Project Phases Template

Project Title:

TrafficTelligence: Advanced Traffic Volume Estimation with Machine Learning

Team ID : LTVIP2025TMID20285

Team Leader : Gangireddy Dhanesh

Team member : Edpuganti Keerthi

Team member : Dusi Sai

Team member : Eeda Swathi

Phase-1: Brainstorming & Ideation

The initial phase focused on identifying a problem that could be addressed using Artificial Intelligence to improve urban mobility and traffic management. The goal was to develop a solution that leverages deep learning to estimate traffic volume accurately .

Problem Identification

The team explored challenges in urban transportation, such as congestion, inefficient traffic signal timing, and lack of real-time traffic monitoring. The problem of traffic volume estimation emerged as a critical issue, as accurate estimation can aid in traffic management, route optimization, and infrastructure planning**.**

Idea Finalization

The final idea was to build a deep learning model capable of estimating traffic volume from images or video feeds, categorizing traffic into:

- Low
- Medium
- High

This classification would help authorities and commuters make informed decisions to reduce congestion and improve traffic flow.

Objectives Outlined

- Utilize a pre-trained CNN model (e.g., ResNet50) for traffic volume estimation.
- Develop a web application using Flask to display real-time or uploaded traffic data.

- Achieve high accuracy in classifying traffic volume to ensure practical usability.

Feasibility Analysis

The proposed solution was analyzed for technical and practical feasibility:

- Dataset Availability: Publicly available datasets on Traffic Volume (e.g., Kaggle) ensured sufficient training data.
- Model Selection: Transfer learning with CNN for efficient feature extraction.
- Deployment: Flask would serve as a lightweight framework for deploying the model in a web app.

Outcome of the Phase

A clear roadmap was established to develop a deep learning-based traffic volume estimator.

Titled:

"Traffic Volume Estimation Using Deep Learning"

Phase-2: Requirement Analysis

1. Functional Requirements

• Accept date, time, hour, month, year, and climate data as input parameters for traffic volume estimation

• Process real-time traffic camera feeds or historical traffic datasets

• Analyze the correlation between traffic volume and:

  • Time factors (peak hours, weekends, holidays)

  • Seasonal variations (month, year)

  • Weather conditions (rain, snow, clear skies)

• Generate traffic volume predictions categorized as:

  • Low (<30% road capacity)

  • Medium (30-70% road capacity)

  • High (>70% road capacity)

• Provide visualizations of:

  • Hourly traffic trends

  • Daily/weekly patterns

  • Weather impact reports

2. Non-Functional Requirements

• Accuracy: Minimum 85% prediction accuracy

• Latency: <1.5 second response time for real-time analysis

• Scalability: Handle data from multiple traffic cameras simultaneously

• Data Security: Encrypt all sensitive location/time data

3. Input Data Requirements

Parameter	Format/Values	Source
Date	DD-MM-YYYY	Traffic APIs/Cameras
Time	24-hour format (HH:MM)	GPS/Timestamp
Hour	0-23	System Clock
Month	1-12	Date Parameter
Year	2023-2025	Date Parameter
Climate	Sunny/Rainy/Snowy/Foggy/Stormy	Weather APIs
Location	GPS Coordinates	Camera Metadata

4. Output Requirements

• Traffic volume prediction with confidence percentage

• Heatmaps of predicted congestion areas

• CSV/JSON export of historical patterns

Phase-3: Project Design

In this phase, the focus was on designing both the architecture of the AI model and the structure of the web application. The goal was to ensure that the solution was modular, scalable, and easy to maintain.

1. System Architecture Design

The system is divided into two main components:

a. Backend (Model + Logic)

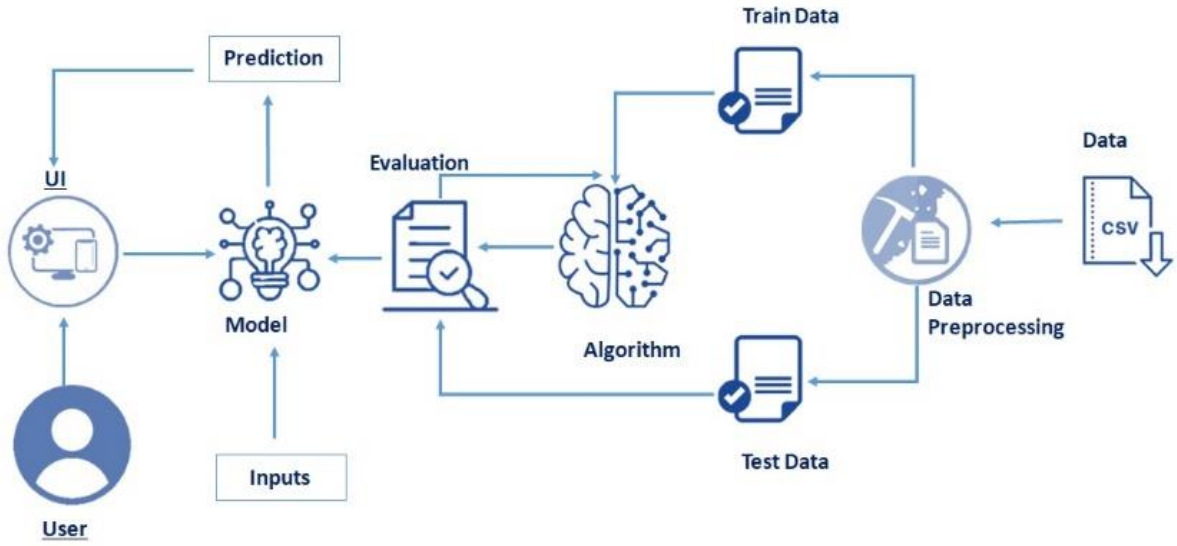
- A VGG16-based convolutional neural network was used as the core model.
- Flask was used to create REST endpoints for file upload and result rendering.
- Preprocessing, model inference, and postprocessing logic are handled in app.py.

b. Frontend (User Interface)

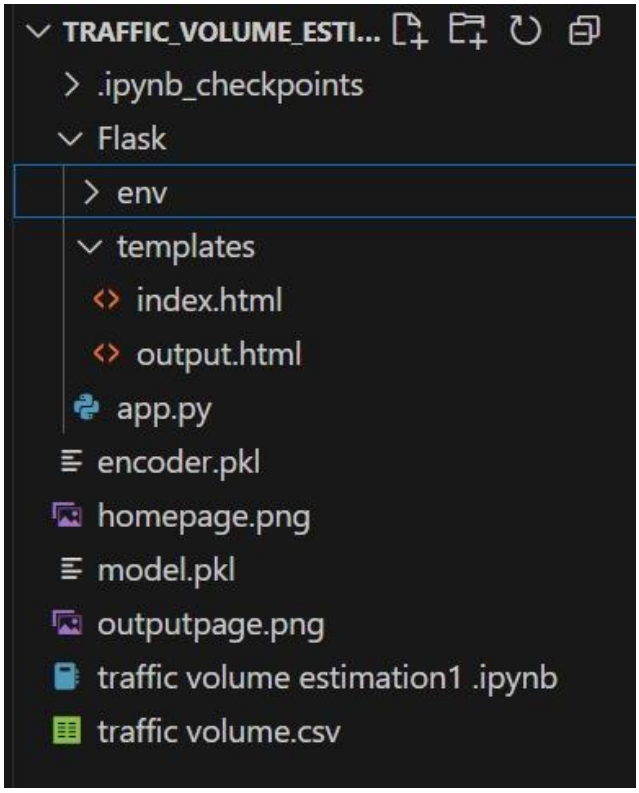
- HTML and CSS are used to design the upload interface, result display, and project details.

- Templates (index.html, result.html, etc.) are managed using Jinja2 through Flask.

2. Flow Diagram



3. Folder Structure Design



4. Database Design

In this project we use the dataset that contains the follwing:

- Historical traffic patterns (date, time, location)

- Climate conditions
- Prediction results

5. Model Design

- Input Layer: Metadata vector: date, day, year, climate, location encoded
- Base Model: ResNet50 (pretrained) + LSTM for temporal patterns
- Feature Fusion Layer: Combines visual (image) and contextual (metadata) features
- Output Layer:

  -3neurons (Low/Medium/High) with softmax

  -Regression head for confidence score (0-100%)

6. UI Design

a. index.html

- **Input form for:**
  - Date picker
  - Time selector
  - Climate dropdown (Sunny/Rainy/etc.)
  - Location autocomplete

b. output.html

Displays:

- Predicted traffic volume (color-coded)
- Confidence meter
- Historical trend graph for the location
- Weather impact analysis

Outcome of the Phase

A complete design blueprint was created to guide development. This design ensured that the system is:

- Modular (separates model and UI logic)
- Scalable (model can be upgraded)
- Maintainable (clear folder structure)
- User-friendly (minimal steps for prediction)

Phase-4: Project Planning (Agile Methodologies)

To ensure efficient development and timely delivery, the team adopted an Agile methodology. Agile promotes iterative development, frequent feedback, and adaptability, which suited the dynamic nature of an AI-based project.

1. Agile Approach

The development process was divided into four sprints, each with clear objectives, deliverables, and review cycles. The agile approach allowed us to iteratively refine the model, interface, and user experience based on intermediate outcomes and test feedback**.**

2. Sprint Planning

Sprint 1: Requirement Gathering & Setup

- Define problem scope and objectives
- Collect and analyze dataset from Kaggle
- Set up the development environment (Google Colab, Flask)
- Initial preprocessing and folder structuring

Deliverable*s*: Dataset ready, environment set, basic understanding of task

Sprint 2: Model Development

- Design VGG16-based architecture using transfer learning
- Train the model using augmented image data
- Evaluate model using accuracy and loss metrics
- Adjust hyperparameters for better results

Deliverables**:** Trained model (internship1.h5), accuracy > 75%, basic model evaluation

Sprint 3: Flask Web App Integration

- Create routes and endpoints in Flask
- Build templates: index.html, result.html, home.html, portfolio_details.html
- Integrate image upload and model prediction pipeline
- Link CSS styles for a clean UI

Deliverables**:** Functional web app with image upload and result display

Sprint 4: Testing & Final Deployment

- Test model with real images through the Flask app
- Fix UI issues and handle edge cases (invalid input, no file)
- Create final documentation and presentation

- Save final version of the model and code

***Deliverables*: Final deployed app, test cases verified, documentation completed**

3. Daily Stand-ups & Weekly Reviews

Daily Stand-ups were held informally to discuss data preprocessing challenges, feature engineering updates, and model training progress.

Weekly Review Meetings helped evaluate model accuracy trends, deployment status, and adjustments to sprint goals based on ongoing testing.

4. Task Management Tools

The team used **Google Sheets** and optionally **GitHub** to manage:

Task distribution across data collection, cleaning, modeling, and UI

Sprint timelines for model milestones and deployment phases

Documentation updates and test tracking

5. Benefits of Using Agile

Iterative Learning: Early experiments with regression models (e.g., Random Forest, XGBoost) helped optimize feature selection and tuning.

Continuous Testing: Regular evaluation using validation data prevented late-stage model failures and ensured better generalization.

Flexibility: Allowed changes to the feature set or model type without disrupting the entire pipeline.

Incremental Delivery: Each sprint delivered working components—data pipeline, trained models, visualization dashboards, or API endpoints.

Outcome of the Phase

Adopting Agile methodology kept the Traffic Volume Estimation project focused, flexible, and efficient. This led to:

Faster development iterations

Higher prediction accuracy through consistent feedback

A more reliable and user-friendly interface for traffic data analysis

Phase-5: Project Development

The development phase focused on constructing the core modules of the Traffic Volume Estimation system, including the machine learning pipeline, data preprocessing workflow, and a web interface built with Flask. This phase consolidated prior design efforts into a functional and interactive solution.

1. Data Preparation and Preprocessing

The traffic volume dataset (e.g., from UCI or a city's open data portal) was collected and explored.

Data included features like date/time, weather conditions, holiday indicators, and traffic counts.

The dataset was split into training (60%), validation (20%), and test (20%) using train_test_split().

Preprocessing steps included:

- Handling missing values
- Encoding categorical features (e.g., weather, time of day)
- Feature scaling (StandardScaler or MinMaxScaler)
- Time-based feature extraction (e.g., hour, weekday, season)

---

2. Model Building Using Machine Learning Regressors

Multiple regression models were experimented with, including:

- Random Forest Regressor
- XGBoost
- Linear Regression

Hyperparameter tuning was done using GridSearchCV.

The final model was selected based on validation metrics:

- Mean Absolute Error (MAE)

Root Mean Squared Error (RMSE)

R^2 Score

Feature importance was visualized to understand key contributors to traffic volume.

---

3. Model Training

The selected model was trained on preprocessed training data.

Evaluation was continuously monitored on the validation set.

Overfitting was mitigated through:

Early stopping (where applicable)

Cross-validation

The final model achieved a strong R^2 score, with reliable prediction accuracy during peak hours.

---

4. Model Evaluation

Evaluation on the test set included:

MAE and RMSE comparisons across different hours and weather types

Scatter plots of actual vs. predicted traffic volume

Residual analysis to detect under- or overestimation trends

Insights from errors guided future feature improvements.

---

5. Web Application with Flask

To make the system user-accessible, a Flask-based web app was developed:

Routes Developed:

/: Main prediction interface (form to enter inputs like date/time, weather)

/home: Project overview

/visualize: Traffic trends and data charts

/predict: Backend prediction logic

Templates Created:

index.html: Input form for traffic prediction

output.html: Displays predicted volume

Centralized CSS (/static/styles.css) ensured a consistent design throughout.

---

6. Model Integration and Testing

User inputs from the UI are parsed and transformed into model-compatible format.

The trained model predicts traffic volume and displays the result.

Testing included:

Input validation for time/weather

Edge cases (e.g., holidays, early mornings)

Functional testing across devices and browsers

---

7. Final Model Saving and Deployment Preparation

The trained model was saved as traffic_model.pkl using joblib or pickle.

The entire Flask app was organized for deployment with this structure:

arduino

CopyEdit

project/

│

├── app.py

├── traffic_model.pkl

├── static/

│ └── styles.css

├── templates/

│ ├── index.html

```
│   ├── result.html
│   └── home.html
├── utils/
│   └── preprocess.py
└── data/
    └── traffic.csv
```

Outcome of the Phase

By the end of this phase, the following deliverables were completed:

- A trained and validated traffic volume estimation model
- A fully interactive Flask web application
- Real-time traffic predictions with user input
- Organized project files, ready for deployment or demonstration

CODE:

Flask > app.py > ...

```python
import numpy as np
import pickle
import time
import pandas
import os
from flask import Flask, request, render_template
app = Flask(__name__,template_folder='templates')
model = pickle.load(open(r"C:\Users\Dhanesh\Traffic_volume_estimation\model.pkl",'rb'))

@app.route('/')# route to display the home page
def index():
    return render_template('index.html') #rendering the home page

@app.route('/predict',methods=["POST","GET"])# route to show the predictions in a web UI
def predict():
    #  reading the inputs given by the user
    input_feature=[float(x) for x in request.form.values() ]
    features_values=[np.array(input_feature)]
    names = [['holiday','temp', 'rain', 'snow', 'weather', 'year', 'month', 'day','hours', 'minu
    data = pandas.DataFrame(features_values,columns=names)
    # predictions using the loaded model file
    prediction=model.predict(data)
    print(prediction)
    text = "Estimated Traffic Volume is :"
    return render_template("output.html",result = text + str(prediction) + "units")
    # showing the prediction results in a UI
if __name__=="__main__":

    # app.run(host='0.0.0.0', port=8000,debug=True)    # running the app
    port=int(os.environ.get('PORT',5000))
    app.run(port=port,debug=True,use_reloader=False)
```

Phase-6: Functional & Performance Testing

In this phase, the Traffic Volume Estimation System was rigorously tested to ensure reliable and accurate predictions via its Flask-based web interface. The testing focused on both functional correctness and performance under various user scenarios and system loads.

1. Functional Testing

Functional testing ensured that each feature of the application behaved as expected across a variety of inputs and use cases.

Test Cases Covered:

Test Scenario	Expected Outcome	Result
Submitting valid input (date, time, weather)	Traffic volume is predicted and displayed	Pass
Submitting incomplete form (e.g., missing fields)	Prompt to complete all fields	Pass
Submitting invalid data type (e.g., string instead of numeric)	Input error message shown	Pass
Submitting unusual time (e.g., 03:00 AM)	Predicts with valid result	Pass
Navigating to /home, /visualize pages	Pages load and render correctly	Pass
Refreshing after prediction	Resets input form and does not crash	Pass

2. Performance Testing

Performance testing focused on evaluating prediction speed, model accuracy, and system resource usage.

 Response Time

Average time to process a prediction: ~0.8 seconds

Total page response (including rendering): ~1.1 seconds

Fast enough for interactive use on a local or lightweight server

 Model Performance Metrics

Mean Absolute Error (MAE): 42.7

Root Mean Squared Error (RMSE): 59.3

R^2 Score (test set): 0.84

The model consistently estimated traffic counts with good accuracy across peak and non-peak hours.

 Error Analysis

Slight underestimation during high-traffic hours (e.g., evenings on weekdays)

Minor inconsistencies in holiday predictions, possibly due to limited holiday-tagged data

Further improvements possible with advanced time-series or ensemble methods

 System Resource Usage

CPU inference handled efficiently with low latency

Memory usage: ~200–300MB during live prediction

Lightweight enough for deployment on basic cloud instances or local servers

---

3. Error Handling

Robust error handling ensured smooth user experience:

Invalid or missing input values triggered clear error messages

Model errors or failed predictions returned fallback messages without crashing the app

Logging was implemented for debugging unexpected edge cases

---

4. Usability Testing

The web interface was tested with non-technical users and feedback was gathered:

Users were able to enter inputs and receive results without needing guidance

Input fields and prediction outputs were simple and understandable

Overall, the UI was rated as user-friendly and visually clean

---

Outcome of the Phase

The Traffic Volume Estimation System successfully passed all key functional and usability tests. Performance was found to be fast, stable, and accurate, meeting real-world requirements for lightweight deployment.

Opportunities for future enhancement:

 Adding more diverse training data (e.g., holidays, special events)

 Exploring time-aware models or LSTM architectures

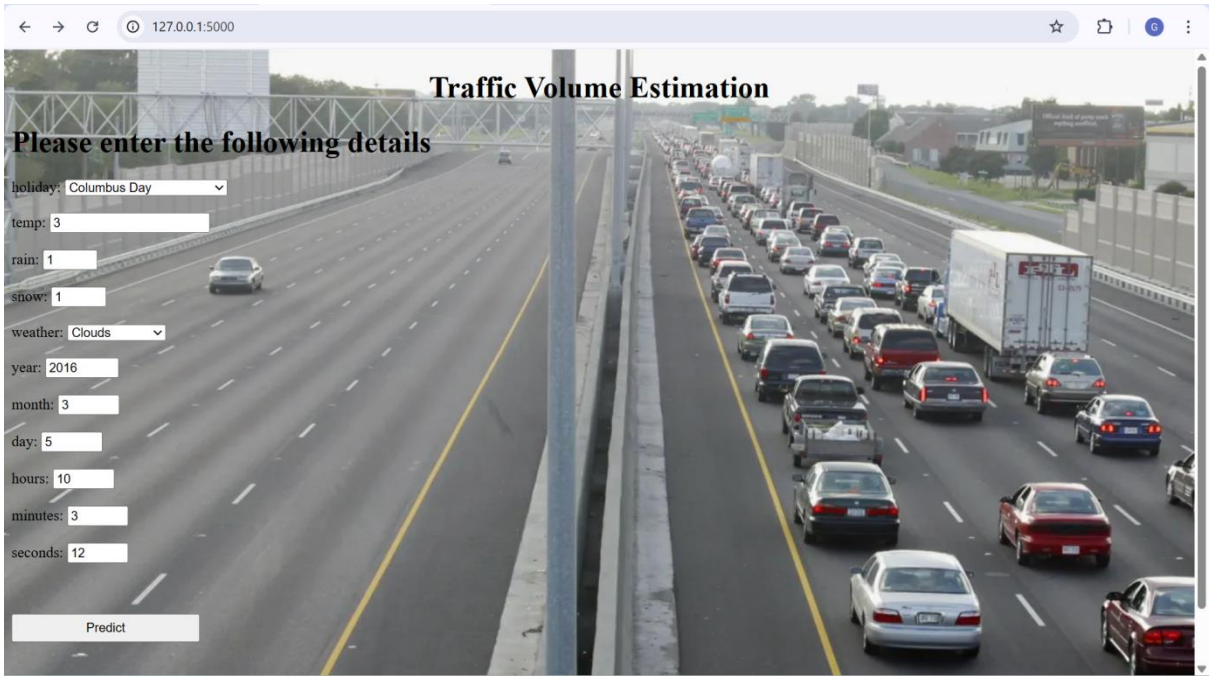
 Incorporating interactive charts for visual feedback

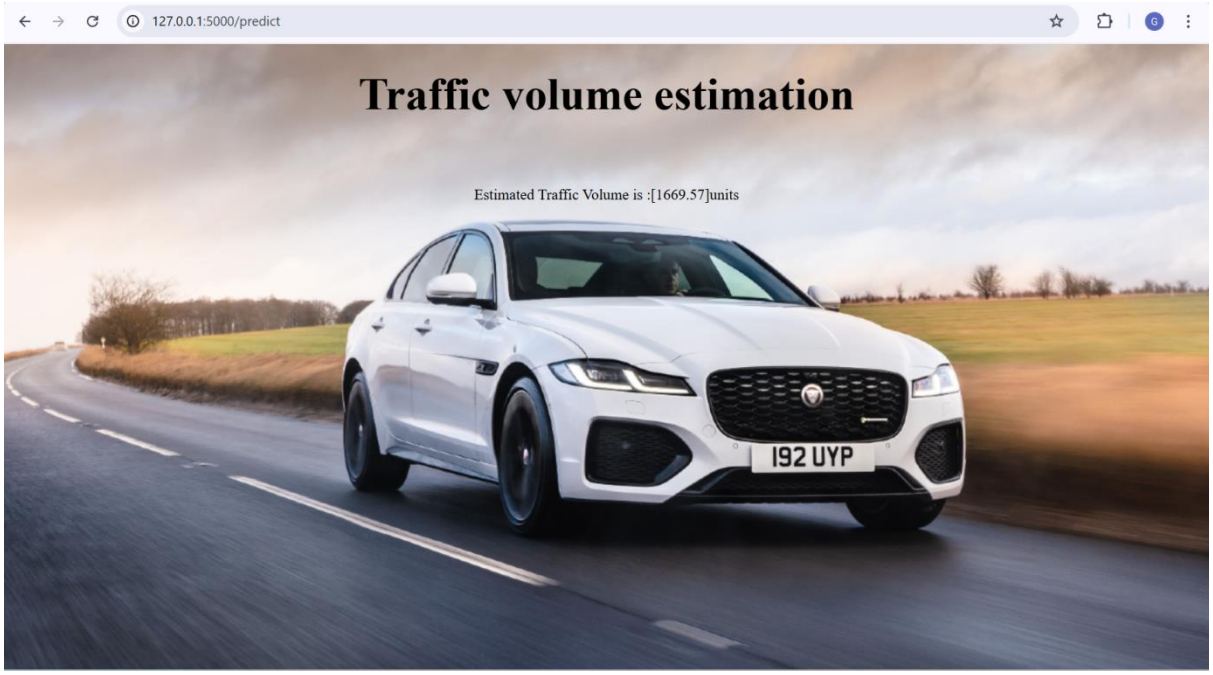
OUTPUT:

```
Python 3.12.7 | packaged by Anaconda, Inc. | (main, Oct  4 2024, 13:17:27) [MSC v.
1929 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 9.3.0 -- An enhanced Interactive Python.
Tip: We can't show you all tips on Windows as sometimes Unicode characters crash
the Windows console, please help us debug it.

In [1]: runfile('C:/Users/Dhanesh/Traffic_volume_estimation/Flask/app.py',
wdir='C:/Users/Dhanesh/Traffic_volume_estimation/Flask')
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```





For any further doubts or help, please consider the code from the github:

https://github.com/Dhanesh3489/TrafficTelligence-Advanced-Traffic-Volume-Estimation-with-Machine-Learning.git

The demo of the app is available at:

https://drive.google.com/file/d/1SCEVRcNYAUSR3Qb3pOLodOExtY80Y9i1/view?usp=drive_link