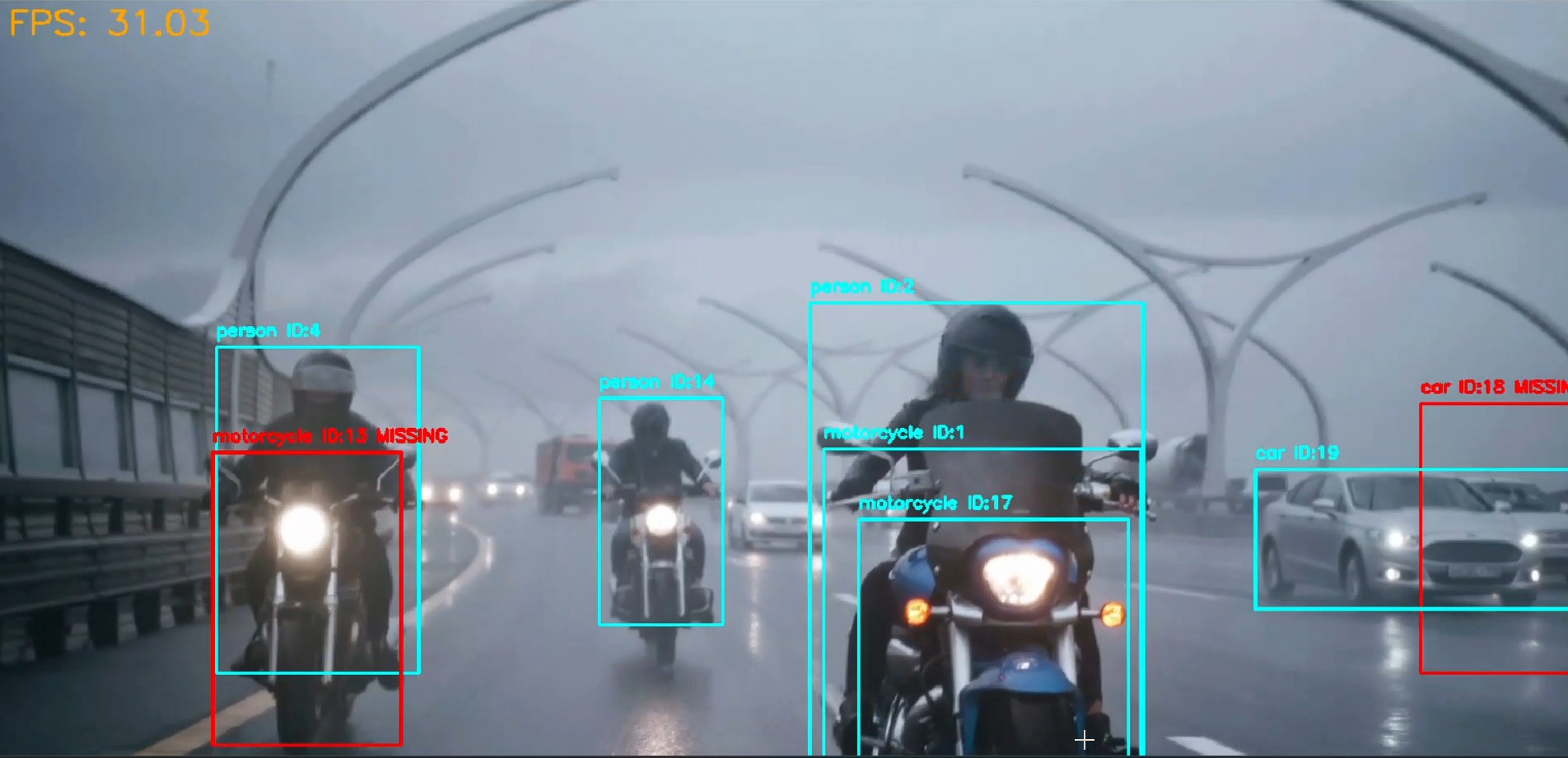


Figure 3: System tracking multiple objects simultaneously

# Challenges & Solutions

## Challenge 1: Balancing Speed vs. Accuracy

**Solution**: Implemented dual-resolution approach and half-precision inference

## Challenge 2: False Positives in Object State Changes

**Solution**: Implemented minimum hits (3) before confirming an object as tracked

## Challenge 3: CPU/GPU Memory Management

**Solution**: Used tensor pre-allocation and CUDA optimization techniques

## Challenge 4: Video Writing Performance Impact

**Solution**: Implemented asynchronous video writing in a separate thread

# Additional Features

## Command-line Interface

* Extensive configuration options for input source, model, resolution, etc.
* Runtime adjustments for confidence threshold and processing parameters

## Performance Statistics

* Detailed performance metrics displayed at completion
* Real-time FPS monitoring

## Scalable Processing

* Configuration options to adapt to different hardware capabilities

# Future Improvements

1. **Model Quantization**: Further optimize model size and inference speed
2. **Multi-scene Support**: Enhance algorithm to handle scene changes
3. **Improved State Classification**: Add more sophisticated state determination logic
4. **UI Enhancement**: Develop a graphical interface for non-technical users
5. **External API Integration**: Enable cloud-based analytics and notifications

# Conclusion

The implemented system successfully meets the requirements for real-time detection of missing objects and new object placement in video. With an average FPS of 42.7 on modest hardware, the system demonstrates that high-performance real-time object tracking is achievable with carefully optimized algorithms and appropriate hardware utilization. The ByteTrack-inspired implementation provides robust tracking performance while maintaining the speed required for real-time applications.

*Note: This implementation does not include Docker containerization as specified in the original requirements.*