## Technical Report

a. Which object detection model did you use and why?  
The project uses the YOLOv8 (You Only Look Once version 8) object detection model. YOLOv8 is chosen for its state-of-the-art accuracy combined with real-time inference speed, making it ideal for video processing and real-time applications. It balances performance and speed, supports transfer learning, and has a mature PyTorch implementation facilitating easy integration.

b. What was the average FPS you achieved?  
The average frames per second (FPS) achieved during video processing ranges from around 8 to 10 FPS depending on the hardware used. On a standard laptop without GPU acceleration, the system maintains practical real-time processing speed sufficient for many applications. The exact FPS is logged dynamically and reported in the analytics.

c. What were the main technical challenges?

1. Integrating object detection with DeepSORT for robust multi-object tracking ensuring consistent IDs across frames.
2. Handling IOU-based assignment of detection boxes to trackers efficiently to assign class labels correctly.
3. Managing video input/output formats and ensuring compatibility, especially for displaying or saving output videos within constrained environments like Google Colab.
4. Implementing real-time performance constraints while maintaining detection accuracy.
5. Designing modular, clean, and extensible code with suitable error handling and configuration options.

d. How would you optimize this for production use?

1. Deploy optimized YOLOv8 variants or convert to efficient formats like TensorRT, ONNX, or OpenVINO for faster inference on production hardware.
2. Use batch processing and asynchronous pipelines to maximize GPU utilization and throughput.
3. Implement adaptive confidence thresholds and dynamic frame skipping for performance tuning under different operational scenarios.
4. Include hardware acceleration support and conduct profiling on target devices.
5. Add a user-friendly interface with live controls for parameters and error logging.
6. Ensure robust handling of corner cases (lost tracks, occlusion) and integrate post-processing analytics exposed via APIs for monitoring.