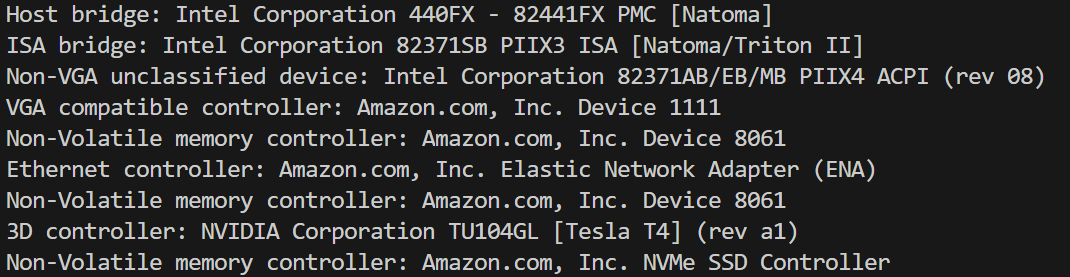
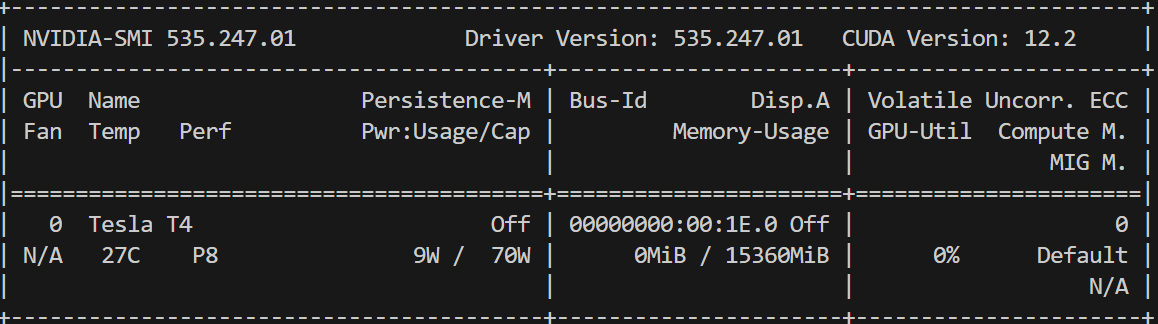
**Samajh.ai Assignment Report**

**Link to github repository:**[**https://github.com/Vaibtan/Realtime-Video-Object-Tracking/tree/main**](https://github.com/Vaibtan/Realtime-Video-Object-Tracking/tree/main)

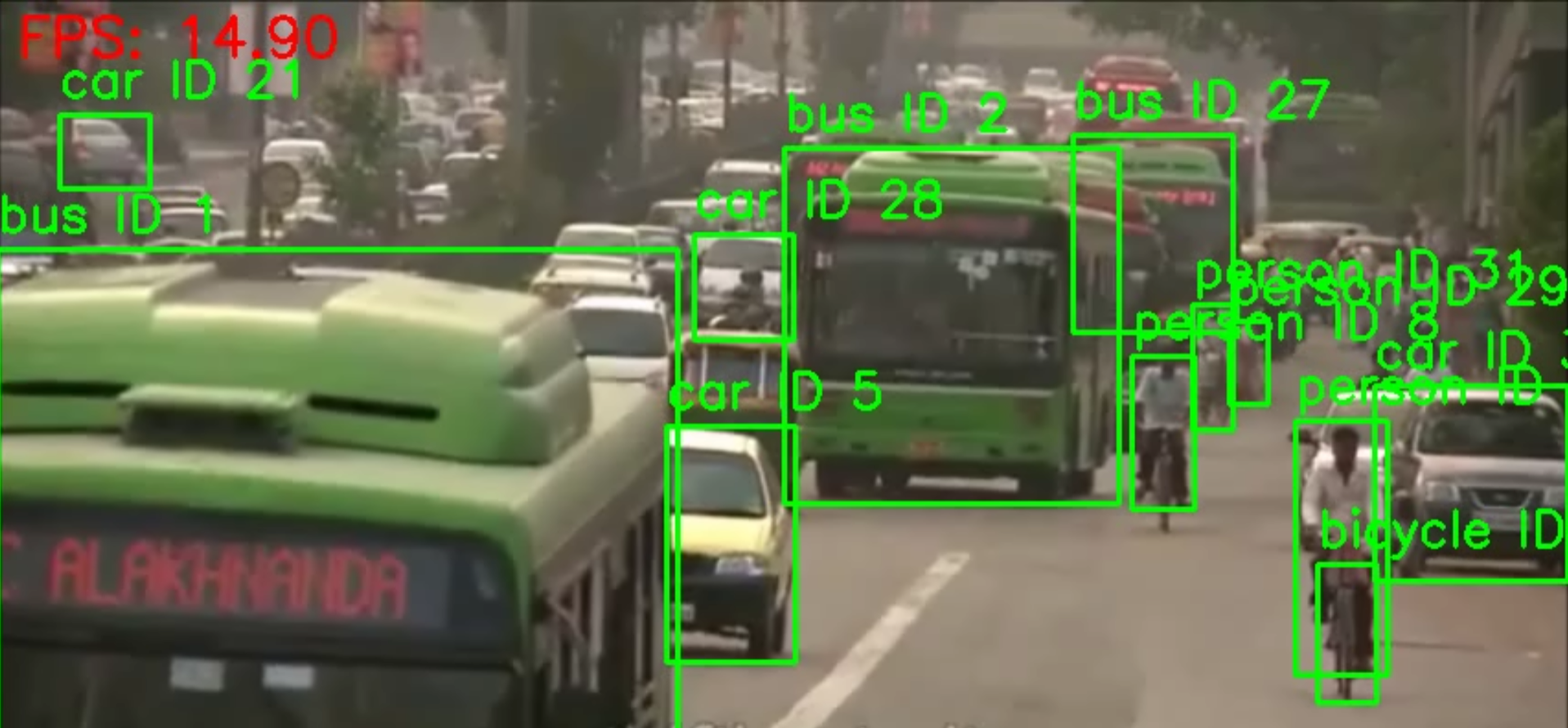
**Link to the sample annotated video:** [**https://drive.google.com/file/d/1WWNhO6Fk0tkgodaqzDfezfPWkv9C7xJp/view?usp=sharing**](https://drive.google.com/file/d/1WWNhO6Fk0tkgodaqzDfezfPWkv9C7xJp/view?usp=sharing)

**System Configuration:  
Used a Lightning.ai studio for this task.**

****

****

**A sample output frame is shown below where a real time counter for FPS is given below.**

****

**Here I am using DeepSORT multi object tracking algorithm which assigns each detected object a persistent tracking ID by combining motion predictions (via a Kalman filter) with appearance embeddings. Here's how it works end-to-end in the pipeline:**

1. **Detection & Embedding**
   * **YOLOv8 produces bounding boxes each frame.**
   * **DeepSORT crops each detection and runs it through a small Re-ID network to produce a feature vector encoding the object’s appearance.**
2. **ID Assignment**
   * **For every new frame, DeepSORT predicts where each existing track should be (motion model), then computes the cosine distance between each track’s appearance vector and each new detection’s vector.**
   * **It also computes IoU between predicted and detected boxes.**
   * **Using a Hungarian algorithm that fuses motion, appearance, and IoU costs, it matches detections to existing track\_ids when similarity is high.**
   * **Unmatched detections are spawned as new tracks with fresh ​track\_id​ values.**
   * **Tracks that go unmatched enter a countdown before being dropped.**

**Optimizations done for optimal inference and minimizing latency:**

* **Set the YOLOv8 model to .eval() and call .fuse() in order to merge the convolutional and batch normalization layers, reducing operator overhead.**
* **Converted the model to FP16 via .half() and wrapped inference in torch.cuda.amp.autocast(), cutting GPU compute time by ~2×.**
* **Disabled autograd (torch.no\_grad()) to skip gradient bookkeeping and free up memory bandwidth.**
* **Batch-loaded all box coordinates, confidences, and classes from PyTorch tensors into NumPy in one shot, avoiding per-box Python calls.**
* **Pushed DeepSORT’s ReID embedder onto the GPU, so appearance features are computed in parallel with detection.**
* **Introduced a three-stage threaded pipeline (capture → process → write) with bounded queues to hide I / O and post-processing behind the GPU.**
* **Pre-allocate and reuse a single VideoWriter, building the output path once and skipping imshow() calls entirely to eliminate GUI stalls.**
* **Tuned DeepSORT’s max\_iou\_distance and missing-frame threshold so tracks are pruned or born quickly, reducing tracker bookkeeping overhead.**

**How to use the pipeline:  
Run the following command:  
python track.py --input path/to/video\_file.mp4 --output results --async   
  
Here the async flag is optional - in case it’s not enabled, the multithreaded pipeline will not be initialized and only the synchronous pipeline will run.**