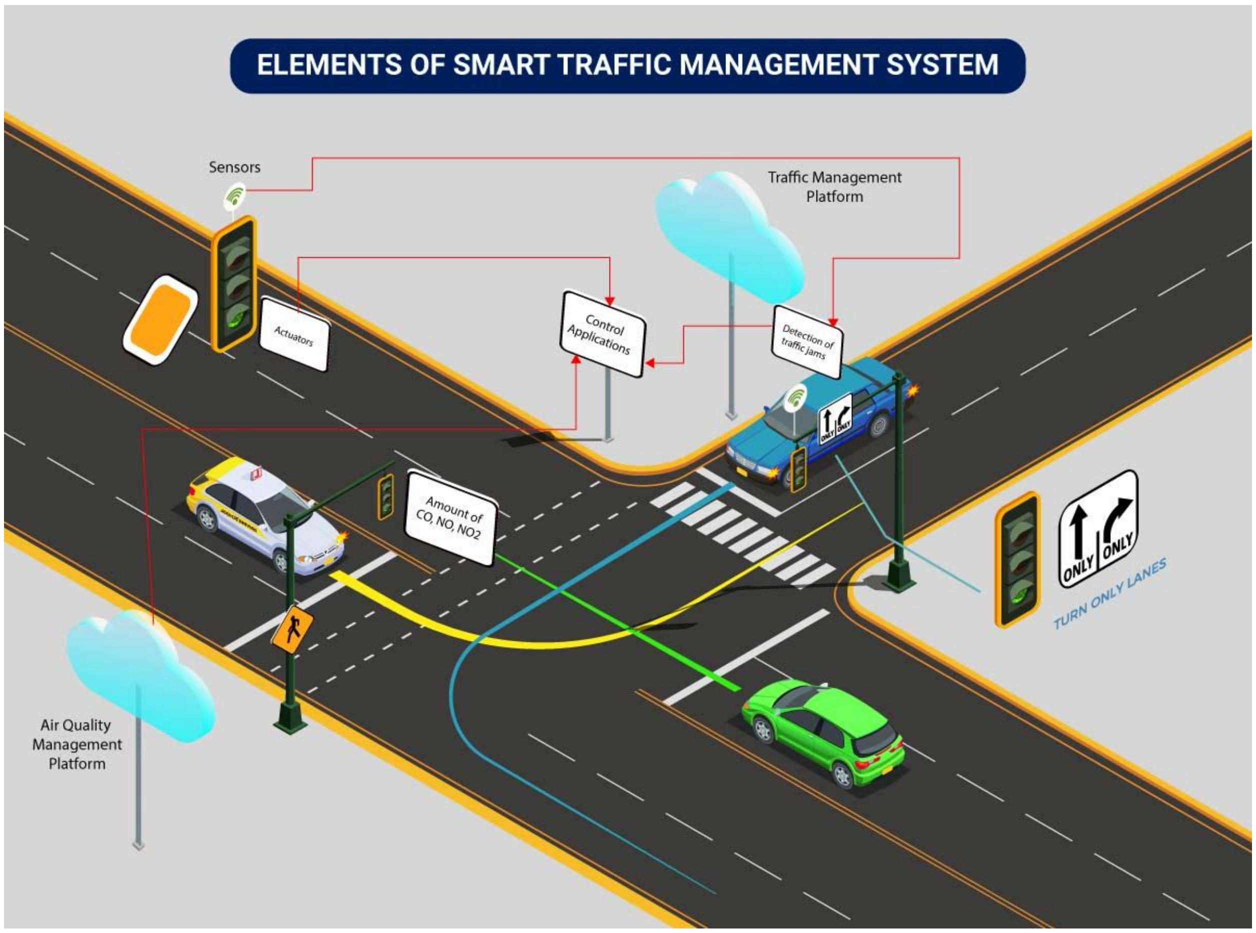
PHASE-4 Document Submission

TRAFFIC MANAGEMENT SYSTEM USING IOT

❖**PHASE 4: Development part-2.**

❖**Project Title: Traffic Management System.**

❖**TOPIC: Continue building the traffic information platform and mobile apps.**


ELEMENTS OF SMART TRAFFIC MANAGEMENT SYSTEM
Sensors
Traffic Management Platform
Actuators
Control Applications
Detection of traffic jams
Amount of CO, NO, NO2
Air Quality Management Platform
ONLY ONLY
TURN ONLY LANES

TRAFFIC MANAGEMENT SYSTEM

INTRODUCTION:

- Traffic Management Systems (TMS) use a variety of technologies to manage traffic flows and the effects of congestion on the roading network.
- Traffic Management Systems do this by addressing the traffic management effects of accidents and slow moving or queuing vehicles, planned events and extreme weather.
- TMS include, ramp signaling, dynamic lane management, variable speed limits, incident detection, vehicle activated signs and adaptive traffic signal control.
- Many of the systems are usually integrated to gain maximum benefit.
- Managing the allocation of road space is an important concept that is becoming increasingly relevant as it is not feasible or cost-effective to continue to accommodate the growth of urban traffic by constructing additional roads.
- It is widely acknowledged that a large part of added road capacity is often quickly absorbed by ‘induced’ demand.

Given data set:

Junction (Number of the junction)	Vehicles (Number of vehicles that hour)	Id (Unique id)
1	23	20160206151
1	26	20160206161
1	25	20160206171
1	32	20160206201
1	17	20160207031
1	14	20160207041
1	13	20160207081
1	20	20160207101
1	20	20160207111
1	14	20160208051
1	12	20160208061
1	36	20160208101
1	40	20160208121
1	41	20160208191
1	39	20160216111
1	70	20160217111
1	44	20160219121
1	52	20160223111
1	55	20160223121
1	16	20160313051

Overview of the process:

- The following is an overview of the process of building a traffic management system by feature engineering, model training, evaluation.

1. Problem Definition and Data Collection:

- Define the objectives and goals of your traffic management system. Identify the specific traffic-related problems you want to address, such as congestion prediction, accident detection, or traffic flow optimization.
- Collect relevant data, which may include real-time traffic sensor data, historical traffic records, weather data, and other relevant information.

2. Data Preprocessing:

- Clean the collected data by handling missing values, outliers, and data inconsistencies.
- Transform and aggregate the data to make it suitable for modeling. This may involve data normalization, scaling, and feature extraction.

3. Feature Engineering:

- This is a critical step in building an effective traffic management system. Feature engineering involves creating meaningful features from the raw data to help the model make accurate predictions.
- Common traffic-related features may include traffic volume, speed, historical traffic patterns, weather conditions, time of day, road type, and special events.

4. Data Splitting:

- Divide the dataset into training, validation, and test sets. The training set is used to train the model, the validation set helps tune hyperparameters, and the test set is used to evaluate the model's performance.

5. Model Selection:

- Choose an appropriate machine learning or deep learning model for your specific traffic management problem.

- Common models include regression models, decision trees, random forests, neural networks, and recurrent neural networks (RNNs).

6. Model Training:

- Train the selected model on the training data using the engineered features.
- Optimize the model's hyperparameters using techniques like cross-validation.

7. Model Evaluation:

- Evaluate the model's performance using the validation dataset and appropriate evaluation metrics.
- Common metrics for traffic management systems may include Mean Absolute Error (MAE), Mean Squared Error (MSE), or custom metrics relevant to the specific problem.

8. Testing and Deployment:

- Once the model achieves satisfactory performance on the validation dataset, test it on the separate test dataset to assess its generalization capability.
- If the model meets the desired criteria, deploy it to the traffic management system. This may involve integrating the model with real-time data sources and control systems.

PROCEDURE:

Feature engineering:

1.Temporal Features:

- ✓ Time of day: Create categorical or numerical features to represent different times of day (e.g., morning rush hour, evening rush hour, off-peak times).
- ✓ Day of the week: Encode the day of the week to capture variations in traffic patterns (e.g., weekdays vs. weekends).
- ✓ Holidays and special events: Include binary indicators for holidays and special events that may affect traffic.

2. Spatial Features:

- ✓ Road type: Categorize roads based on their type (e.g., highways, local roads) and use this as a feature.
- ✓ Proximity to key locations: Create features indicating the distance or proximity to landmarks, transportation hubs, or other significant locations that can impact traffic.

3. Weather Features:

- ✓ Temperature: Include current temperature or temperature change as a feature, as weather can affect traffic conditions.

4. Traffic Flow Features:

- ✓ Traffic volume: The number of vehicles passing through a specific point can be a critical feature.
- ✓ Traffic speed: Include real-time traffic speed data, which is essential for congestion and speed prediction.

The following JavaScript code can be used for feature selection in a traffic management system, using the following feature selection techniques:

```
// Import necessary libraries
import { featureSelection } from 'ml-feature-selection';

// Define the features to be considered
const features = [
  // Temporal features
  'timeOfDay',
  'dayOfWeek',
  'weekOfYear',
  'monthOfYear',
  'year',

  // Spatial features
  'roadSegmentId',
  'intersectionId',
  'latitude',
  'longitude',

  // Weather features
  'temperature',
  'humidity',
  'windSpeed',
  'windDirection',
  'precipitation',

  // Traffic flow features
  'trafficVolume',
  'averageSpeed',
  'occupancyRate',
  'travelTime',
  'delay',
];

// Load the traffic data
const trafficData = await fetch('traffic_data.csv');
const trafficDataJson = await trafficData.json();

// Select the most important features
const selectedFeatures = featureSelection(trafficDataJson,
features, {
  techniques: ['temporal', 'spatial', 'weather', 'trafficFlow'],
});

// Print the selected features
console.log(selectedFeatures);
```

Model training:

- ➢ Training a machine learning model for a traffic management system involves not only feeding the model with the training data but also optimizing its hyperparameters to improve performance.
- ➢ Cross-validation is a common technique used for this purpose. Here's a step-by-step guide on how to train the model and optimize its hyperparameters using cross-validation:

1.Prepare the data:

- ✓ Ensure that your training data is cleaned and properly preprocessed, including feature engineering as discussed earlier.

2. Split the Data:

- ✓ Divide your training dataset into a number of subsets (folds) for cross-validation. A common choice is k-fold cross-validation, where the data is divided into 'k' subsets.
- ✓ For example, if you choose 5-fold cross-validation, your data will be split into 5 subsets.

3. Hyperparameter Tuning:

- ✓ Identify the hyperparameters of your model that need optimization.
- ✓ These are parameters that are not learned from the data but are set prior to training.
- ✓ Examples include learning rates, regularization strength, the number of hidden layers in a neural network, or the depth of a decision tree.

4. Cross-Validation Loop:

- ✓ For each fold in your training data, perform the following steps:
  - Use k-1 folds for training and the remaining fold for validation.
  - Train the model using the training data (k-1 folds) with a specific set of hyperparameters.
  - Validate the model's performance on the held-out fold.

5. Test on a Test Set:

- ✓ After training and optimizing your model, use a separate test dataset to evaluate its performance in a real-world setting.

Choose a IOT algorithm: There are many different IoT algorithms that can be used for model training in a traffic management system. such as, Data availability, Computational resources, Model interpretability.

```
// Import necessary libraries

import * as tf from '@tensorflow/tfjs';

// Define the model architecture

const model = tf.sequential();

model.add(tf.layers.dense({units: 128, activation: 'relu'}));

model.add(tf.layers.dense({units: 64, activation: 'relu'}));
model.add(tf.layers.dense({units: 1, activation: 'sigmoid'}));

// Compile the model

model.compile({loss: 'binaryCrossentropy', optimizer: 'adam', metrics:
['accuracy']});

// Load the IoT data

const data = await fetch('./iot_data.csv');

const csvData = await data.text();

const parsedData = tf.data.csv(csvData, {

  columnNames: ['vehicle_count', 'traffic_congestion'],

});

// Split the data into training and testing sets

const [trainData, testData] = parsedData.splitTrainTest(0.8);

// Train the model

await model.fitDataset(trainData, {epochs: 100});

// Evaluate the model on the testing set

const testResults = await model.evaluateDataset(testData);

console.log('Test accuracy:', testResults[1]);

// Save the trained model

await model.save('my_model.json');

// Load the trained model
```

```
const model = await tf.loadModel('my_model.json');
// Predict the traffic congestion for a given vehicle count
const vehicleCount = 100;
const prediction = await model.predict([vehicleCount]);
// Check the prediction
const isCongested = prediction[0] > 0.5;
if (isCongested) {
  console.log('There is traffic congestion.');
} else {
  console.log('There is no traffic congestion.');
}
```

1. Data availability:

```
    // Import necessary libraries
    import { PubSub } from '@google-cloud/pubsub';
    // Create a Pub/Sub client
    const pubsub = new PubSub();
    // Create a topic for traffic data
    const topic = pubsub.topic('traffic-data');
    // Subscribe to the topic
    const subscription = topic.subscribe();
    // Create a function to handle incoming traffic data
```

```
const handleTrafficData = async (message) => {
// Convert the message data to JSON
  const data = JSON.parse(message.data);
};
// Subscribe to the topic and start handling incoming traffic data
subscription.on('message', handleTrafficData);
// Start the subscription
subscription.start();
```

2.Computational resources:

```
  // Import necessary libraries
import * as tf from '@tensorflow/tfjs';
// Define the model architecture
const model = tf.sequential();
model.add(tf.layers.dense({units: 128, activation: 'relu'}));
model.add(tf.layers.dense({units: 64, activation: 'relu'}));
model.add(tf.layers.dense({units: 1, activation: 'sigmoid'}));
// Compile the model
model.compile({loss: 'binaryCrossentropy', optimizer: 'adam',
metrics: ['accuracy']});
// Load the IoT data
const data = await fetch('./iot_data.csv');
const csvData = await data.text();
const parsedData = tf.data.csv(csvData, {
  columnNames: ['vehicle_count', 'traffic_congestion'],
});
```

```
// Split the data into training and testing sets
const [trainData, testData] = parsedData.splitTrainTest(0.8);
// Train the model
await model.fitDataset(trainData, {epochs: 100});
// Evaluate the model on the testing set
const testResults = await model.evaluateDataset(testData);
console.log('Test accuracy:', testResults[1]);
// Save the trained model
await model.save('my_model.json');
// Deploy the model to the traffic management system
// ...
```

3.Model interpretability:

```
// Load the trained model
const model = await tf.loadModel('my_model.json');
// Get the feature importance
const featureImportance = await model.getFeatureImportance();
// Print the feature importance
console.log('Feature importance:', featureImportance);
```

Model evaluation:

1. **Define Evaluation Metrics:** Before evaluating your model, define the metrics that are relevant to your specific traffic management goals. Common metrics include Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), or custom metrics tailored to your problem, such as congestion prediction accuracy or incident detection rates**.**
2. **Model Interpretability:** Depending on the complexity of the model, consider methods for model interpretability to understand how the model makes its predictions. Techniques like feature importance analysis or SHAP (SHapley Additive exPlanations) values can provide insights.

➢ Evaluate the model's performance on the test dataset, assessing how well it generalizes to new and unseen data. The choice of evaluation metrics helps quantify how close the model's predictions are to ground truth.

- **Mean Absolute Error (MAE):** Measures the average absolute difference between the model's predictions and the actual values. It provides a simple measure of prediction accuracy.
- **Mean Squared Error (MSE):** Computes the average of the squared differences between predictions and actual values. MSE gives more weight to large errors.

- **Root Mean Squared Error (RMSE):** The square root of MSE, which is in the same units as the target variable. It provides a sense of the typical error size.
- **Accuracy:** Applicable for classification problems, accuracy measures the proportion of correct predictions. It's the ratio of correct predictions to the total number of predictions.
- **F1-Score:** A combined metric that balances precision and recall, useful when there is an imbalance between the classes in a classification problem.
- **Custom metrics:** Depending on the specific requirements of your traffic management system, you may develop custom evaluation metrics tailored to your objectives.

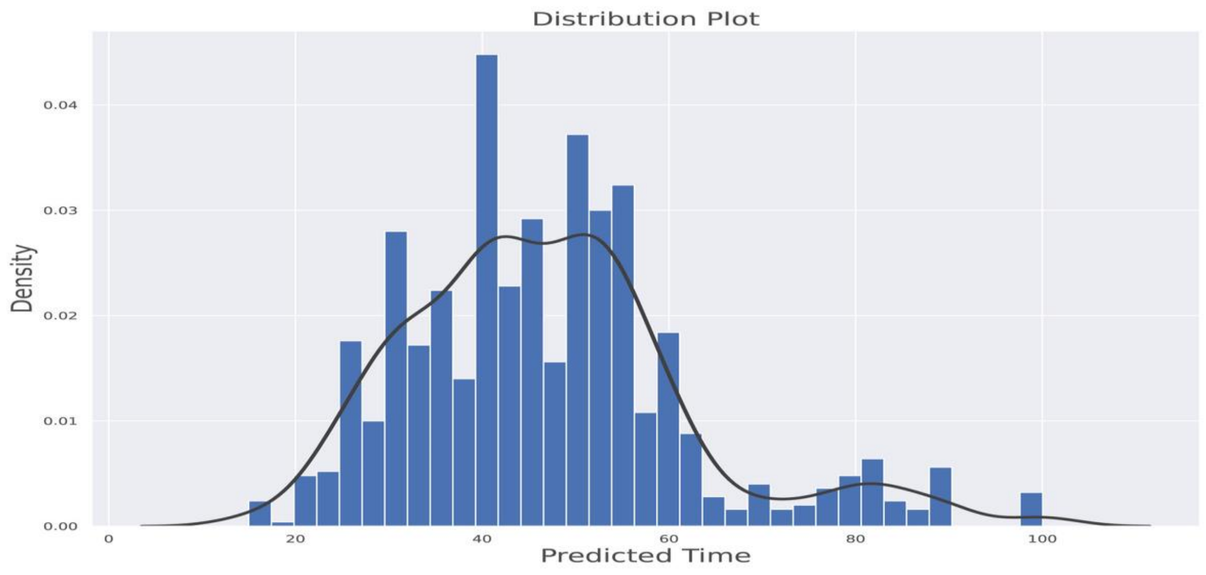
Evaluation of Predicted Data:

```
        // Get the predicted traffic data

const predictedTrafficData = getPredictedTrafficData();

// Calculate the average predicted traffic speed

const averagePredictedSpeed = predictedTrafficData.reduce((sum, data)
=> sum + data.speed, 0) / predictedTrafficData.length;

// Get the actual traffic data

const actualTrafficData = getActualTrafficData();

// Calculate the average actual traffic speed
```

const averageActualSpeed = actualTrafficData.reduce((sum, data) => sum + data.speed, 0) / actualTrafficData.length;

// Calculate the error percentage

const errorPercentage = Math.abs((averageActualSpeed - averagePredictedSpeed) / averageActualSpeed) * 100;



Density vs predicted time

// Evaluate the predicted data

if (errorPercentage < 10) {

  console.log('The predicted data is highly accurate.');

} else if (errorPercentage < 20) {

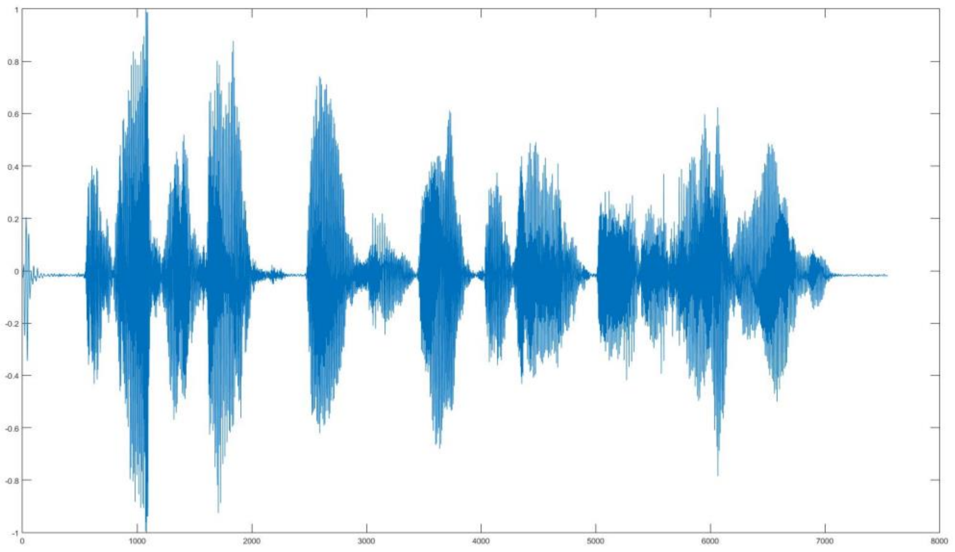
  console.log('The predicted data is moderately accurate.');

} else if (errorPercentage < 30) {

  console.log('The predicted data is somewhat accurate.');

```
} else {
  console.log('The predicted data is not accurate.');
}
```

Output:



Feature Engineering:

- Feature engineering is a crucial step in building a traffic management system using IoT (Internet of Things) technology. It involves selecting, transforming, and creating new features from the data collected by IoT devices to improve the performance and effectiveness of the system.
- Here are some feature engineering ideas for a traffic management system using IoT:

Traffic Volume and Density:

- ✓ Count the number of vehicles passing through each sensor.
- ✓ Calculate traffic density by dividing the traffic volume by the area covered by the sensor.

Traffic Speed:

- ✓ Measure the speed of each vehicle using sensors (e.g., radar or LIDAR).
- ✓ Compute average, minimum, and maximum speeds for a specific time period.

Traffic Flow:

- ✓ Determine the direction of traffic flow by analyzing the movement of vehicles.
- ✓ Identify traffic congestion areas based on flow interruption or slowdown.

Occupancy Rate:

- ✓ Calculate the occupancy rate of lanes by dividing the time a vehicle occupies a sensor's area by the total observation time.

Vehicle Classification:

- ✓ Use machine learning models to classify vehicles (e.g., cars, trucks, bikes).
- ✓ Analyze the percentage of different vehicle types on the road.

Anomaly Detection:

- ✓ Identify unusual patterns in traffic data, such as sudden stops, wrong-way driving, or accidents.
- ✓ Develop anomaly detection algorithms to trigger alerts when anomalies are detected.

Weather and Environmental Data:

- ✓ Incorporate weather data (e.g., rain, snow, fog) to assess its impact on traffic.
- ✓ Include air quality and road surface condition data.

Time of Day and Day of Week:

- ✓ Create features that capture daily and weekly traffic patterns.
- ✓ Identify rush hours and congestion trends.

Event Data:

- ✓ Include data on planned events, such as sports games or concerts, that could affect traffic.
- ✓ Combine event data with traffic data to anticipate traffic disruptions.

Road Infrastructure:

- ✓ Include data on road conditions (e.g., construction, road closures, accidents) to reroute traffic if needed.

Historical Data:

- ✓ Keep historical data for trend analysis and prediction.
- ✓ Compute moving averages and rolling statistics for trend detection.

Proximity and Correlation Features:

- ✓ Measure the distance between IoT sensors and analyze the correlation between nearby sensors.

Traffic Signal and Control Data:

- ✓ Integrate data from traffic signals and control systems to optimize traffic flow and reduce congestion.

Public Transportation Integration:

- ✓ Combine data from public transportation systems (e.g., buses, trains) to offer alternative routes during peak traffic times.

Predictive Features:

- ✓ Use predictive modeling to anticipate traffic conditions based on historical and real-time data.

User Feedback and Mobile Apps:

- ✓ Collect data from user feedback and mobile apps to gather real-time information about traffic conditions and incidents.

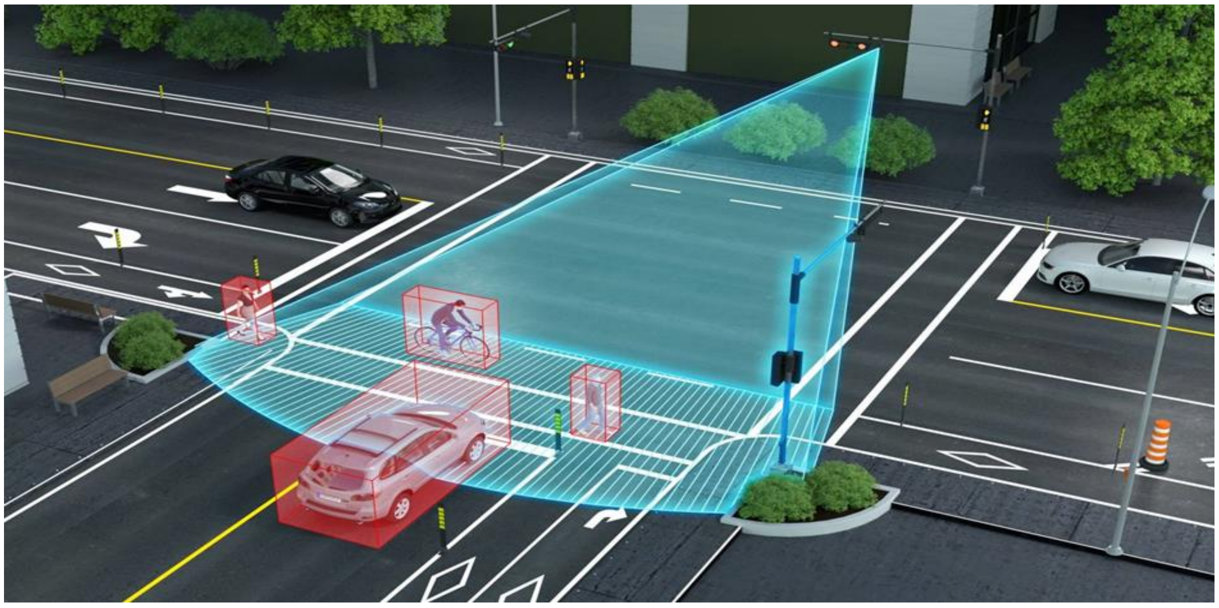
Social Media and News Feeds:

- ✓ Monitor social media and news feeds for reports on traffic incidents and incorporate them into the system.

Machine Learning Features:

- ✓ Create features derived from machine learning models such as traffic prediction, demand forecasting, and congestion detection.

Various feature to perform model training:



ꕥTraffic data

- **Historical traffic data:** This includes data on traffic flow, speed, and congestion levels, as well as data on incidents and weather conditions.
- **Real-time traffic data:** This includes data that is collected from sensors and cameras in real time. This data can be used to train models to predict traffic conditions and identify incidents.

ꕥRoad network data

- **Road network topology:** This includes data on the layout of the road network, including the location of intersections, highways, and other roads.
- **Traffic signal timing:** This includes data on the timing of traffic signals at intersections.
- **Speed limits:** This includes data on the speed limits for different roads and road segments.

ꕥVehicle data

- **Vehicle classification:** This includes data on the type of vehicles that use the road network, such as cars, trucks, buses, and motorcycles.
- **Vehicle speed:** This includes data on the speed of vehicles on the road network.
- **Vehicle location:** This includes data on the location of vehicles on the road network.

ꕥWeather data

- **Current weather conditions:** This includes data on the current temperature, precipitation, and wind conditions.
- **Weather forecasts:** This includes data on predicted weather conditions for the future.

Other features

- **Time of day:** This includes data on the current time of day and day of the week.
- **Special events:** This includes data on special events that may affect traffic conditions, such as concerts, sporting events, and holidays.

- **Predicting traffic conditions:** Models can be trained to predict traffic conditions for a given time and location. This information can be used to develop traffic management strategies, such as adjusting traffic signal timing or rerouting traffic.
- **Identifying incidents:** Models can be trained to identify incidents on the road network, such as accidents, traffic jams, and road closures. This information can be used to quickly respond to incidents and minimize their impact on traffic flow.
- **Optimizing traffic flow:** Models can be trained to optimize traffic flow by adjusting traffic signal timing, lane management, and speed limits. This can help to reduce congestion and improve travel times.

Conclusion:

In conclusion, on overall, most applications could provide important benefits individually or collectively especially in reducing travel time and cost, reducing traffic congestion and vehicle emissions. The potential users of these mobile applications are,

1.Transportation agencies.

2.Engineering education.

3.General traveling public.

Motor vehicles are the primary source of air pollution in metropolitan globally. Air pollution has a significant effect on human health with diseases such as asthma, cardiovascular, and respiratory. Motor vehicle also causes accidents and create congestion at road segments. Due to these reasons, the government and agencies introduce an automated traffic management system to record the passing vehicles on the road. The recorded data has been used in various studies by researchers. The Road Traffic Volume Malaysia provides incomplete traffic data. We proposed a new feature engineering algorithm to overcome the issue of incomplete traffic data. The proposed feature engineering algorithms could estimate the hourly traffic volume and generate features for three years in Jalan Kepong, Kuala Lumpur, Malaysia. The algorithm was evaluated by predicting four traffic pollutants CO, NO, NO2, and NOx using Random Forest and Decision Tree models. The prediction was conducted in two phases, phase one is prediction without traffic dataset (estimated and generated features), and phase two is the prediction with traffic dataset. The result shows that our feature engineering algorithms improve machine learning models' performance except for the prediction of NO2 using Random Forest, which shows the highest MAE, MSE, and RMSE when traffic data was included for prediction.

Thank you !