Intel Unnati Industrial Training Programme – 2024

Project report - Team Techsmashers

Problem statement title: PS-13 Vehicle Movement Analysis and Insight Generation in a College Campus using Edge AI

Team Mentor : Archana Vaidheeswaran,

archana.vaidheeswaran@unnatiindustrialtraining2024.com

Internal Mentor : Dr T V RAJINI KANTH, Professor & Head,

Department of CSE-AI&ML

rajinikanthtv@sreenidhi.edu.in, 9849414375

Team members:

1. Mohammed Moiz Pasha - 21311A6601@sreenidhi.edu.in, CSE-AI&ML

2. Gurram Bharath - 21311A6603@sreenidhi.edu.in, CSE-AI&ML

3. Ramachandruni Sumedha - 21311A6662@sreenidhi.edu.in, CSE-AI&ML

4. Mendi Pavan Kumar - 21311A6639@sreenidhi.edu.in, CSE-AI&ML

5. Gugulothu Vijaya - 21311A6643@sreenidhi.edu.in, CSE-AI&ML

Institute Name: Department of CSE-AI&ML, Sreenidhi Institute of Science and Technology, Yamnampet, Ghatkesar, Hyderabad - 501301

Problem Description:

Managing vehicle movement and parking in a college campus can be a challenging task. An intelligent system that can analyse vehicle movement, monitor parking occupancy, and match vehicles to an approved database can significantly improve campus security and management. Our task is to develop an Edge AI-based solution that can analyse vehicle movement using data from cameras capturing vehicle photos and license plates. The solution should be capable of processing image data in real-time and provide insights on:

- Vehicle Movement Patterns: The solution should be able to analyse the frequency and timing of vehicle movement in and out of the campus. It should be able to identify peak times and patterns.
- Parking Occupancy: The solution should be able to monitor the occupancy of parking lots in real time. It should be able to identify which parking lots are frequently occupied and at what times.
- Vehicle Matching: The solution should be able to match captured vehicle images and license plates to an approved vehicle database. It should be able to identify unauthorized vehicles.

Dataset description:

Link to datasets used: https://drive.google.com/drive/folders/1YbZ3_s_dBmEuzfxu8qAOClS74C3n5T46

Sources for all datasets have been listed in the “Data collection” subsection below.

For the various submodules of this project, a total of 4 datasets have been used for training, validation and for testing/demonstration.

1. License Plate Recognition dataset:

- A collection of 22,068 images of various vehicles with license plate bounding boxes, annotated in the YOLO format.
- Labels: 0 – license_plate
- Image size: 640x640, 3 channels (scaled using preprocessing methods discussed below)

2. License Plate OCR dataset

- 26,022 images of license plates, saved with their file names as the OCR text of the number plate, followed by an integer indicating license plate size (1 for single-line, 2 for two-line plates), e.g. "AP13AN4322_1"
- 967 real plates and 25,055 generated plates, stored without augmentation (augmentation applied during training for the LPRNet model)
- Image size: 94x24, 3 channels.

3. Parking spaces dataset

- 35,191 images of parking lots, each containing multiple parking spaces in both occupied and unoccupied states, annotated in the YOLO format
- Labels: 0 – empty, 1 - occupied
- Image size: 640x640, 3 channels

4. Vehicle movement dataset:

- 1000 rows of randomly generated vehicle movement data used for testing insight generation, stored in the CSV format.
- Fields: Date, Day, Vehicle Name, License Plate number, In time, Out Time, Authorization status

Methodology:

The methodology employed to address the problem statement and develop the project is as follows:

1. Data Collection
2. Data Preprocessing
3. Model training
4. Model optimization for Edge device inference
5. Model deployment for data collection
6. Insight generation using collected data

Additional details and results for the methodology are listed in the appendix.

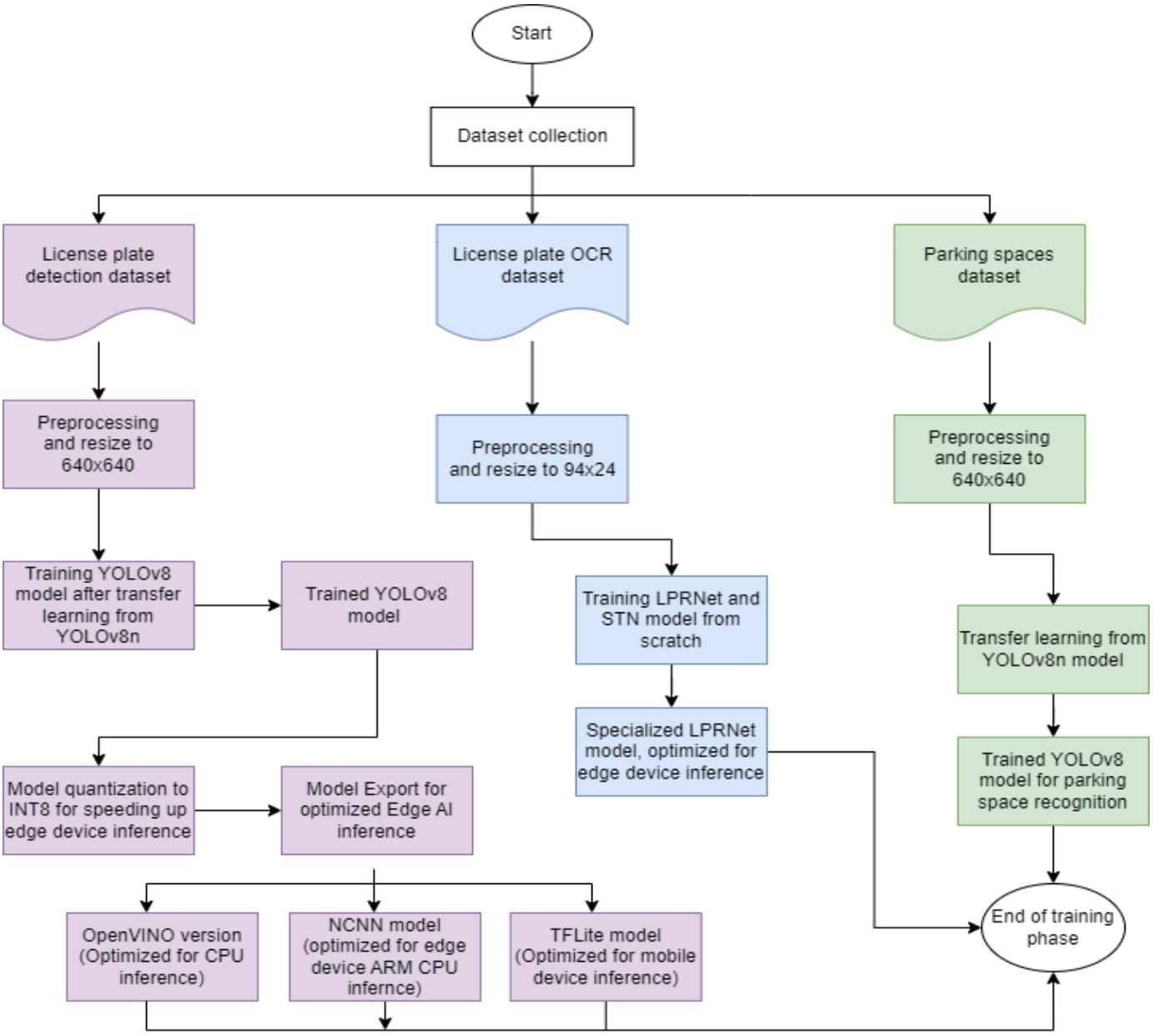


Fig 1: Model training flowchart

A. Data collection

The datasets used in the project contain both real and generated images, to increase model robustness. The data was collected by compiling existing open-source datasets, along with data collected by us in real life. The sources for the datasets have been listed below:

1. License Plate Recognition dataset:

- https://universe.roboflow.com/william-i5nyg/licens-plate-detection-vn5sg

  License: Public Domain

- https://universe.roboflow.com/atif-siddiqui/indian-number-plates-nrsft

  License: CC BY 4.0
- https://universe.roboflow.com/datacluster-labs-agryi/indian-number-plates-9oobq
  License: CC BY 4.0
- https://universe.roboflow.com/license-plates-vcmxh/indian-pm6w4

License: CC BY 4.0
- https://universe.roboflow.com/indiannumberplatesdetection/indian-car-bike-number-plate
  License: CC BY 4.0
- https://universe.roboflow.com/augmented-startups/vehicle-registration-plates-trudk
  License: CC BY 4.0

The remainder of the images were generated by **augmentation**, detailed in the preprocessing section below.

2. License plate OCR dataset

- https://www.kaggle.com/datasets/saisirishan/indian-vehicle-dataset
  License: Unknown

The remainder of the images, which make up about **96% of the total dataset size**, were **generated through code using PIL**. The generated images contain single-line and two-line number plates, with randomly placed separators and colors (yellow, black, green, white), to represent diversity of Indian number plates. A small subset (~1%) was collected from real-life images and labelled by hand.

3. Parking spaces dataset

- https://universe.roboflow.com/car-parking-yolov8/ml-8n5gd
  License: CC BY 4.0
- https://universe.roboflow.com/almutasem-bellah-enad/real-time-car-parking
  License: CC BY 4.0
- https://universe.roboflow.com/aiml-the-lebron-project/parking-finder
  License: CC BY 4.0
- https://universe.roboflow.com/sep-gruppe-pb6fz/to_merge
  License: CC BY 4.0
- https://universe.roboflow.com/imad-eze/carsparking
  License: CC BY 4.0

B. Dataset preprocessing

The dataset preprocessing techniques applied on the raw images collected for training are:

1. Missing class labelling: The images containing missing class labels (<100 images) were manually labelled using the Roboflow annotation tool.

2. Resizing: The datasets for parking spaces and license plate detection were resized to 640x640, while the dataset for license plate OCR was resized to 94x24, using OpenCV

3. Augmentation: To make the models robust, the following augmentation steps were applied to all training datasets:

i. Jitter: Varying the brightness, hue and saturation of the image

ii. Rotate: Rotating the image by a random angle

iii. Perspective: Warping the alignment of the image

iv. Crop: Cropping margins of the image by random amounts

v. HSV space variation: Changing HSV color values of the image randomly, and applying Gaussian noise

For the License Plate OCR model, two additional steps were applied for preprocessing:

4. Normalization: Normalizing the pixel values of the image between [0, 1], to enable faster convergence.

5. Batch dimension: Adding a batch dimension to the image, to make the images compatible with the STN and LPRNet models.

6. Removal of outliers: The dataset with timestamps for insight generation was plotted and outliers in the dataset removed, in order to generate accurate insights

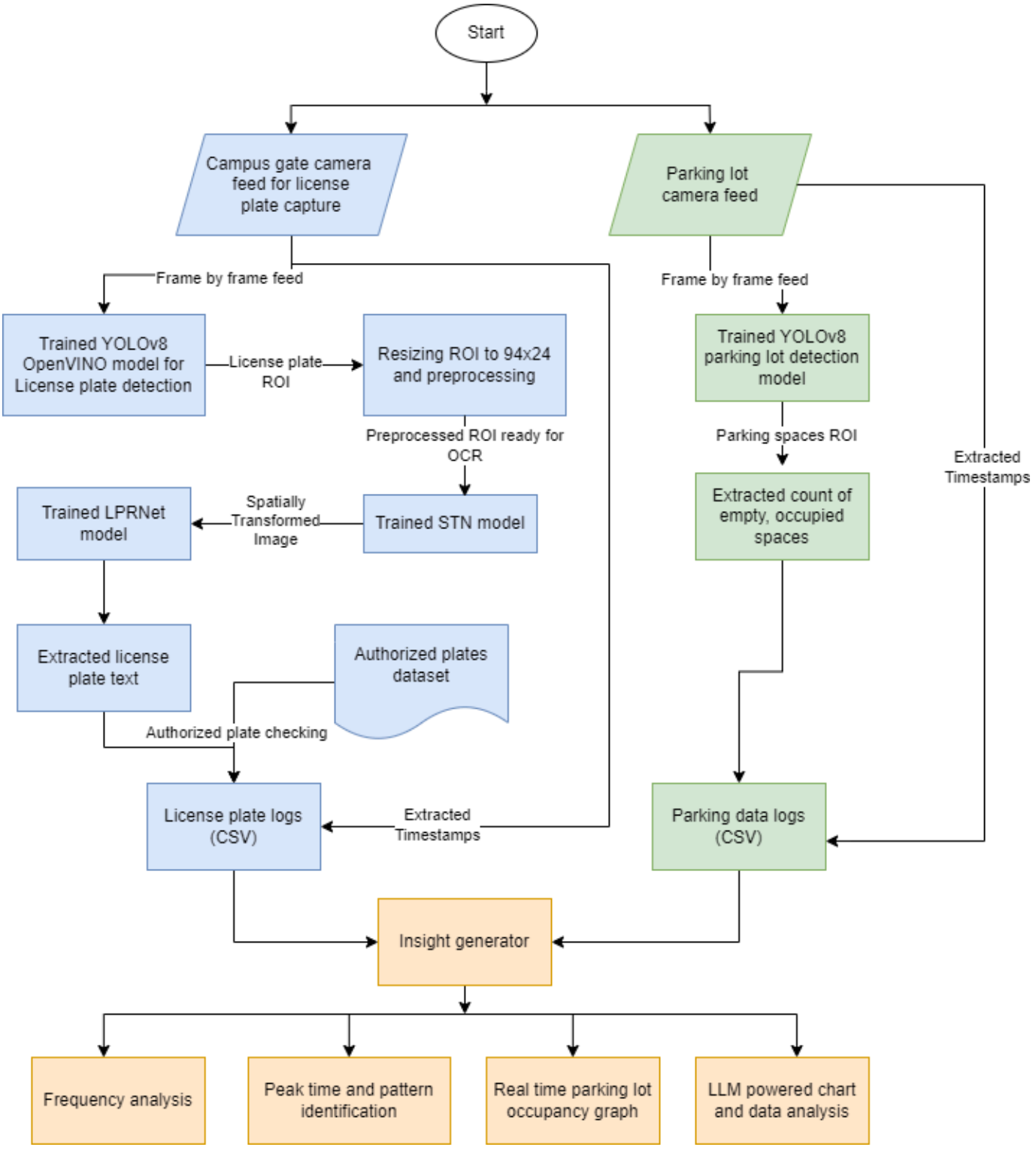


Fig 2: Inference and Insight generation flowchart

C. Model Training:

(Metrics of model training present in the Results section)

1. License Plate OCR model:

The model we chose for License plate text OCR was **LPRNet,** developed by the Intel IOTG Computer vision group [2] (Link to research paper in references). We chose this model specifically for the following reasons:

i. The trained model only required **0.34 GFLops** for a single forward pass, making it fast and extremely efficient for processing on edge devices such as the Raspberry Pi and mobile devices

ii. Unlike other OCR models such as Pytesseract and EasyOCR, the LPRNet model runs completely **on device** and is made for the specific purpose of recognizing license plate text, hence reducing model weight sizes and processing overhead. Preliminary testing also showed it to be more accurate compared to baseline EasyOCR and Pytesseract models.

To improve the accuracy of the model even further, we decided to implement a **Spatial Transformer Network (STN)** into the OCR pipeline. The STN model reorients the input license plate image spatially, hence enabling high accuracy character recognition of pictures of license plates from different angles as well.

Fig 3. Inference process for the OCR models

The models were trained for 1200 epochs on a T4 GPU on Google Colab, and are saved as a PyTorch model checkpoint (.ckpt) format.

2. License Plate Detection model:

For license plate detection from images of cars, we performed transfer learning on a YOLOv8 nano model. The model was trained for 120 epochs, and was initially saved as a PyTorch model format (.pt).

3. Parking Spaces Detection model:

For detecting the parking spaces, similar to the license plate detection model, we performed transfer learning on the YOLOv8n model. The model detects empty spaces and occupied spaces in a given image of a parking lot. This model was trained for 200 epochs on a P100 GPU, and can detect the occupancy of multiple parking spaces simultaneously.

D. Model optimization for Edge device inference

In order to optimize inference on edge devices, we performed **INT8 quantization** on the License plate model, and then exported the model to the following formats (see appendix for further information):

1. Intel OpenVINO: OpenVINO is an open-source software toolkit for optimizing and deploying deep learning models. We found that the OpenVINO models performed best for CPU inference.

2. NCNN: NCNN is a high-performance neural network inference computing framework optimized for mobile platforms. We found this model to perform best on ARM based devices, such as the Raspberry PI

3. TFLite: TensorFlow Lite (TFLite) is a collection of tools to convert and optimize TensorFlow models to run on mobile and edge devices. The TFLite model is specifically optimized for mobile devices such as smartphones .

The LPRNet and STN models are already optimized for high inference speeds, due to the nature of their architecture. The parking spaces model was quantized and exported to OpenVINO to speed up inference (s.

E. Model deployment for data collection

The models are deployed in the form of a CLI and GUI based format. The License plate detection model, the LPRNet model and the STN model are combined into one file, which detects the license plate texts from the given input images. The GUI format of the model is deployed as a Streamlit app, which provides the capability of both video and image inference in a user-friendly format.

The output of these models is then used to log data, along with the timestamp, for generation of insights from the data captured. Sample videos and images are taken to emulate the operation of the project in real-time environments, and the data logged is saved as a CSV file using Pandas.

F. Insight generation using collected data

In order to generate insights from collected data, we first created a prototype of the insights engine by running it on a randomly generated dataset of a 1000 records (as described in the datasets section above). We then used the following in order to generate insights from the data:

1. Pandas: Pandas is a popular open source data analysis and manipulation tool. We used pandas in order to load the dataset from the CSV file, and perform **aggregation functions** such as **min, max and mean for frequency analysis, data transformations** for converting raw data into visualizable formats, and for storing and updating new data from the trained models.

2. Plotly express: Plotly Express is a high-level interface for the plotly module, which provides functions to visualize a variety of types of data. We used Plotly to visualize the dataset, and present it to the user in the form of **bar charts, pie charts and line graphs.**

3. PyPlot: The PyPlot submodule of the Matplotlib module is a popular data visualization tool. We used pyplot for **visualizing aggregated data**, which changes based on user input selections

4. Streamlit: In order to create a **GUI for the project**, we used the Streamlit library, which is a framework for building interactive data apps.

5. Gemini: Gemini is a multimodal, generalized chat model developed by Google. We used it to give the user the ability to get answers to questions regarding the data used.

Features of Insight generation:

1. Displays frequency of vehicles coming in and leaving the campus, grouped by days, weeks and months.

2. Identifies patterns in vehicle frequencies, and displays them in the form of line graph and stacked bar chart visualizations

3. Shows the count of authorized and unauthorized vehicles entering the campus, and shows the change compared to the previous day.

4. Displays the parking occupancy over the duration of the day, i.e. count of occupied and unoccupied parking spaces, grouped by parking lot.

5. Provides interactive chat mode, powered by Gemini, to help users ask questions regarding the data in natural language.

Results:

1. Model training results:

License plate recognition model:

The best iteration of the license plate recognition model had a **precision(B) score of 0.9767, recall(B) score of 0.95942 and mAP50 (B) score of 0.98568**

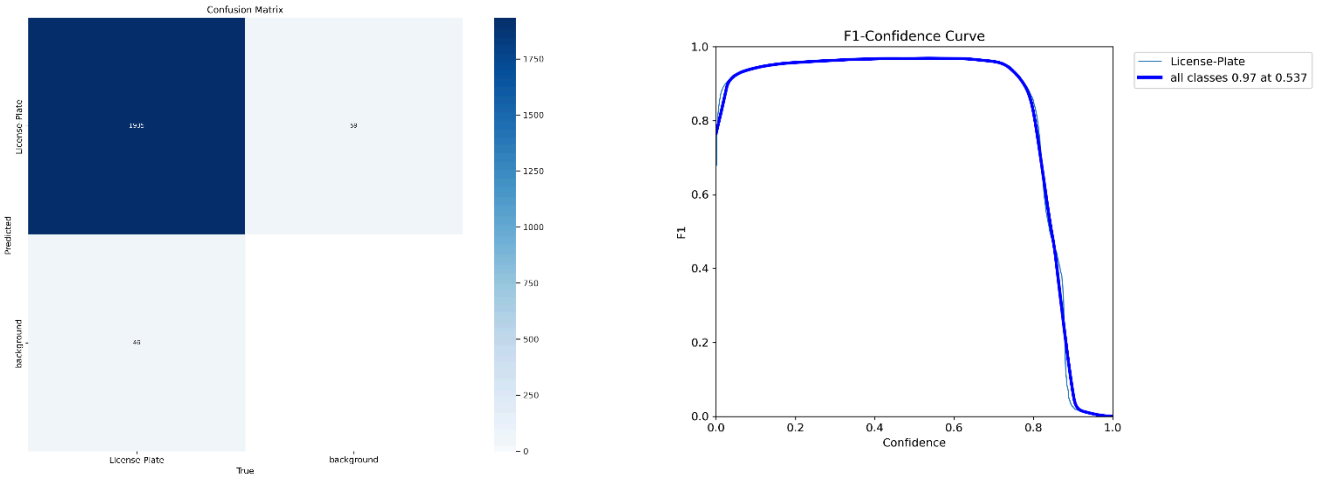


Fig 4. Confusion matrix and F1-Confidence Curve for LPR model

2. LPRNet + STN model

The best iteration of the model gave an accuracy score of 0.88, measured using the CTC loss function. The STN model is trained parallelly with the LPRNet model, hence the model's accuracy is directly proportional to the LPRNet model and cannot be measured in the traditional sense.

3. Parking Space detection model:

The best iteration of the license plate recognition model had a **precision(B) score of 0.98861, recall(B) score of 0.98787 and mAP50 (B) score of 0.9924**

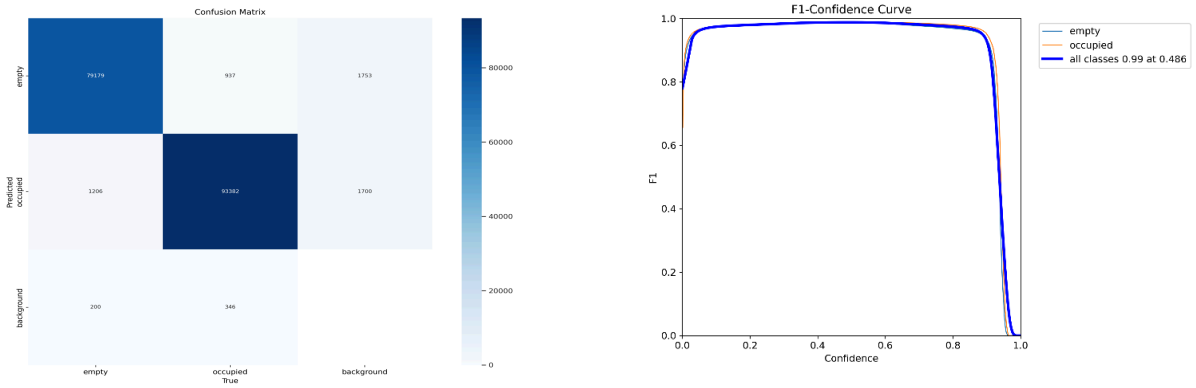
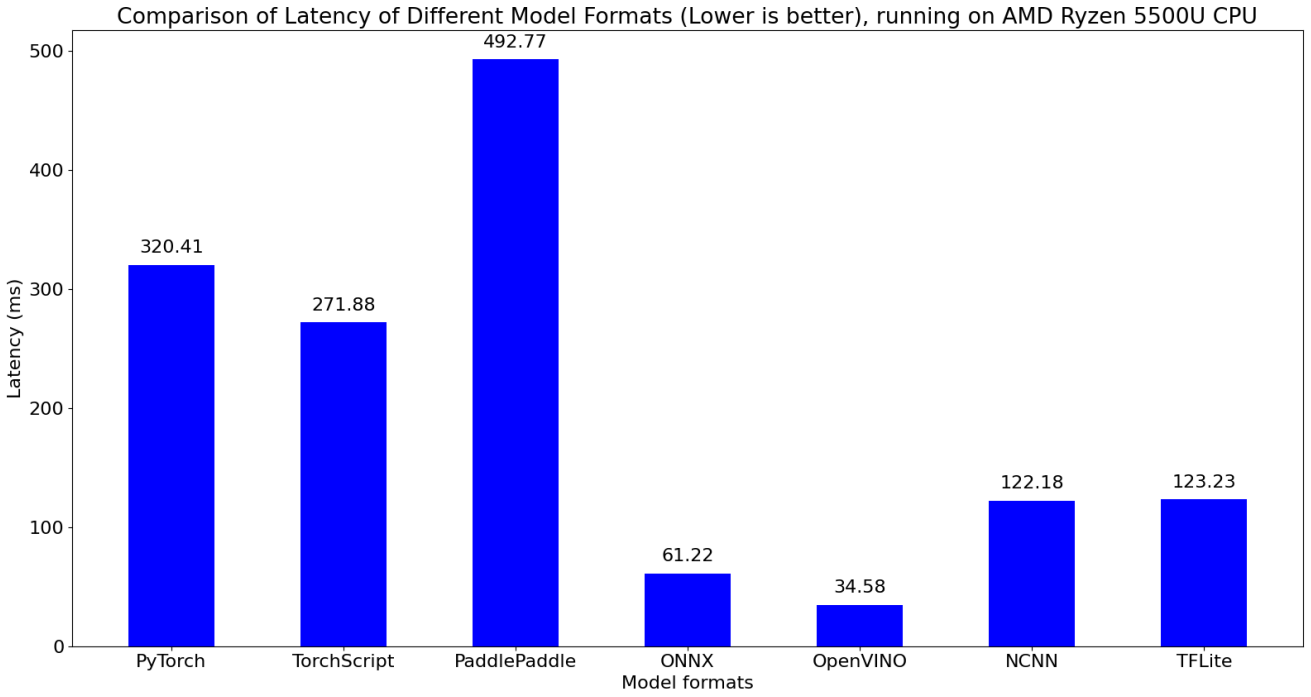


Fig 5. Confusion matrix and F1-Confidence Curve for parking space detection model

2. Model inference results:

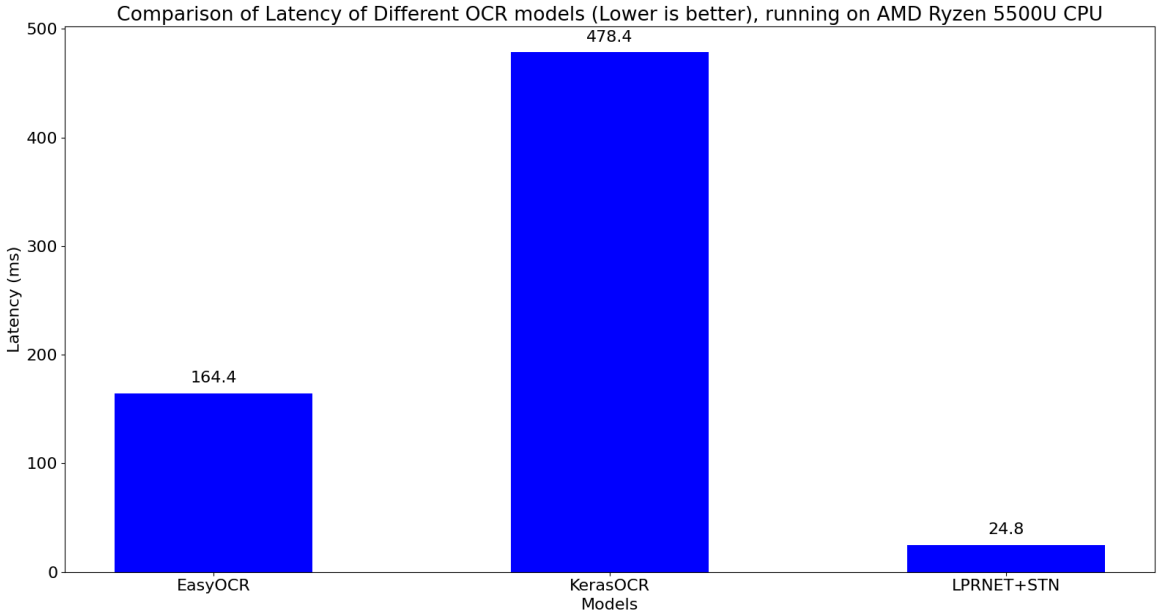
The following bar graph shows the model inference results, depicted by latency in milliseconds, after exporting to the various formats for fast edge device inference:



As per our testing, the **Intel OpenVINO** format provided the best latency on CPU inference, achieving **near real-time operation**, while the **NCNN model** provided the best performance on **edge devices**.

LPRNet model inference:

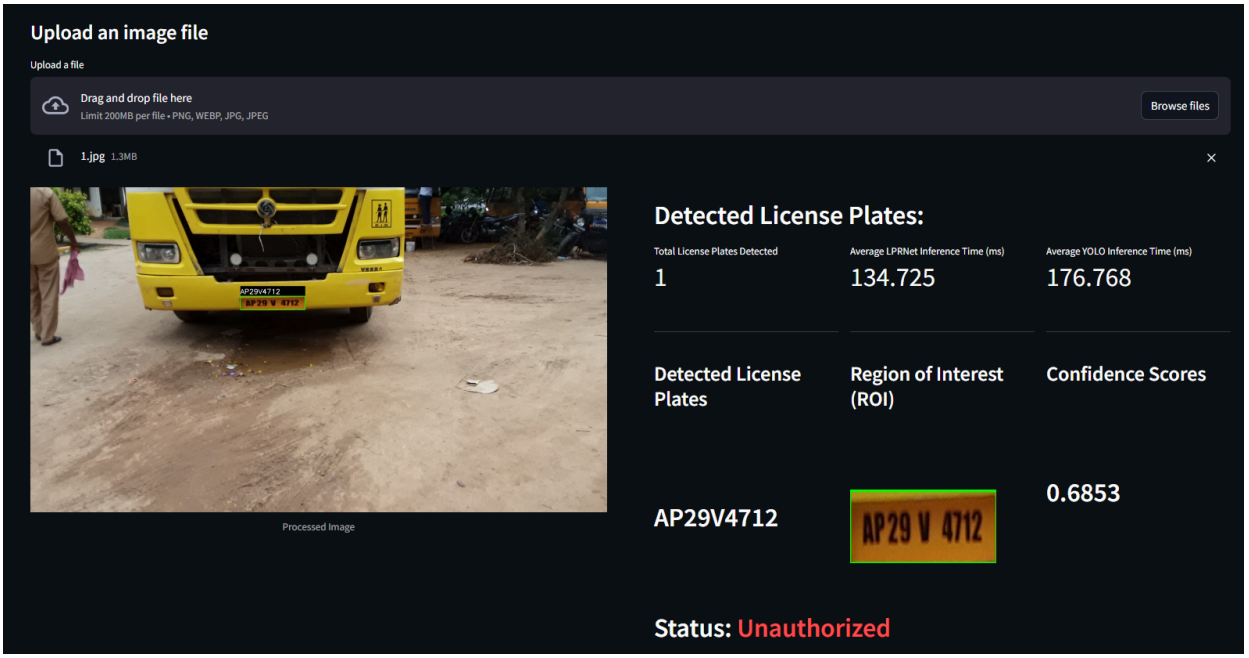
The following figure shows the inference latency in milliseconds, for three of the models we tested: EasyOCR, KerasOCR and our STN+LPRNet model.

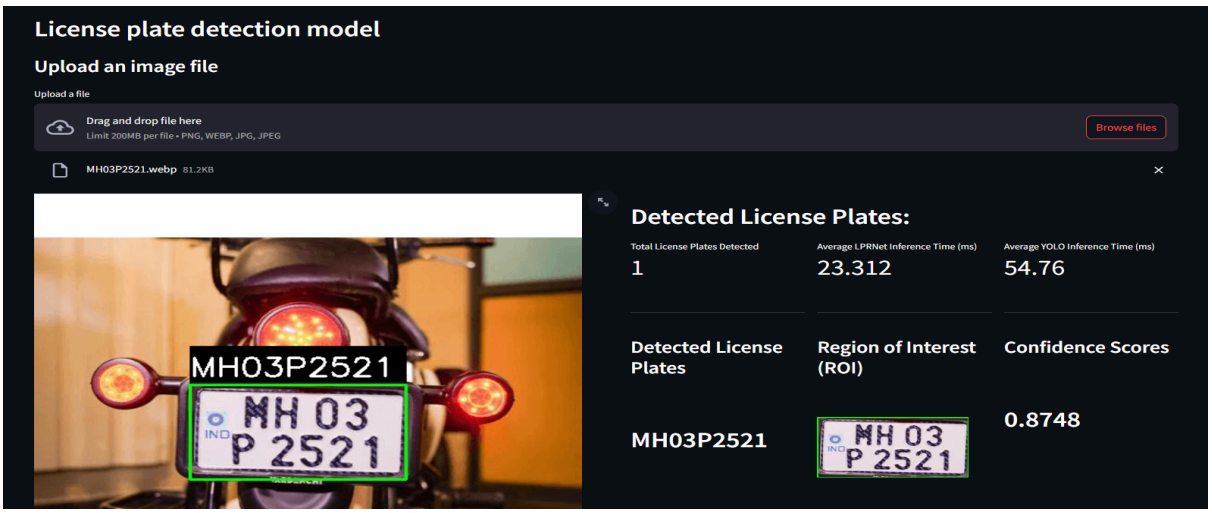


Our model achieved a latency of just **24.8 milliseconds** on a Ryzen 5 CPU, making it far superior to other generalized OCR models and highly optimized for Edge inference.

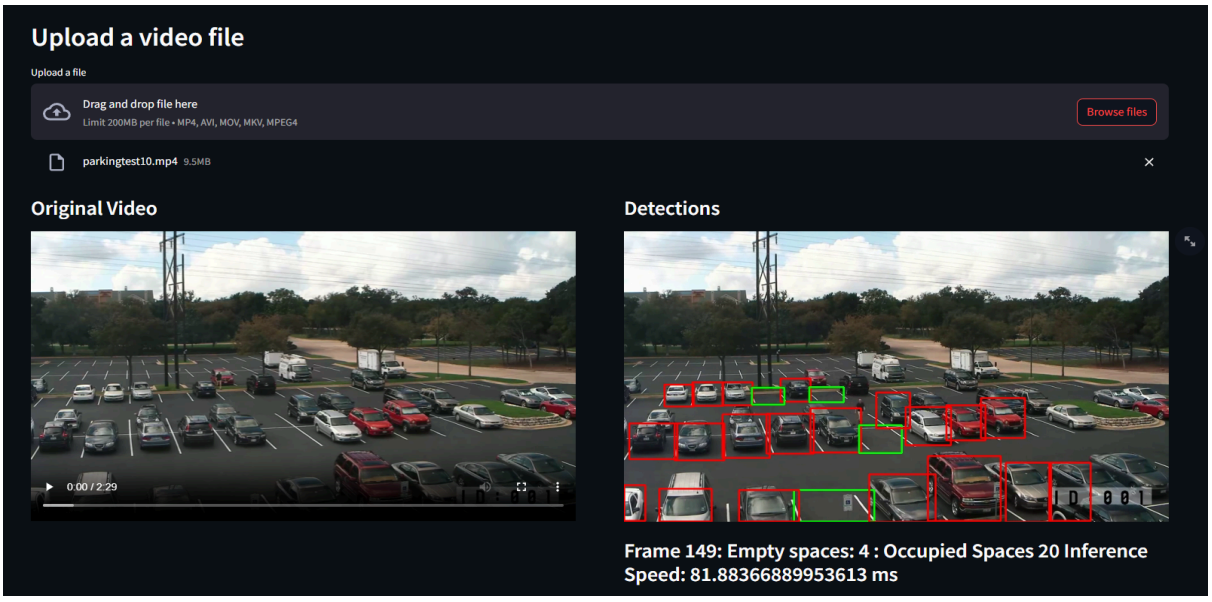
We also benchmarked our models on edge devices, the results of which are in Appendix-B

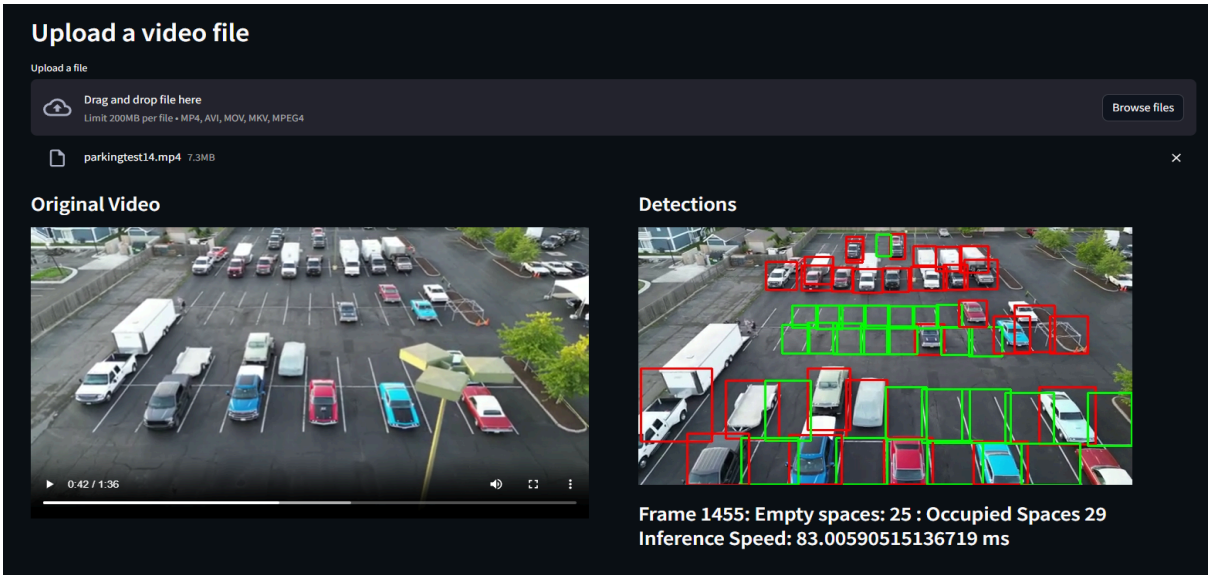
The following are screenshots of the GUI for our detection models, using the YOLOv8 License detector model + STN + LPRNet OCR on both real-life, and stock images from our college campus:





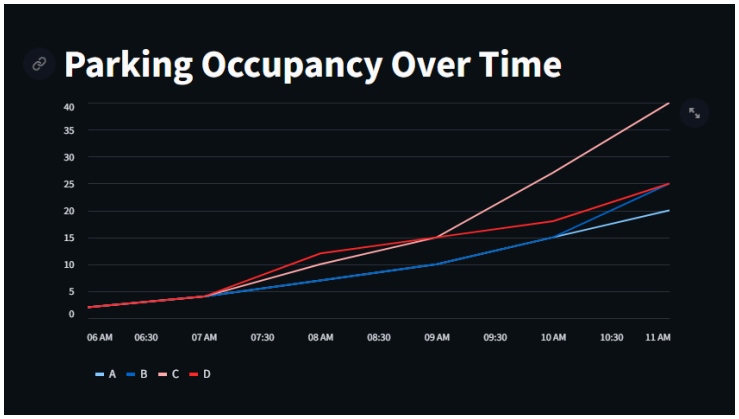
Parking spaces inference model (inferring from external security camera video file):

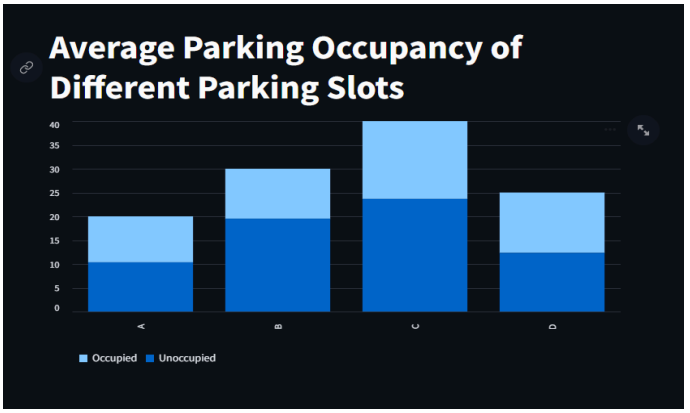


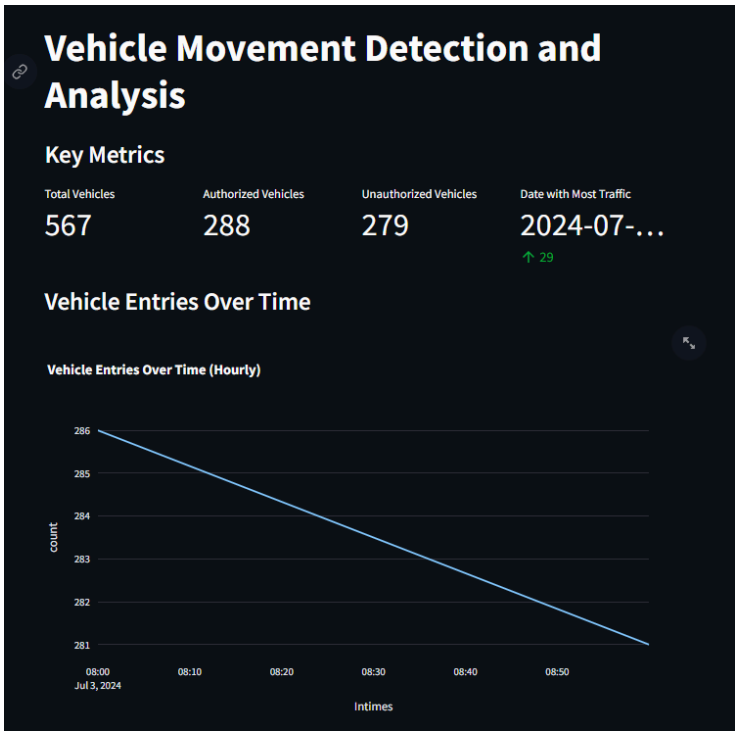


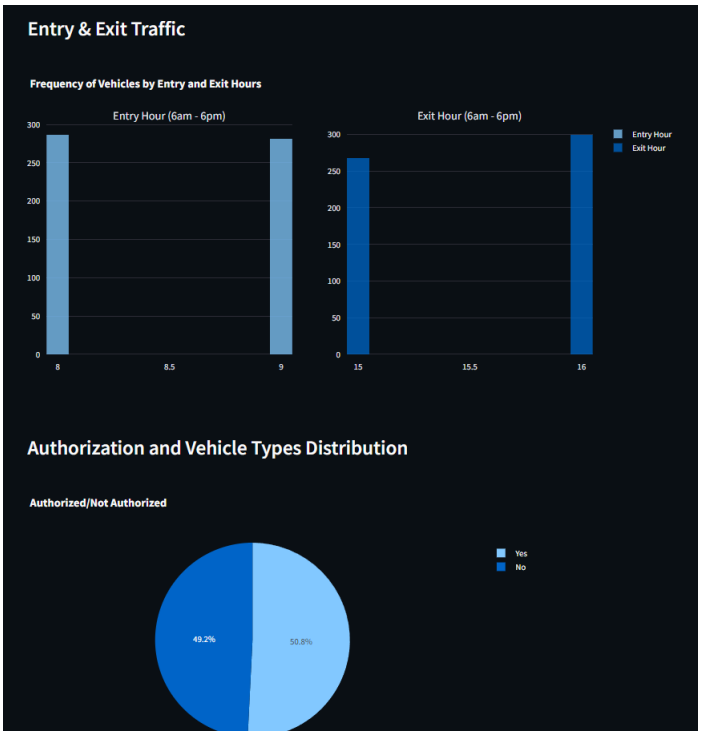
Insights visualization:

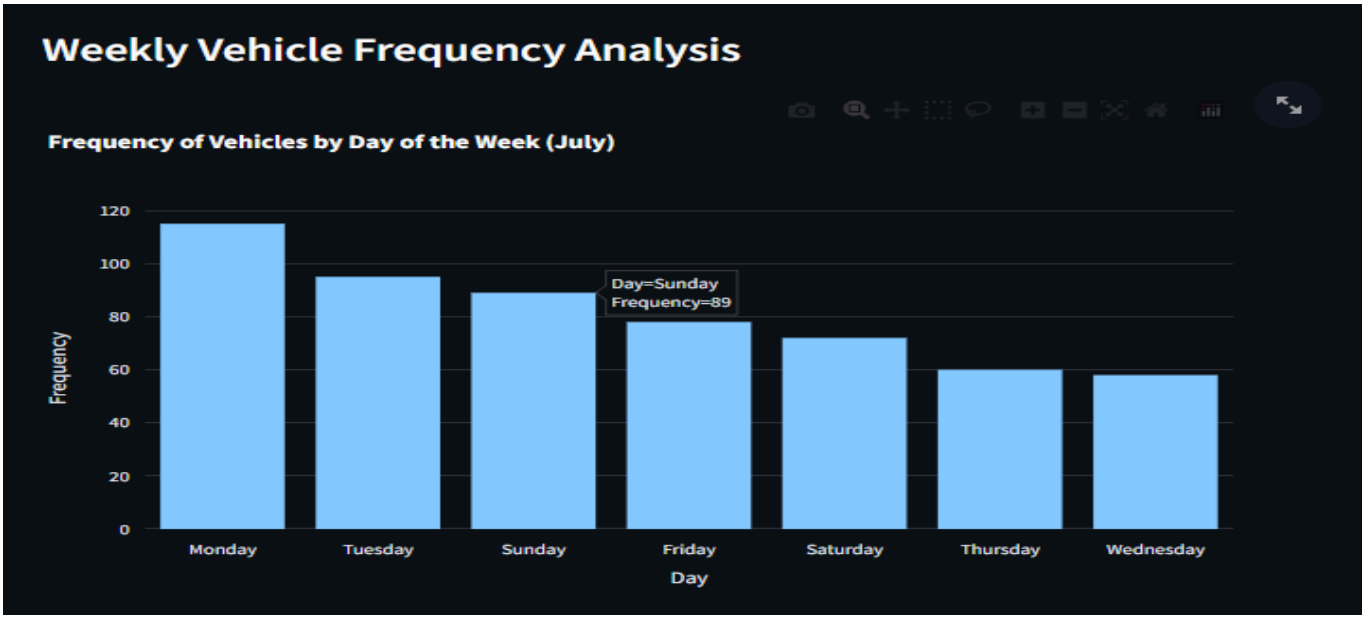
The following screenshots are the version of the insights generation module, deployed on Streamlit:











As mentioned above, more details regarding the approach can be found in the appendix.

Conclusion:

In this project, we implemented a vehicle movement and parking space detection module, which is used to detect incoming vehicles and parking spaces respectively. We also implemented an insights generation module, which uses the data collected from the vehicle movement and parking space modules, to provide meaningful insights in the form of visualizations, and interactive chat, using techniques such as aggregation. We also optimized the modules for low latency, on-device edge device inference, by using techniques such as export to fast inference formats and INT8 quantization.

Future scope:

We wish to experiment with various novel object detection models, such as MobileNet and NanoDet plus, in order to optimize the inference speed on edge devices even further. We also wish to implement even more features in the insight generation module, such as intelligent recognition of reasons for abnormal statistics (such as sudden spike or drop in incoming vehicle frequency), by using a combination of RAG and generative AI models.

References:

[1] Feng, H.; Mu, G.; Zhong, S.; Zhang, P.; Yuan, T. Benchmark Analysis of YOLO Performance on Edge Intelligence Devices. *Cryptography* **2022**, *6*, 16. https://doi.org/10.3390/cryptography6020016

[2] Zherzdev, Sergey, and Alexey Gruzdev. "Lprnet: License plate recognition via deep neural networks." *arXiv preprint arXiv:1806.10447* (2018).

[3] https://github.com/sirius-ai/LPRNet_Pytorch

[4] https://docs.ultralytics.com/modes/export/

[5] https://github.com/xuexingyu24/License_Plate_Detection_Pytorch

[6] https://docs.streamlit.io/

Link to code repo: https://github.com/mopasha1/IntelUnnati2024TTS

Link to datasets: https://drive.google.com/drive/folders/1YbZ3_s_dBmEuzfxu8qAOClS74C3n5T46

END OF REPORT. APPENDIX FOLLOWS

APPENDIX - A

LPRNet model training:

The LPRNet model was originally developed for fast inference on single line, fixed size number plates. However, Indian number plates can contain 8-11 characters, and can be of two lines or a single line.

In order to address this problem, we fixed the length of the output tensor of the LPRNet model as 18. The model was then trained in such a way that the remaining spaces of the output tensor, apart from the predicted codes, were filled with a blank label '-'

In order to tackle the problem of two-line number plates, we generated multiple two line number plates, of various colors, fonts and separators.

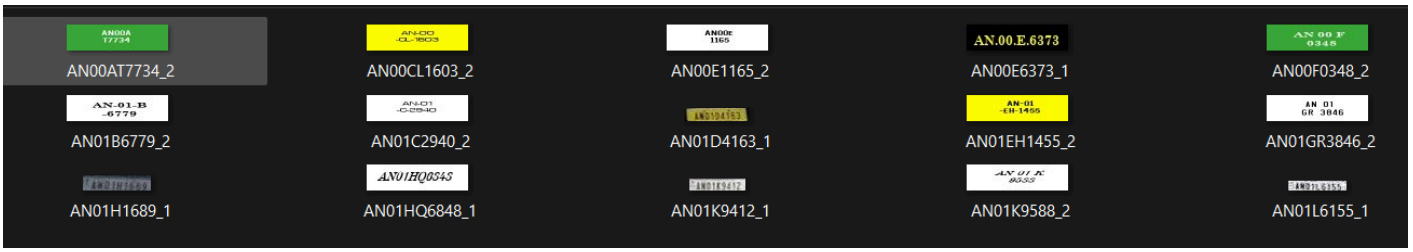


Fig 1: Examples from the License Plate OCR generated dataset

This allowed us to achieve much better results on the test dataset of Indian number plates. Moreover, the use of different fonts made the model more robust, and able to recognize different types of license plate fonts as well.

APPENDIX - B

Model benchmarking on Edge devices

In order to test the efficiency of the models, we ran our code on a Raspberry Pi 4, with 8 GB RAM

Fig 2: Photo of the Raspberry Pi used for benchmarking

We found that the entire process of inference took 701 ms on average, including recognition, OCR detection and post processing, resulting in around ~1.42 FPS throughput. Since this is a completely unaccelerated version of the Raspberry Pi on an older version, we believe that this performance can be highly improved by attaching hardware accelerators, or using an updated processor such as Pi 5.

APPENDIX-C

Parking Lot YOLO model

The parking detection model training was initially done on a dataset of ~5000 images, however, we were unable to get good precision values. This was because of the variety in parking spaces types, camera orientations, weather conditions, lighting conditions, etc.

Hence, we decided to compile multiple open-source datasets, and ended up with the final dataset which encompassed all the above mentioned conditions, and helped us achieve good precision and recall scores.

APPENDIX-D

Insight generation:

In order to provide a wide variety of results to the user, we decided to implement an LLM-powered analysis of the dataset gathered. For this purpose, we used Gemini, which is a large multi-modal model developed by Google. This provides a chat module for the users to interact with data, and ask queries about the data in natural language.

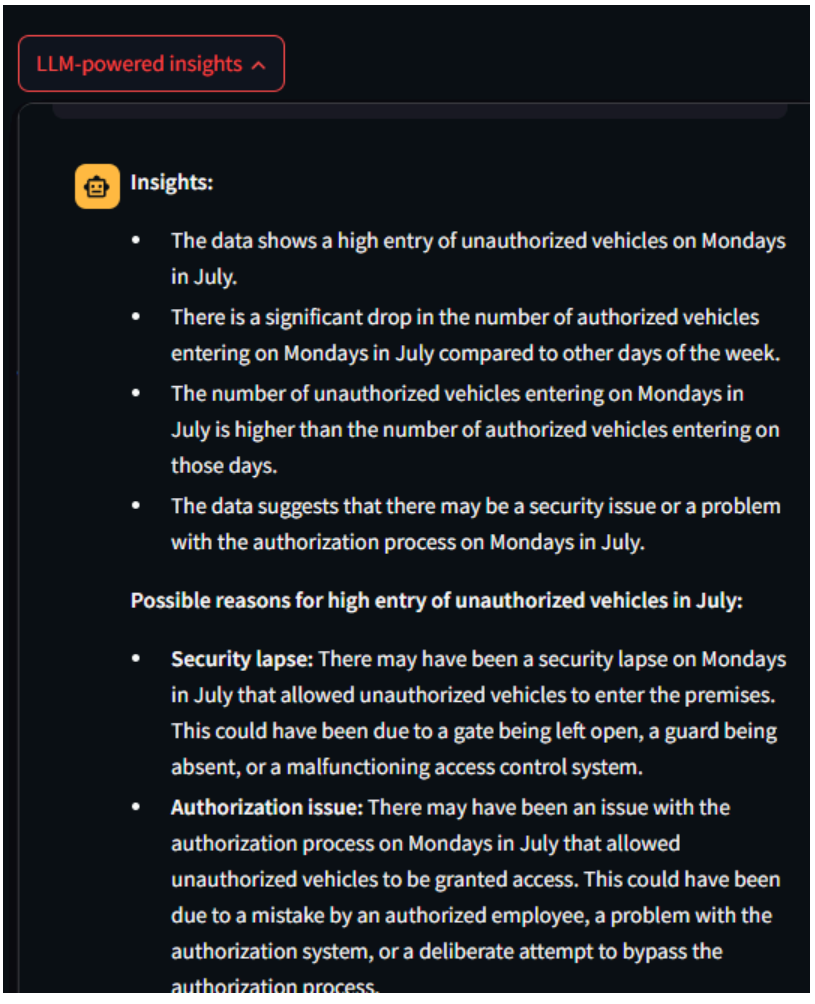


Fig 3: LLM-powered insight generation

APPENDIX-E

Checking for authorized/unauthorized vehicles and logging:

Another major aspect of the problem statement is the checking of authorized/unauthorized vehicles entering into the campus. This problem we solved by storing a dataset of authorized vehicles as a CSV file, and checking if each detected license plate was present in the dataset using dataframe slicing:

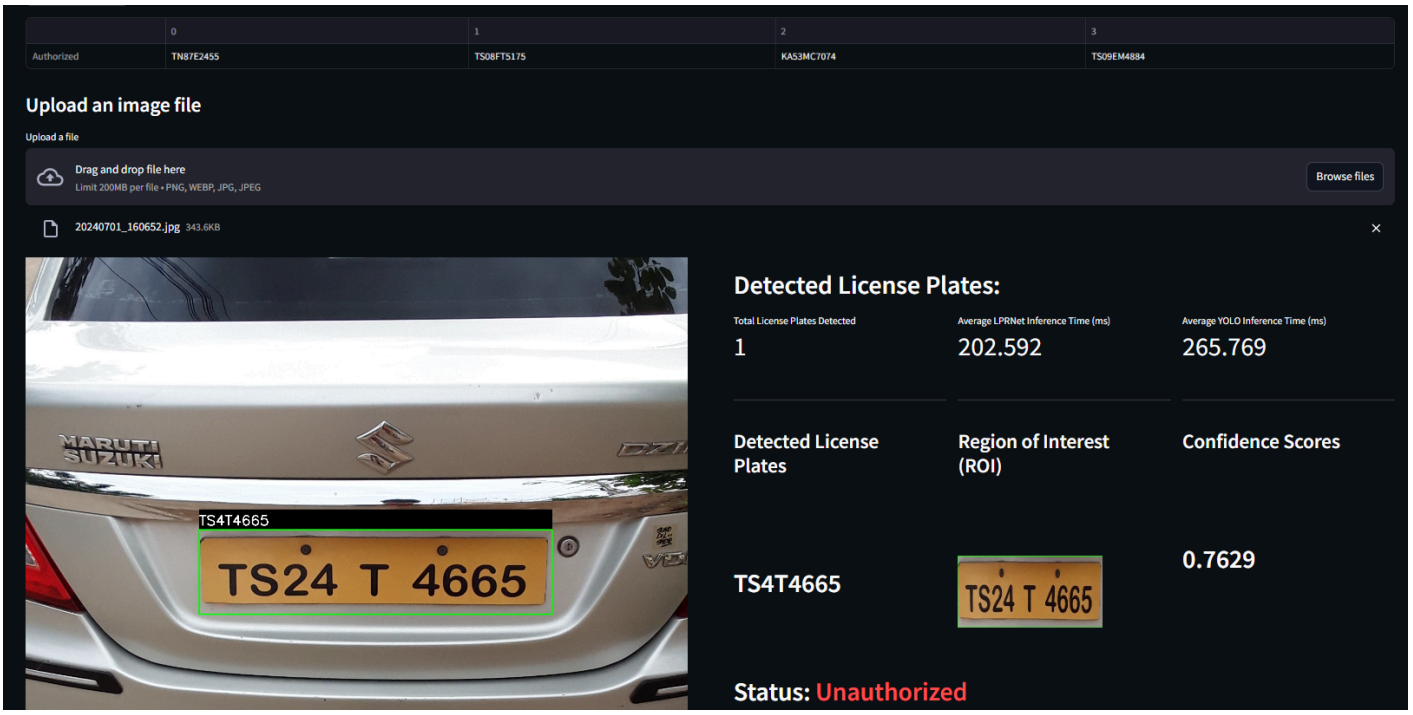


Fig 4: Authorization checking

Since our design is modular, all of the different parts of our project are compatible with one another. Hence, the video_inference model can directly be connected to the insights generation model, and connecting it with a real time video feed results in logs which are stored as a CSV file that can be passed to the insights engine..

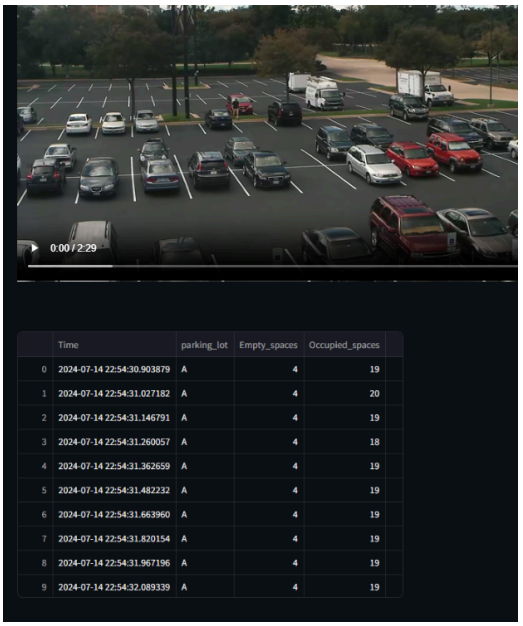


Fig 5: Data logging