VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Winter Semester 2024 – 2025

Smart Traffic Simulator

A Comprehensive Project on Real-Time Traffic Simulation

Team Members

Registration Number	Name
24MCA1018	Sivas
24MCA1031	Gokulnath

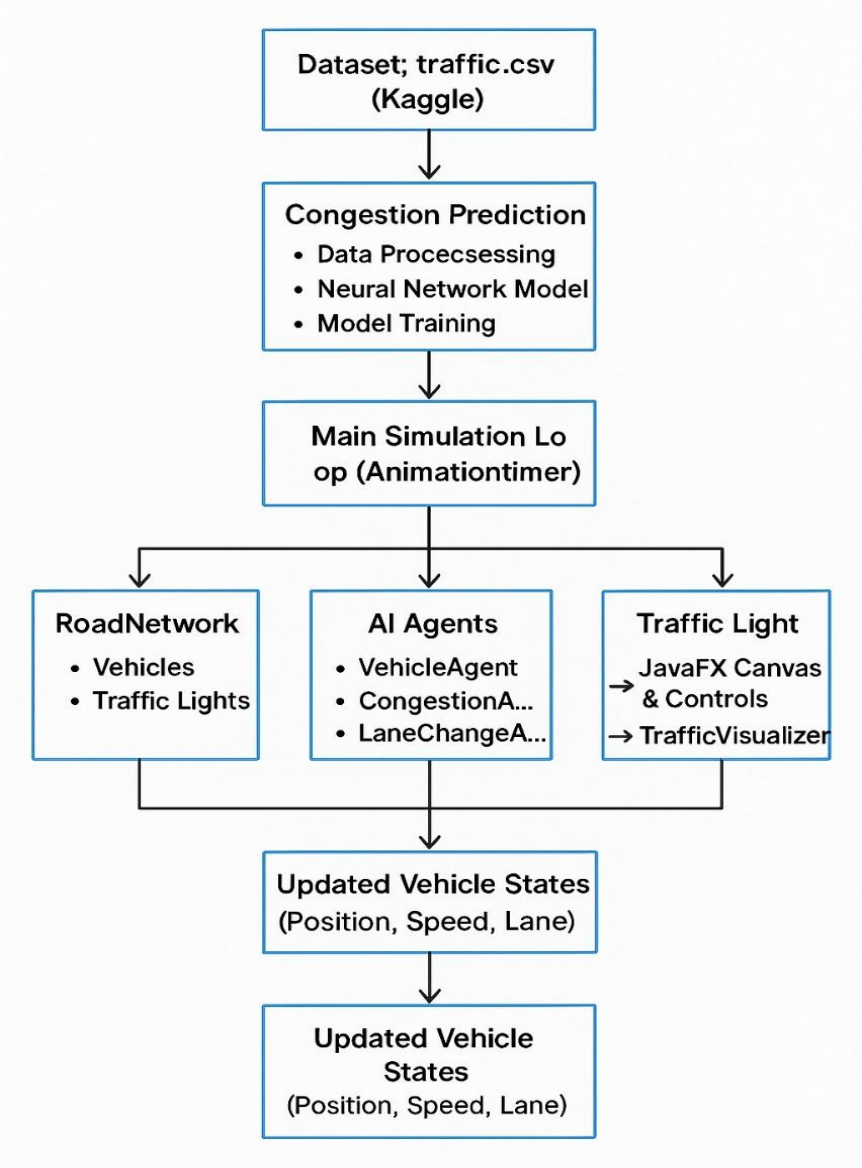
AIM:

To develop an intelligent Smart Traffic Simulator system that uses Artificial Intelligence (AI), Machine Learning (ML), and Neural Networks to efficiently manage traffic flow, reduce congestion, and improve road safety by simulating real-time scenarios.

Abstract:

The Smart Traffic Simulator is an AI-powered system designed to simulate and optimize traffic flow in real-time. It mimics real-world traffic scenarios using intelligent agents and machine learning to reduce congestion, improve road safety, and manage traffic signals efficiently. By integrating artificial intelligence and real-time decision-making, the simulator helps urban planners and traffic authorities visualize, analyze, and enhance city traffic systems.

Architecture Diagram:



Workflow:

Dataset Input (traffic.csv):
The simulation begins with the historical traffic dataset from Kaggle. The CSV file contains fields such as DateTime and Vehicles that are used in data preprocessing.

TrafficPredictor (ML Component):

- This module reads and preprocesses the dataset (e.g., extracting the hour from DateTime, normalizing vehicle counts and speed).
- A neural network is trained with these inputs to learn a relationship between traffic features and congestion.
- The output is a congestion prediction (a value between 0 and 1), which reflects conditions such as rush-hour (informed by the DateTime feature) and vehicle density.

Main Simulation Loop (AnimationTimer):

- The simulation loop (in MainApp.java) continuously updates the simulation state.
- It fetches the congestion value from the ML module on each frame, which is then used to adjust vehicle behavior via the AI agents.

 RoadNetwork:

- Maintains the central state including the list of vehicles and traffic lights.
- Provides the current simulation environment for the AI agents.

 AI Agents:

- **VehicleAgent:** Implements basic vehicle rules (move if no red light is close).
- **CongestionAwareAgent:** Adjusts vehicle speed based on the congestion value.
- **LaneChangeAgent:** Monitors for slow vehicles ahead and changes lanes to avoid traffic jams.
- These agents update the vehicle states (position, speed, lane) according to both the environment and the predicted congestion.

 Traffic Light:

- A component within the RoadNetwork that cycles between states (GREEN, YELLOW, RED) to influence vehicle movement.
- AI agents use the state of the traffic light as part of their decision-making process.

 UI Layer (JavaFX):

- **TrafficVisualizer:** Draws the current state of the simulation on a canvas (vehicles, grid, traffic lights, congestion bar).
- **Controls:** Provide user interactivity for adding vehicles, toggling the light, and adjusting parameters like spawn rate and light timing.
- The visual output updates based on the current simulation state maintained by the RoadNetwork and modified by the AI agents.

 Updated Vehicle States:

- After the AI agents update the vehicle attributes, the new positions and speeds feed back into the simulation.
- These updated states are then rendered on the UI, completing the cycle

AI Components

The AI in this project is implemented through individual agent classes that control how each vehicle behaves in the simulation. Their logic determines whether a vehicle moves

normally, slows down under high congestion, or changes lanes if obstructed by slower traffic.

Agent: Role-based AI agent

- **VehicleAgent.java**
    - **Purpose:**
      Implements basic vehicle behavior by checking if a vehicle is near a red traffic light.
    - **Logic:**
        - If the vehicle is within 20 pixels of a traffic light and the light is not green, the vehicle stops.
        - Otherwise, the agent calls the vehicle's move() method.
    - **Role:**
      Provides a baseline agent that enforces traffic rules without dynamic speed adjustments.
- **CongestionAwareAgent.java**
    - **Purpose:**
      Adjusts the vehicle's behavior based on congestion levels.
    - **Logic:**
        - Similar to the VehicleAgent regarding red light detection.
        - When congestion is high (above a threshold of 0.8), the agent scales down the vehicle's moving speed by applying a factor (e.g., reducing speed by half).
    - **Role:**
      Helps simulate realistic driving where heavy congestion affects vehicle speed.
- **LaneChangeAgent.java**
    - **Purpose:**
      Introduces lane-changing behavior by considering the relative position of other vehicles.
    - **Logic:**
        - Scans for slower vehicles ahead in the same lane within a close distance.
        - If a slow vehicle is detected (vehicle ahead less than 30 pixels away), it alters the vehicle's vertical position (y-coordinate) to attempt a lane change.

- **Role:**
  Adds an extra layer of dynamics by enabling vehicles to avoid traffic jams by switching lanes, which can lead to smoother overall traffic flow.

The AI agents simulate varied driver behaviors, from basic adherence to traffic signals to adaptive responses to congestion and lane dynamics. This modular approach allows for future enhancements—for instance, more sophisticated decision-making or context-aware lane selection.

Machine Learning (ML) Component

The machine learning part is dedicated to predicting congestion levels based on real-world data. The predictor uses data from a CSV file (from Kaggle's Traffic Prediction Dataset) to learn patterns in traffic behavior. The integration of ML helps the simulation adjust vehicle behavior dynamically based on the predicted level of congestion.

- **TrafficPredictor.java**
  - **Purpose:**
    Trains and uses a neural network model to output a congestion value (ranging from 0 to 1) based on several traffic inputs.
  - **Data Input:**
    - **CSV Data:**
      Reads from a CSV file with fields: DateTime, Junction, Vehicles, and ID.
    - **Feature Extraction:**
      - **Vehicles:** The number of vehicles, normalized by a maximum count (100).
      - **DateTime:** The hour is extracted from the timestamp. This helps simulate rush-hour effects by adjusting the base speed and adding a time weight.
      - **Speed:** A computed value derived from the base speed (30 mph during rush hours, 50 mph otherwise) and vehicle count.
      - **isGreen:** A binary flag indicating the traffic light status (randomly generated during training, as the dataset does not provide this).
  - **Neural Network Model:**

        - **Architecture:**
            - Input layer: 3 features (normalized vehicle count, normalized speed, and isGreen flag).
            - One hidden dense layer with 4 neurons using ReLU activation.
            - Output layer: 1 neuron with sigmoid activation which outputs a congestion value.
        - **Training:**
            - The dataset is split into training (80%) and testing (20%) samples.
            - Trained over 100 epochs using the Adam optimizer and Mean Squared Error (MSE) as the loss function.
    - **Prediction:**
        - At runtime, the predictor computes the congestion based on the current vehicle count, average speed, and the traffic light status.
        - This congestion value is then used by the AI agents to alter vehicle behavior (e.g., slowing down when congestion is high).

The ML component adds a layer of adaptive realism to the simulation by integrating historical patterns (like rush hours) into a dynamic prediction model. This prediction is crucial for simulating real-world traffic conditions and adjusting the AI agents accordingly.

Java (Core Application & UI)

The main backbone of the project is built using Java with JavaFX for the graphical user interface (GUI). The Java/JavaFX portion ties together the simulation's models, AI agents, and machine learning predictions into one interactive environment.

- **MainApp.java (UI Layer)**
    - **Purpose:**
      Acts as the entry point for the simulation, orchestrating the initialization, simulation loop, and user interactions.
    - **Responsibilities:**
        - **Initialization:**
            - Constructs the road network and creates initial vehicles and a traffic light.

        - Instantiates the TrafficPredictor and triggers data generation and model training.
        - Creates an assortment of AI agents (VehicleAgent, CongestionAwareAgent, LaneChangeAgent), assigning them to vehicles.
    - **User Controls:**
        - Provides interactive controls through a control pane (buttons to add vehicles, toggle the traffic light, and sliders to adjust spawn rate and light toggle timing).
    - **Animation Loop:**
        - Uses an AnimationTimer to update the simulation approximately 60 times per second.
        - Handles toggling of traffic lights and spawning of new vehicles.
        - Invokes the ML predictor to compute congestion, then updates all agents and visualizes the state.
    - **Status Display:**
        - The status bar displays real-time metrics such as congestion level and current vehicle count.

- **TrafficVisualizer.java (Rendering)**
    - **Purpose:**
      Draws the current state of the simulation onto a JavaFX canvas.
    - **Visual Features:**
        - **Background and Grid:**
            - A dark gray background with a grid layout to represent lanes and road divisions.
        - **Traffic Lights:**
            - Drawn as colored squares (green or red) to indicate the current state of the traffic light.
        - **Vehicles:**
            - Rendered as ovals with ID labels. Their color (green or dark red) reflects the congestion condition.
        - **Congestion Bar:**
            - A visual indicator that shows the current congestion as a colored bar.

- **Supporting Resources (styles.css)**
    - **Purpose:**
      Defines the styling of the UI components to maintain a clean and consistent look.
    - **Details:**
        - Sets the background, button styles, slider aesthetics, and label fonts.

The Java/JavaFX portion of the project is responsible for tying all components together. It ensures the simulation is visually represented and interactive. From setting up the models to handling real-time updates and user controls, it provides the framework for the entire project, allowing AI and ML components to interact dynamically during the simulation.

Code:

Output:


Smart Traffic Simulator
Controls
Add Vehicle
Toggle Light
Spawn Rate
0
5
Spawn Rate (vehicles/sec): 1.6
Light Timing
2
10
Light Toggle (sec): 5.0
V43
V19
V20
V26
V21
V15
V22
V40
V23
V41
V42
V36
V24
V17
V32
Congestion: 0.41 Vehicles: 43