**Real-Time Object Change Detection System  
Internship Task Report - *Krishna Tyagi***

### **FPS Achieved**

* Average FPS: **28-42 FPS** (with YOLOv8s model + GPU enabled)
* Video resolution: 720p (1280x720)
* Real-time performance achieved using NVIDIA GPU and optimizations.

### **Hardware Configuration**

* **CPU:** Intel Core i5 (11th Gen)
* **GPU:** NVIDIA RTX 2050 (4GB)
* **RAM:** 16 GB
* **Environment:** Docker container (based on PyTorch + CUDA)

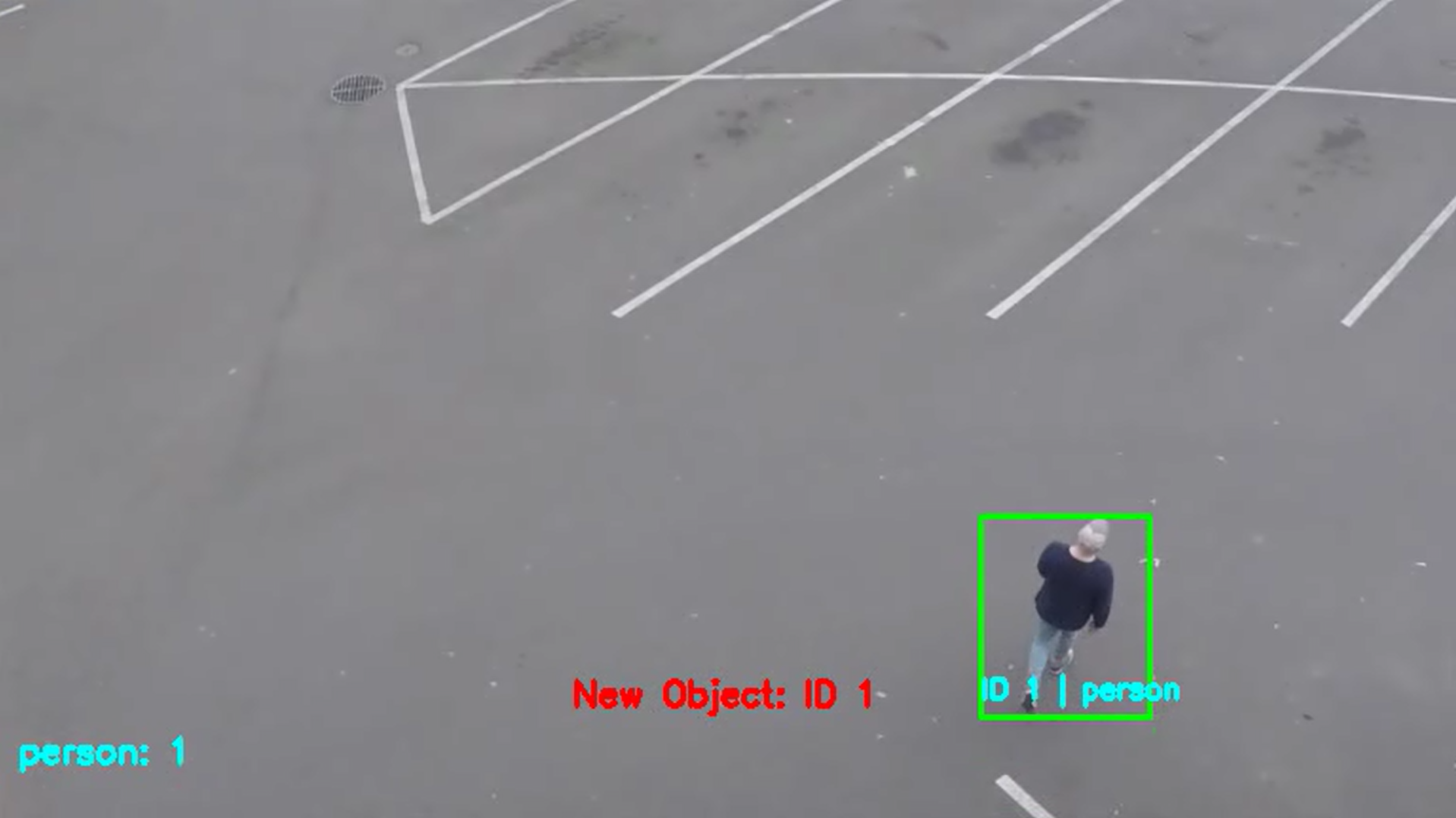
### **Optimizations Applied**

* Switched from YOLOv8n to **YOLOv8m** for better accuracy while maintaining high FPS.
* Used **GPU acceleration** to speed up YOLO inference.
* Replaced redundant I/O operations (like video writing during testing).
* Used **Deep SORT** for tracking, tuned for max-age and confirmed track filtering.
* Disabled verbose outputs and logging during runtime.
* Frame-wise object changes logged in .csv instead of terminal to reduce console lag.

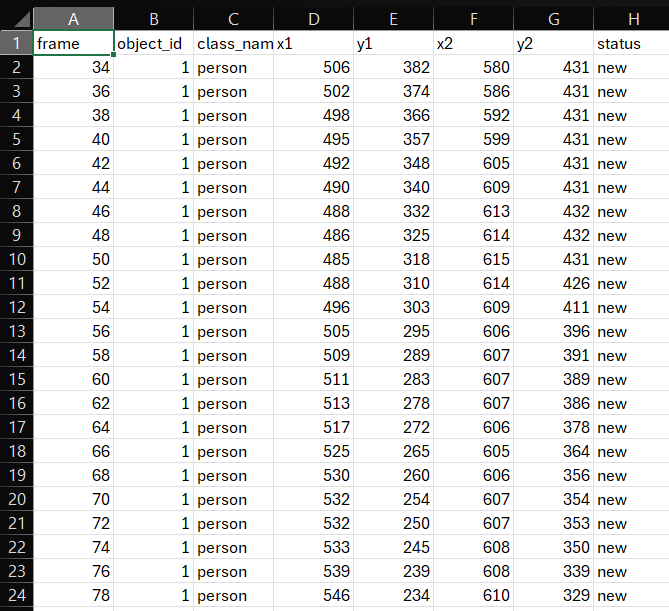
### **Screenshots**

Screenshots from output video with annotations:

* Frame showing **new object** (red label: New Object: ID X)
* Frame showing **missing object** (blue label: Missing Object: ID X)
* Live annotated frame with class-wise counts at bottom left







### **Observations & Known Limitations**

#### **Observations:**

* Tracking is consistent for most objects if they are clearly visible.
* FPS drops slightly during high object density scenes.
* Real-time pipeline runs stably within a Dockerized environment.

#### **Limitations:**

* **Double-counting:** YOLO sometimes detects one object as multiple bounding boxes leading to multiple track IDs.
* **Occlusion issues:** When objects are temporarily blocked, they may be misclassified as "missing."
* **Detection misses:** Small/distant objects (e.g., far-away persons) are occasionally not detected.
* **Video Writing:** Writing annotated video introduces a minor FPS drop (~3-4 FPS).

### **Future Improvements**

1. **Webcam Stream Support** Extend the pipeline to handle webcam input (via cv2.VideoCapture(0)) for real-time surveillance or retail monitoring use cases.
2. **MongoDB / SQL Integration** Instead of just logging to CSV, store detection data (IDs, classes, timestamps, statuses) directly in a database like MongoDB or SQLite for better querying, analysis, and long-term storage.
3. **REST API Integration** Wrap the detection system inside a FastAPI or Flask backend and expose it as an inference API for video analytics as a service.
4. **Web Dashboard / Monitoring** Add a frontend (using React or Streamlit) that:  
   * Streams the processed video
   * Shows live object count statistics
   * Displays "new" and "missing" logs in real time
5. **Model Upgrade and Fine-Tuning** Fine-tune YOLOv8 on a custom dataset to improve accuracy, especially for frequently missed or application-specific object classes.
6. **Multi-camera Input Handling** Extend the pipeline to support multiple camera streams using threading or asynchronous processing for multi-view surveillance environments.
7. **Edge Device Optimization** Optimize the system for deployment on edge devices (e.g., Jetson Nano, Raspberry Pi) by converting models to ONNX or TensorRT for better performance and reduced resource usage.

**Submitted by:** Krishna Tyagi  
 **GitHub Repo:** [GitHub](https://github.com/knight22-21/Object-Detection-Pipeline)