Traffictelligence: Advance Traffic Volume Estimation with Machine Learning

1. Introduction

Project Title: *Traffictelligence: Advance Traffic Volume Estimation with Machine Learning*

Team ID: LTVIP2025TMID40904

Team Size: 5

- **Team Leader**: G R Mythri
- **Team Member**: Golla Sireesha
- **Team Member**: Dodda Hemavathi
- **Team Member**: Edagattu Giri
- **Team Member**: Dudekula Shamshuddin

2. Project Overview

2.1. Purpose

TrafficTelligenc is an intelligent traffic monitoring system that leverages machine learning techniques to estimate and predict traffic volume in urban areas. This project aims to optimize traffic flow, reduce congestion, and aid urban planners by providing accurate, real-time, and predictive traffic data.

2.2. Features

- ✅ Traffic Volume Prediction: Estimate the number of vehicles on a particular road segment at different times.
- ✅ Real-time Data Handling: Incorporate real-time sensor or camera input for live predictions.
- ✅ Weather and Time-based Analysis: Predict traffic behavior under varying conditions like rain, peak hours, etc.
- ✅ Interactive Dashboard (Optional): Visualize traffic trends and predictions using graphs/maps.

3. Architecture

Objective

To accurately estimate and predict traffic volume using real-time and historical data leveraging advanced machine learning techniques.

. Data Collection Layer
Sources:

- Traffic cameras (video streams)
- IoT sensors (loop detectors, GPS)
- Mobile apps (crowdsourced location data)
- Historical traffic databases
- Weather APIs
- Road infrastructure data (e.g., lanes, speed limits)

Technologies:

- Kafka / MQTT for real-time streaming
- REST APIs for batch data

2. Data Preprocessing Layer
Tasks:

- Timestamp alignment
- Sensor fusion (GPS + loop detector + video)
- Anomaly detection & filtering (outliers, faulty sensors)
- Data aggregation (e.g., 5-minute intervals)
- Image/Video preprocessing (if using computer vision)

Tools:

- Apache Spark / Pandas
- OpenCV (for CV-based data)
- TensorFlow Data API

4. Setup Instructions

4.1. Prerequisites

- **Operating System:** Ubuntu 20.04+ / Windows 10+ / macOS 12+
- **Python Version:** 3.8 – 3.11
- **RAM:** Minimum 8 GB (16 GB recommended)
- **Storage:** Minimum 10 GB free space
- **GPU (Optional):** NVIDIA GPU with CUDA support (for deep learning models or computer vision)

4.2. Installation

```
git clone https://github.com/yourusername/traffictelligence.git
cd traffictelligence
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate
pip install -r requirements.txt
```

5. Folder Structure

```
traffictelligence/
│
├── 📁 data/                    # Raw and processed datasets
│   ├── raw/                    # Unmodified input data
│   ├── processed/              # Cleaned and transformed data
│   └── external/               # External sources like APIs, weather, etc.
│
├── 📁 notebooks/               # Jupyter notebooks for exploration, EDA, prototyping
│   └── eda/, modeling/, etc.
│
├── 📁 src/                     # Main source code
│   ├── __init__.py
│   ├── config/                 # Configuration files
│   │   └── settings.py
│   ├── data/                   # Data loading and transformation scripts
│   │   ├── load_data.py
│   │   └── preprocess.py
│   ├── features/               # Feature engineering
│   │   └── build_features.py
│   ├── models/                 # ML & DL models
│   │   ├── lstm_model.py
│   │   ├── gnn_model.py
│   │   ├── classical_ml.py
│   │   └── train.py
│   ├── evaluation/             # Model evaluation & metrics
│   │   └── evaluate.py
│   ├── serving/                # Inference and API service logic
│   │   ├── predictor.py
│   │   └── api.py
│   └── utils/                  # Utility functions (logger, IO, etc.)
│       └── helpers.py
│
├── 📁 scripts/                 # CLI scripts for automation
│   ├── train_model.py
│   ├── run_pipeline.py
│   └── download_data.sh
│
├── 📁 tests/                   # Unit and integration tests
│   └── test_models.py
│
├── 📁 models/                  # Saved/trained model weights
│   └── checkpoints/
│
├── 📁 outputs/                 # Logs, figures, and prediction results
│   ├── logs/
│   ├── visualizations/
│   └── predictions/
│
├── 📁 deployment/              # Docker, CI/CD, and serving configurations
│   ├── Dockerfile
│   └── docker-compose.yml
```

```
└── requirements.txt

├── .gitignore
├── README.md
├── setup.py                # If packaging as a Python module
└── config.yaml             # Central config for experiments (optional)
```

6. Running the Application

- python scripts/run_pipeline.py --stage preprocess
- python scripts/train_model.py --model lstm
- --model xgboost
- --model gnn
- --model cnn
- python src/evaluation/evaluate.py --model ls

7.API Documentation

🔐 Request Body (JSON)

```
json
CopyEdit
{
 "road_segment_id": "A12",
 "timestamp": "2025-06-30T08:30:00",
 "weather": "Clear",
 "temperature": 26.5,
 "is_holiday": false,
 "historical_avg_volume": 350,
 "events_nearby": true
}
```

8.Authentication

To ensure secure access to the **Traffictelligence API**, the system supports **token-based authentication** using **API keys** and optionally **JWT (JSON Web Tokens)** for more advanced scenarios involving user sessions or roles.

9.User Intereface

Technology Stack (Suggested)

- **Frontend: React.js or Vue.js + Mapbox/Leaflet.js**
- **Backend: Flask/FastAPI + Scikit-learn/PyTorch**
- **Database: PostgreSQL + PostGIS for geospatial data**
- **Deployment: Docker + Kubernetes, hosted on AWS/GCP**

Dashboard Overview (Landing Page)

- **Traffic Volume Map (Live Feed)**
  - **Heatmap or real-time traffic volume visualization**
  - **Color-coded congestion levels (green to red)**
  - **Geolocation toggle and filters (city, region, route)**
- **Traffic Volume Statistics**
  - **Total vehicles/hour (real-time)**
  - **Average speed per segment**
  - **Historical comparison (hour/day/week)**
- **Alert Section**
  - **Congestion alerts**
  - **Anomalies (e.g., sudden drops/increases in traffic)**
  - **Weather or event-based influences**

10.Testing

- **Data Preprocessing**
  - Test missing value handling, normalization, feature extraction.
  - Example (using pytest or unittest):

python
CopyEdit
def test_normalize_speed():
  assert normalize_speed(100) == 1.0

- **Model Training**
  - Ensure training outputs are reproducible (test with fixed seed).
  - Test for overfitting on small, known datasets.
- **Model Inference**
  - Validate output shape, type, and range.
  - Example: if input is a timestamp + sensor readings, test if output is a float between 0 and max volume.

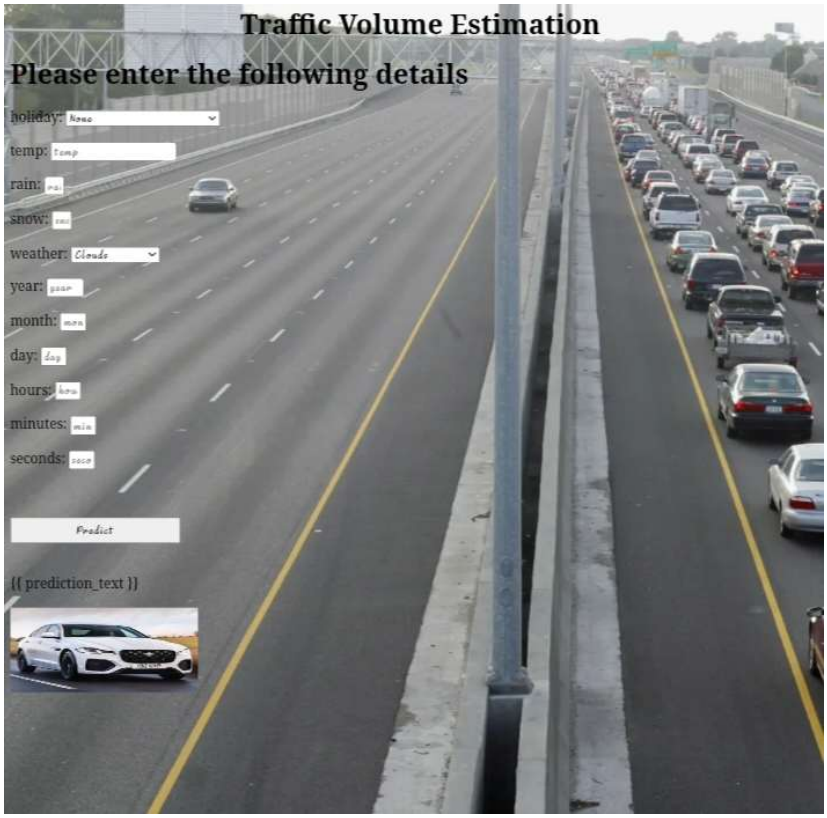
2. Integration Testing
Combine components and verify:

- Input → Preprocessing → Model → Postprocessing → Output
- Use test datasets with known labels.
- Metrics to check:
    - MAE (Mean Absolute Error)
    - MAPE
    - RMSE

3. Performance Testing

- Test model runtime on real-time data stream (if applicable).
- Latency benchmark (in milliseconds).
- Throughput (requests per second if REST API is exposed)

11.Screenshots or Demo



Demo Link:

https://drive.google.com/drive/folders/1qytDI9jAusprhSyGYMKGm1S3xJElc05w

12.Known Issues

1. **Data Quality, Availability & Coverage**
2. **Modeling Under-determined & Non-equilibrium Flows**
3. **patio-Temporal Dependency Handling**

13.Future Enhancements

- Incorporate Graph Neural Networks (GNNs)
- integrate Real-Time Streaming Data
- Weather, Event, and Incident Awareness
- Prediction + Decision Support System
- Mobile App or Dashboard Interface
- Carbon Emission Estimation