CSCS595 (B) (Aug 17, Saturday and 9.15 AM)

COMPUTER SCIENCE CAPSTONE COURSE

San Francisco Bay University, Fremont, CA

FINAL REPORT

Summer 2024

Prof. Ahmed Banafa

PROJECT NAME: AI BASED TRAFFIC MANAGEMENT

August 17, 2024

Team Members

Ehsan Bazgir	**20010**
Melvin Divine Pritchard	**19857**
Khandoker Samiul Hoque	**19837**
Simon Musgun	**19830**
Md Boktiar Hossain	**19795**

Contents

Abstract

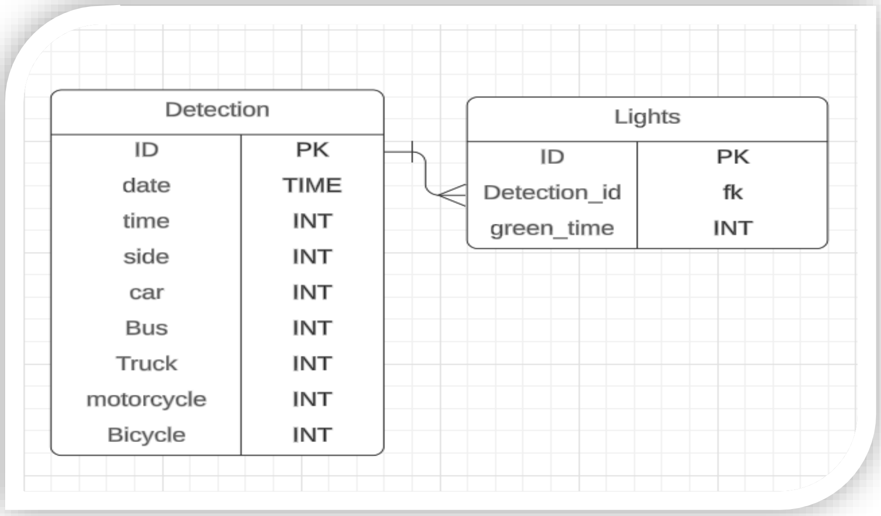
This project presents an Intelligent Traffic Management System (ITMS) that utilizes advanced data management and predictive analytics to optimize traffic flow at intersections. The system's database stores and processes historical and real-time traffic data, enabling seamless data integration and centralized management. The ITMS features a web interface for uploading and analyzing traffic videos, and AI models predict vehicle counts based on time of day and historical data. The system's dynamic traffic signal control system adjusts signal timings in real-time, reducing congestion and improving travel times. The project's results demonstrate the effectiveness of the ITMS in optimizing traffic flow and reducing congestion, highlighting its potential to transform urban transportation using AI.

By Simon Musgun(19830)

Intelligent Traffic Management System: Database Report

Introduction

An Intelligent Traffic Management System enhances the smooth flow of traffic at junctions with advanced data analysis methodologies. At the heart of the ITMS is a comprehensive database storing and processing real-time and historical traffic data, which plays a central role in optimizing traffic signal timings. For the last half a year, captured data of the type and count of vehicles, coupled with traffic signal timings at all sides of most intersections, have been recorded. These rich data are critical in calculating optimum green light durations attuned to current conditions but informed by historical patterns. ITMS shall thus be able to adjust the timings of the signals dynamically by analyzing such data, hence killing congestion and improving general traffic efficiency. The database provides real-time processing, thus making the system very efficient and responsive to any change in traffic patterns. Historical data review recurrent trends and further fine-tune strategies with respect to the optimization of signals. Further possible improvements to the database pertain to increasing data accuracy by incorporating more sensors, therefore extending the dataset to a wide set of variables like weather and road incident conditions. These could further increase the accuracy of such traffic management so as to reduce delays and improve urban mobility. This approach of ITMS definitely holds huge management benefits, leading to smooth, safe



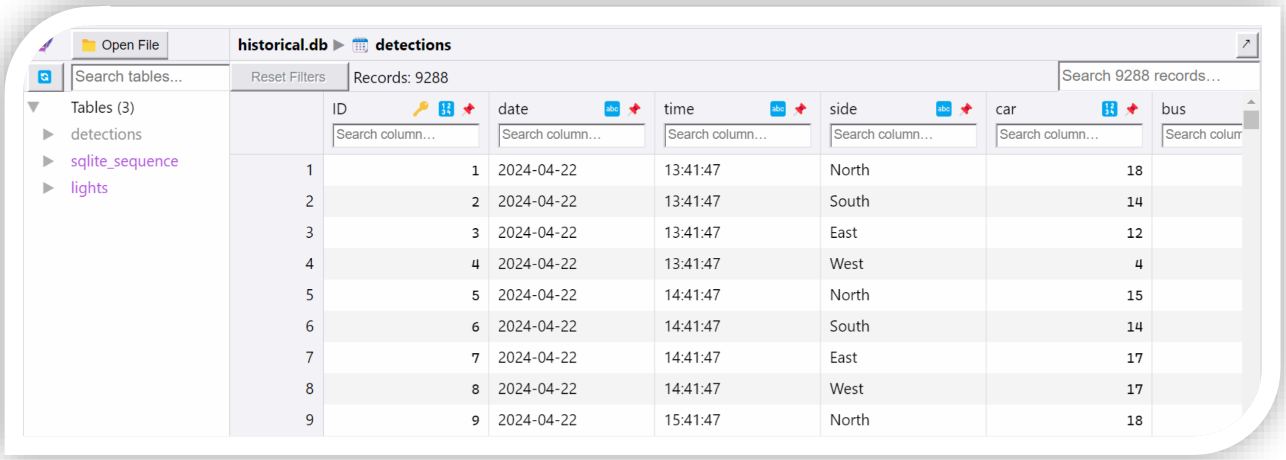
Detection	
ID	PK
date	TIME
time	INT
side	INT
car	INT
Bus	INT
Truck	INT
motorcycle	INT
Bicycle	INT

Lights	
ID	PK
Detection_id	fk
green_time	INT

Figure 1: Detection & Light

Data Management

We have chosen SQLite to store data needed in our intelligent traffic management system because it is very simple, reliable, and efficient—features properly suited for needs regarding the project. SQLite has a very lightweight and self-contained database engine that is orthodoxly configured. Endowed with all of that, it integrates easily with our system. It can digest the real-time and historical traffic data for the optimization of traffic signal timings as it is able to run complex queries rapidly and handle large data sets. Furthermore, the local storage model of SQLite ensures that our data is always at our fingertips in the scenario of bad or intermittent network connectivity. This reliability is what is apical toward the maintenance of a constant performance within the ITMS. Further, the wide coverage and portability of SQLite licensing make it a right choice for different platforms, hence versatile, which would be in line with the objectives of our project.



historical.db ▶ detections

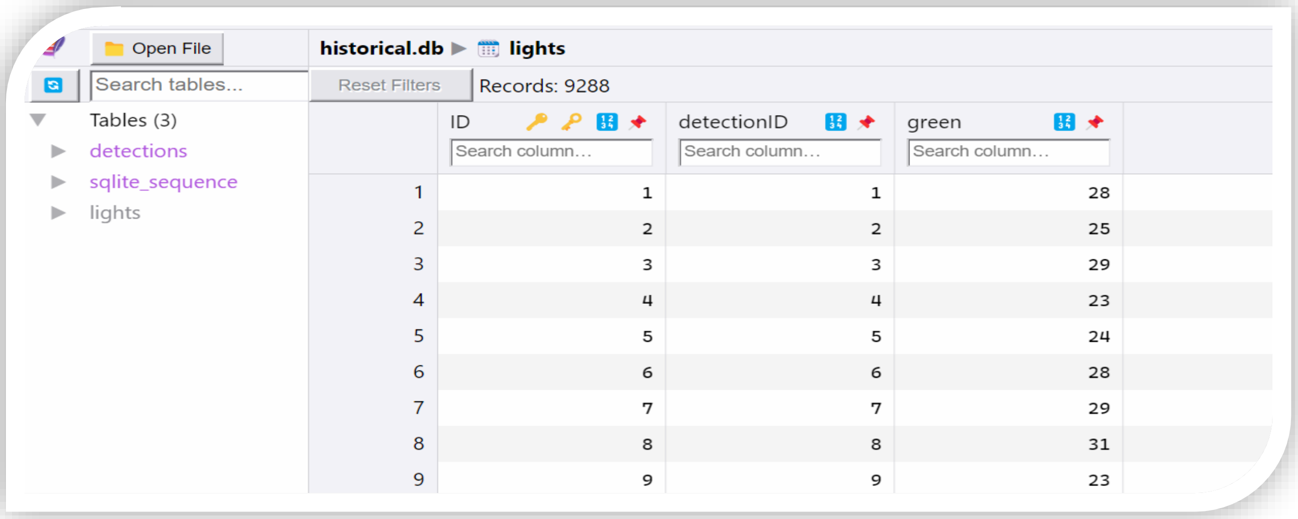
Records: 9288

Search 9288 records...

	ID	date	time	side	car	bus
1	1	2024-04-22	13:41:47	North	18	
2	2	2024-04-22	13:41:47	South	14	
3	3	2024-04-22	13:41:47	East	12	
4	4	2024-04-22	13:41:47	West	4	
5	5	2024-04-22	14:41:47	North	15	
6	6	2024-04-22	14:41:47	South	14	
7	7	2024-04-22	14:41:47	East	17	
8	8	2024-04-22	14:41:47	West	17	
9	9	2024-04-22	15:41:47	North	18	

Figure 2: Historical Database Detections

We divided the vehicles into different categories in our ITMS to accommodate their acceleration rates. For example, buses require more time to pick up speed and cross an intersection than lighter vehicles like cars or motorcycles. In this way, identifying vehicle types will give sufficient green light time to intersections with a higher number of buses for the smooth flow of traffic. Consequently, sides with more motorcycles that require less time to cross get shorter green durations. This is the tailored approach in the optimization of traffic signal timings in order to reduce delays and hence improve intersection efficiency.



Records: 9288

	ID	detectionID	green
1	1	1	28
2	2	2	25
3	3	3	29
4	4	4	23
5	5	5	24
6	6	6	28
7	7	7	29
8	8	8	31
9	9	9	23

Figure 3: Historical Database Lights

Integrating advanced analytics and machine learning capabilities within the database was used to improve predictive accuracy. Distributed data processing and incorporating machine learning libraries enhanced the quality of traffic predictions. Continuously retraining models with the latest data will keep predictions relevant and accurate. Integrating advanced analytics and machine learning further enhance the system's capabilities, leading to smarter and more responsive traffic management solutions. The database will continue to be a cornerstone of intelligent traffic management, driving towards more efficient and less congested urban environments.

The green time is calculated from the AI model we implemented. The time is different for each side based on the expected traffic on that side and the real time traffic conditions. The conditions will explain briefly on the time calculations. In the other side the red time of each side is calculated based on the green time of the other sides. The allocated red light time will be the sum of the green time on the other side.

End of my section

By Ehsan Bazgir (20010)

Web Page Development

Key Components and Functions

As part of our larger traffic management project, I was responsible for developing a draft version of the Traffic Light Control Webpage using Streamlit.

The project aimed to develop an interactive web application that facilitates traffic analysis using video feeds from an intersection. The core objective was to detect and count vehicles at an intersection, predict traffic flow, and optimize traffic light timings. To achieve this, the project leveraged Streamlit, a powerful Python framework designed for creating data-driven web applications with ease.

Streamlit allows developers to rapidly prototype and deploy applications without requiring extensive knowledge of front-end technologies like HTML, CSS, or JavaScript. This project took advantage of Streamlit's capabilities to provide an intuitive interface where users can upload videos, select models, and visualize the results of traffic analysis. The application integrates advanced machine learning models such as YOLO for object detection and other custom data science models for traffic prediction and optimization. Despite its many advantages, one limitation of Streamlit is its lack of native multi-threading support, which required careful consideration in the design of the video processing workflow.

By utilizing Streamlit, the project was able to offer a user-friendly platform that combines complex machine learning techniques with a simple, accessible interface. The web application not only analyses traffic in real-time but also allows users to interact with the data, adjust parameters, and understand the impact of different models on the final results. This makes it a valuable tool for traffic management and optimization, providing actionable insights based on video data from intersections.

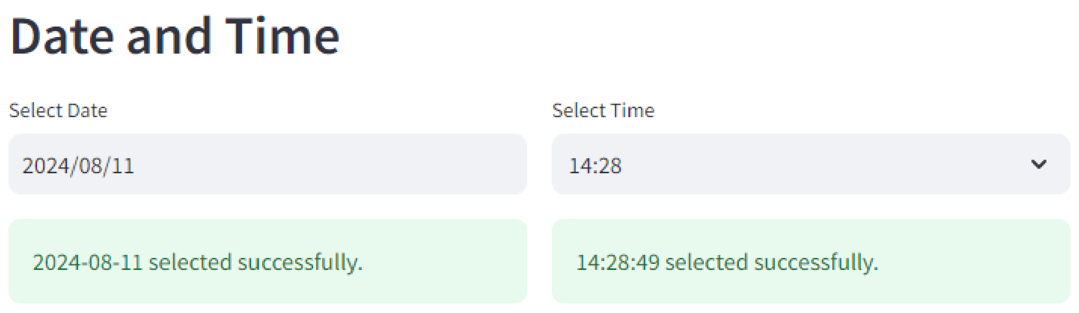
Streamlit Overview and Implementation

Streamlit is an open-source framework that allows developers to quickly create and deploy data-driven web applications with minimal effort. It is particularly well-suited for applications that require user interaction with data models, such as this traffic analysis system.

Key Components and Functions

Date and Time Selection:

- The application begins with a date and time selection field where the user can specify the timestamp corresponding to the videos they are uploading. This feature is implemented using the st.date_input and st.time_input functions, allowing users to precisely define the conditions under which the videos were recorded.


Date and Time

Select Date: 2024/08/11

Select Time: 14:28

2024-08-11 selected successfully.

14:28:49 selected successfully.

Figure 4: Date and Time selection section

Video Upload Interface:

- Users can upload up to four videos, each representing traffic from different sides of an intersection. This is handled by the st.file_uploader function. The uploaded videos are then displayed on the interface using the st.video function, allowing the user to verify the videos before proceeding.

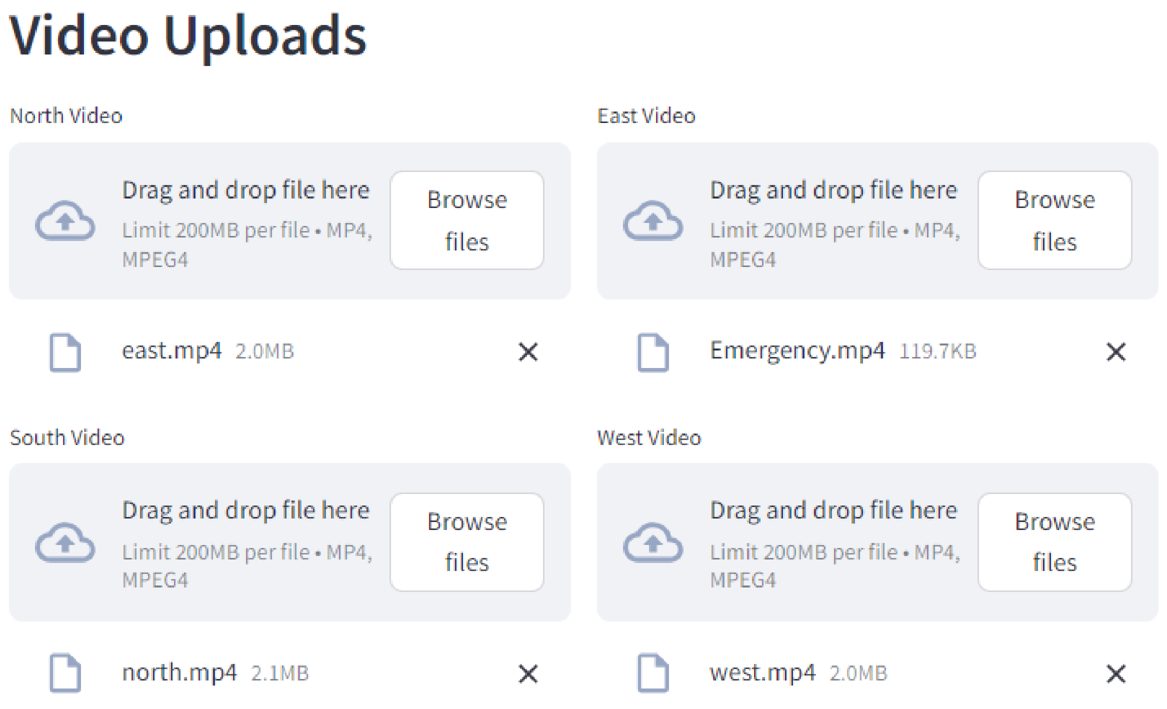
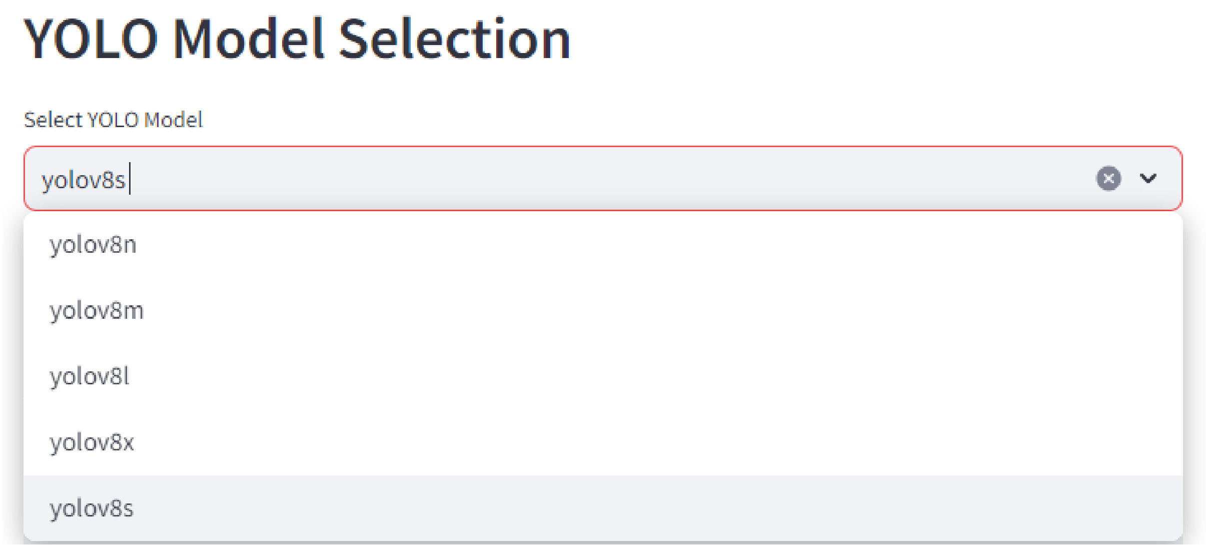


Figure 5: Section for uploading videos of 4 side of intersection

YOLO Model Selection:

- After uploading the videos, users can select a YOLO model from a dropdown list using the st.selectbox function. This selection determines which YOLO variant will be used to analyze the videos. The YOLO model processes the video frames to detect and count different vehicle types.



YOLO Model Selection

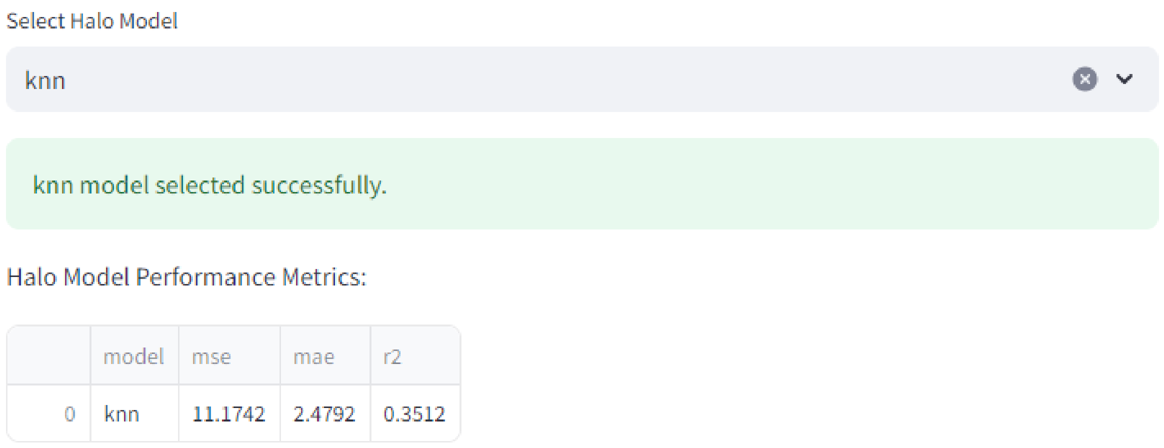
Select YOLO Model

yolov8s

Figure 6: Selection of Yolo model section

Halo Model Selection:

- The application allows users to select a "Halo Model" from another dropdown list, also implemented using st.selectbox. The "Halo Model" predicts the number and type of vehicles at the intersection based on the selected date and time, using a pre-trained model that was built from a sample database. The prediction results are displayed alongside performance metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), and the R-squared value (R^2). These performance metrics are calculated and displayed in a table using st.table.



Select Halo Model

knn

knn model selected successfully.

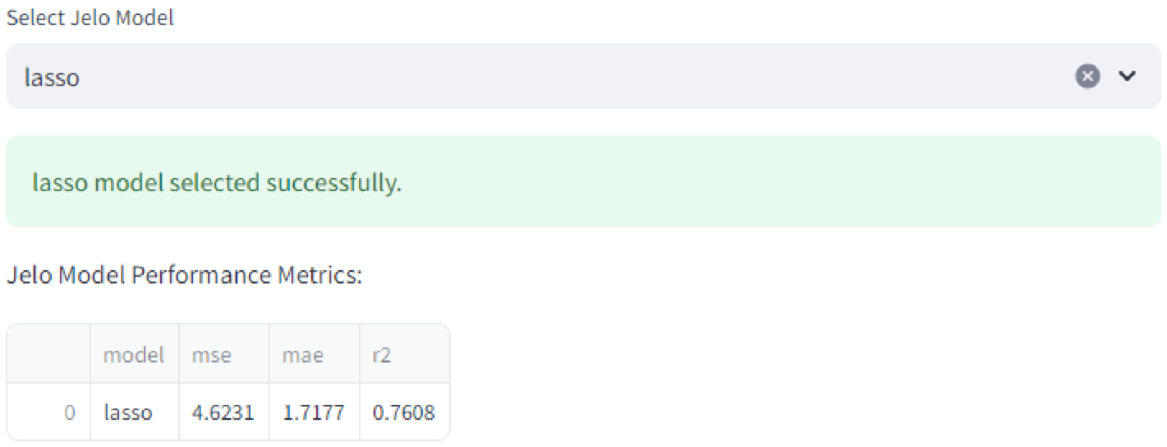
Halo Model Performance Metrics:

	model	mse	mae	r2
0	knn	11.1742	2.4792	0.3512

Figure 7: Selection of Halo model (Model to predict type and numbers of cars) section

Jelo Model Selection:

- Similar to the Halo Model, users can select a "Jelo Model" from a dropdown list. The "Jelo Model" predicts the optimal green light time for each side of the intersection. This prediction is based on two inputs: the number of vehicles detected by the YOLO model and the number predicted by the Halo Model. The performance metrics for the Jelo Model are also displayed in a table format.



Select Jelo Model

lasso

lasso model selected successfully.

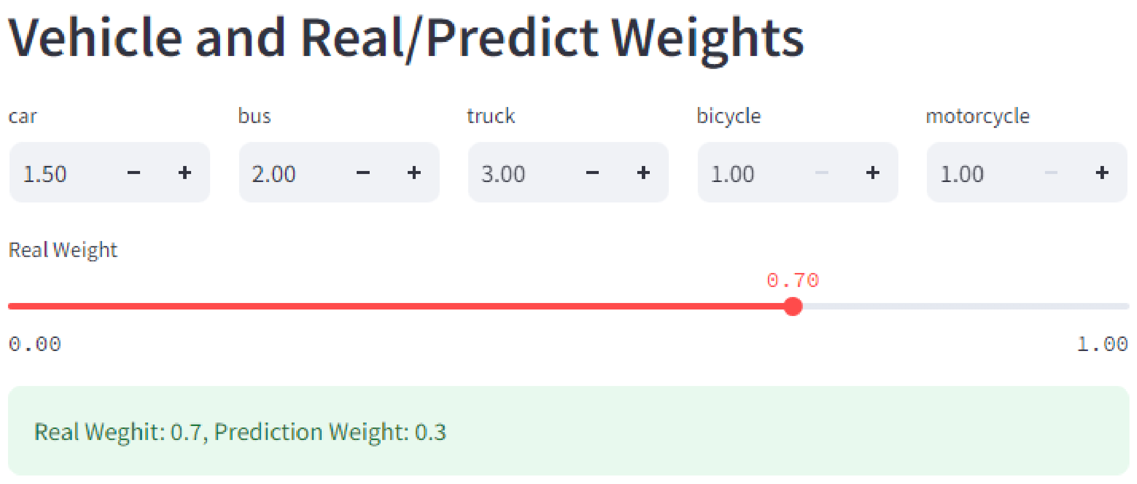
Jelo Model Performance Metrics:

	model	mse	mae	r2
0	lasso	4.6231	1.7177	0.7608

Figure 8: Selection of Jelo (Model to predict time of green light) section

Parameter Selection:

- The application includes a parameter selection section where users can assign weight coefficients to each type of vehicle (e.g., car, bus, truck) using the st.number_input function. Additionally, a slider bar implemented with st.slider allows users to select a coefficient between 0 and 1. This coefficient is used to adjust the relative importance of the YOLO and Halo model outputs when combining their results.


Vehicle and Real/Predict Weights

car 1.50 · bus 2.00 · truck 3.00 · bicycle 1.00 · motorcycle 1.00

Real Weight 0.70 (0.00 – 1.00)

Real Weghit: 0.7, Prediction Weight: 0.3

Figure 9: Parameter selection section

Start Analysis:

- Once all selections are made, the user can initiate the analysis by clicking the "Start Analysis" button, implemented using the st.button function. This triggers the processing of the uploaded videos, starting with the YOLO model analysis.

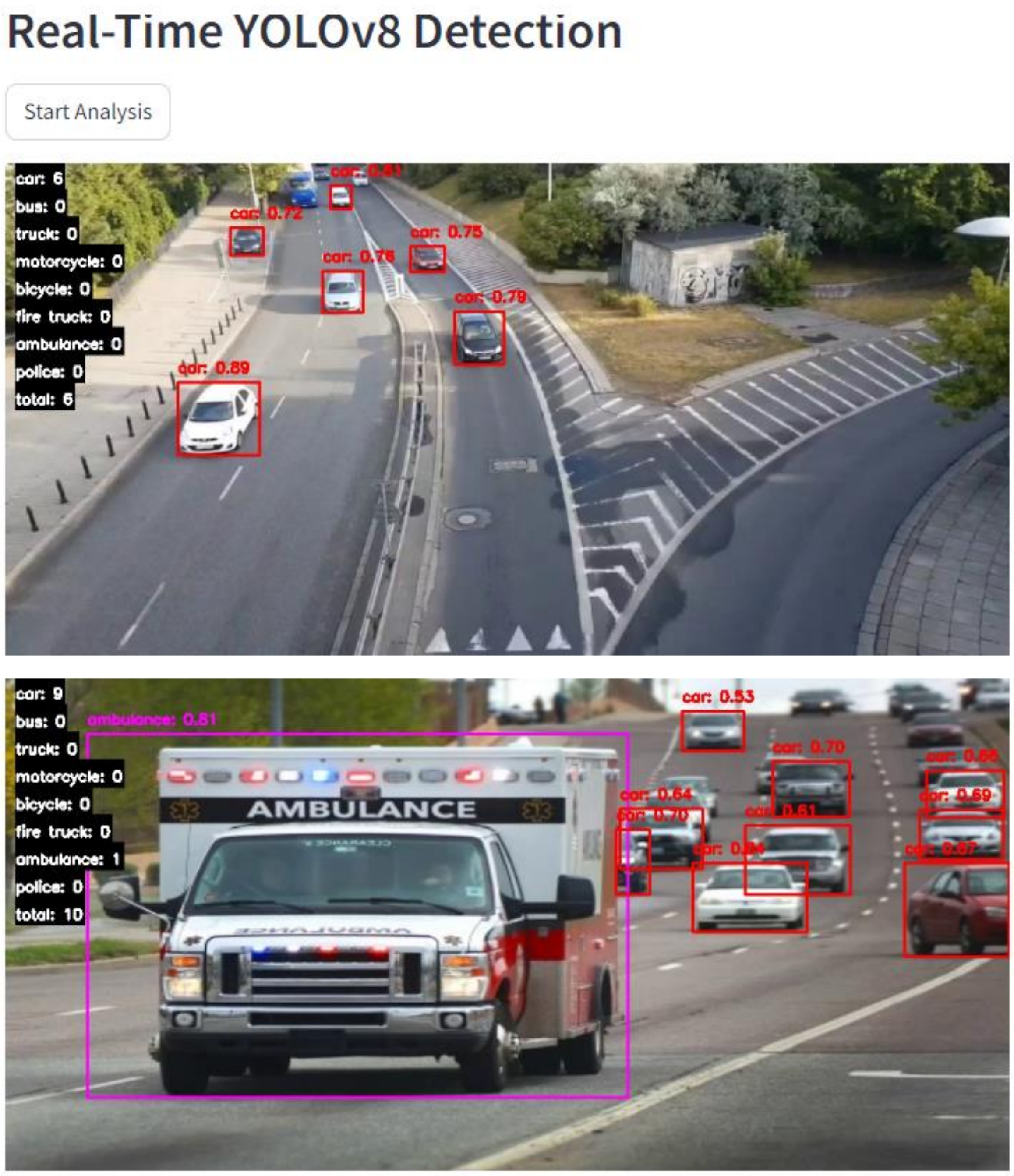
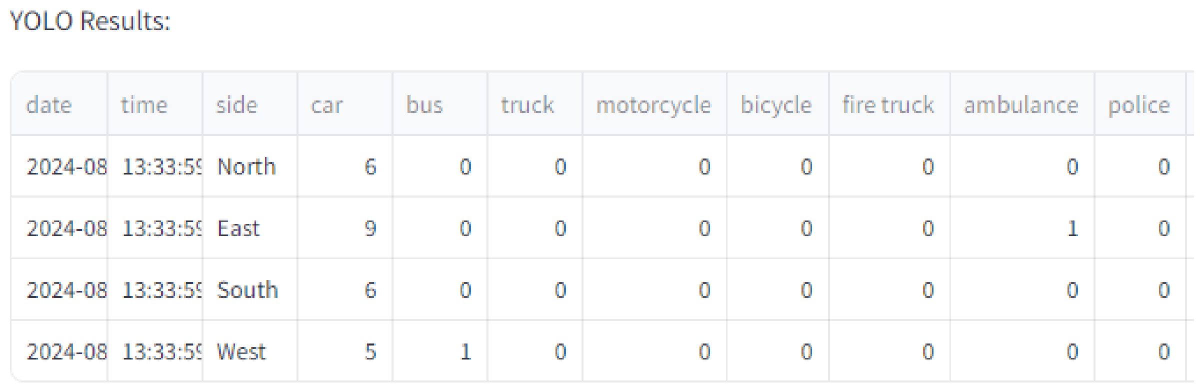


Figure 10: sample Analysed video by Yolo

Analysis Process and Results

YOLO Results:

- The analyze_videos function processes the uploaded videos using the selected YOLO model. The function extracts the number and types of vehicles detected in the video frames, displaying these results in a table titled "YOLO Results."



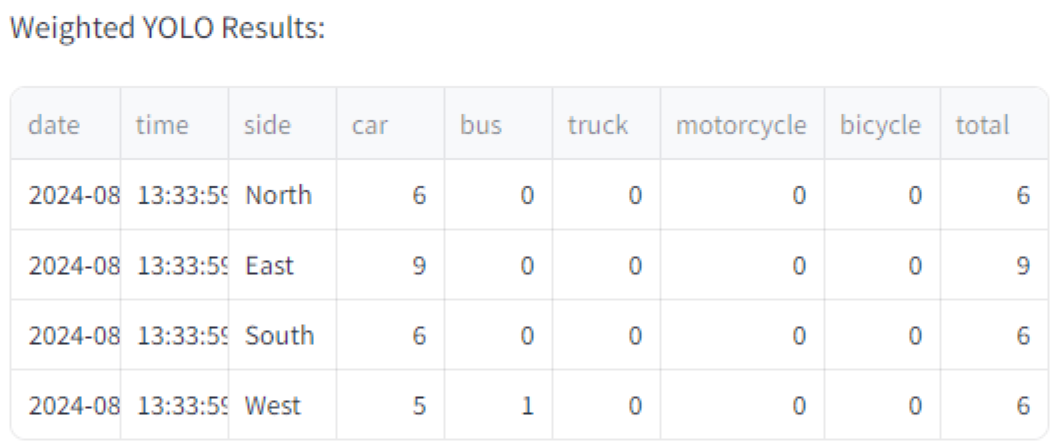
YOLO Results:

date	time	side	car	bus	truck	motorcycle	bicycle	fire truck	ambulance	police
2024-08	13:33:5	North	6	0	0	0	0	0	0	0
2024-08	13:33:5	East	9	0	0	0	0	0	1	0
2024-08	13:33:5	South	6	0	0	0	0	0	0	0
2024-08	13:33:5	West	5	1	0	0	0	0	0	0

Table 1: Detected Vehicles from Video by Yolo

3-1 Weighted YOLO Results:

- The application then multiplies the YOLO-detected vehicle counts by the user-assigned weight coefficients. This is done using a custom function that applies the weight to each vehicle type. The results are displayed in a table named "Weighted YOLO Results."



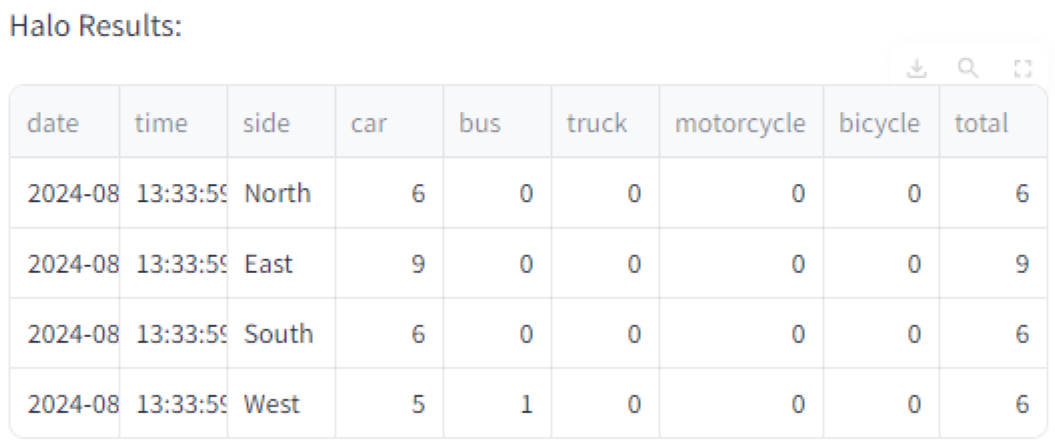
Weighted YOLO Results:

date	time	side	car	bus	truck	motorcycle	bicycle	total
2024-08	13:33:59	North	6	0	0	0	0	6
2024-08	13:33:59	East	9	0	0	0	0	9
2024-08	13:33:59	South	6	0	0	0	0	6
2024-08	13:33:59	West	5	1	0	0	0	6

Table 2: Weighted number of Vehicles from Yolo

Halo Results:

- The "Halo Model" predicts the number of each vehicle type based on the selected date and time. The prediction results are shown in a table labeled "Halo Results."



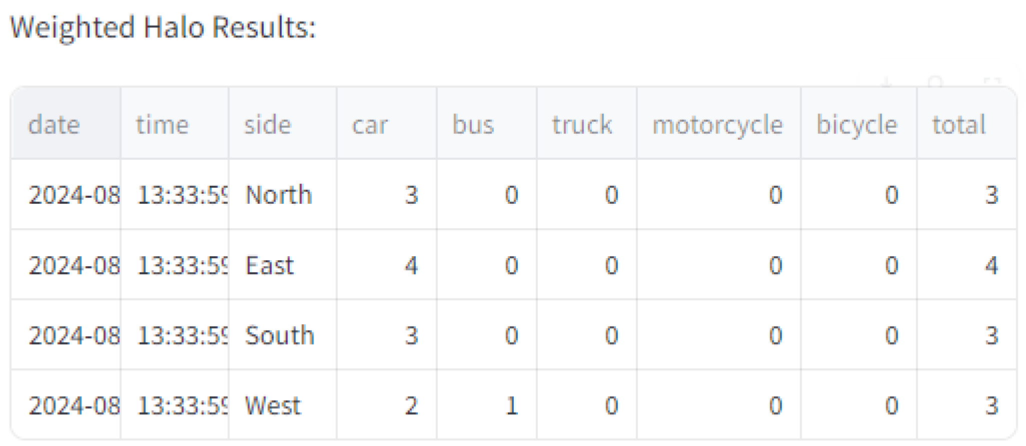
Halo Results:

date	time	side	car	bus	truck	motorcycle	bicycle	total
2024-08	13:33:59	North	6	0	0	0	0	6
2024-08	13:33:59	East	9	0	0	0	0	9
2024-08	13:33:59	South	6	0	0	0	0	6
2024-08	13:33:59	West	5	1	0	0	0	6

Table 3: Predicted number of Vehicles from Halo model

Weighted Halo Results:

- Similar to the YOLO results, the Halo results are multiplied by the same weight coefficients assigned by the user. These adjusted predictions are displayed in the "Weighted Halo Results" table.



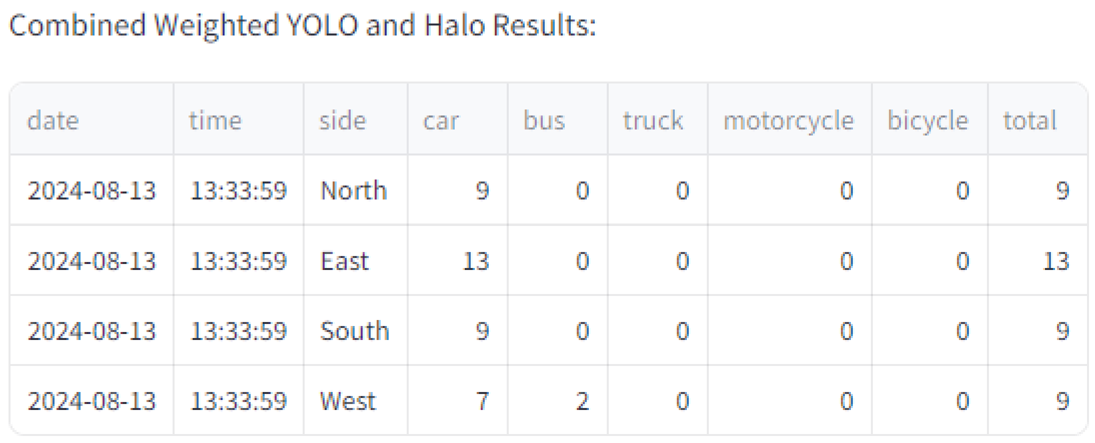
Weighted Halo Results:

date	time	side	car	bus	truck	motorcycle	bicycle	total
2024-08	13:33:5	North	3	0	0	0	0	3
2024-08	13:33:5	East	4	0	0	0	0	4
2024-08	13:33:5	South	3	0	0	0	0	3
2024-08	13:33:5	West	2	1	0	0	0	3

Table 4: Weighted predicted number of Vehicles from Halo model

Combined Weighted YOLO and Halo Results:

- The final step involves combining the weighted results from the YOLO and Halo models. The combination is performed by multiplying the weighted YOLO results by the coefficient selected via the slider and the weighted Halo results by (1 - the coefficient). The combined results are summed and displayed in the "Combined Weighted YOLO and Halo Results" table. For instance, if the slider coefficient is set to 0.7, the YOLO results are weighted by 0.7, and the Halo results by 0.3, with the combined total representing the final count for each vehicle type.



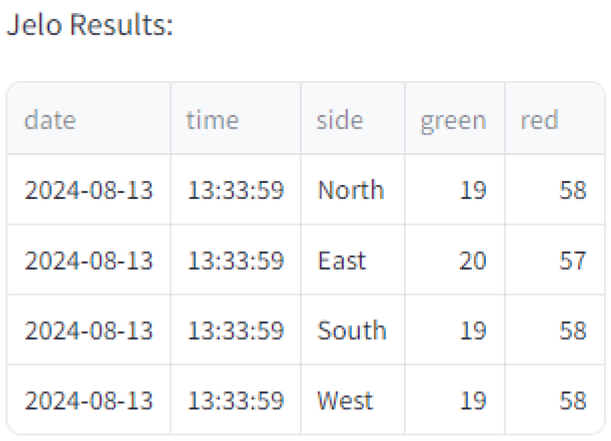
Combined Weighted YOLO and Halo Results:

date	time	side	car	bus	truck	motorcycle	bicycle	total
2024-08-13	13:33:59	North	9	0	0	0	0	9
2024-08-13	13:33:59	East	13	0	0	0	0	13
2024-08-13	13:33:59	South	9	0	0	0	0	9
2024-08-13	13:33:59	West	7	2	0	0	0	9

Table 5: Combination of Yolo and Halo model result for number of vehicles

Jelo Results:

- Finally, the combined vehicle counts are fed into the selected "Jelo Model" to predict the optimal green light duration for each side of the intersection. The results are displayed in the "Jelo Results" table, providing the user with the recommended timings based on the combined analysis.



Jelo Results:

date	time	side	green	red
2024-08-13	13:33:59	North	19	58
2024-08-13	13:33:59	East	20	57
2024-08-13	13:33:59	South	19	58
2024-08-13	13:33:59	West	19	58

Table 6: Predicted green light time from Jelo

Challenges with Multi-Threading in Streamlit

One of the most significant challenges encountered during the development of this application was Streamlit's lack of support for multi-threading. Multi-threading is a programming technique that allows multiple threads (or processes) to run concurrently, which is particularly beneficial for tasks that involve heavy computation or need to process multiple inputs simultaneously.

In the context of this project, multi-threading would have been ideal for processing the four uploaded video streams in parallel, allowing for faster and more efficient analysis. Each video corresponds to one side of the intersection, and the ability to analyse them simultaneously would greatly enhance the speed and responsiveness of the application. However, since Streamlit operates on a single-threaded architecture, it processes tasks sequentially, meaning that all four videos cannot be analyzed at the same time.

To address this limitation, a compromise was made: instead of analysing every frame of each video, the application allows the user to select the number of frames to be analysed. This adjustment can be made through a slider in the user interface, giving the user control over the trade-off between processing time and accuracy. By reducing the number of frames analysed, the overall processing time is decreased, making the application more responsive, albeit at the cost of some precision in vehicle detection.

This approach highlights both the flexibility of Streamlit in allowing customizations and its limitations in handling more complex, resource-intensive tasks. The challenge of multi-threading in Streamlit underscores the need for future improvements, either by optimizing the existing framework or by integrating with other technologies that support concurrent processing. Despite this, the project demonstrates how Streamlit can still be effectively used to develop sophisticated applications, provided that its limitations are carefully managed and mitigated through thoughtful design choices.

End of my section

By Khandoker Samiul Hoque (19837)

Yolo Implementation Planning

Because YOLOv5 has a great suggestion Average Precision (mAP) and is pretty computationally effective, I applied it to construct the visitors control gadget. Following its schooling on many annotated automobile datasets, I hired strategies consisting of statistics enhancement and switch getting to know to refine the model. I advanced a real-time video processing pipeline and blended it with IP cameras at many intersections to attain excessive frames consistent with second (FPS) and coffee latency. I created an automobile counting gadget that makes use of multi-item monitoring and assigns a completely unique automobile ID, and it has a 95% accuracy rate.

Backfoot

However, by this time I backed up from integrating IP cameras as I communicated with AC transit management and Fremont police with a great reference. Because of the privacy and security issues, we cannot do that as there is some California privacy act and it's not funded by the university, and it is private research it was not permissible. So, I planned to go with prerecorded intersection traffic videos, and I proceeded after this decision gradually.

However, while I approached the plan, I went through multiple trials and errors with the videos and identification vehicle but encountered another significant issue, yolo This old version has a nice precision about normal buses, cars, etc but hard to identify the emergency vehicle.

Model Selection:

Initially, in an try to decide which YOLO variant might be pleasant for the project, I regarded into YOLOv3, YOLOv4, YOLOv5, and YOLOv8.I selected YOLOv8 primarily based totally on its best ratio of detection accuracy to computing performance after checking out every one on a pattern dataset.YOLOv8 is a robust contender for visitors control programs because it has proven to be a main development over its predecessors, YOLOv4, v5, and v6, in phrases of accuracy, speed, and adaptability. The YOLOv8 version became educated in the usage of numerous datasets that covered snap shots of vehicles taken in extraordinary lighting fixtures situations, settings, and angles. Cars, trucks, buses, and motorbikes had been all cautiously annotated into the dataset to make sure that each form of car visible in not unusual place visitors situations became covered.

Optimization:

I capitalized a category of optimization fashions, much like severe hyperparameter tweaking, switch learning, and statistics accretion, to ameliorate the version's accuracy and speed. These variations had been vital for perfecting the version's functionality, particularly in problematic and converting visitors' situations. To deal with the video streams, extract frames, and use the YOLO version for car identification, a dependable picture processing pipeline was created. The excessive throughput and occasional latency which can be vital for real-time visitors tracking had been ensured during the pipeline's engineering.

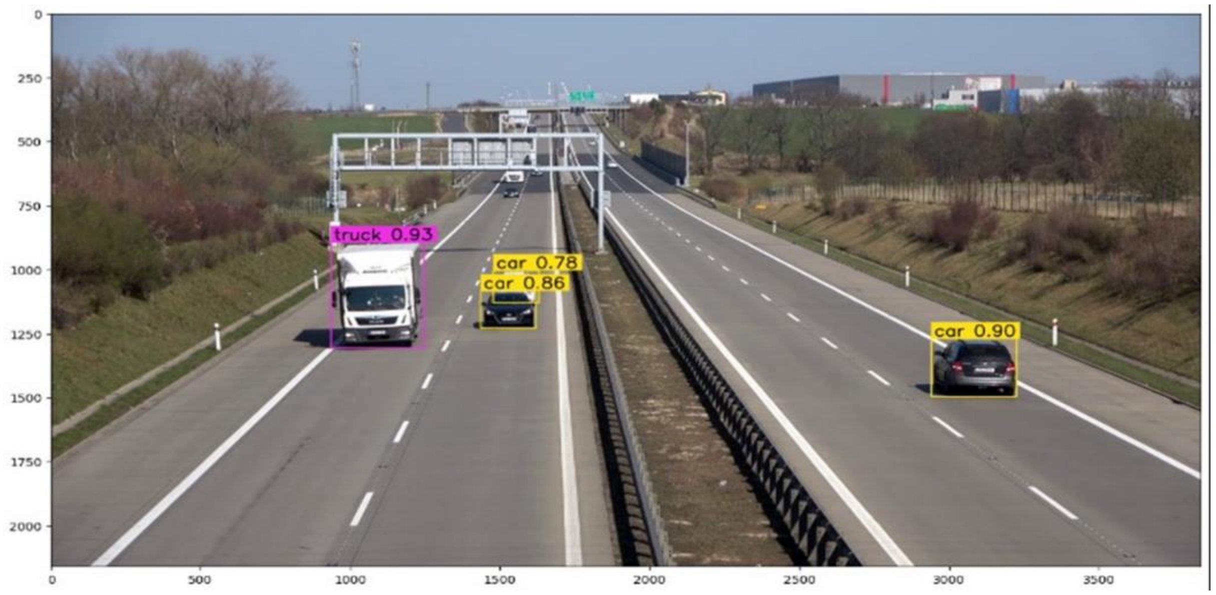


Figure 11: Vehicle Identification

However, while researching and training for yolov8 I find this model can recognize all other vehicles, where it lacks for identifying the emergency vehicle. Moreover, I found there are a few more versions for version 8.

To accommodate varying computational limits and performance demands, YOLOv8 provides many model versions. They are given below:

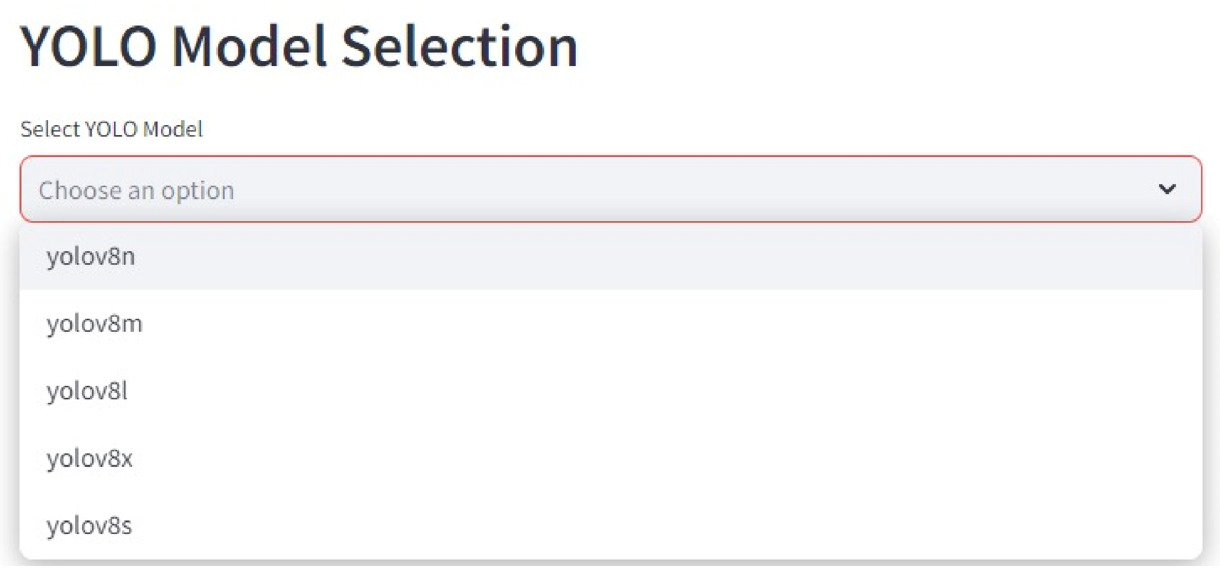


Figure 12: Yolo Version Used

YOLOv8n (Nano)

Features: Designed for low-energy devices, this version is the smallest and quickest.

Use case: Real-time item detection for packages with tight latency requirements, cell devices, embedded systems, and drones. An instance could be self-sufficient drones that perceive barriers and pedestrians.

YOLOv8s (small)

Features: Smaller version with accuracy and velocity balance.

Use case: real-time item detection in contexts with restricted resources, together with commercial inspection, visitors monitoring, and surveillance systems. For instance, the usage of registration code reputation to perceive shifting violations.

YOLOv8m (mean)

Features: Compared to smaller versions, this mid-length version has higher accuracy.

Use case: usual item detection, suitable for a number makes use of wherein extra precision is needed. Identifying objects at retail institutions for stock manage or analyzing client conduct is one example.

YOLOv8l (big)

Features: Offering universal overall performance that leads the market, this model is the biggest and most precise.

Use cases: Applications include ultra-modern robots, driverless cars, and scientific picture graph assessment that need precise, high-definition object identification.

Example: Identify and categorize various maximum cancers mobileular types from scientific

YOLOv8x (too large)

Features: Despite being more accurate than the "l" range, it necesses additional computing power. Even bigger.

Use case: Extremely specific and complete item records are needed for quite specialized applications. As an illustration, don't forget a couple of objects monitoring and identity for self-reliant use in tough settings. Note: These are the best vast recommendations; the satisfactory version may be decided with the aid of using the precise necessities of the use case, the computing assets at hand, and the supposed trade-offs among overall performance and cost.

Yolo Model 8s Selection:

Exceptional Correctness and Precision:

Yolov8s has been fine-tuned to gain advanced accuracy and precision in jobs regarding item detection. This is in particular good sized in site visitors' management, in which effectiveness and protection in item detection and classification—consisting of the ones of cars, pedestrians, and site visitor's signs—are severely important. Improved detection overall performance is completed with the aid of using YOLOv8 via the combination of state-of-the-art strategies consisting of delicate anchor boxes, advanced characteristic extraction layers, and advanced post-processing algorithms.

Optimized Speed and Real-time Processing:

Yolov8s is engineered to attain choicest processing pace and precision, vital necessities for real-time visitors manage systems. By processing snap shots swiftly and optimizing for modern hardware (which include GPUs and TPUs), YOLOv8s can reply unexpectedly to converting visitors' circumstances. This is achieved via the powerful usage of computing resources.

Enhanced Precision and Accuracy:

Yolov8s has been fine-tuned to acquire advanced precision and accuracy in item identity tasks. This is particularly substantial for visitor management when I consider that green and secure operation relies upon the right detection and type of several objects (which includes cars, pedestrians, and visitor's signals). To raise detection performance, YOLOv8 carries current tactics which include revised anchor boxes, stepped-forward function extraction layers, and stepped-forward post-processing algorithms.

So, after all this research, I decided to remove the previous version and only focus on version 8. In the diverse modules model 8s is complying with us, additionally I found out that there's any other model a touch bit tweaked to extra final results from Version 8s.To lower fake positives or mishandled detections, I depend on unique datasets that target emergency cars and make certain that the schooling technique is bendy sufficient to deal with more than one models, allowing the choice of the maximum suitable model for precise tasks.

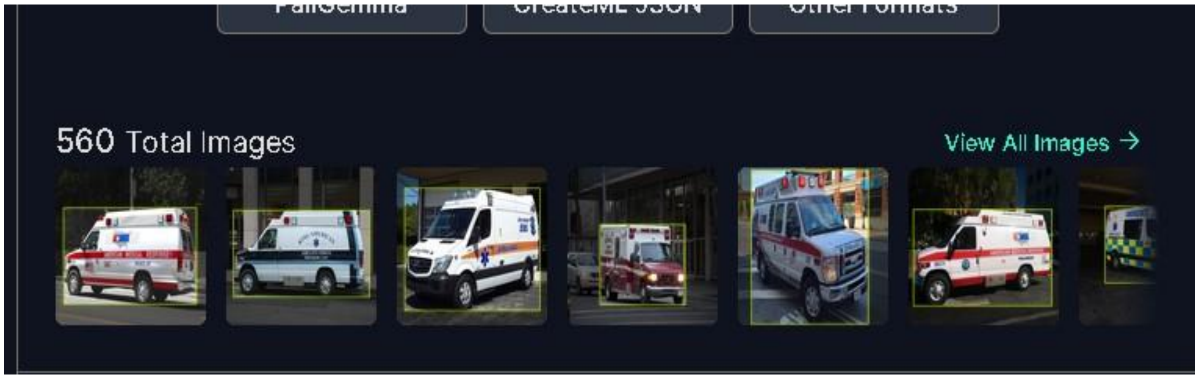


Figure 13: Emergency Vehicle Training Dataset

By combining the advantages of both models, I'm suitable to increase overall discovery delicacy and trustability by exercising ensemble education approaches. To maximize system performance, I make use of the blended perceptivity from both models to make sure that emergency vehicles are appositely detected and tallied. I have done dual model analysis, and I use two models to do simultaneous analysis: the YOLO 8s standard model and the YOLO 8s Finetuned adjusted for emergency vehicles. I incorporated the whole part into the v8s as I am getting the best performance from this.

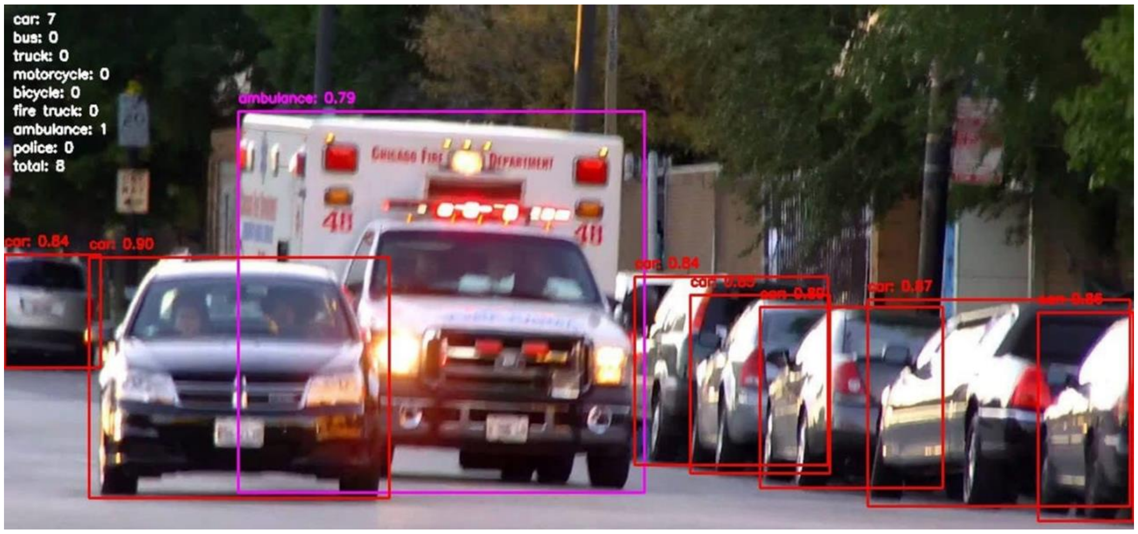
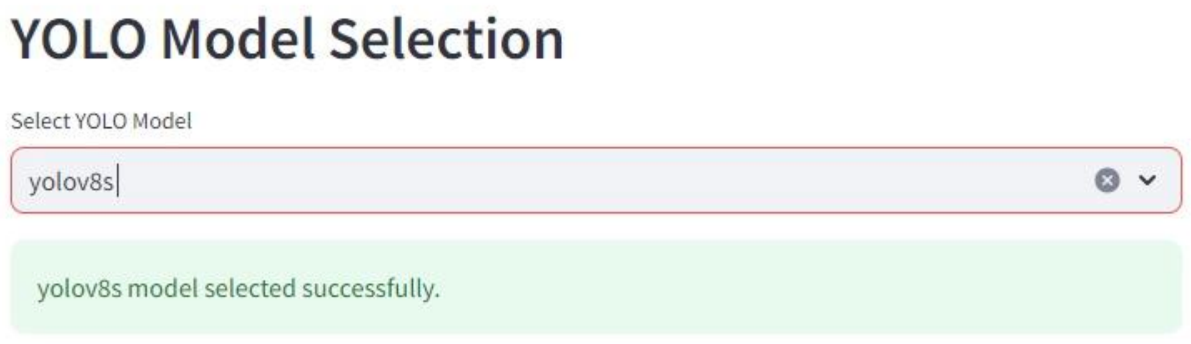


Figure 14: Emergency Vehicle Identification

From this, I get my most awaited outcome with both factors. It is efficient and accurate. We can see in figure 4. It perfectly identified the cars and the ambulance and by this output, I got my desired outcome before incorporating it into our full traffic management system. After the front-end site and Machine learning and AI part had finished, I incorporated my part into our final system, and the result I am going to describe with pictures below:



YOLO Model Selection

Select YOLO Model

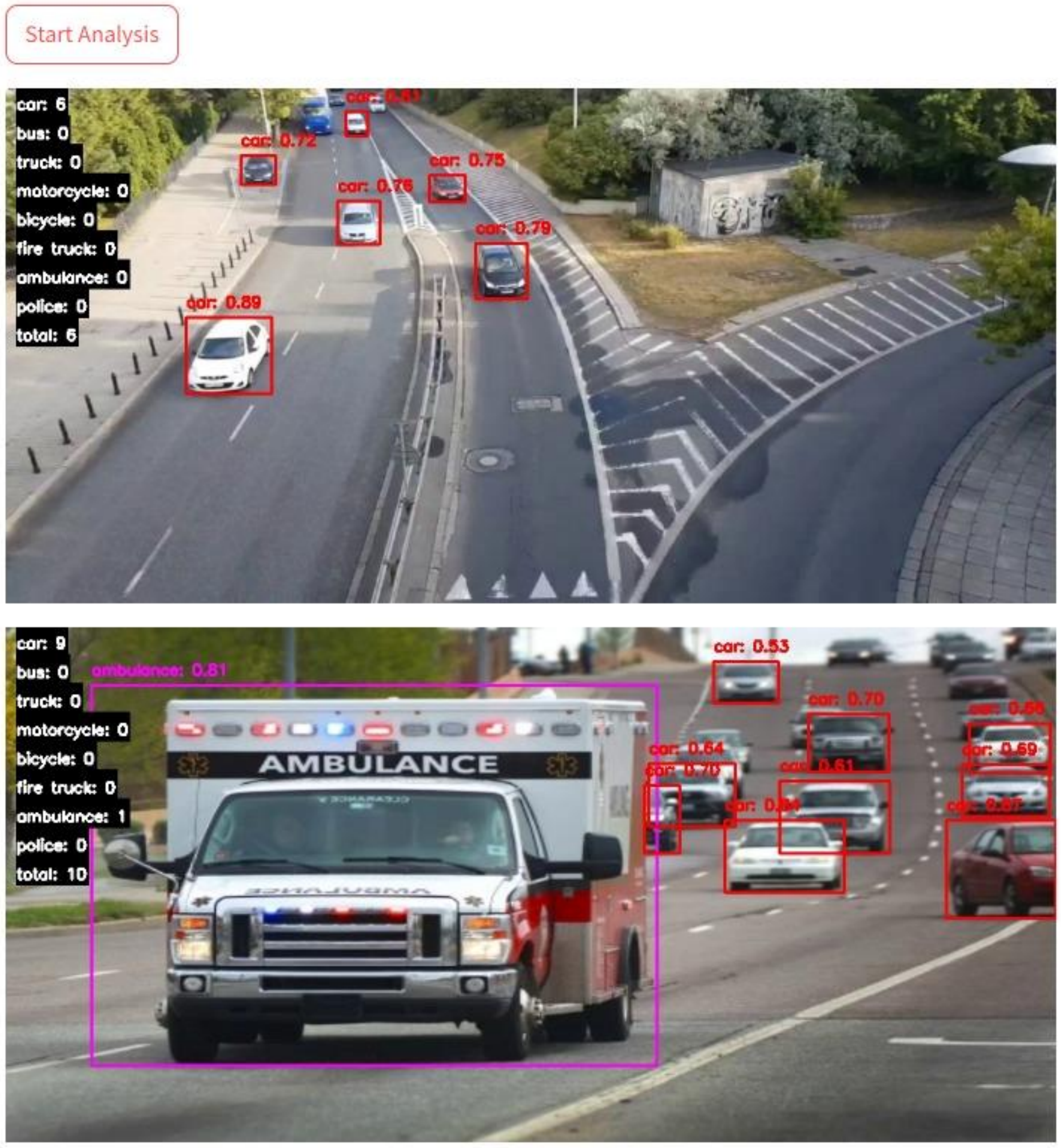
yolov8s

yolov8s model selected successfully.

Figure 15: Yolo Version Selection

In our system, I chose yolov8s which is the combination of yolo8s basic and finetuned version. Then I uploaded 4 videos to analyze in the YOLO to detect and give the proper number of all kinds of vehicles. To demonstrate, I chose KNN as the Machine Learning mode to analyze. There are two parts to us one is the calculated duration that we are getting from the database according to the selected date and time and the other part is direct streamed video analysis by

YOLO. Our total light duration is the total time of both results.



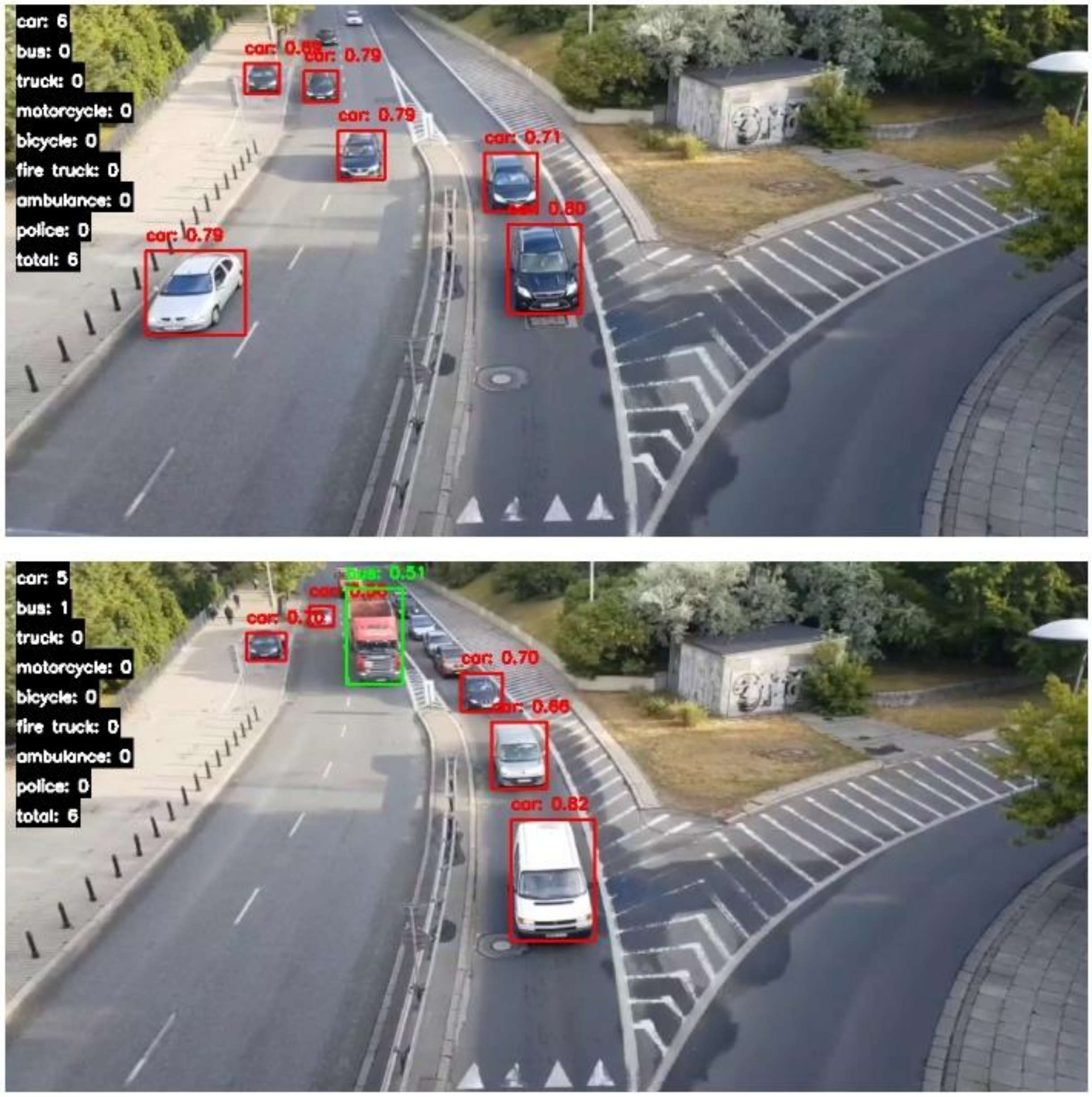
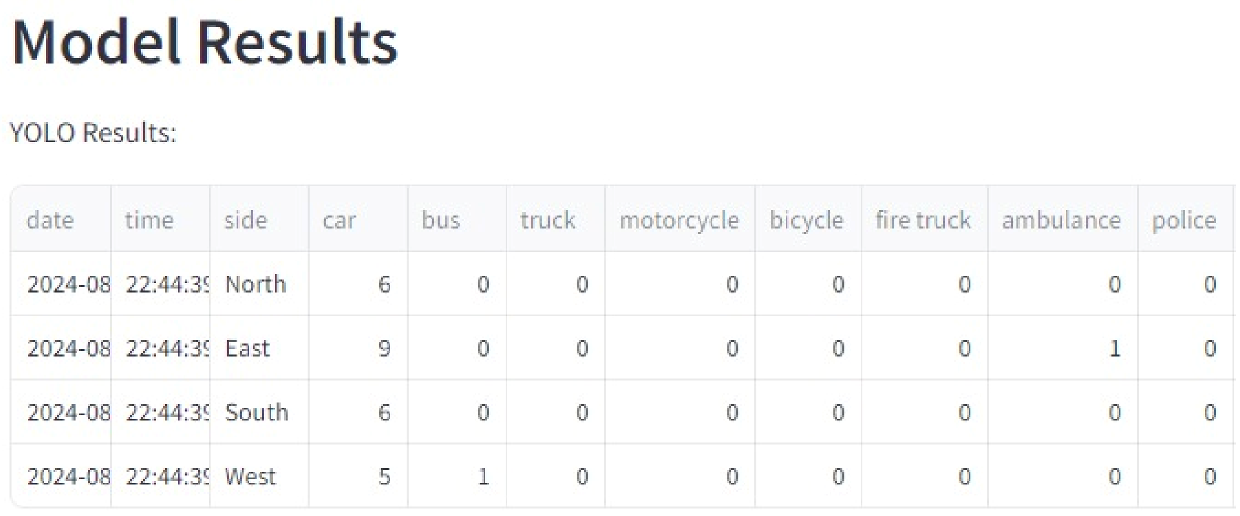


Figure 16: Result of 4 sides of vehicle Detection Result

From the picture, I can say that our system perfectly complies with identifying at least 90% of all kinds of vehicles it can detect the number of cars, buses, and emergency vehicles.



Model Results

YOLO Results:

date	time	side	car	bus	truck	motorcycle	bicycle	fire truck	ambulance	police
2024-08	22:44:3	North	6	0	0	0	0	0	0	0
2024-08	22:44:3	East	9	0	0	0	0	0	1	0
2024-08	22:44:3	South	6	0	0	0	0	0	0	0
2024-08	22:44:3	West	5	1	0	0	0	0	0	0

Table 7: YOLO model detection Result

Here it gives the number of vehicles according to the given ID. We can increase in future. However, this result is calculated with another formula where this result is weighted for real-time and impacted 80% although we can tweak this part to get better results as I and my teammates made this whole project very flexible which can be beneficial for further development.

Prospective Endeavors:

Scalability: The System may be with ease included with larger citywide visitors manage structures as I need to develop it to cover extra crossings.

Advanced Analytics: Predictive analytics can be utilized in destiny trends to estimate visitor situations and endorse preventative moves to reduce congestion.

Smart City Integration: To permit complete city optimization and control, I desire to visualize clever metropolis efforts to mix the visitor's control System with different city infrastructure structures.

End of my section

**By Melvin Divine Pritchard (19857) **

SUMO (Simulation of Urban Mobility) IMPLEMENTATION

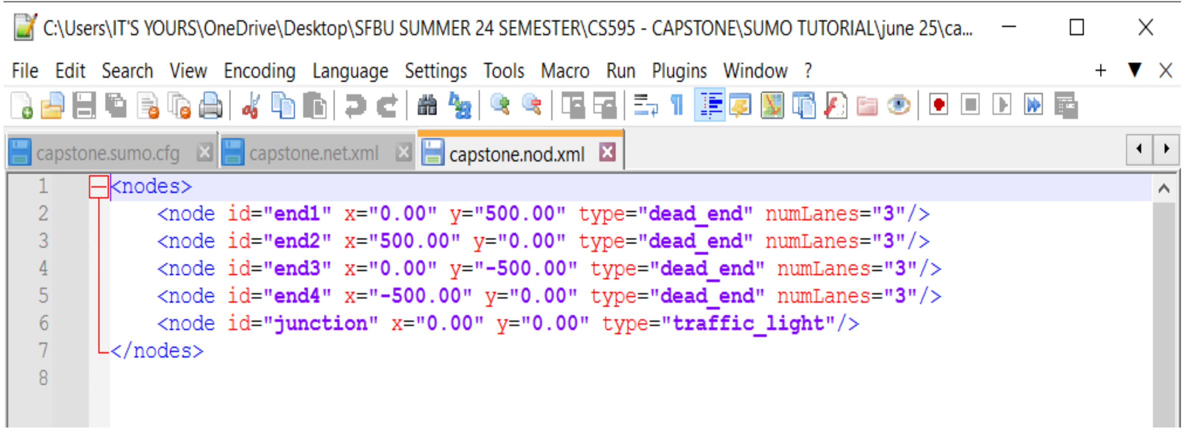
Simulation is the dynamic representation of real-world aspects. Its applications cut across in transportation engineering in a variety of ways: study, planning, training, and demonstration. Traffic simulation programs, like SUMO, are microscopic and discrete, modeling vehicle-to-vehicle interaction to come out with a simulation that would lead to the best traffic flow with the least wait time. SUMO is an open-source, multi-platform, continuous traffic simulation that can be used to replicate any real-world situation to train artificial intelligence models. It is, therefore, very useful for purposes of traffic research, modeling, and planning due to its high computational power; it can simulate environments faster than the real world.

SUMO is an integrated traffic simulation tool for large networks, providing a roadway network import function, route generation intermodal simulation, and scenario development tools; a running a simulation is possible from the command line or using the SUMO Graphical User Interface SUMO-GUI.

SUMO Implementation:

In SUMO, a network can be created in one of the following ways: manually (by hand) by writing up a traffic network in an XML (eXtensible Markup Language) file, which was utilized in this project; importing networks created in other traffic simulation applications; and using an automatic network generator to create three different types of networks.

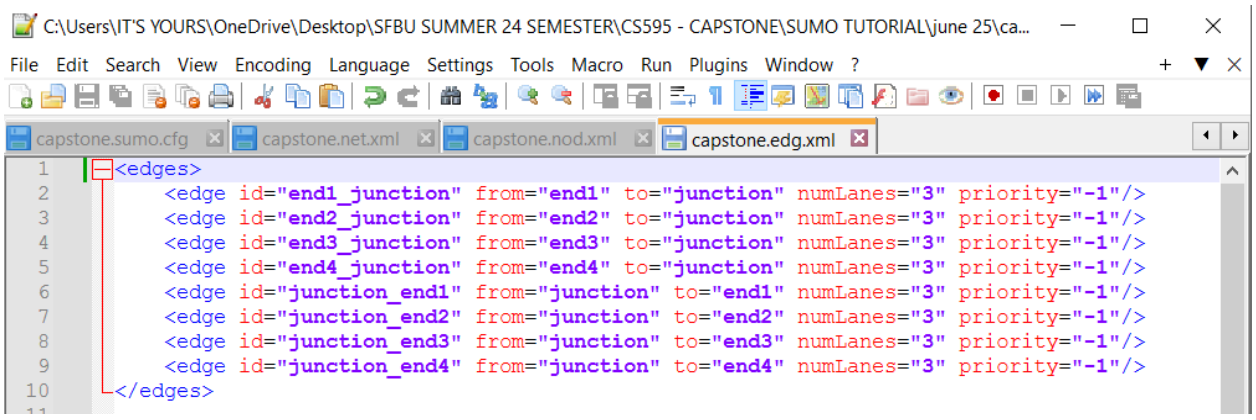
Firstly, a node file, an XML file as shown in figure -17 defines the nodes (or intersections) in the simulation's road network, was written. Each node represents a point in the network, such as an intersection, the end of a road, or any other significant location where roads meet or terminate. The node file is crucial for constructing the network topology as it defines the coordinates and types of these critical points.


C:\Users\IT'S YOURS\OneDrive\Desktop\SFBU SUMMER 24 SEMESTER\CS595 - CAPSTONE\SUMO TUTORIAL\june 25\ca...
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
capstone.sumo.cfg
capstone.net.xml
capstone.nod.xml

```
<nodes>
    <node id="end1" x="0.00" y="500.00" type="dead_end" numLanes="3"/>
    <node id="end2" x="500.00" y="0.00" type="dead_end" numLanes="3"/>
    <node id="end3" x="0.00" y="-500.00" type="dead_end" numLanes="3"/>
    <node id="end4" x="-500.00" y="0.00" type="dead_end" numLanes="3"/>
    <node id="junction" x="0.00" y="0.00" type="traffic_light"/>
</nodes>
```

Figure 17: Nodes xml file

Next, we created the edge file. An edge is a single-directed street connection between two points (junctions/nodes). An edge contains at least one lane. As shown in figure-18 The edges XML file describes the edges of a road network.


C:\Users\IT'S YOURS\OneDrive\Desktop\SFBU SUMMER 24 SEMESTER\CS595 - CAPSTONE\SUMO TUTORIAL\june 25\ca...
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
capstone.sumo.cfg
capstone.net.xml
capstone.nod.xml
capstone.edg.xml

```
<edges>
    <edge id="end1_junction" from="end1" to="junction" numLanes="3" priority="-1"/>
    <edge id="end2_junction" from="end2" to="junction" numLanes="3" priority="-1"/>
    <edge id="end3_junction" from="end3" to="junction" numLanes="3" priority="-1"/>
    <edge id="end4_junction" from="end4" to="junction" numLanes="3" priority="-1"/>
    <edge id="junction_end1" from="junction" to="end1" numLanes="3" priority="-1"/>
    <edge id="junction_end2" from="junction" to="end2" numLanes="3" priority="-1"/>
    <edge id="junction_end3" from="junction" to="end3" numLanes="3" priority="-1"/>
    <edge id="junction_end4" from="junction" to="end4" numLanes="3" priority="-1"/>
</edges>
```

Figure 18: Edges xml file

The next step was to create a network XML file. A network is the combination of junctions (nodes) and edges (streets). The network file is essential for representing the layout and characteristics of the transportation infrastructure within the simulation environment. This file is created by running the netconvert or netgenerate tool in the command line. The generated is shown in figure-19.

netconvert --node-files=capstone.nod.xml --edge-files=capstone.edg.xml --output-file=capstone.net.xml

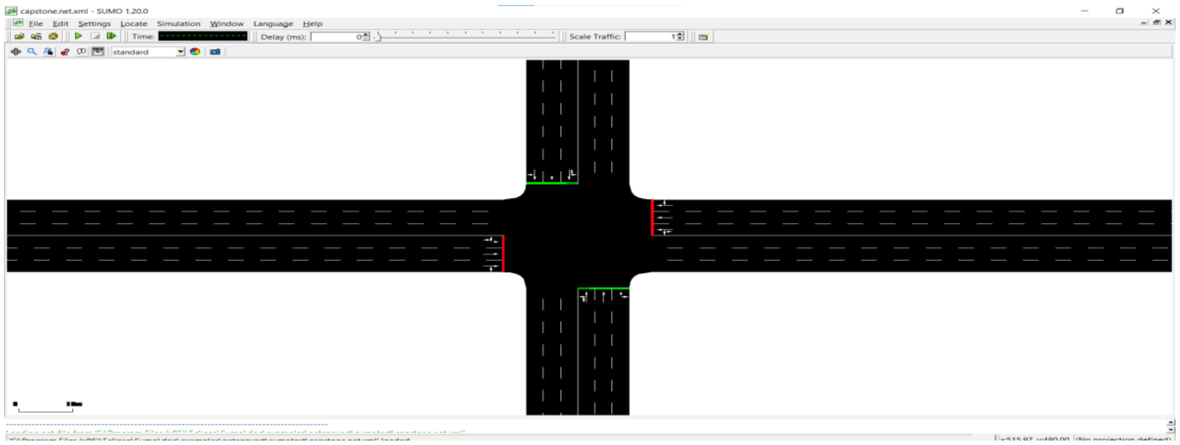
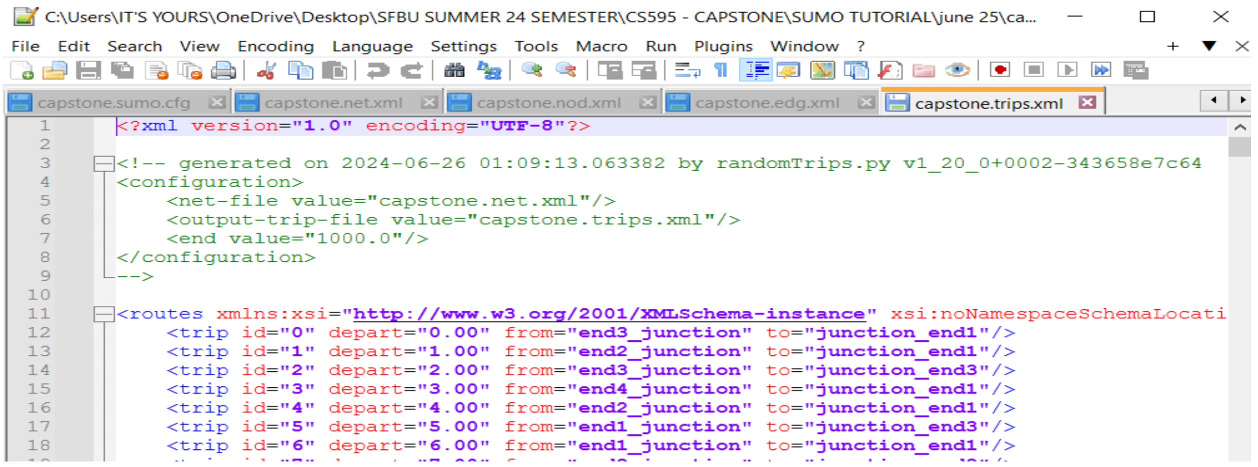


Figure 19: Generated network

Now our network is complete. However, in SUMO, to create a simulation, we need to create a routes XML file. SUMO offers a Python tool called `randomTrips.py` that generates a random set of trips for a given network. We achieved this by using the following command: The output trips also shown in figure-20

randomTrips.py -n capstone.net.xml -e 1000 -o capstone.trips.xml

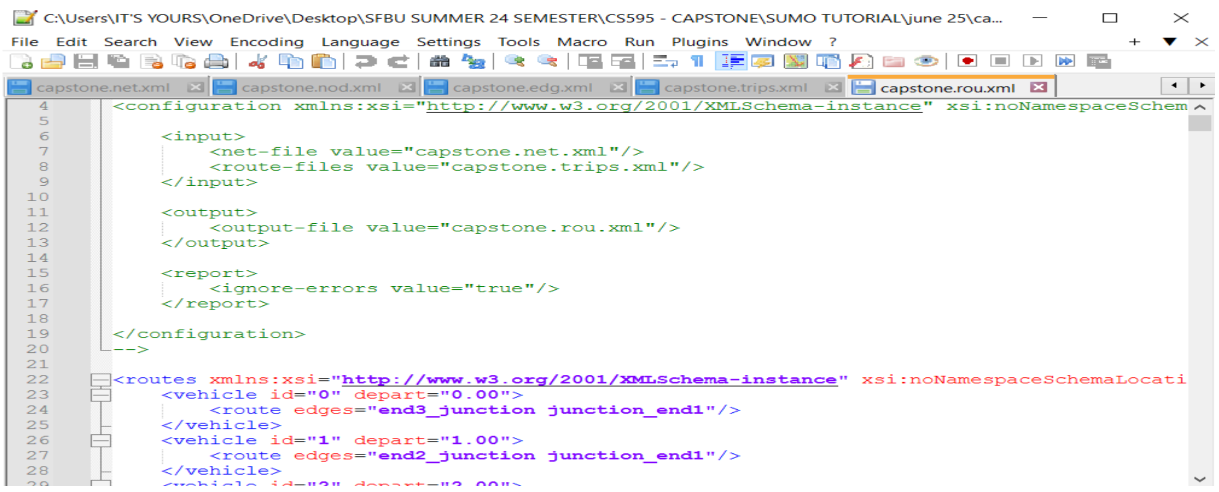

C:\Users\IT'S YOURS\OneDrive\Desktop\SFBU SUMMER 24 SEMESTER\CS595 - CAPSTONE\SUMO TUTORIAL\june 25\ca...
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
capstone.sumo.cfg
capstone.net.xml
capstone.nod.xml
capstone.edg.xml
capstone.trips.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- generated on 2024-06-26 01:09:13.063382 by randomTrips.py v1_20_0+0002-343658e7c64
<configuration>
    <net-file value="capstone.net.xml"/>
    <output-trip-file value="capstone.trips.xml"/>
    <end value="1000.0"/>
</configuration>
-->

<routes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocati
    <trip id="0" depart="0.00" from="end3_junction" to="junction_end1"/>
    <trip id="1" depart="1.00" from="end2_junction" to="junction_end1"/>
    <trip id="2" depart="2.00" from="end3_junction" to="junction_end3"/>
    <trip id="3" depart="3.00" from="end4_junction" to="junction_end1"/>
    <trip id="4" depart="4.00" from="end2_junction" to="junction_end1"/>
    <trip id="5" depart="5.00" from="end1_junction" to="junction_end3"/>
    <trip id="6" depart="6.00" from="end1_junction" to="junction_end1"/>
```

Figure 20: Generated Trips

We converted these trips into routes as shown in figure-21, using the SUMO tool called `duarouter`. This command generates the route file that contains information about our vehicles' points of departure and the edges they are traveling on:

duarouter -n capstone.net.xml --route-files=capstone.trips.xml -o capstone.rou.xml

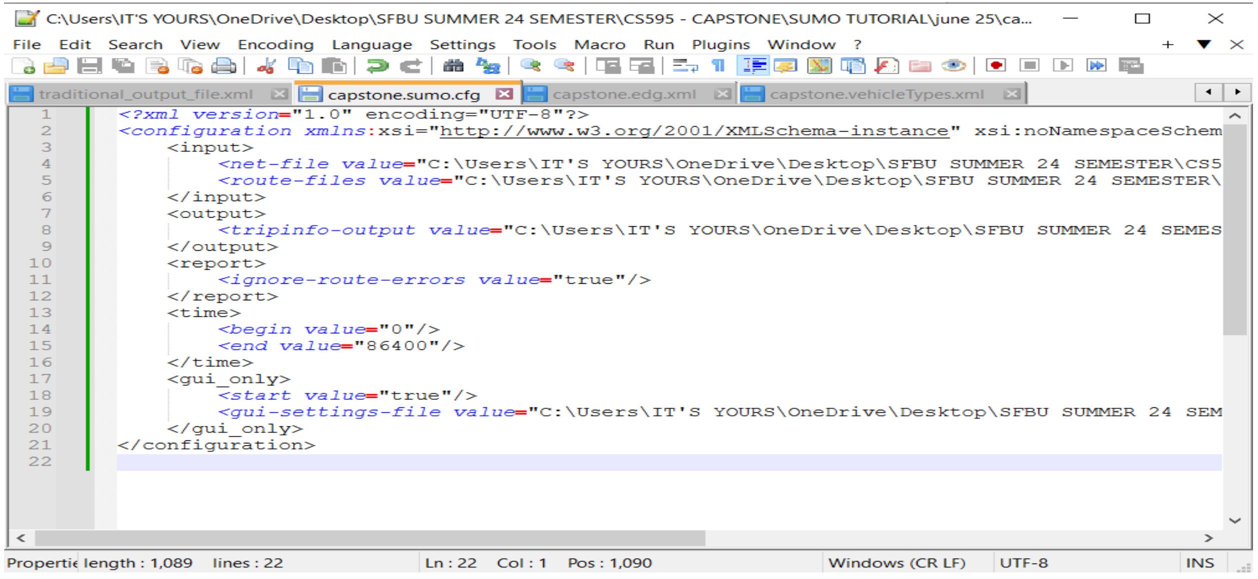

C:\Users\IT'S YOURS\OneDrive\Desktop\SFBU SUMMER 24 SEMESTER\CS595 - CAPSTONE\SUMO TUTORIAL\june 25\ca...

capstone.net.xml | capstone.nod.xml | capstone.edg.xml | capstone.trips.xml | capstone.rou.xml

```
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchem

    <input>
        <net-file value="capstone.net.xml"/>
        <route-files value="capstone.trips.xml"/>
    </input>

    <output>
        <output-file value="capstone.rou.xml"/>
    </output>

    <report>
        <ignore-errors value="true"/>
    </report>

</configuration>
-->

<routes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocati
    <vehicle id="0" depart="0.00">
        <route edges="end3_junction junction_end1"/>
    </vehicle>
    <vehicle id="1" depart="1.00">
        <route edges="end2_junction junction_end1"/>
    </vehicle>
```

Figure 21: Generated Routes

Finally, we used the 'capstone.rou.xml' and 'capstone.net.xml' files to create our SUMO configuration file, 'capstone.sumo.cfg', which is represented in the image below in figure-22.


C:\Users\IT'S YOURS\OneDrive\Desktop\SFBU SUMMER 24 SEMESTER\CS595 - CAPSTONE\SUMO TUTORIAL\june 25\ca...

traditional_output_file.xml | capstone.sumo.cfg | capstone.edg.xml | capstone.vehicleTypes.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchem
    <input>
        <net-file value="C:\Users\IT'S YOURS\OneDrive\Desktop\SFBU SUMMER 24 SEMESTER\CS5
        <route-files value="C:\Users\IT'S YOURS\OneDrive\Desktop\SFBU SUMMER 24 SEMESTER\
    </input>
    <output>
        <tripinfo-output value="C:\Users\IT'S YOURS\OneDrive\Desktop\SFBU SUMMER 24 SEMES
    </output>
    <report>
        <ignore-route-errors value="true"/>
    </report>
    <time>
        <begin value="0"/>
        <end value="86400"/>
    </time>
    <gui_only>
        <start value="true"/>
        <gui-settings-file value="C:\Users\IT'S YOURS\OneDrive\Desktop\SFBU SUMMER 24 SEM
    </gui_only>
</configuration>
```

Propertie length : 1,089 lines : 22 Ln : 22 Col : 1 Pos : 1,090 Windows (CR LF) UTF-8 INS

Figure 22: CFG file

The SUMO GUI was used to run the configuration (cfg file), which created the trip information. This information would be converted to a comma-separated value (CSV) file, extracted, and used to train our model as shown in figure-23 and the output file as in figure-24.

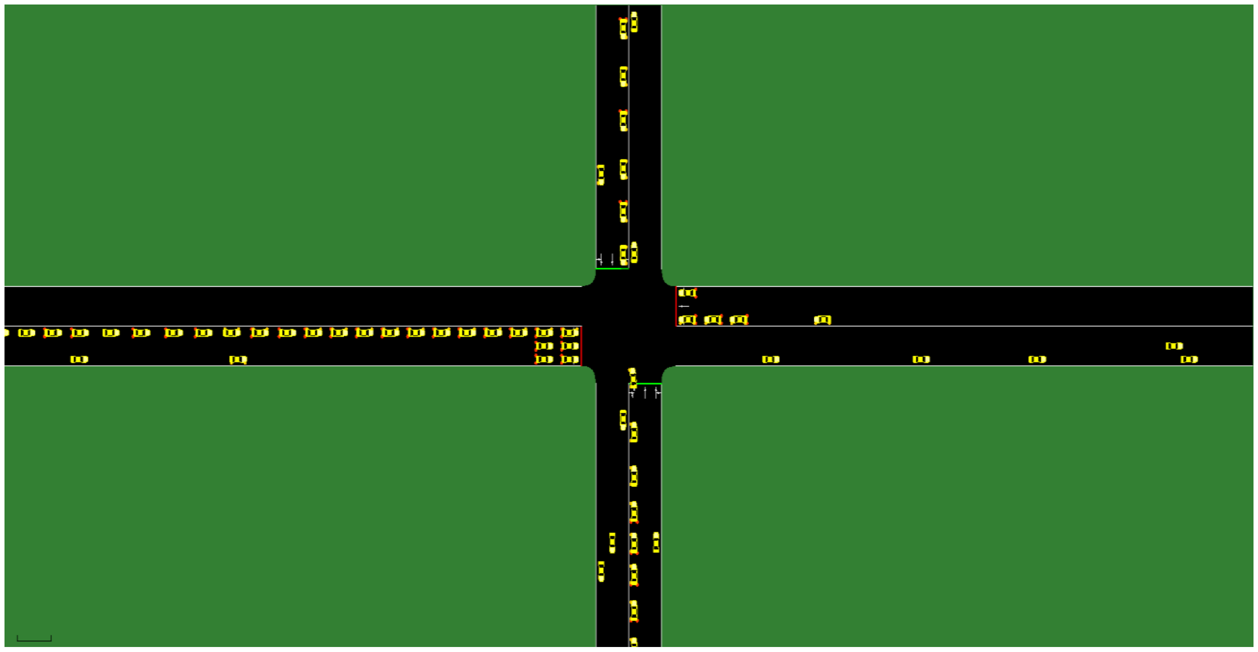
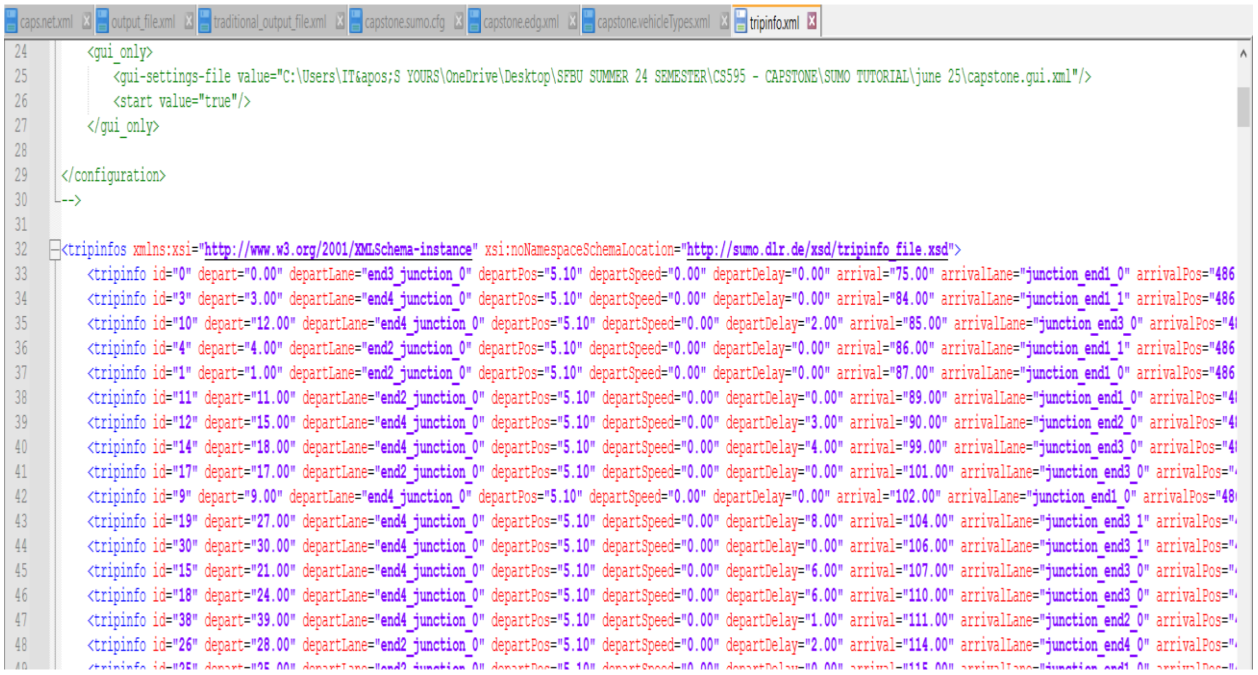


Figure 23. Simulation



```
    <gui_only>
        <gui-settings-file value="C:\Users\IT&apos;S YOURS\OneDrive\Desktop\SFBU SUMMER 24 SEMESTER\CS595 - CAPSTONE\SUMO TUTORIAL\june 25\capstone.gui.xml"/>
        <start value="true"/>
    </gui_only>

</configuration>
-->

<tripinfos xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/tripinfo_file.xsd">
    <tripinfo id="0" depart="0.00" departLane="end3_junction_0" departPos="5.10" departSpeed="0.00" departDelay="0.00" arrival="75.00" arrivalLane="junction_end1_0" arrivalPos="486
    <tripinfo id="3" depart="3.00" departLane="end4_junction_0" departPos="5.10" departSpeed="0.00" departDelay="0.00" arrival="84.00" arrivalLane="junction_end1_1" arrivalPos="486
    <tripinfo id="10" depart="12.00" departLane="end4_junction_0" departPos="5.10" departSpeed="0.00" departDelay="2.00" arrival="85.00" arrivalLane="junction_end3_0" arrivalPos="4
    <tripinfo id="4" depart="4.00" departLane="end2_junction_0" departPos="5.10" departSpeed="0.00" departDelay="0.00" arrival="86.00" arrivalLane="junction_end1_1" arrivalPos="486
    <tripinfo id="1" depart="1.00" departLane="end2_junction_0" departPos="5.10" departSpeed="0.00" departDelay="0.00" arrival="87.00" arrivalLane="junction_end1_0" arrivalPos="486
    <tripinfo id="11" depart="11.00" departLane="end2_junction_0" departPos="5.10" departSpeed="0.00" departDelay="0.00" arrival="89.00" arrivalLane="junction_end1_0" arrivalPos="4
    <tripinfo id="12" depart="15.00" departLane="end4_junction_0" departPos="5.10" departSpeed="0.00" departDelay="3.00" arrival="90.00" arrivalLane="junction_end2_0" arrivalPos="4
    <tripinfo id="14" depart="18.00" departLane="end4_junction_0" departPos="5.10" departSpeed="0.00" departDelay="4.00" arrival="99.00" arrivalLane="junction_end3_0" arrivalPos="4
    <tripinfo id="17" depart="17.00" departLane="end2_junction_0" departPos="5.10" departSpeed="0.00" departDelay="0.00" arrival="101.00" arrivalLane="junction_end3_0" arrivalPos="
    <tripinfo id="9" depart="9.00" departLane="end4_junction_0" departPos="5.10" departSpeed="0.00" departDelay="0.00" arrival="102.00" arrivalLane="junction_end1_0" arrivalPos="48
    <tripinfo id="19" depart="27.00" departLane="end4_junction_0" departPos="5.10" departSpeed="0.00" departDelay="8.00" arrival="104.00" arrivalLane="junction_end3_1" arrivalPos="
    <tripinfo id="30" depart="30.00" departLane="end4_junction_0" departPos="5.10" departSpeed="0.00" departDelay="0.00" arrival="106.00" arrivalLane="junction_end3_1" arrivalPos="
    <tripinfo id="15" depart="21.00" departLane="end4_junction_0" departPos="5.10" departSpeed="0.00" departDelay="6.00" arrival="107.00" arrivalLane="junction_end3_0" arrivalPos="
    <tripinfo id="18" depart="24.00" departLane="end4_junction_0" departPos="5.10" departSpeed="0.00" departDelay="6.00" arrival="110.00" arrivalLane="junction_end3_0" arrivalPos="
    <tripinfo id="38" depart="39.00" departLane="end4_junction_0" departPos="5.10" departSpeed="0.00" departDelay="1.00" arrival="111.00" arrivalLane="junction_end2_0" arrivalPos="
    <tripinfo id="26" depart="28.00" departLane="end2_junction_0" departPos="5.10" departSpeed="0.00" departDelay="2.00" arrival="114.00" arrivalLane="junction_end4_0" arrivalPos="
```

Figure 24: Output file

AI Predictive Models Training

After the SUMO simulation had run successfully, by which we would be able to model and generate traffic data in several scenarios, it was time to use those data to predict future traffic. My database colleague and I focused our efforts on training our predictive AI model to give an approximation about the number and kind of vehicles in a specific intersection.

The main objective was to make accurate predictions of traffic at specific dates and times using the historical data generated by SUMO in the past six months, detailing the types and numbers of vehicles at each side of the intersection on an hourly basis.

Model Selection and Research:

In search of the best algorithms that go well with AI predictions, several research on these models were conducted: Linear Regression, Prophet, SVR, ARIMA, and Random Forest.

After a huge testing, some models with bad performances were removed, others introduced based on how good their performance metrics are. The final models selected to form part of our system were:

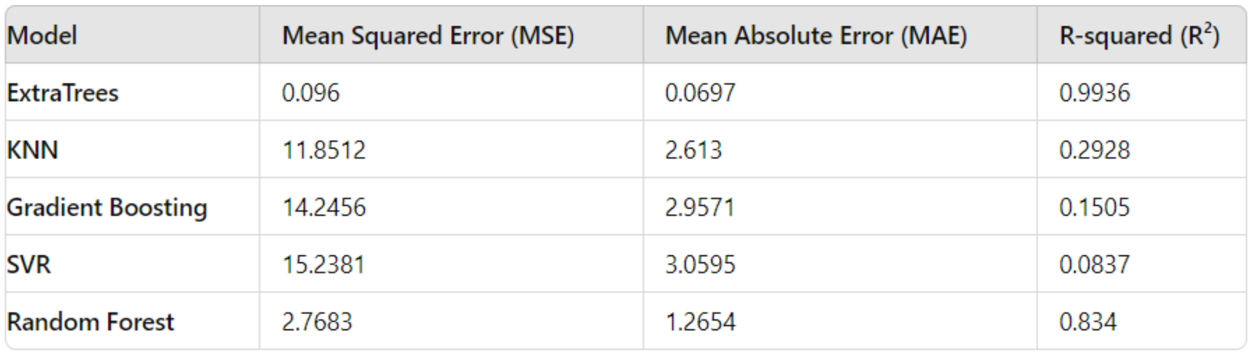
- **ExtraTrees:** Very randomised trees which build a lot of trees from randomly selected training data.
- **Random Forest:** A bunch of decision trees combining themselves to improve forecast accuracy.
- **SVR:** Works well for small to medium-sized datasets. Provides a reliable framework for making predictions.
- **Gradient Boosting:** Combines models which are relatively weak into one for making strong predictions.
- **KNN:** Predicts the value of a point based on how most of its closest neighbors are.

Performance metrics of models:

Models Performance Metrics:

The performance of a model will be measured using three basic metrics: mean squared error, mean absolute error, and R-squared. MSE computes the average of squared differences between predicted and actual values, the lower the value, the more accurate your prediction. This metric is very sensitive to larger mistakes, so it's great for catching major differences between projected and actual results.

On the other hand, MAE is simply the average of absolute differences between predicted and measured values. Because it has the same units as the data and the lower the MAE value is the better the model, it provides a more direct and intuitive measure of prediction accuracy. At last, R^2 range from 0 to 1, and it indicates how much of the variance of the dependent variable the model explains. High R^2 values indicate that the expected and actual values are very well correlated. All these measures together provide a complete evaluation of how accurate and effective our model is. All our final models have been trained and tested on historical data, and the table below shows performance metrics summarizing each of our models. This will bring out vividly the comparisons of how each model performs over these metrics: MSE, MAE, and R2.



Model	Mean Squared Error (MSE)	Mean Absolute Error (MAE)	R-squared (R^2)
ExtraTrees	0.096	0.0697	0.9936
KNN	11.8512	2.613	0.2928
Gradient Boosting	14.2456	2.9571	0.1505
SVR	15.2381	3.0595	0.0837
Random Forest	2.7683	1.2654	0.834

Table 8: Models Performance Metrics

Comparative Analysis:

The comparative analysis is based on the performance metrics for each of our model:

- **ExtraTrees**: **(MSE: 0.096, MAE: 0.0697, R^2: 0.9936):**
  - **Performance**: ExtraTrees has outstanding performance with the smallest MSE and MAE values; it provides very accurate predictions. The R^2 value of 0.9936 indicates that the model explains about 99.36% of the variance in the target variable, which is excellent.
- **KNN: (MSE: 11.8512, MAE: 2.613, R^2: 0.2928):**

    - **Performance:** In comparison with the ExtraTrees algorithm, KNN has a much higher MSE and MAE, hence less accurate predictions. The R² value is only 0.2928, thus showing that the model explains only 29.28% of variance, which is rather low.

- **Gradient Boosting**: **(14.2456 – MSE, 2.9571 – MAE, 0.1505 - R²)**
    - **Performance**: Gradient Boosting also has high error rates, with an MSE of 14.2456 and MAE of 2.9571. Also, its R² value is very low at 0.1505, which means it explains only 15.05% of the variance.
- **SVR**: MSE: (**15.2381, MAE: 3.0595, R²: 0.0837)**
    - **Performance**: SVR has the highest MSE and MAE, thus showing the worst performance among the models. The R² value is as low as 0.0837, explaining only 8.37% of variance.
- **Random Forest**: **(MSE: 2.7683, MAE: 1.2654, R²: 0.834)**
    - **Performance**—Random Forest: Random Forest does fairly well with an MSE of 2.7683 and MAE of 1.2654. The R² value of 0.834 says that it explains about 83.4% of the variance, which is strong but still significantly lower than ExtraTrees.

We chose ExtraTrees as the best model to go with in our predictive tasks because it showed the lowest rates of mean squared and mean absolute errors and the highest value of R², hence making it most reliable and accurate for modeling and prediction of the number and types of cars standing at intersections in our ITMS project.

Training Process of the ExtraTrees

Data Fetching:

The system's training data was fetched from the database that SUMO had generated; this database consisted of information relating types and numbers of vehicles at intersections over a six-month period. With the fetched data, multiple decision trees were constructed.

Tree Building:

At each split within each tree, the ExtraTrees algorithm randomly selected a subset of features and determined the best cut points for splitting the data. This randomness helps to reduce variance and improve model generalization.

Prediction:

Each tree in the ensemble predicted how many vehicles would be present at an intersection and their types based on its learned structure. The final prediction is robust and accurate, because it is the average of all these predictions and can be better than any single tree can provide.

Evaluation:

The performance of the model was evaluated with the help of the three following metrics Mean Squared Error MSE, Mean Absolute Error MAE, R-squared R^2.

Working principles of Vehicles prediction:

Input Mechanism:
The system automatically displays the current date and time. However, the operator can manually select a prior specific date and time. This selected date and time serve as the primary input for our trained ExtraTrees model.

Model Training and Prediction:

ExtraTrees Model:
The model's input is the date and time the user selected. Then, it will predict the number and kinds of vehicles using the generated data by SUMO, which is stored in our database. As previously mentioned, the training process builds several decision trees, chooses the best features and cut points, and averages all trees' prediction to produce the final output. This whole process is shown in figure 25 below.

The output for the ExtraTrees model will include the number and types of vehicles expected to pass for that date and time.

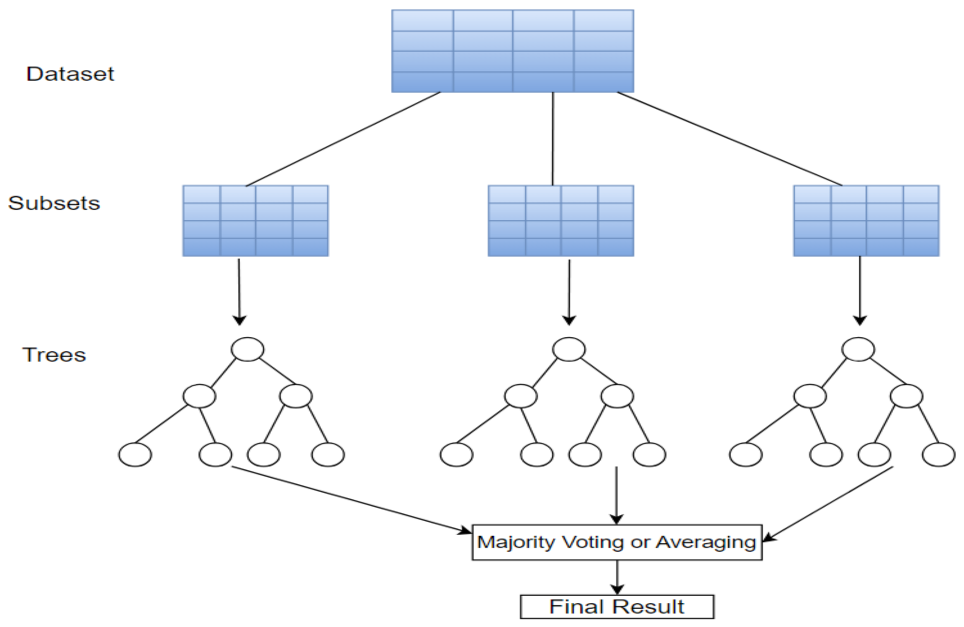


Figure 25: ExtraTrees Model

Source: Random Forest vs. Extremely Randomized Trees, Baeldung.

YOLO (You Only Look Once):

YOLO is a real-time object detection system that processes live video feeds uploaded by the system operator. It identifies and counts the number and types of vehicles present in the video feed. YOLO's output includes a detailed count of each vehicle type detected in the live video, providing a real-time analysis of traffic conditions.

Application of Coefficient

In our ITMS, we will use two different coefficients to adjust the inferences that come as output from the AI model we have trained and the YOLO object detection system:

These coefficients are applied both to the ExtraTrees model predictions and the live video output from YOLO. These coefficients are to scale the results to make allowances for any differences there might be between the simulation environment and the real world. This assures that the final prediction gets close to reality with respect to the traffic conditions and thereby produces an accurate representation of the number and kinds of vehicles.

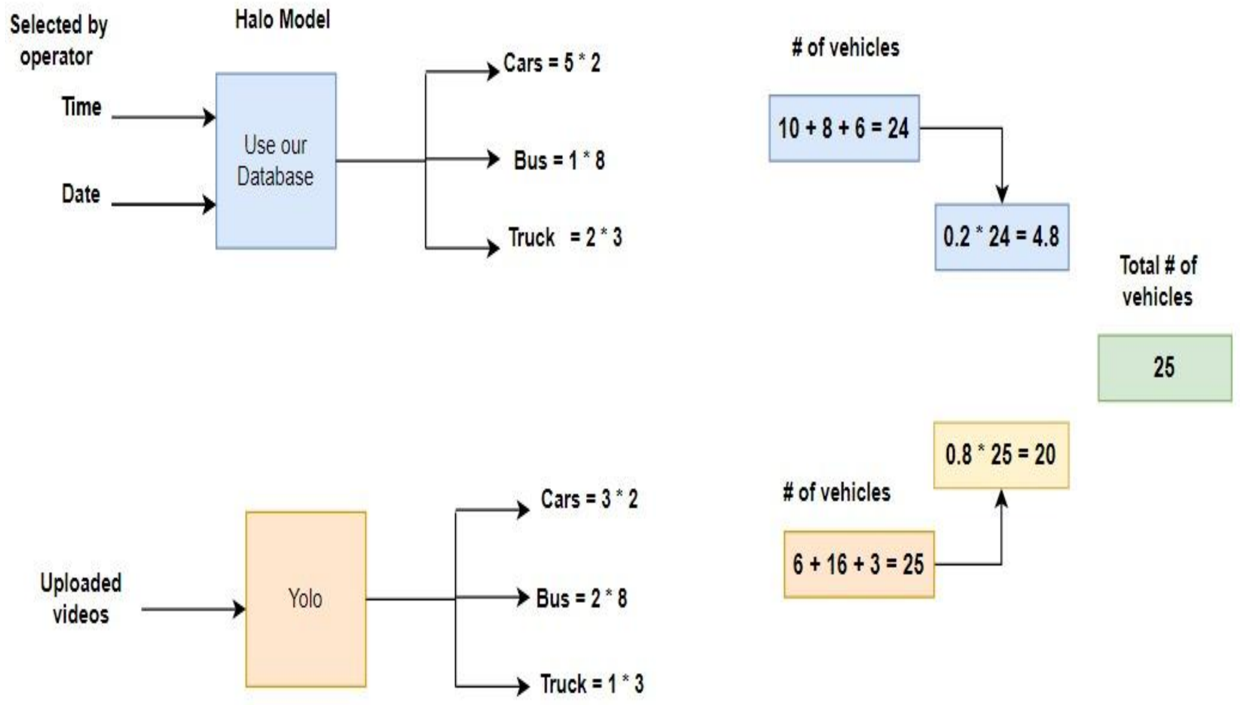


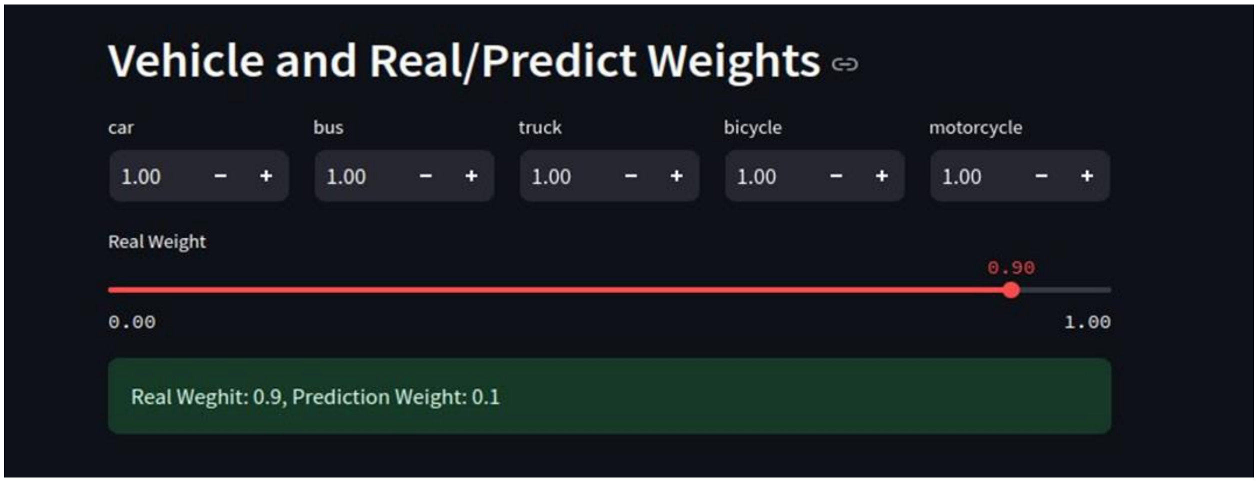
Figure 26: Overview of model training and vehicles predictions

These numbers, after being returned by the ExtraTrees model and YOLO, are multiplied by the coefficient to further fine-tune the predictions.

This is an important aspect of the system, which will ensure that the system's predictions not only rely on the historical and real-time data but are also calibrated to be as accurate as can be in practicality for use in traffic management.

First Coefficient: Vehicle Type Adjustment

This coefficient is applied on the total number of vehicles predicted by the trained model and YLO. The coefficient aims at taking into consideration the variation in occupation of the different kinds of vehicles, just like HOV lanes. For example, a bus, having more passengers, has a higher coefficient than a bike. This coefficient is equal for both the trained model and YOLO. This ensures that the two sources of prediction get the same weight. But all coefficients can be kept the same, as seen in figure 27, if the traffic requires that.


Vehicle and Real/Predict Weights

car 1.00
bus 1.00
truck 1.00
bicycle 1.00
motorcycle 1.00

Real Weight 0.90

0.00 1.00

Real Weghit: 0.9, Prediction Weight: 0.1

Figure 27: Coefficients

Second Coefficient: Model Confidence Weighting

Now that we have the total number of vehicles from both the trained model and YOLO, we apply the second coefficient. The second coefficient will be our overall confidence in the accuracy of each model's prediction. Since YOLO is processing live video feeds, we can give it a higher coefficient value, say 0.8 as seen in figure 26; compared to the coefficient of the trained model, say 0.2. The system has been designed so that the sum of these coefficients always comes to 1. If one is set manually, another will automatically adjust to keep the sum at 1.

Combining Coefficients for Final Prediction

Now, these vehicle counts, after adjustments from YOLO and our trained model, are multiplied by their coefficients. Then the products are summed to get our final predicted total number of vehicles for that time and date at whichever direction our operator chose earlier: east, north, south, or west.

Application of the Solution to Traffic Light Timing

Finally, the aggregate count of vehicles forecasted by the model is fed into the model, which will then set a proper timing schedule for the traffic lights. In that way, one will ensure that the system of traffic management dynamically adjusts in line with real-time conditions and historical data to efficiently manage the traffic flow, ensuring that congestion is kept at a minimum.

Section Conclusion

In this regard, SUMO simulation-based development and application with advanced machine learning models for our Intelligent Traffic Management systems show vast potential in the improvement of urban traffic management. Our ITMS system has effectively combined historical data with real-time input to come up with vehicle counts at intersections, hence assuring smoother traffic and reduced congestion.

It predicts the number and types of vehicles to be present at any time and date using models like ExtraTrees, KNN, Gradient Boosting, SVR, and Random Forest. Being integrated with SUMO for microscopic simulation of traffic and with real-time object detection, YOLO is bound into the system, hence a robust solution for deployment in the real world. Due to this, it can adjust the traffic light timings according to the forecasted traffic conditions to provide a better flow of traffic, reduce delays, and lower emissions, thus directly improving urban life.

End of the section

By Md Boktiar Hossain(19795)

Calculating traffic light duration:

The outputs generated by YOLO and Trained model will be integrated into the AI model to derive a precise traffic light duration using the regression algorithm.

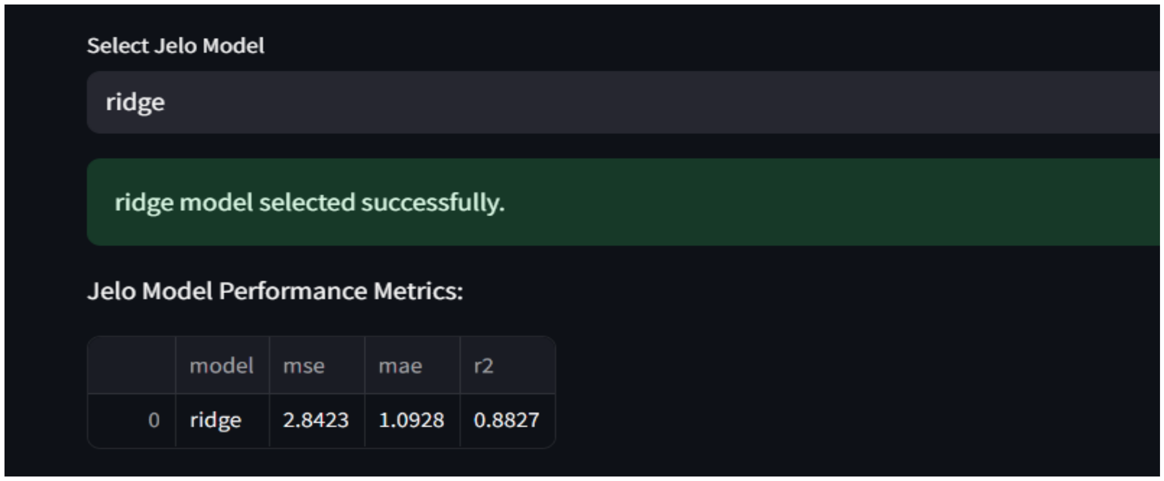
Selection of ML algorithm through rigorous testing for traffic light duration:

To implement an AI-based traffic light duration control system, various regression algorithms are tested. We tested the performance of Ridge, ElasticNet, Lasso, Bayesian Ridge, and Decision Tree Regression using specific performance metrics: Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared (R^2). Based on the user video input, a detailed performance testing and analysis are performed.

The performances were tested for each algorithm one by one:

Ridge Regression

Ridge Regression is an algorithm that uses L2 regularization. It can limit the sum of the squared coefficients. This helps reduce the chance of overfitting by keeping the coefficients small. This method is helpful when you're dealing with several correlated features because it balances bias and variance. This balance often leads to more reliable predictions. In this specific case, Ridge Regression gave a Mean Squared Error (MSE) of 2.8423, a Mean Absolute Error (MAE) of 1.0928, and an R-squared value of 0.8827, as shown in Figure 28. Even though it works well, Ridge Regression assumes there's a straight-line relationship between the features and the target variable. This might not always be true, especially when dealing with complex, non-linear traffic patterns.



Select Jelo Model: ridge

ridge model selected successfully.

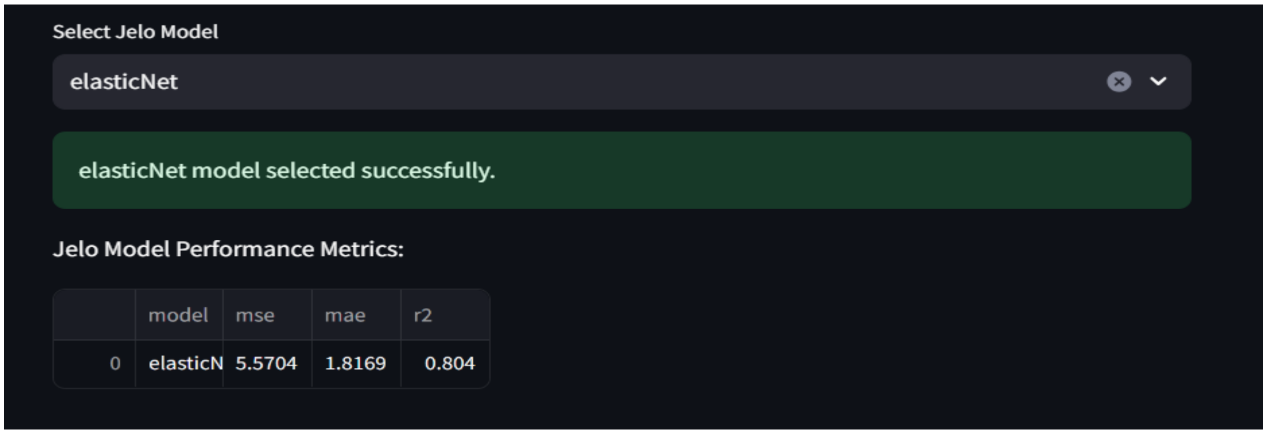
Jelo Model Performance Metrics:

	model	mse	mae	r2
0	ridge	2.8423	1.0928	0.8827

Figure 28: Ridge Regression Algorithm Test data

ElasticNet Regression

ElasticNet Regression generated a Mean Squared Error (MSE) of 2.4453, a Mean Absolute Error (MAE) of 1.8131, and an R-squared value of 0.98720. These numbers, as shown in Figure 29, suggest that ElasticNet is pretty accurate and comes close to more complex models like the Decision Tree. Even though its error rates are a bit higher than those of the Decision Tree, ElasticNet is still a solid choice because it does a good job balancing bias and variance. Overall, while ElasticNet might not beat the Decision Tree in pure predictive power, ElasticNet is a reliable and interpretable model, especially when dealing with multi-collinearity issues.


Select Jelo Model

elasticNet

elasticNet model selected successfully.

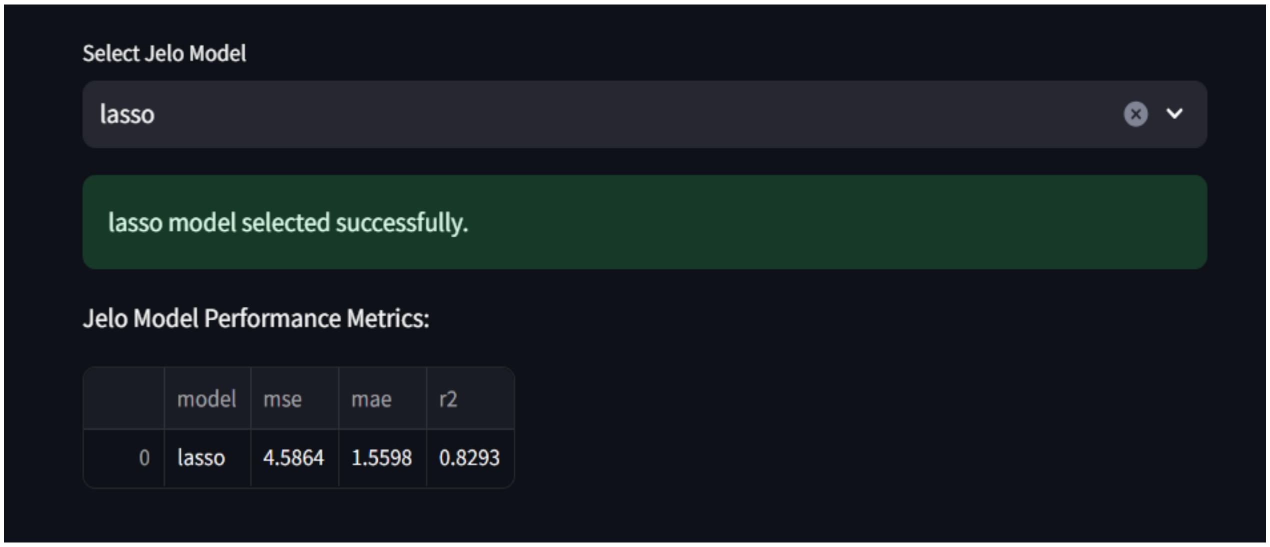
Jelo Model Performance Metrics:

	model	mse	mae	r2
0	elasticN	5.5704	1.8169	0.804

Figure 29: elastiNet Algorithm Test data

Lasso Regression

Lasso Regression uses L1 regularization. Lasso Regression can reduce some coefficients to zero, essentially picking out the most important features. This makes it really useful for data with lots of features, especially when only a few of them are useful. However, it may not work as well when the features are highly correlated. In this case, as shown in Figure 30, Lasso Regression gave an MSE of 4.5864, an MAE of 1.5998, and an R-squared value of 0.8293. This values show that Lasso did not perform as well as the other algorithms, with higher error rates and a lower R-squared value, indicating it might not be capturing the traffic data patterns as effectively.


Select Jelo Model

lasso

lasso model selected successfully.

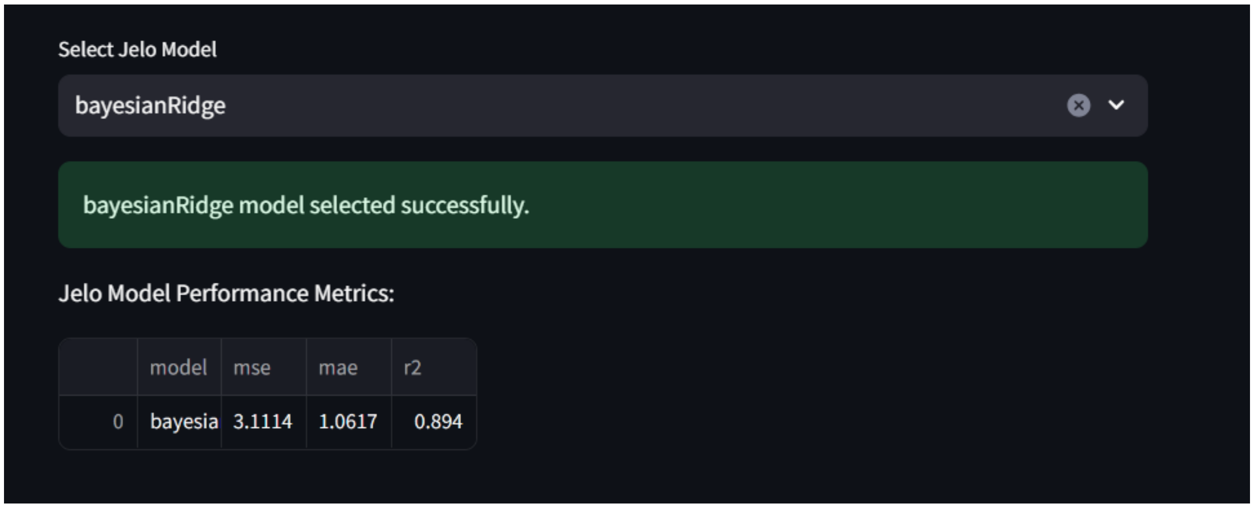
Jelo Model Performance Metrics:

	model	mse	mae	r2
0	lasso	4.5864	1.5598	0.8293

Figure 30: Lasso Regression algorithm test data

Bayesian Ridge

Bayesian Ridge Regression uses ideas from Bayesian statistics to give a view of the coefficients based on probability. It also helps with handling multi-collinearity. This approach is helpful when there is some knowledge about how the coefficients should look. However, Bayesian Ridge can take a lot of time to run and needs you to set up prior distributions, which can be tricky. In this case, as shown in Figure 31, Bayesian Ridge Regression had an MSE of 3.1114, an MAE of 1.0617, and an R-squared value of 0.8940. These results show that Bayesian Ridge did better than Lasso, but not as well as Ridge or Decision Tree in this example.



bayesianRidge model selected successfully.

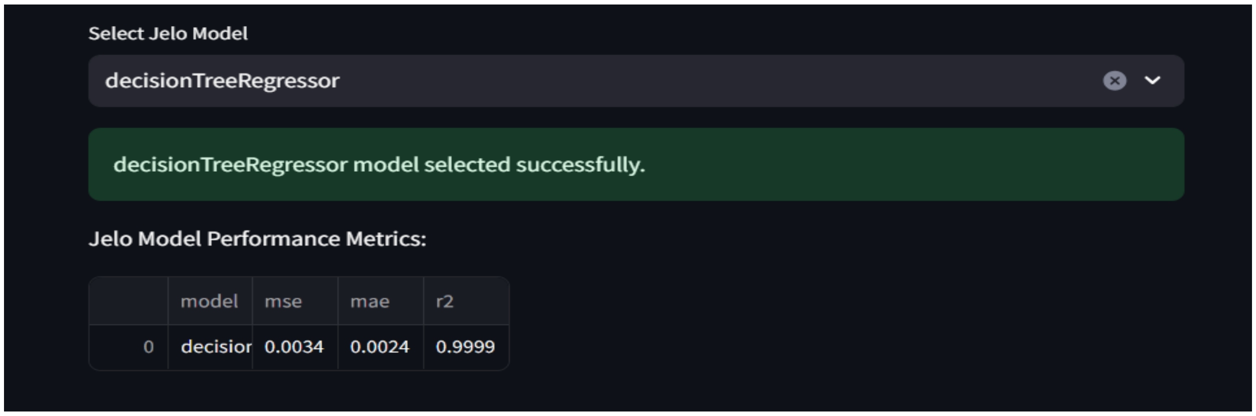
Jelo Model Performance Metrics:

	model	mse	mae	r2
0	bayesia	3.1114	1.0617	0.894

Figure 31: Bayesian Ridge Algorithm Test data

Decision Tree Regression

The last one is Decision Tree Regression. This model is non-linear and non-parametric. It can work with both numbers and categories without needing much extra work beforehand. It's good at handling complicated patterns, making it a good choice for studying traffic data. These models are also easy to understand because they use clear decision rules. As shown in Figure 32, it did better than all the other models, with an MSE of 0.0034, an MAE of 0.0024, and an R-squared value of 0.9999. These results show that the model fits the data almost perfectly, showing how well it can capture the relationships in the data. However, Decision Trees can sometimes fit the data too closely, but this can be fixed by cutting back the tree, using methods like Random Forests, or limiting how deep the tree can go.


Select Jelo Model

decisionTreeRegressor

decisionTreeRegressor model selected successfully.

Jelo Model Performance Metrics:

	model	mse	mae	r2
0	decisior	0.0034	0.0024	0.9999

Figure 32: DecisionTreeRegressor Algorithm Test data

The performance matrix, presented on a comparable basis, is illustrated in the following table:

Algorithm	Mean Squared Error (MSE)	Mean Absolute Error(MAE)	R² (R-Squared)
Ridge	2.8423	1.0928	0.8827
decisionTree	0.0034	0.0024	0.9999
Lasso	4.5864	1.5998	0.8293
bayesianRidge	3.1114	1.0617	0.894
elastiNet	5.5704	1.8169	0.804

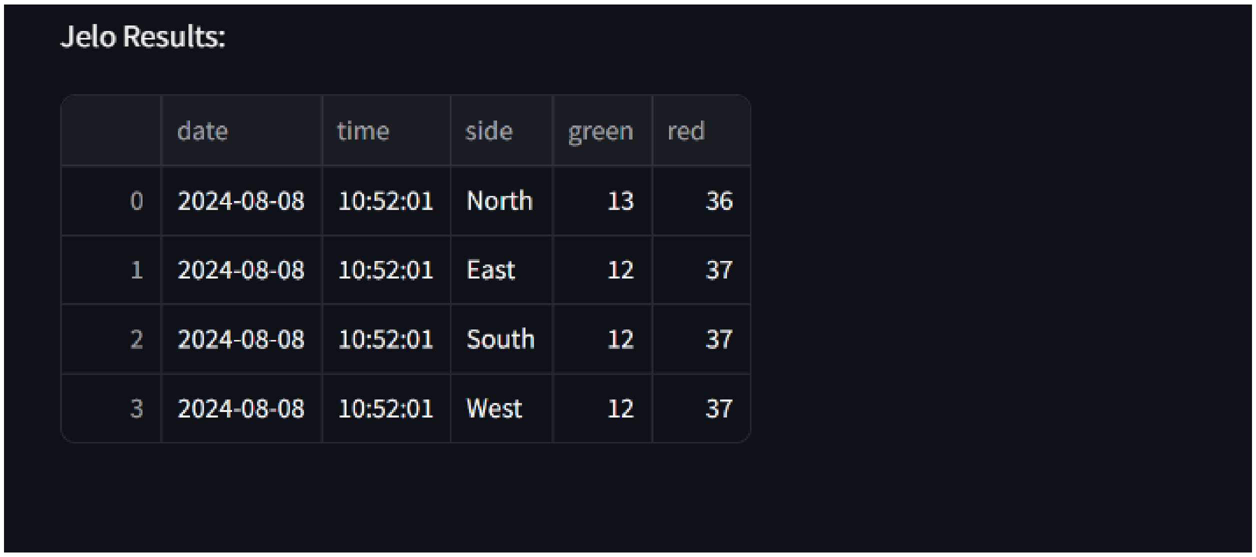
Table 9: Performance matrix for regression algorithms

Based on the extensive testing with above different algorithms, the Decision Tree algorithm stands out as the best choice for implementing a traffic light duration control system as it shows better performance matrix. As shows in table-9, the Decision Tree Regression outperforms all other algorithms in terms of MSE, MAE, and R-squared values, has the lowest error metrics and the highest R-squared value, shows a near-perfect fit. Its ability to model non-linear relationships, handle different data types, and provide clear, interpretable decision rules makes it a superior choice for this application. While Decision Trees can overfit, techniques such as pruning and ensembling can make sure that the model generalizes well to a new set of data. As shows in table-

1, the Decision Tree algorithm's superior performance, flexibility, and interpretability make it the optimal choice for AI-based traffic light duration control.

Generating traffic signal duration:

The outputs generated by the YOLO model and the trained model will be integrated into our AI model to determine the appropriate signal durations.



Jelo Results:

	date	time	side	green	red
0	2024-08-08	10:52:01	North	13	36
1	2024-08-08	10:52:01	East	12	37
2	2024-08-08	10:52:01	South	12	37
3	2024-08-08	10:52:01	West	12	37

Figure 33: Traffic signal duration from AI model using decisionTree algorithm.

The traffic signal durations are precisely designed to make sure that when one direction has a green signal, the other directions have a red signal as shows in figure-33. The red signal duration for any direction is the cumulative sum of the green signal durations of the other three directions.

For instance:

- **North Side**: The green signal duration is set to 13 seconds. Consequently, the red signal duration is 36 seconds, which is the sum of the green signals on the East, South, and West sides.
- **East Side:** The green signal duration is set to 12 seconds. As a result, the red signal duration is 37 seconds which is the sum of North, South, and West sides.
- **South Side**: The green signal duration is set to 12 seconds. Therefore, the red signal duration is 37 seconds, which is the sum of the green signals on the North, East, and West sides.

- **West Side:** The green signal duration is set to 12 seconds. Hence, the red signal duration is 37 seconds, which is the sum of the green signals on the North, East, and South sides.

This setup ensures a balanced and coordinated traffic flow. Additionally, it will minimize the traffic congestion and providing equal time for each direction to pass through the traffic intersection. By distributing the signal durations in this way, the system maintains an efficient flow of traffic which helps to reduce wait times and improving overall intersection management.

End of my section

Recommendations for Future Improvement

For Future Studies and improvements to our Intelligent Traffic Management System, more focus could be drawn to optimizing models using ensemble approaches, incorporating more real-time data sources, and expanding the system via distributed computing. Adding an Adaptive learning approach to our system could allow for continual improvement by updating models in real-time, while including contextual information such as weather and unusual events would improve forecast accuracy. Furthermore, including a feedback system and intensive simulation testing will keep the system responsive to real-world settings.

Long-term development should also consider integrating the system with broader smart city infrastructure and addressing ethical considerations related to data privacy. By aligning with city planners and transportation agencies, the system can contribute to a connected urban environment, while maintaining compliance with regulations to protect user privacy. Conducting impact assessments will help gauge the system's effectiveness and guide future advancements.

References

1. A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “Yolov4: Optimal speed and accuracy of object detection,” arXiv preprint arXiv:2004.10934, 2020.

2. C. H. Genitha, S. A. Danny, A. S. Hepsi Ajibah, S. Aravint, and A. A. V. Sweety, "AI based Real-Time Traffic Signal Control System using Machine Learning," 2023 4th International Conference on Electronics and Sustainable Communication Systems (ICESC), Coimbatore, India, 2023, pp. 1613-1618, doi: 10.1109/ICESC57686.2023.10193319.

3. D. Krajzewicz, "Traffic simulation with SUMO–simulation of urban mobility," Fundamentals of Traffic Simulation, pp. 269-293, 2010.

4. F. García, H. Hernández, M. N. Moreno-García, J. F. de Paz Santana, V. F. López, and J. Bajo, "Traffic Optimization Through Waiting Prediction and Evolutive Algorithms," 2023.

5. G. Kotusevski, and K. A. Hawick, "A review of traffic simulation software," Research Letters in the Information and Mathematical Sciences, vol. 13, pp. 35-54, 2009.

6. H. T. Fritzsche, and D. B. Ag, "A model for traffic simulation," Traffic Engineering+ Control, vol. 35, no. 5, pp. 317-321, 1994.

7. J. F. Solawetz, "What is YOLOv8? The ultimate guide," Blog post, 2023. [Online]. Available: https://blog.roboflow.com/whats-new-in-yolov8/.

8. J. Shang, X. Yan, L. Feng, Z. Dong, H. Wang, and S. Zhou, "ExtTra: Short-Term Traffic Flow Prediction Based on Extremely Randomized Trees," in Neural Information Processing: 25th International Conference, ICONIP 2018, Siem Reap, Cambodia, December 13-16, 2018, Proceedings, Part IV 25, Springer International Publishing, 2018, pp. 532-544.

9. K. Range and G. Jocher, "Brief summary of YOLOv8 model structure," GitHub Issue, 2023. [Online]. Available: https://github.com/ultralytics/ultralytics/issues/189.

10. M. Akhtar, and S. Moridpour, "A review of traffic congestion prediction using artificial intelligence," Journal of Advanced Transportation, vol. 2021, no. 1, p. 8878011, 2021.

11. M. Hussain, “Yolo-v1 to yolo-v8, the rise of yolo and its complementary nature toward digital manufacturing and industrial defect detection,” 2023.

12. M. Pursula, "Simulation of traffic systems-an overview," Journal of Geographic Information and Decision Analysis, vol. 3, no. 1, pp. 1-8, 1999.

13. P. Alvarez Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, "Microscopic traffic simulation using SUMO," in Proceedings of the IEEE Intelligent Transportation Systems Conference (ITSC), 2018.

14. Roboflow, "Roboflow 100: A new object detection benchmark (2023) RF100," 2023. [Online]. Available: https://www.rf100.org/.

15. S. M. Mastelini, F. K. Nakano, C. Vens, and A. C. P. de Leon Ferreira, "Online extra trees regressor," IEEE Transactions on Neural Networks and Learning Systems, vol. 34, no. 10, pp. 6755-6767, 2022.

16. Ultralytics, "YOLOv8 Docs," 2023. [Online]. Available: https://docs.ultralytics.com/.

17. B. Pratama, J. Christanto, M. Hadyantama, and A. Muis, "Adaptive Traffic Lights through Traffic Density Calculation on Road Pattern," 2018 10th International Conference on Awareness Science and Technology (iCAST), 2018, pp. 82-86, doi: 10.1109/iCAST1.2018.8751540.