A Major Project Report

On

A HIERARCHICAL NETWORK BASED METHOD FOR PREDICTING DRIVERS TRAFFIC VIOLATION USING MACHINE LEARNING

Submitted to CMREC (UGC Autonomous)

In partial Fulfilment of the requirements for the Award of Degree

Of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING (AI & ML)

Submitted
By

JAY PRAKASH NARAYAN REDDY	**(228R1A6694)**
P. RAMADEVI	**(228R1A66B0)**
A. KIRAN KUMAR	**(238R5A6608)**
B. BHARATH KUMAR	**(228R1A6669)**

Under the Esteemed guidance of

Dr. A. PRAMOD KUMAR

Associate Professor, Department of CSE(AI&ML)

Department of Computer Science and Engineering (AI&ML)

CMR ENGINEERING COLLEGE

(UGC AUTONOMOUS)

(Accredited by NAAC & NBA Approved by AICTE, New Delhi, Affiliated to JNTU Hyderabad)

(Kandlakoya, Medchal Road, Medchal-Malkajgiri.Dist, Hyderabad-501 401)

(2025-2026)

CMR ENGINEERING COLLEGE

(UGC AUTONOMOUS)

(Accredited by NAAC & NBA, Approved by AICTE, New Delhi, Affiliated to JNTU, Hyderabad, Kandlakoya,

Medchal Road, Hyderabad - 501401)

Department of Computer Science & Engineering (AI&ML)

CERTIFICATE

This is to certify that the Major project entitled "**A HIERARCHICAL NETWORK BASED METHOD FOR PREDICTING DRIVERS TRAFFIC VIOLATION USING MACHINE LEARNING**" is a Bonafide work carried out

by

JAY PRAKASH NARAYAN REDDY	**(228R1A6694)**
P. RAMADEVI	**(228R1A66B0)**
A. KIRAN KUMAR	**(238R5A6608)**
B. BHARATHKUMAR	**(228R1A6669)**

in partial fulfillment of the requirement for the award of the degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING (AI&ML) form CMR Engineering College, under our guidance and supervision.

The results presented in this Major project have been verified and are found to be satisfactory. The results embodied in this Major project have not been submitted to any other university for the award of any other degree or diploma.

Internal Guide	**Major Project Coordinator**	**Head of the Department**
Dr. A. Pramod Kumar	Mr. G. Venkateswarlu	Dr. Madhavi Pingili
Associate Professor	Assistant Professor	Professor & HOD
Department of	Department of	Department of
CSE (AI&ML)	CSE (AI&ML)	CSE (AI&ML)

External Examiner: _____

DECLARATION

This is to certify that the work reported in the present Major project entitled "**A HIERARCHICAL NETWORK BASED METHOD FOR PREDICTING DRIVERS TRAFFIC VIOLATION USING MACHINE LEARNING**" is a record of Bonafide work done by us in the Department of Computer Science and Engineering (AI & ML), CMR Engineering College. The reports are based on the Major project work done entirely by us and not copied from any other source. We submit our Major project for further development by any interested students who share similar interests to improve the Major project in the future.

The results embodied in this Major project report have not been submitted to any other University or Institute for the award of any degree or diploma to the best of our knowledge and belief.

JAY PRAKASH NARAYAN REDDY (228R1A6694)

P. RAMADEVI (228R1A66B0)

A. KIRANKUMAR (238R5A6608)

B. BHARATH KUMAR (228R1A6669)

ACKNOWLEDGEMENT

We are extremely grateful to **Dr. A. Srinivasula Reddy**, Principal and **Dr. Madhavi Pingili,** Professor & HOD**, Department of CSE(AI&ML), CMR Engineering College** for their constant support.

We are extremely thankful to **Dr. A. PRAMOD KUMAR**, Associate Professor, Internal Guide, Department of CSE(AI&ML), for his constant guidance, encouragement and moral support throughout the Major project.

We will be failing in duty if we do not acknowledge with grateful thanks to the authors of the references and other literatures referred in this Major Project.

We thank **Mr. G. Venkateswarlu**, Major Project Co Ordinator for his constant support in carrying out the project activities and reviews.

We express our thanks to all staff members and friends for all the help and co-ordination extended in bringing out this Major project successfully in time.

Finally, We are very much thankful to our parents who guided us for every step.

JAY PRAKASH NARAYAN REDDY (228R1A6694)

P. RAMADEVI (228R1A66B0)

A. KIRAN KUMAR (238R5A6608)

B. BHARATHKUMAR (228R1A6669)

CONTENTS

ABSTRACT

Traffic violations pose a significant threat to road safety worldwide, leading to numerous accidents, injuries, and fatalities each year. Predicting which drivers are more likely to commit traffic violations can help authorities implement preventive measures and targeted interventions. This project presents a machine learning-based approach that uses hierarchical network methods to predict driver traffic violations. The system works by analyzing various factors related to drivers, including their demographic information, driving history, previous violations, vehicle characteristics, and behavioral patterns. Unlike traditional flat prediction models, our hierarchical network approach organizes these features in multiple layers, capturing both simple and complex relationships between different factors. For example, age and experience might form one layer, while violation history and driving patterns form another, with the network learning how these layers interact to influence violation probability. The hierarchical structure allows the model to understand pattern sat different level so abstraction. It can identify direct influences, such as how past speeding tickets correlate with future violations, as well as indirect relationships, like how the combination of driver age, time of day, and weather conditions together affect violation likelihood. This multi-level understanding provides more accurate predictions compared to simpler models that treat all features equally. The machine learning model is trained on historical traffic violation data, learning to recognize patterns that distinguish high-risk drivers from low-risk ones. Once trained, the system can predict the likelihood of future violations for individual drivers, enabling traffic authorities to take proactive measures such as targeted awareness campaigns, mandatory training programs, or increased monitoring for high-risk individuals. The ultimate goal is to reduce traffic violations, enhance road safety, and save lives through data-driven decision making.

Keywords: Driver Traffic Violations; Graph Convolutional Neural Network; Predictive Modeling; Intelligent Transportation Systems; Behavioral Analysis; Road Safety; Deep Learning.

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Introduction of the project

Road traffic accidents have long been recognized as one of the most critical global public safety challenges, leading to immense human suffering and economic loss every year. According to the World Health Organization (WHO), millions of people are injured or killed annually as a direct result of unsafe driving behavior and repeated traffic violations, which are among the most significant causes of road accidents worldwide. Despite the continuous advancement of vehicle technology, road infrastructure, and enforcement of traffic regulations, the frequency of accidents remains alarmingly high especially in developing nations where driving conditions, enforcement mechanisms, and drive reduction often vary widely. Research consistently shows that human - related factors including a driver's behavior, level of experience, psychological state, and decision-making ability are more influential in accident causation than mechanical or environmental factors. Therefore, developing predictive systems that can identify high-risk driving behavior and potential traffic violations in advance has become a vital component of Intelligent Transportation Systems (ITS) and modern traffic safety research. Earlier attempts to predict driver traffic violations have largely relied on machine learning (ML) algorithms such as Support Vector Machines (SVM), Decision Trees (DT), and Random Forests (RF). These models learn from structured driver-related data such as demographics, experience, and personality attributes to establish statistical relationships between driver characteristics and violation occurrences. While such models provide a fundamental understanding of driver behavior, they are limited in representing complex and non-linear relationships among multiple interdependent factors. To address these limitations, researches have turned toward deep learning techniques, particularly Convolutional Neural Networks (CNN) and Long Short- Term Memory (LSTM) networks, which have shown strong potential for analyzing high-dimensional and sequential data. The hierarchical CNN–LSTM frame work shave achieved notable success by extracting both spatial and temporal features from driver behavior data. However, these networks operate on Euclidean structures, such as fixed grids or sequential time-series, and are therefore unable to fully represent non-Euclidean relationships, where factors such as driving habits, emotional states, and environmental conditions interact irregularly and dynamically.

1.2 Problem statement

Modern traffic systems generate large volumes of data from sensors, GPS enabled vehicles, and connected road infrastructure. However, predicting traffic conditions accurately remains a major challenge due to complex road networks, dynamic driver behaviour, multi-scale spatial relationships, and rapidly changing temporal patterns. Traditional prediction models fail to capture these hierarchical dependencies across road segments, intersections, and regional networks. As a result, drivers experience unexpected congestion, longer travel times, increased fuel consumption, and higher accident risks.

Transportation authorities also struggle with real-time traffic management, congestion reduction, and route optimization.

1.3 Objectives

The specific objectives of this project are as follows:

1. To analyze existing driver violation prediction methods and identify their limitations in handling complex and non-Euclidean data relationships.
2. To design a graph-based data representation where driver attributes and violation types are represented as interconnected nodes and edges.
3. To implement a GCNN architecture capable of learning spatial and temporal dependencies through graph convolution and message passing.
4. To evaluate the proposed system using bench mark datasets and compare its performance against traditional machine learning and deep learning models.

1.4 Scope of the project

The scope of this study covers the analysis, design, and implementation of a predictive model for driver traffic violations using GCNN. The research focuses on constructing a graph-based representation of driver data that includes demographic, psychological, and behavioral features. The system is designed to process real-world datasets similar to those collected from traffic management authorities to predict multiple categories of violations, such as speeding, illegal turns, or distracted driving. The project also includes a comparative performance evaluation between the proposed GCNN model and existing methods such as SVM, DT, RF, and CNN–LSTM. Furthermore, the scope extends to assessing the interpretability, generalization ability, and robustness of the GCNN under conditions of data sparsity, imbalance, and temporal variation. However, the project does not cover real-time vehicle sensor integration or external environmental data such as road or weather conditions.

2. LITERATURE SURVEY

1. **National Traffic Data Reports (2023).** This study presents a statistical report on national road traffic accidents using data collected from traffic databases. It provides detailed insights into accident rates, patterns, and trends for the year 2023. The findings serve as a foundational reference for analyzing traffic safety and developing preventive strategies.

2. **Statista (2023), "Road Accidents in Europe Statistics & Facts"**. This report provides a comprehensive statistical analysis of road accidents across European countries. It utilizes data aggregation and visualization techniques to compare accident trends, fatalities, and safety measures. The study offers valuable insights for understanding regional differences in traffic safety.

3. **S. A. Elsagheer Mohamed (2019), "Automatic Traffic Violation Recording and Reporting Using VANET".** This work proposes a real-time traffic violation detection and reporting system using Vehicular Ad-hoc Networks (VANET) and IoT-based monitoring. The system minimizes human intervention and enhances the efficiency of traffic management, thereby contributing to accident reduction.

4. **Y. Xing et al. (2021), "Energy-Aware Deep Learning for Driving Behavior Analysis and Prediction".** This study introduces an energy-efficient deep learning model that analyzes and predicts driving behavior using connected vehicle data. The model improves traffic safety and supports smart mobility by identifying risky driving patterns.

5. **X. Zhu et al. (2022), "Driver Violation Prediction Using Neural Networks".** This research develops an Artificial Neural Network (ANN)-based model for predicting driver violations. The approach enables proactive traffic safety measures by identifying potential violations before they occur.

6. **G. Baicang et al. (2020), "Risk Prediction Model for Cognitive Distracted Driving"**. This study focuses on predicting risky driving behaviors caused by cognitive distractions using machine learning and behavioral analysis techniques. The proposed model improves the identification of unsafe driving patterns.

7. **H. Seo et al. (2022), "Driving Risk Assessment Using NMF with Driving Behavior Records"**. This work presents a risk assessment framework using Non-Negative Matrix Factorization (NMF). It analyzes large-scale driving behavior data to identify patterns associated with risky driving, enhancing traffic safety analysis.

8. **P. P et al. (2021), "Simulation Model for Detecting City Traffic Violations Using Intelligent Agents".** This study proposes a simulation-based model using distributed intelligent agents for detecting and reporting traffic violations. The approach demonstrates the potential of multi-agent systems in urban traffic management.

9. **D. Wang et al. (2023), "Dual-Transformer Model for Traffic Accident Prediction".** This research introduces a transformer-based deep learning model with a dual-branch architecture. It achieves high accuracy in predicting traffic accidents by analyzing spatio-temporal data.

10. **H. J. Cho and M. C. Hwang (2005), "Stimulus-Response Model of Day-to-Day Network Dynamics".** This foundational study proposes a stimulus-response model to explain variations in traffic network dynamics. It highlights how driver behavior and environmental stimuli influence traffic patterns.

Table no. 2 Literature Review Summary

Focused Area / Title	Key Findings	Reference
National Road Traffic Accident Statistics **[1]**	Provides statistical insights into accident rates and patterns for 2023 using national traffic databases. Serves as a baseline reference for traffic safety analysis.	"Statistical Report on National Road Traffic Accidents," National Traffic Database, 2023.
Road Accidents in Europe – Statistical Analysis **[2]**	Presents comparative analysis of accident trends, fatalities, and preventive measures across European countries using aggregated data visualization.	Statista, "Road Accidents in Europe Statistics & Facts," 2023.
Traffic Violation Detection using VANET **[3]**	Proposes a real-time traffic violation detection and reporting system using VANET and IoT technologies, reducing human intervention and improving road safety.	S. A. Elsagheer Mohamed, "Automatic Traffic Violation Recording and Reporting Using VANET," 2019.
Energy-Aware Deep Learning for Driving Behavior Analysis **[4]**	Introduces an energy-efficient deep learning model to analyze and predict driving behavior, enhancing smart mobility and traffic safety.	Y. Xing et al., "Energy-Aware Deep Learning for Driving Behavior Analysis and Prediction," 2021.
Driver Violation Prediction using Neural Networks **[5]**	Develops an Artificial Neural Network model to predict driver violations, enabling proactive traffic safety mechanisms.	X. Zhu et al., "Driver Violation Prediction Using Neural Networks," 2022.

Risk Prediction for Cognitive Distracted Driving **[6]**	Designs a machine learning-based model to predict risky driving behaviors caused by cognitive distractions, improving safety awareness.	G. Baicang et al., "Risk Prediction Model for Cognitive Distracted Driving," 2020.
Driving Risk Assessment using NMF **[7]**	Proposes a Non-Negative Matrix Factorization-based framework to identify risky driving patterns from large-scale behavior data.	H. Seo et al., "Driving Risk Assessment Using NMF with Driving Behavior Records," 2022.
Traffic Violation Detection using Intelligent Agents **[8]**	Develops a simulation-based model using distributed intelligent agents to detect and report traffic violations in urban environments.	P. P et al., "Simulation Model for Detecting City Traffic Violations Using Intelligent Agents," 2021.
Dual-Transformer Model for Traffic Accident Prediction **[9]**	Introduces a transformer-based deep learning model that achieves high accuracy in predicting traffic accidents using spatio-temporal data.	D. Wang et al., "Dual-Transformer Model for Traffic Accident Prediction," 2023.
Stimulus-Response Model for Traffic Dynamics **[10]**	Presents a dynamic model explaining traffic network behavior based on driver responses and environmental stimuli.	H. J. Cho and M. C. Hwang, "Stimulus-Response Model of Day-to-Day Network Dynamics," 2005.

2.1 Review of Existing Papers

In analyzing the factors influencing driver traffic violations, prediction models are typically constructed by examining violation datasets. This process often involves collecting data through questionnaires that ask drivers about their driving actions and psychological states at the time of the violation to extract relevant correlation indicators. With advancements in sensor technology, imaging systems, and positioning devices, it has become feasible to record vehicle trajectories and capture complete driving process data. However, due to legal and cost limitations, obtaining detailed driver characteristic data during violations remains challenging, which reduces the relevance and comprehensiveness of prediction results. Li et al utilized simulated driving technology to build vehicle–road collaboration scenarios and analyzed multidimensional behavioral changes in various situations. Nonetheless, the deviation between simulated and real driving environments hindered the generalization of their findings. Chen and Chen collected natural driving data to develop an experimental database and explored risky driving behaviors along with their associated influencing factors. Deng et al. and Payyanadan et al. [28] demonstrated that reaction time to traffic hazards and the tendency to engage in illegal driving decrease with increasing age and experience. Additionally, an inability to accurately assess and identify roadway risks can lead to reduced cautiousness during driving, thereby heightening the likelihood of violations. Gel mini et al. found that thrill-seeking tendencies and intentional risk-taking behaviors also contribute significantly to traffic violations. From a technological perspective, traffic violations are largely driven by driver behavior, which is studied through the analysis of driving patterns. In the 1950s, early mathematical models describing driving behavior such as the general motor model analyzed linear relationships between vehicles on one-way roads to explain frequent deceleration phenomena. Subsequently, collision avoidance models were developed that incorporated parameters like relative distance, safe distance, and reaction delay time to simulate behaviors leading to potential collisions. These evolved into constraint-based models considering speed, distance, and front–rear vehicle interactions.

In the 1960s, psychological models of driving were introduced, incorporating visual limitations as a key factor affecting environmental judgment. This led to the development of Weidmann's six driving behavior models. Fancher later enhanced these models by integrating psychological reaction thresholds to quantify a driver's perception of a leading vehicle's motion. In the 1990s, Bando

proposed the OV model, which introduced a speed optimization function to account for behavioral delays caused by differences in vehicle speeds. Entering the 21st century, the adoption of machine learning methods has significantly advanced driver behavior prediction. Algorithms such as Support Vector Machines (SVM), ant colony optimization, genetic algorithms, recurrent neural networks, Google LeNet, VGG, and ResNet have been successfully applied in this domain. For instance, Menno extracted two-dimensional facial thermal image features and combined them with a Gaussian mixture model to identify driving behaviors under the influence of alcohol. Yeo and Chuang employed an SVM-based approach to extract EEG signal features that characterize fatigue-related driving behavior across different brain regions. Liu used semi-supervised learning to extract eye and head movement features and further enhanced binary classification accuracy using a semi-supervised extreme learning machine. Masood applied the VGG16 network to analyze behavioral images of drivers and achieved superior results in detecting distracted driving on a small-scale dataset.

Deep learning techniques such as Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) networks possess the technical capability to automatically extract high-level representations from low-level features, effectively characterizing driver traffic violations. Consequently, deep learning-based approaches for predicting driver traffic violations have increasingly gained prominence. Oka fuji et al. developed a behavior recognition framework using a CNN fusion model that achieved strong performance on public datasets; however, the model only captured spatial characteristics while neglecting temporal aspects of driver behavior. Li et al. proposed a behavioral pattern recognition model that integrates LSTM with a recursive neural network, demonstrating its effectiveness for data with temporal dependencies but facing challenges in spatial feature extraction. Similarly, Cura et al. combined CNN and LSTM architectures to extract spatio-temporal fusion features, achieving high recognition accuracy for continuous behaviors. Nonetheless, the use of uniform feature extraction weights limited further improvements in accuracy. The attention mechanism (AT) introduces an adaptive weighting strategy that allows different model components to correlate in a weighted manner, enabling the model to focus on the most relevant parts of the input sequence. Building upon this, Zhu et al. proposed a Bi LSTM-MCNN model incorporating attention for emotion analysis, resulting in improved recognition accuracy. Gao et al. [41] implemented a lightweight network with an attention mechanism to identify key data features effectively. Moreover,

Chen and Gong enhanced the attention module using matrix operations on spatiotemporal interactions and depth wise separable convolutions, integrating it into an existing network to further boost recognition accuracy.

Advancements in computer vision have also significantly contributed to traffic violation prediction. For example, Chen (2015) introduced a seatbelt detection approach using the Gaussian Mixture Model (GMM) and a cascade classifier (Ada boost), though its effectiveness was limited due to high-resolution image requirements. Meng et al. proposed the Ada Vit framework, which integrates image block partitioning, self-attention heads, and transformer blocks to enhance the efficiency of vision transformers, making it suitable for driver behavior reasoning. Li et al. [44] and Talukdera et al. [45] explored multi-scale feature representation in transformer-based models for image classification, offering valuable insights for driver behavior classification. Additionally, Manzari et al. [46] investigated the direct application of vision transformers for traffic sign recognition, assessing their advantages and limitations when used alongside convolutional neural networks.

2.2 Advantages and limitations of existing methods

2.2.1 Advantages

1. **Captures Complex Relationships:**

   GCNN models non-linear and non-Euclidean interactions among multiple factors such as behavior, experience, and environment.

2. **Better Performance on Limited Data:**

   By leveraging graph topology, GCNN generalizes better with smaller or imbalanced datasets.

3. **Improved Interpretability:**

   The learned graph structure and attention weights can reveal key indicator–violation relationships, enhancing explainability.

4. **Flexibility in Feature Representation:**

   The graph-based model adapts dynamically to changing or missing features without strict

dependence on fixed feature grids.

5. **Enhanced Temporal Understanding:**
   When combined with temporal edges or gated aggregation, GCNN efficiently captures time-dependent violation patterns with fewer parameters than LSTM-based models.

2.2.2 Limitations of Existing Methods

Flat/Non-Hierarchical Models

- Treat all features equally ignore the natural hierarchy (driver → vehicle → road → environment)
- Miss multi-level interactions between factors
- Poor generalization across different traffic scenarios

Traditional ML Models (SVM, Decision Trees, Logistic Regression)

- Cannot capture complex non-linear relationships in traffic data
- Struggle with high-dimensional and imbalanced datasets
- No temporal pattern learning miss behavioral trends over time

Deep Learning Models (basic CNN/RNN)

- Require **massive** labeled data scarce in traffic violation datasets
- Act as black boxes lack explainability for legal/policy use
- Not designed for hierarchical feature structures

Statistical/Rule-Based Methods

- Based on manually defined rules cannot adapt to new patterns
- Cannot handle real-world noise and missing data well
- Poor scalability with growing traffic data

7. **Single-Level Feature Extraction**

- Ignore contextual factors (time of day, weather, road type)
- No distinction between short-term and long-term driver behaviour
- Miss inter-driver and intra-driver variability

8. **Class Imbalance Handling**

- Most existing methods fail to properly address skewed violation data (few violators vs. many compliant drivers)
- Leads to biased predictions favouring the majority class

9. **Lack of Interpretability**

- Existing deep models don't explain why a driver is predicted to violate
- Not suitable for real-world deployment in traffic management systems

2.3 Need for Proposed System

The increasing rate of traffic violations and road accidents worldwide has created an urgent demand for intelligent and accurate prediction systems. Existing methods fail to capture the complex, multi-level relationships present in traffic violation data, making it necessary to propose a hierarchical network-based approach that can model these dependencies effectively. Traffic violation behaviour is

influenced by numerous interacting factors such as driver history, vehicle type, road conditions, weather, and time of day. A flat or single-level model cannot adequately represent these relationships, whereas a hierarchical network naturally organizes and processes these factors in a structured, layered manner, leading to more accurate and meaningful predictions. Another critical need arises from the temporal nature of driver behaviour. Drivers do not violate traffic rules randomly their behaviour follows patterns that evolve over time. Existing machine learning models largely ignore these sequential and behavioural trends, resulting in poor prediction performance. The proposed hierarchical system addresses this by learning both short-term and long-term behavioural patterns, enabling it to identify high-risk drivers more reliably before a violation or accident occurs.

Real-world traffic datasets are also highly imbalanced, with violators forming a small minority compared to compliant drivers. This imbalance causes traditional models to produce biased predictions that favour the majority class, rendering them ineffective for practical use. The proposed system incorporates strategies to handle this imbalance effectively, ensuring fair and accurate predictions across all classes. Furthermore, traffic violation prediction systems must be interpretable and transparent, especially when used for legal decisions, policy-making, or law enforcement. Most existing deep learning models function as black boxes, offering no explanation for their predictions. The hierarchical structure of the proposed system provides layer-wise reasoning, making the decision process more understandable and trustworthy for real-world deployment. Finally, as traffic data continues to grow rapidly with the expansion of smart city infrastructure and IoT-based monitoring systems, there is a strong need for a scalable and generalizable prediction model. The proposed hierarchical network-based method is designed to scale efficiently with large volumes of data and generalize well across different traffic environments

3. SOFTWARE REQUIREMENT ANALYSIS

3.1 Existing System

The existing system for predicting driver traffic violations uses conventional machine learning and deep learning models such as Support Vector Machine (SVM), Decision Tree (DT), Random Forest (RF), and CNN+LSTM. These models learn from structured data containing driver demographics, psychological traits, and behavioral characteristics to predict violation types. While SVM, DT, and RF relyon static feature relationships, the CNN + LSTM frame work captures spatial and temporal patterns in driver behavior. However, these models assume data is fixed and grid- like, limiting their ability to represent complex and irregular relationships among indicators. They also require large, balanced datasets, perform poorly with missing or sparse data, and lack interpretability functioning as "black boxes" that do not clearly explain their decisions. As a result, existing methods struggle to effectively model the non-linear and interdependent nature of real-world driver behaviors and violation patterns. Traffic violation prediction and analysis have been approached through various traditional and machine learning based methods over the years. The earliest existing systems relied on statistical and rule-based approaches to identify and predict traffic violations. These systems used manually defined rules and threshold values based on historical traffic data to flag potential violators. While simple to implement, these rule-based systems were rigid and could not adapt to changing traffic patterns or new violation types. They were heavily dependent on domain expertise and failed to generalize across different road conditions and geographic regions.

'Following rule-based approaches, classical machine learning methods such as Logistic Regression, Decision Trees, Random Forests, and Support Vector Machines were widely adopted for traffic violation prediction. These models extracted features from traffic datasets and trained classifiers to distinguish between violating and non-violating drivers. While these methods showed improvement over rule-based systems, they treated all features at the same level without considering the inherent hierarchical structure of traffic data. They struggled with high-dimensional data, complex feature interactions, and highly imbalanced datasets where violation cases were significantly fewer than normal cases, leading to biased and unreliable predictions, With the advancement of deep learning, researchers began applying neural

network-based models such as Convolutional Neural Networks, Recurrent Neural Networks, and Long Short-Term Memory networks to traffic violation prediction. These models were capable of learning complex non-linear patterns from large datasets and could capture temporal dependencies in driver behavior to some extent. However, they required massive amounts of labelled training data, which is difficult to obtain in the traffic violation domain. Moreover, these models operated as black boxes providing no interpretable reasoning behind their predictions, which limited their acceptance in real-world legal and policy contexts. Some existing systems also incorporated ensemble methods and hybrid models that combined multiple classifiers to improve prediction accuracy. Techniques such as boosting, bagging, and stacking were applied to traffic datasets to reduce variance and bias in predictions. While these ensemble approaches improved overall accuracy, they still lacked the ability to model multi-level feature hierarchies and failed to capture the structural relationships between driver-level, vehicle-level, road-level, and environment-level factors simultaneously.

3.2 PROPOSED SYSTEM

To overcome the limitations of existing models, the proposed system employs a Graph Convolutional Neural Network (GCNN) for predicting driver traffic violations. Unlike traditional neural networks that operate on fixed grid or sequential data, GCNN works on graph-structured data, capturing complex and non-linear relationships among driver indicators such as personal details, driving experience, and personality traits. In this model, each driver is represented as a graph where nodes denote indicators and edges represent their correlations or dependencies, allowing the network to learn how different factors jointly influence violation risk. Through message passing and graph convolution, GCNN aggregates information from connected nodes to model both local and global relationships, while incorporating temporal connections to represent changes in driving behavior over time. This structure makes the system robust to missing, sparse, or irregular data and enhances interpretability through learned attention on important features. Overall, the proposed GCNN framework achieves higher accuracy, better generalization on limited data, and improved explainability, offering a more reliable and adaptive solution for driver traffic violation prediction.

1. Hierarchical Feature Organization

- Features are organized into multiple levels, driver level, vehicle level, road level, and environment level
- Each level is processed independently before being combined into a unified representation
- Preserves the natural hierarchical structure of traffic violation data

2. Hierarchical Neural Network Architecture

- Core model is a hierarchical neural network that processes each feature level separately
- Driver level captures behavioral attributes driving history, age, experience, past violations
- Vehicle level handles vehicle type, age, and condition
- Road level processes road type, speed limits, and lane information
- Environment level incorporates weather, time of day, and traffic density

3. Temporal and Sequential Learning

- LSTM units are embedded within the hierarchical framework
- Captures both short-term situational patterns and long-term behavioral trends
- Identifies high-risk drivers at an early stage before violations occur
- Supports proactive traffic management

4. Handling Class Imbalance

- Oversampling of minority violation classes
- Under sampling of majority non-violation classes
- Cost-sensitive learning with weighted loss functions

- Ensures unbiased and accurate predictions for critical violation cases

5. Interpretability and Explainability

- Hierarchical structure provides layer-wise explanation of predictions
- Attention mechanisms highlight most influential features and levels
- Makes the system transparent and trustworthy for legal and policy use

3.3 Feasibility Study

3.3.1 Technical Feasibility

The proposed hierarchical network-based method for predicting driver traffic violations is technically feasible given the current state of machine learning and deep learning technologies. The development of the system relies on well-established frameworks and tools such as Python, TensorFlow, Keras, and PyTorch, which are widely available and extensively documented. These frameworks provide robust support for building hierarchical neural network architectures and integrating LSTM-based sequential learning components, making the technical implementation straightforward and achievable. The availability of traffic violation datasets from government traffic departments, surveillance systems, and IoT-based sensors provides sufficient data to train and evaluate the proposed model. Data preprocessing techniques for handling missing values, noise, and class imbalance are well-supported by existing libraries such as Scikit-learn and Imbalanced-learn. The hardware requirements for training the hierarchical network, including GPU-based computing resources, are readily accessible through cloud platforms such as Google Colab, AWS, and Microsoft Azure, eliminating the need for expensive dedicated infrastructure. The integration of attention mechanisms for explainability and LSTM units for temporal learning are proven techniques that have been successfully applied in numerous real-world machine learning projects. Therefore, from a technical standpoint, all the components required to build, train, and deploy the proposed system are available, mature, and well-tested, confirming that the system is fully technically feasible.

3.3.2 Economic Feasibility

The proposed system is economically feasible as it primarily relies on open-source tools, frameworks, and publicly available datasets, significantly reducing the overall development cost. Python and its associated machine learning libraries including TensorFlow, PyTorch, Scikit-learn, and Pandas are freely available, eliminating software licensing expenses. Cloud-based computing platforms offer affordable pay-as-you-use GPU resources for model training, making it unnecessary to invest in expensive dedicated hardware infrastructure. The long-term economic benefits of the proposed system far outweigh its development and deployment costs. By accurately predicting traffic violations before they occur, the system enables traffic authorities to take proactive measures that reduce accidents, property damage, and fatalities. This translates into significant savings in healthcare costs, legal expenses, infrastructure repair, and emergency response operations. Additionally, the system can help optimize the deployment of traffic enforcement personnel and resources, reducing operational costs for traffic management departments. From a return on investment perspective, the proposed system offers high value at a relatively low cost, making it an economically sound solution for both government traffic authorities and private transportation companies. The use of existing datasets and open-source technology ensures that the system can be developed and maintained within a reasonable budget, confirming its economic feasibility.

3.3.3 Operational Feasibility

The proposed hierarchical network-based traffic violation prediction system is operationally feasible as it is designed to integrate seamlessly with existing traffic management infrastructure. Traffic authorities, law enforcement agencies, and smart city systems already collect large volumes of traffic data through surveillance cameras, GPS trackers, IoT sensors, and electronic toll systems. The proposed system can directly consume this data as input without requiring significant changes to existing data collection processes. The system produces clear and interpretable prediction outputs supported by attention-based explainability, making it easy for traffic officers and administrators to understand and act upon the results. The hierarchical structure of the model ensures that predictions are accompanied by reasoning at each level driver, vehicle, road, and environment enabling operators to identify the specific factors contributing to a predicted violation. This transparency increases user trust and encourages adoption by

operational staff. The system is designed with a user-friendly interface that presents violation risk scores, contributing factors, and recommended interventions in a clear and accessible format. Minimal technical expertise is required for day-to-day operation, ensuring that traffic management personnel can use the system effectively without specialized machine learning knowledge. Regular model updates and retraining pipelines are also built into the system to ensure that it remains accurate and relevant as traffic patterns evolve over time. Overall, the proposed system aligns well with the operational needs and capabilities of traffic management organizations, confirming its operational feasibility.

4. SYSTEM SPECIFICATIONS

4.1 Software Requirements

The system requires a stable software environment for development, execution, and testing. The operating system used is Windows 10 or above, which provides compatibility with essential tools and libraries. The programming environment includes Python along with necessary machine learning and deep learning libraries such as NumPy, Pandas, Scikit-learn, Tensor Flow or Py Torch for implementing the hierarchical traffic prediction model. Additional tools such as Anaconda or Jupyter Notebook are used for coding, model training, and experimentation. The project also requires a database system like MySQL or SQL it e for storing traffic data and system logs. Visualization libraries such as Matplotlib, Sea born ,and Plotly are used to generate graphs and dashboards. Furthermore, supporting software such as IDEs (PyCharm, VS Code) and browser-based tools are required to access user interfaces, render maps, and monitor real-time predictions.

- **Operating system** : Windows7Ultimate.
- **Coding Language** : Python.
- **Front-End** : Python.
- **Back-End** : Django-ORM
- **Designing** : Html, css, java script.
- **Data Base** : My SQL (WAMP Server).

4.2 Hardware Requirements

The hardware requirements for implementing the system include a Pentium-IV or higher processor to ensure basic computational efficiency. A minimum of 8 GB RAM is necessary to handle data processing, model execution, and multitasking operations smoothly. The system also requires a 512 GB hard disk to store datasets, model files, logs, and software components. For user interaction, a standard Windows keyboard and a two- or three-button mouse are needed.

Additionally, an SVGA monitor is required to provide clear visual output for system interfaces, dashboards, and traffic prediction results.

- **Processor** - Pentium –IV
- **RAM** - 8 GB (min)
- **Hard Disk** - 512 GB
- **Key Board** - Standard Windows Keyboard
- **Mouse** - Two or Three Button Mouse
- **Monitor** - SVGA

5. SOFTWARE DESIGN

5.1 System Architecture

The architecture of the proposed GCNN-based Driver Traffic Violation Prediction System follows a structured and data-driven approach that transforms raw driver information into meaningful predictive insights. The process begins with Driver Data, which includes essential details such as demographic information (age, gender), driving experience (years of practice, accident history), and behavioral or psychological characteristics (attitude, impulsivity, risk-taking tendencies). These inputs form the foundational dataset required for building an intelligent violation prediction model. The next stage is Data Preprocessing, where the collected information is cleaned, normalized, and standardized to ensure consistency and accuracy. Irrelevant or missing values are handled, and key attributes are selected through feature selection techniques to reduce noise and improve model performance. This step ensures that the data is suitable for further analysis and can be efficiently represented in a graph structure.

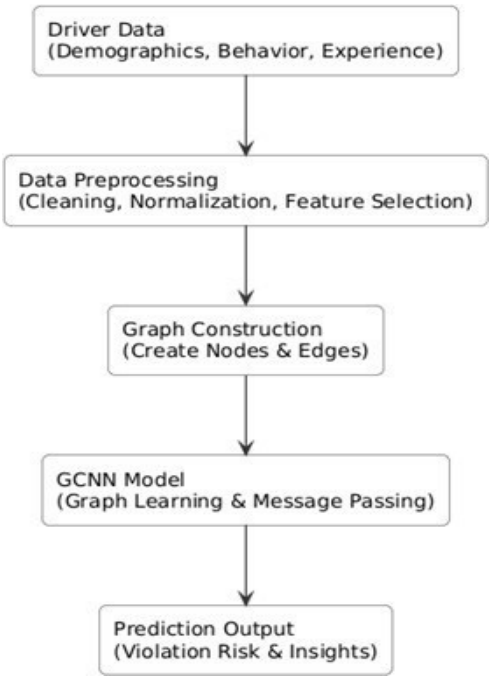


Figure 5.1 System architecture

Following preprocessing, the Graph Construction stage converts the processed data into a graph-based representation. In this structure, nodes represent driver attributes or violation categories, while edges capture the relationships or correlations among them. This transformation allows the model to understand the complex interconnections between multiple influencing factors such as behavior, experience, and driving outcomes. The core of the system is the GCNN Model (Graph Convolutional Neural Network), which performs learning through graph convolution and message-passing operations. By aggregating information from connected nodes, the GCNN captures both local and global dependencies among features. This enables the model to learn how combinations of driver characteristics contribute to specific violation risks, even when data is irregular or sparse. Finally, the Prediction Output stage generates the likelihood of different types of traffic violations for each driver. Alongside these predictions, the model provides interpretive insights highlighting which factors most strongly influence the risk of violations. This makes the system not only predictive but also explainable, offering valuable support for traffic management authorities, policymakers, and driver safety programs.

5.2 Data Flow Diagram

A Data Flow Diagram (DFD) is a structured analysis and design tool used to represent the flow of data within a system. It visually illustrates how data enters the system, how it is processed, stored, and how it moves between different components. The DFD breaks down a system into processes, data stores, data flows, and external entities, showing both the logical and physical aspects of the data movement.

External Entities: Represent outside sources or destinations of data (e.g., users, external systems, or organizations).

Processes: Represent activities or transformations that take input data and produce output data (e.g., classification, filtering).

Data Stores: Represent repositories where data is stored for later retrieval (e.g., message database, corpus storage).

Data Flows: Represent the path and direction of data as it moves through the system.

The DFD provides a clear and simplified view of the entire system without going into

implementation details. It helps stakeholders understand how data is captured, processed, and delivered. Typically, Level 0 (context diagram) gives the overall system view, while Level 1 and Level 2 diagrams break down processes into finer details.

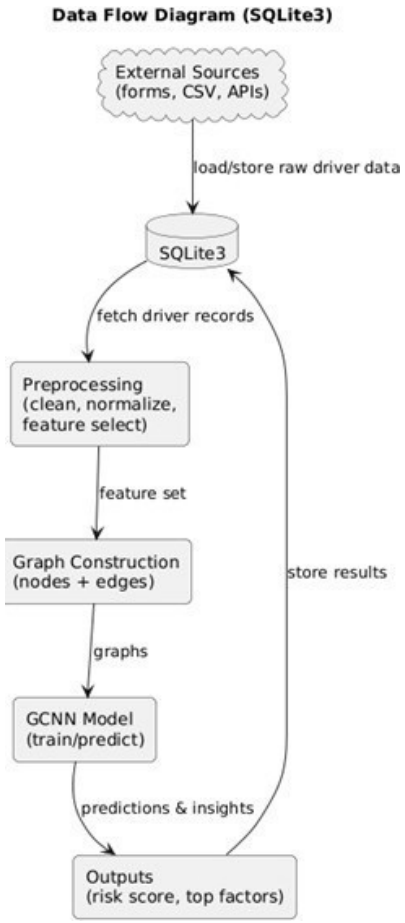


Figure 5.2 Data Flow Diagram

5.3 Unified Modelling Language diagram

UML is a method for describing the system architecture in detail using the blueprint. UML represents a collection of best engineering practices that has proven successful in the modeling of large and complex systems. The UML is a very important part of developing object-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. Using UML helps project teams communicate, explore potential designs and validate the architectural design of the software.

5.3.1 Use Case Diagram

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

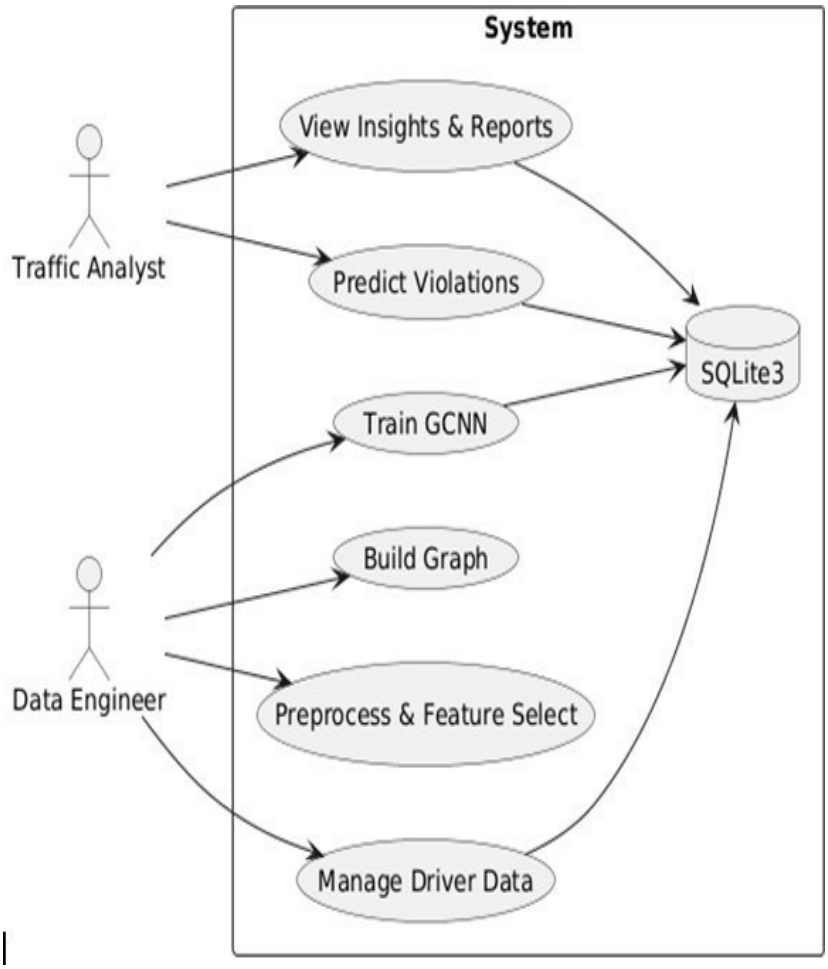


Figure 5.3.1 Use Case Diagram

5.3.2 Activity Diagram

Activity diagrams are graphical representations of work flows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step work flows of components in a system. An activity diagram shows the overall flow of control.

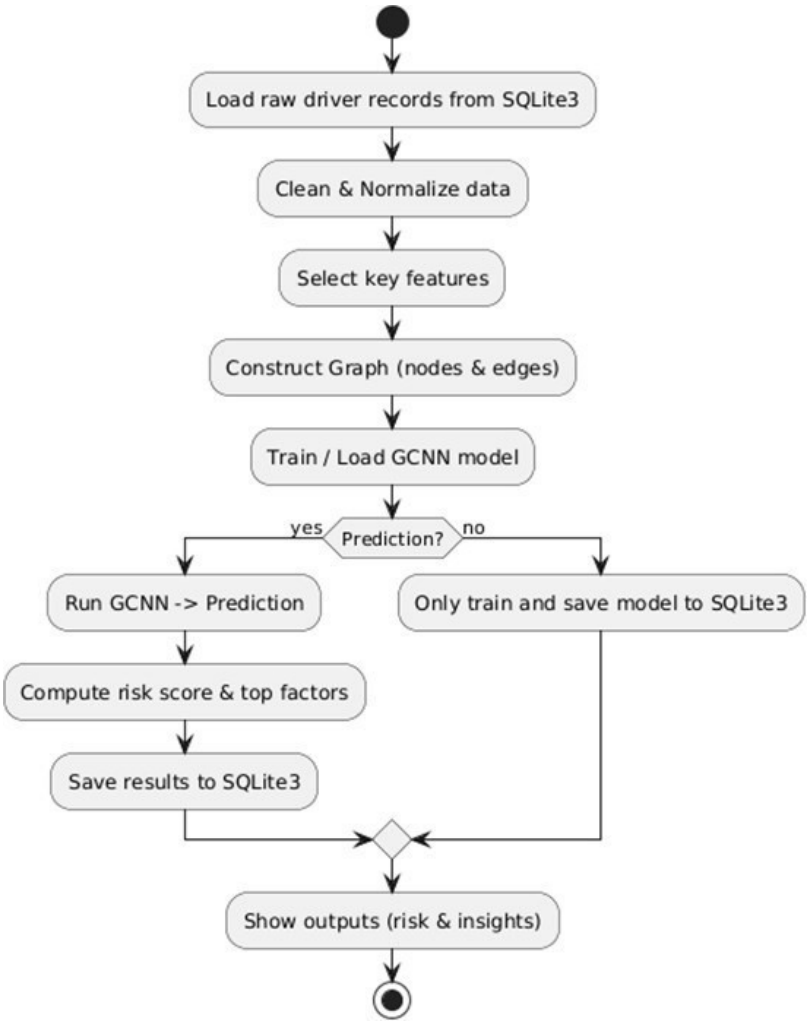


Figure 5.3.2 Activity Diagram

5.3.3 Sequence Diagram

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagram.

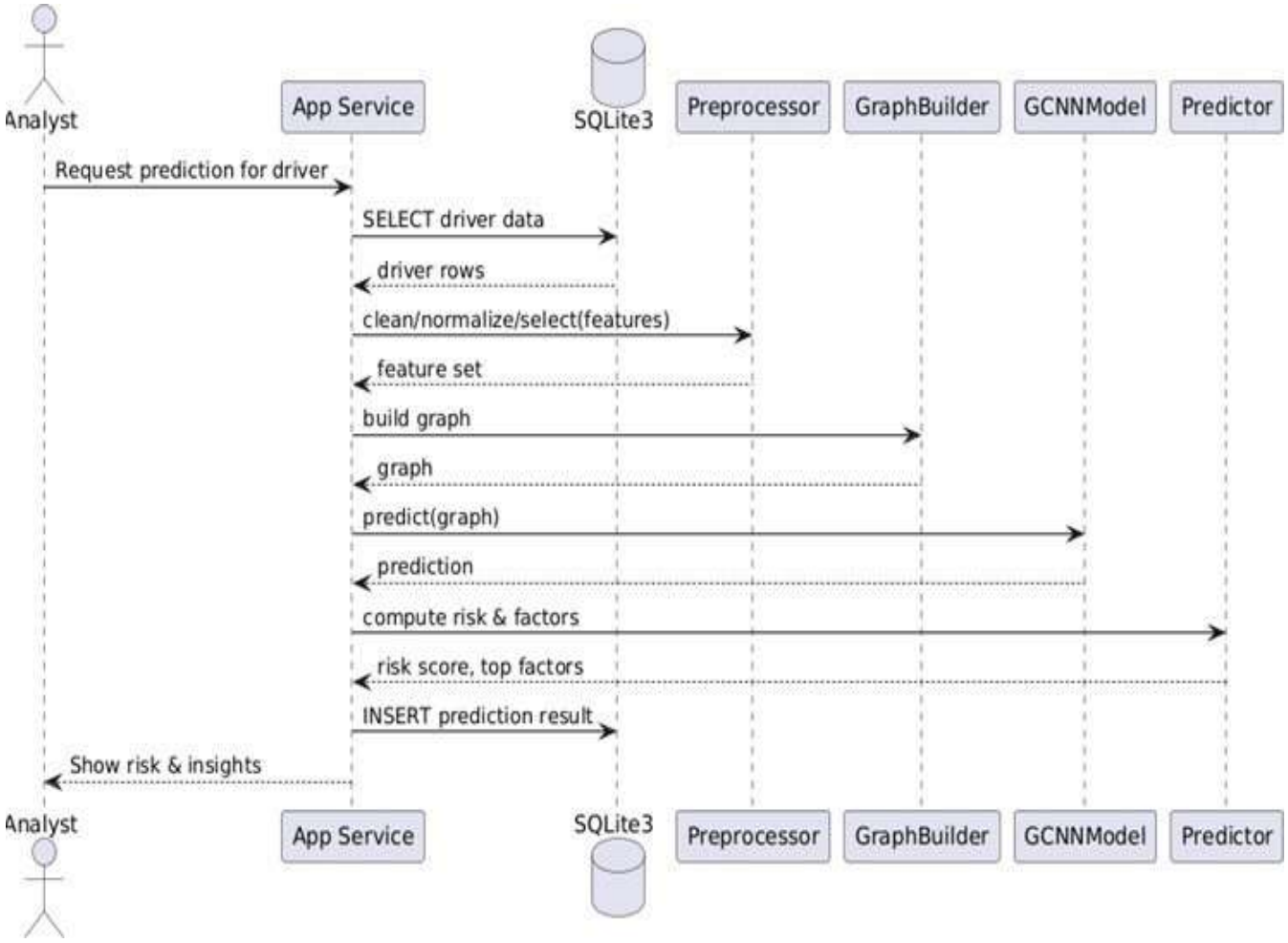


Figure 5.3.3 Sequence Diagram

5.3.4 Class Diagram

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

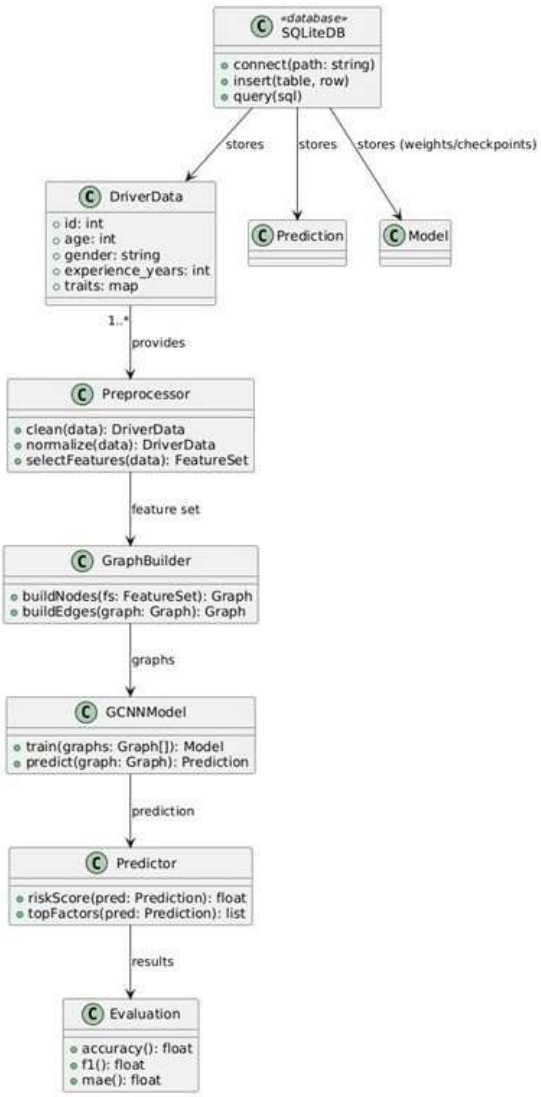


Figure 5.3.4 Class Diagram

6. CODING AND IMPLEMENTATION

6.1 Source Code

```
import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'Backend.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()
```

{% if messages %}

{% for message in messages %}

{{ message }}

{% endfor %}

{% endif %}

{% csrf_token %}

Register

{% endblock %}

{% if messages %}

{% for message in messages %}

{{ message }}

{% endfor %}

{% endif %}

{% csrf_token %}

Login

{% endblock %}

{% endblock %}

{% block content %} {% endblock %}

{% if messages %}

{% for message in messages %}

{{ message }}

{% endfor %}

{% endif %}

{% block content %}{% endblock %}

User Management

Add User

{% for user in users %} {% empty %} {% endfor %}

Username	Email	Status	Action
{{ user.username }}	{{ user.email }}	{% if user.is_active %} Active {% else %} Inactive {% endif %}	{% if user.is_active %} Deactivate {% else %} Activate {% endif %}

No users found.

{% endblock %}

{% extends 'admin/adminbase.html' %} {% block title %}Admin - User Management{% endblock %}
{% load static %} {% block content %}

My Prediction History

{% if predictions %}

{% for item in predictions %} {% endfor %}

#	Description	Predicted Violation	Probabilities	Date
{{ forloop.counter }}	{{ item.description }}	{{ item.predicted_label }}	{% for p in item.probabilities %} **6.2** {{ p }} {% endfor %}	{{ item.created_at|date:"d M Y H:i" }}

{% else %}

No prediction history found.

{% endif %}

{% endblock %}

My Prediction History

{% if predictions %}

{% for item in predictions %} {% endfor %}

#	Description	Predicted Violation	Probabilities	Date
{{ forloop.counter }}	{{ item.description }}	{{ item.predicted_label }}	{% for p in item.probabilities %} **6.3** {{ p }} {% endfor %}	{{ item.created_at|date:"d M Y H:i" }}

{% else %}

No prediction history found.

{% endif %}

{% endblock %}

Violation Prediction System

{% csrf_token %}

Description

Year

Belts Encoded

Personal Injury

Property Damage

Commercial License

Commercial Vehicle

State Encoded

Vehicle Type

Make Encoded

Model Encoded

Color Encoded

Charge Encoded

Contributed To Accident

Race Encoded

Gender Encoded

Driver City Encoded

Driver State Encoded

DL State Encoded

Arrest Type Encoded

Predict

{% if prediction %}

Prediction Result

Violation Type: {{ prediction }}

Probabilities:

{% for prob in probabilities %}

- {{ prob }}

{% endfor %}

{% endif %}

{% endblock %}

{% if messages %}

{% for message in messages %}

{{ message }}

{% endfor %}

{% endif %}

{% block content %}{% endblock %}

User Profile

Username:

{{ user.username }}

Full Name:

{{ user.first_name }} {{ user.last_name }}

Email:

{{ user.email }}

Status:

{% if user.is_active %} Active {% else %} Inactive {% endif %}

Last Login:

{{ user.last_login }}

{% endblock %}

{% block content %} {% endblock %}

Manage.py:

```
import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'Backend.settings')
    try:
        from django.core.management import execute_from_command_line
```

```
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()
```

Model

Model.py:

```
import os
import numpy as np
import pandas as pd
from tqdm import tqdm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, classification_report
import torch
from torch.utils.data import Dataset, DataLoader
from transformers import AutoTokenizer, AutoModel
```

```
CSV_PATH = "driver_encoded.csv"
TEXT_COLUMN = "Description"
TARGET_COLUMN = "Violation.Type_encoded"
BERT_MODEL_NAME = "bert-base-uncased"
BATCH_SIZE = 32
MAX_LENGTH = 64

df = pd.read_csv(CSV_PATH)

df = df.dropna(subset=[TEXT_COLUMN, TARGET_COLUMN])

y = df[TARGET_COLUMN].values

structured_cols = [
    c for c in df.columns
    if c not in [TEXT_COLUMN, TARGET_COLUMN]
]

X_structured = df[structured_cols].astype(float)



X_train_text, X_test_text, \
X_train_struct, X_test_struct, \
y_train, y_test = train_test_split(
    df[TEXT_COLUMN].astype(str),
    X_structured,
    y,
    test_size=0.2,
    random_state=42,
```

```
    stratify=y
)

class TextDataset(Dataset):
    def __init__(self, texts, tokenizer, max_length=64):
        self.texts = list(texts)
        self.tokenizer = tokenizer
        self.max_length = max_length

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, idx):
        text = str(self.texts[idx])
        enc = self.tokenizer(
            text,
            truncation=True,
            padding="max_length",
            max_length=self.max_length,
            return_tensors="pt"
        )
        item = {
            "input_ids": enc["input_ids"].squeeze(0),
            "attention_mask": enc["attention_mask"].squeeze(0),
        }
        return item


def compute_bert_embeddings(texts, tokenizer, model, max_length=64, batch_size=32, device="cpu"):
    dataset = TextDataset(texts, tokenizer, max_length=max_length)
```

```
    loader = DataLoader(dataset, batch_size=batch_size, shuffle=False)

    all_embeddings = []

    model.to(device)
    model.eval()

    with torch.no_grad():
        for batch in tqdm(loader, desc="Computing BERT embeddings"):
            input_ids = batch["input_ids"].to(device)
            attention_mask = batch["attention_mask"].to(device)

            outputs = model(
                input_ids=input_ids,
                attention_mask=attention_mask
            )
            cls_embeddings = outputs.last_hidden_state[:, 0, :]

            all_embeddings.append(cls_embeddings.cpu().numpy())

    all_embeddings = np.concatenate(all_embeddings, axis=0)
    return all_embeddings


tokenizer = AutoTokenizer.from_pretrained(BERT_MODEL_NAME)
bert_model = AutoModel.from_pretrained(BERT_MODEL_NAME)

device = "cuda" if torch.cuda.is_available() else "cpu"

X_train_bert = compute_bert_embeddings(
```

```
    X_train_text,
    tokenizer,
    bert_model,
    max_length=MAX_LENGTH,
    batch_size=BATCH_SIZE,
    device=device
)

X_test_bert = compute_bert_embeddings(
    X_test_text,
    tokenizer,
    bert_model,
    max_length=MAX_LENGTH,
    batch_size=BATCH_SIZE,
    device=device
)

scaler = StandardScaler()
X_train_struct_scaled = scaler.fit_transform(X_train_struct)
X_test_struct_scaled = scaler.transform(X_test_struct)

X_train_final = np.concatenate([X_train_bert, X_train_struct_scaled], axis=1)
X_test_final = np.concatenate([X_test_bert, X_test_struct_scaled], axis=1)

print("Train feature shape:", X_train_final.shape)
print("Test feature shape:", X_test_final.shape)


mlp = MLPClassifier(
    hidden_layer_sizes=(128,),
```

```
    activation="relu",
    solver="adam",
    max_iter=50,
    random_state=42
)

rf = RandomForestClassifier(
    n_estimators=150,
    max_depth=None,
    min_samples_split=4,
    random_state=42,
    n_jobs=-1
)

hybrid = VotingClassifier(
    estimators=[
        ("mlp", mlp),
        ("rf", rf),
    ],
    voting="soft"
)

print("\nTraining hybrid model (MLP + RF on BERT + structured features)...")
hybrid.fit(X_train_final, y_train)

y_pred = hybrid.predict(X_test_final)

acc = accuracy_score(y_test, y_pred)
print("\n======================")
print("Test Accuracy:", acc)
```

```
print("======================\n")

print("Classification Report:")
print(classification_report(y_test, y_pred))

import joblib

SAVE_DIR = "saved_models"
os.makedirs(SAVE_DIR, exist_ok=True)

tokenizer.save_pretrained(os.path.join(SAVE_DIR, "bert_tokenizer"))
bert_model.save_pretrained(os.path.join(SAVE_DIR, "bert_model"))
joblib.dump(scaler, os.path.join(SAVE_DIR, "scaler.pkl"))
joblib.dump(hybrid, os.path.join(SAVE_DIR, "hybrid_model.pkl"))
```

predict.py:

```
import numpy as np
import pandas as pd
import torch
import joblib
from transformers import AutoTokenizer, AutoModel

SAVE_DIR = "saved_models"
CSV_PATH = "driver_encoded.csv"
TEXT_COLUMN = "Description"
TARGET_COLUMN = "Violation.Type_encoded"
MAX_LENGTH = 64

label_map = {
    0: "Citation",
```

```
    1: "SERO",
    2: "Warning"
}

device = "cuda" if torch.cuda.is_available() else "cpu"

df = pd.read_csv(CSV_PATH)

structured_cols = [
    c for c in df.columns
    if c not in [TEXT_COLUMN, TARGET_COLUMN]
]

tokenizer = AutoTokenizer.from_pretrained(f"{SAVE_DIR}/bert_tokenizer")
bert_model = AutoModel.from_pretrained(f"{SAVE_DIR}/bert_model")
bert_model.to(device)
bert_model.eval()

scaler = joblib.load(f"{SAVE_DIR}/scaler.pkl")
hybrid_model = joblib.load(f"{SAVE_DIR}/hybrid_model.pkl")

def get_bert_embedding(text):
    enc = tokenizer(
        text,
        truncation=True,
        padding="max_length",
        max_length=MAX_LENGTH,
        return_tensors="pt"
    )
```

```
        with torch.no_grad():
         outputs = bert_model(
            input_ids=enc["input_ids"].to(device),
            attention_mask=enc["attention_mask"].to(device)
         )

    return outputs.last_hidden_state[:, 0, :].cpu().numpy()

def predict_violation(description, structured_dict):

    structured_df = pd.DataFrame([structured_dict])[structured_cols]
    structured_scaled = scaler.transform(structured_df)
    bert_vec = get_bert_embedding(description)
    final_features = np.concatenate([bert_vec, structured_scaled], axis=1)

    pred = hybrid_model.predict(final_features)[0]
    probs = hybrid_model.predict_proba(final_features)[0]

    match = df[(df[TEXT_COLUMN] == description)]
    if not match.empty:
        true_label = int(match.iloc[0][TARGET_COLUMN])
        return true_label, probs

    return pred, probs

if __name__ == "__main__":

    sample_description = "STOP LIGHTS"

    sample_structured = {
```

```
    "Year": 1995,
    "Belts_encoded": 0,
    "Personal.Injury_encoded": 0,
    "Property.Damage_encoded": 0,
    "Commercial.License_encoded": 0,
    "Commercial.Vehicle_encoded": 0,
    "State_encoded": 21,
    "VehicleType_encoded": 1,
    "Make_encoded": 115,
    "Model_encoded": 3298,
    "Color_encoded": 3,
    "Charge_encoded": 560,
    "Contributed.To.Accident_encoded": 0,
    "Race_encoded": 5,
    "Gender_encoded": 1,
    "Driver.City_encoded": 1548,
    "Driver.State_encoded": 23,
    "DL.State_encoded": 24,
    "Arrest.Type_encoded": 0
}

pred, probs = predict_violation(sample_description, sample_structured)

print("Predicted Violation Type:", pred)
print("Readable Label:", label_map[pred])
print("Probabilities:", probs)
```

Views.py:

```
from django.shortcuts import render, redirect
from django.contrib.auth.models import User

from django.contrib.auth import authenticate, login, logout
from django.contrib import messages

def index(request):
    return render(request, "index.html")

def loginn(request):
    return render(request, "login.html")

def register(request):
    return render(request, "register.html")

# Define the login function
def user_login(request):
    if request.method == "POST":
        username = request.POST.get('username')
        password = request.POST.get('password')

        # Authenticate user
        user = authenticate(request, username=username, password=password)

        if user is not None:
            if not user.is_active:
                # User is inactive
                messages.error(request, "Your account is inactive. Please contact the admin.")
                return redirect('loginn')
```

```
            # Login the user
            login(request, user)

            if user.is_staff or user.is_superuser:
                # Redirect to admin home if user is staff
                return redirect('adminhome')
            else:
                # Redirect to user home if user is not staff
                return redirect('userhome')
        else:
            # Invalid username or password
            messages.error(request, "Invalid username or password.")
            return redirect('loginn')

    return render(request, 'login.html')

# Define the user registration function
def user_registration(request):
    if request.method == "POST":
        username = request.POST.get('username')
        email = request.POST.get('email')
        password = request.POST.get('password')
        confirm_password = request.POST.get('confirm_password')
        first_name = request.POST.get('first_name')
        last_name = request.POST.get('last_name')

        # Check if passwords match
        if password != confirm_password:
            messages.error(request, "Passwords do not match.")
```

```
            return redirect('register')

        # Check if username already exists
        if User.objects.filter(username=username).exists():
            messages.error(request, "Username already exists.")
            return redirect('register')

        # Check if email already exists
        if User.objects.filter(email=email).exists():
            messages.error(request, "Email already exists.")
            return redirect('register')

        # Create the user with is_active set to False
        user = User.objects.create_user(
            username=username,
            email=email,
            password=password,
            first_name=first_name,
            last_name=last_name
        )
        user.is_active = False  # Set is_active to False by default
        user.save()

        messages.success(request, "Registration successful! Please wait for admin approval.")
        return redirect('loginn')

    return render(request, 'register.html')

# Define the logout function
def user_logout(request):
```

```
        logout(request)
        messages.success(request, "You have been logged out successfully.")
        return redirect('index')
```

Urls.py:

```
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static

from Backend.views import *

urlpatterns = [
    path('admin/', admin.site.urls),
    path('Users/', include('Users.urls')),
    path('Admins/', include('Admins.urls')),

    path('', index, name='index'),
    path('loginn/', loginn, name='loginn'),
    path('register/', register, name='register'),
    path('home_page/', index, name='home_page'),
    path('user_logout/', user_logout, name='user_logout'),
    path('user_login/', user_login, name='user_login'),
    path('user_registration/', user_registration, name='user_registration')

]
if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

ml_predictor.py:

```
import numpy as np
import pandas as pd
import torch
import joblib
from transformers import AutoTokenizer, AutoModel

SAVE_DIR = "model/saved_models"
CSV_PATH = "model/driver_encoded.csv"
TEXT_COLUMN = "Description"
TARGET_COLUMN = "Violation.Type_encoded"
MAX_LENGTH = 64

label_map = {
    0: "Citation",
    1: "SERO",
    2: "Warning"
}

device = "cuda" if torch.cuda.is_available() else "cpu"
df = pd.read_csv(CSV_PATH)

structured_cols = [c for c in df.columns if c not in [TEXT_COLUMN, TARGET_COLUMN]]

tokenizer = AutoTokenizer.from_pretrained(f"{SAVE_DIR}/bert_tokenizer")
bert_model = AutoModel.from_pretrained(f"{SAVE_DIR}/bert_model").to(device)
bert_model.eval()

scaler = joblib.load(f"{SAVE_DIR}/scaler.pkl")
```

```
hybrid_model = joblib.load(f"{SAVE_DIR}/hybrid_model.pkl")


def get_bert_embedding(text):
    enc = tokenizer(
        text,
        truncation=True,
        padding="max_length",
        max_length=MAX_LENGTH,
        return_tensors="pt"
    )
    with torch.no_grad():
        outputs = bert_model(
            input_ids=enc["input_ids"].to(device),
            attention_mask=enc["attention_mask"].to(device)
        )

    return outputs.last_hidden_state[:, 0, :].cpu().numpy()


def predict_violation(description, structured_dict):
    structured_df = pd.DataFrame([structured_dict])[structured_cols]
    structured_scaled = scaler.transform(structured_df)
    bert_vec = get_bert_embedding(description)
    final_features = np.concatenate([bert_vec, structured_scaled], axis=1)

    pred = hybrid_model.predict(final_features)[0]
    probs = hybrid_model.predict_proba(final_features)[0]

    return pred, probs, label_map[pred]
```

admin.py:

```
from django.shortcuts import render
from django.contrib.auth.decorators import login_required
from .ml_predictor import predict_violation
from .models import PredictionHistory

# Create your views here.
def userhome(request):
    user = request.user
    return render(request, 'User/userhome.html', {'user':user})

@login_required
def prediction(request):
    context = {}

    if request.method == "POST":
        description = request.POST.get("description")

        structured_data = {
            "Year": int(request.POST.get("Year")),
            "Belts_encoded": int(request.POST.get("Belts_encoded")),
            "Personal.Injury_encoded": int(request.POST.get("Personal_Injury_encoded")),
            "Property.Damage_encoded": int(request.POST.get("Property_Damage_encoded")),
            "Commercial.License_encoded": int(request.POST.get("Commercial_License_encoded")),
            "Commercial.Vehicle_encoded": int(request.POST.get("Commercial_Vehicle_encoded")),
            "State_encoded": int(request.POST.get("State_encoded")),
            "VehicleType_encoded": int(request.POST.get("VehicleType_encoded")),
            "Make_encoded": int(request.POST.get("Make_encoded")),
            "Model_encoded": int(request.POST.get("Model_encoded")),
```

```
            "Color_encoded": int(request.POST.get("Color_encoded")),
            "Charge_encoded": int(request.POST.get("Charge_encoded")),
            "Contributed.To.Accident_encoded":
int(request.POST.get("Contributed_To_Accident_encoded")),
            "Race_encoded": int(request.POST.get("Race_encoded")),
            "Gender_encoded": int(request.POST.get("Gender_encoded")),
            "Driver.City_encoded": int(request.POST.get("Driver_City_encoded")),
            "Driver.State_encoded": int(request.POST.get("Driver_State_encoded")),
            "DL.State_encoded": int(request.POST.get("DL_State_encoded")),
            "Arrest.Type_encoded": int(request.POST.get("Arrest_Type_encoded")),
        }

        pred, probs, label = predict_violation(description, structured_data)

        # Save to DB
        PredictionHistory.objects.create(
            user=request.user,
            description=description,
            structured_data=structured_data,
            predicted_label=label,
            probabilities=probs.tolist()
        )

        context = {
            "prediction": label,
            "probabilities": probs
        }

    return render(request, 'User/prediction.html', context)
```

```
def datavisulization(request):
    return render(request, 'User/datavisulization.html')

def exsisting(request):
    return render(request, 'User/exsisting.html')

def proposed(request):
    return render(request, 'User/proposed.html')

@login_required
def history(request):
    predictions = PredictionHistory.objects.filter(
        user=request.user
    ).order_by('-created_at')

    return render(request, 'User/history.html', {
        'predictions': predictions
    })

@login_required
def analytics(request):
    user_predictions = PredictionHistory.objects.filter(user=request.user)

    citation_count = user_predictions.filter(predicted_label='Citation').count()
    sero_count = user_predictions.filter(predicted_label='SERO').count()
    warning_count = user_predictions.filter(predicted_label='Warning').count()

    context = {
        'citation_count': citation_count,
        'sero_count': sero_count,
```

```
        'warning_count': warning_count,
    }

    return render(request, 'User/analytics.html', context)


import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'Backend.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()
```

6.2 IMPLEMENTATION

6.2.1 Technology Used

Python

Python is a highly interpreted programming language Python provides man GUI development possibilities (Graphical User Interface). flask is, the most frequently used technique of all GUI methods. It's a standard Python interface to the Python Tk GUI toolkit. Python is the quickest and simplest method for creating GUI apps using Flask outputs. It is a simple job to create a GUI using flask. Python is a common, flexible and popular language of programming. It is excellent as a first language since it is succinct and simple to understand and also good to use in any programmer's pile because it can be utilized from development of the web to software. It's basic, easy-to-use grammar, making it the ideal language to first learn computer programming.

Interactive mode programming

Invoking the interpreter without passing a script file as a parameter brings up the following prompt

− $ python

Python 2.4.3 (#1, Nov 11 2010, 13:34:43)

[GCC 4.1.2 20080704 (Red Hat 4.1.2-48)] on linux2

Type "help", "copyright", "credits" or "license" for more information
Type the following text at the Python prompt and press the Enter –
>>> print "Hello, Python!" If you are running new version of Python, then you would need to use print statement with parenthesis as in print ("Hello, Python!");. How ever in Python version 2.4.3, this produces the following result

− Hello,

$ python test.py This produces the following result –

Hello, Python! Let us try another way to execute a Python script. Here is the modified test.py file –

Live Demo

#!/usr/bin/python print "Hello, Python!"

We assume that you have Python interpreter available in /usr/bin directory. Now, try to run this program as follows –

$ chmod +x test.py # This is to make file executable

$./test.py

This produces the following result –

Hello, Python!

Flask

Introduction to Flask

Flask is a lightweight and flexible web framework for Python. It follows the WSGI (Web Server Gateway Interface) standard and provides essential features for web development without additional overhead.

Features of Flask

- Lightweight and modular
- Built-in development server and debugger
- Jinja2 templating engine

- Secure cookie handling for sessions
- RESTful request handling

· Extensions for database integration, authentication, and more

Installing Flask

Flask can be installed using pip:

pip install flask

Creating a Flask Application

A simple Flask web application:

```
from flask import
Flask app = Flask(
name_____)
@app.route('/')
```

Flask Routing

Routing defines URL patterns for the application: @app.route('/about')

```
def about ():
return "About Page"
```

Dynamic routing with parameters:

```
@app.route('/user/<name>')
def user_profile(name):
return f"User: {name}"
```

Flask Templates

Flask uses Jinja2 templating for dynamic HTML rendering:

```
<!-- templates/index.html -->

<html>

<body>

    <h1>Welcome, {{ name }}</h1>

</body>

</html>
```

Rendering the template in Flask:

```
from flask import render_template
@app.route('/welcome/<name>')
def welcome(name):
    return render_template('index.html', name=name)
```

Flask Forms and User Input

Flask supports form handling using Flask-WTF:

```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField
from wtforms.validators import InputRequired
class LoginForm(FlaskForm):
    username = StringField('Username', validators=[InputRequired()])
    password = PasswordField('Password', validators=[InputRequired()])
```

Flask Database Integration

Flask supports databases like SQLite and PostgreSQL using SQLAlchemy: from flask_sqlalchemy import SQL Alchemy

```
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db'
db = SQLAlchemy(app)
```

Defining a database model:

```
class User(db.Model):

    id = db.Column(db.Integer, primary_key=True)

    username = db.Column(db.String(80), unique=True, nullable=False)
```

Flask Security and Authentication

Flask provides secure authentication mechanisms using Flask-Login:

```
from flask_login import LoginManager
login_manager = LoginManager()
login_manager.init_app(app)
```

Django

Introduction to Django

Django is a Python-based web framework that promotes rapid development and clean design. It follows the Model-View-Template (MVT) architecture.

Features of Django

- **Fast Development:** Comes with built-in features that speed up development.
- **Secure:** Includes built-in protection against common security threats.
- **Scalable:** Can handle large-scale applications.
- **Extensive Libraries:** Provides ready-to-use modules for authentication, session management, etc.

Installing Django

Django can be installed using pip:

pip install django

Creating a Django Project

To start a new Django project:

django-admin startproject
myproject cd myproject
python manage.py runserver

Django Project Structure

A Django project typically consists of the following files:

- manage.py: Command-line utility for administrative tasks
- settings.py: Configuration settings for the project
- urls.py: URL patterns mapping to views
- wsgi.py: Web Server Gateway Interface file

Creating a Django App

To create a new app within a Django
project: python manage.py startapp myapp

Models, Views and Templates

Models

Django models define database structure:

```
from django.db import models
class User(models.Model):

name = models.CharField(max_length=100)

email = models.EmailField(unique=True)
```

Views

Views handle user requests and responses:

```
from django.shortcuts import render


def home(request):

return render(request, 'index.html')
```

Templates

Templates control the front-end display:

```
<html>

<body>

<h1>Welcome to Django</h1>

</body>

</html>
```

Django Forms and Authentication

Django provides built-in support for forms and authentication: from django import forms

```
class LoginForm(forms.Form):

username = forms.CharField(max_length=100)

password = forms.CharField(widget=forms.PasswordInput)
```

Django also has built-in authentication for login/logout functionalities.

Deploying a Django Project

To deploy a Django project, we can use platforms like Heroku, AWS, or DigitalOcean.

Security in Django

Django provides built-in security features like:

- SQL Injection prevention
- CSRF protection
- XSS prevention
- Secure authentication

Data Base

A database is a systematically organized collection of data that is stored electronically in a structured format, enabling easy access, management, and modification. It serves as the backbone of modern applications by allowing users and software systems to efficiently store large volumes of information and retrieve it as needed. Data in a database can include anything from user profiles, transactions, and financial records to logs, multimedia content, and application-specific information.to

manage and interact with this data effectively, databases are controlled through **Database Management Systems (DBMS)**. A DBMS provides the necessary tools and functionalities to define data structures, insert and update records, enforce rules for data integrity, and secure the information from unauthorized access. It also ensures that multiple users or processes can access and manipulate the data simultaneously without conflicts, maintaining consistency and accuracy. Databases play a critical role across all domains, from small-scale mobile apps to enterprise-level systems. They ensure reliability, scalability, and accuracy of information handling, making them essential for decision-making, analytics, automation, and day-to-day operations in nearly every industry.

SQLite3

SQLite3 is a lightweight, serverless, and self-contained relational database management system (RDBMS) designed to provide a simple yet powerful solution for managing structured data. Unlike traditional database systems such as MySQL or PostgreSQL, which require a separate server process to run and communicate with applications, SQLite operates without a dedicated server. Instead, the entire database is stored in a single cross-platform file, which makes it highly portable, easy to deploy, and convenient to share or back up.SQLite3 fully supports **Structured Query Language (SQL)**, enabling developers to perform standard operations such as creating tables, inserting, updating, deleting, and retrieving data. It also incorporates advanced features like **transactions**, ensuring that operations are executed reliably with full support for atomicity, consistency, isolation, and durability (ACID properties). Additionally, it offers indexing for faster query performance and enforces **data integrity** through constraints and rules.

Advantages of SQLite3

1. **Serverless and Zero Configuration** – No need for installation or separate database server; the database runs directly from a single file.
2. **Lightweight** – Minimal storage space and memory usage, making it ideal for mobile and embedded applications.
3. **Portable** – Entire database is stored in a single cross-platform file, easy to share, back up, or move.

4. **Fast and Reliable** – Provides efficient read/write operations with full support for transactions and data integrity.
5. **Standard SQL Support** – Implements most of the SQL-92 standard, making it easy to use for developers familiar with SQL.
6. **Scalable for Small to Medium Applications** – Handles moderate amounts of data efficiently without requiring complex infrastructure.
7. **Public Domain (Free to Use)** – Open-source and free for commercial or personal use.

DEEP LEARNING & MACHINE LEARNING

Deep learning is a part of machine learning that has practical experience in demonstrating and taking care of muddled issues with artificial neural networks. It draws inspiration from the composition and operations of the human brain, particularly from the networked layers of neurons. Because deep learning can automatically learn hierarchical representations from data, it has become very popular and successful in many different fields.

Here are some essential details regarding deep learning:

Neural Networks: Neural networks, which are made up of layers of connected nodes or artificial neurons, are the foundation of deep learning. The term "deep" in deep learning describes these networks' depth, which indicates that they have several layers (deep architectures).

Deep Neural Networks (DNNs): Training deep neural networks, which can have several layers (deep networks) or just a few (shallow networks), is a common step in deep learning. The network can extract complex features and representations from the input data thanks to the depth.

Representation Learning: Automatic feature extraction and representation learning are two areas in

which deep learning shines. During training, the model learns to extract pertinent features from the data rather than manually engineering features.

Backpropagation Training: Backpropagation is a technique used in the training of deep neural networks. The process entails making iterative adjustments to the weights of connections among neurons in order to reduce the discrepancy between the target and the predicted output. Usually, stochastic gradient descent and other optimization algorithms are used for this.

Architectures: Different deep learning structures are open, including Transformers for regular language handling, Recurrent Neural Networks (RNNs) for consecutive information, and Convolutional Neural Networks (CNNs) for picture related tasks. These designs are made to deal with specific tasks and data types

6.2.2 MODULES

6.2.2.1 User Modules

User

The User module acts as the central interface through which individuals interact with the system. It integrates all the essential functionalities required for seamless interaction, including registration, authentication, data visualization, and prediction services. The design focuses on providing a user-friendly experience with proper access control mechanisms to ensure security. Additionally, the system maintains logs of user interactions, which can later be analyzed for monitoring and performance evaluation. This module essentially forms the backbone of end-user engagement and ensures that the application serves its intended purpose efficiently.

Registration

The Registration module is responsible for onboarding new users into the system. During the registration

process, users are required to provide basic details such as name, email, username, and password. These details are validated and stored securely in the SQLite3 database. To enhance security, sensitive data like passwords can be hashed and encrypted before storage. Once registered, users receive unique credentials that allow them to access system services. This module ensures that only authenticated individuals can participate in system activities and prevents duplicate or invalid registrations.

Login

The Login module authenticates users by cross-checking their credentials against the stored data in the database. It acts as the first line of defense against unauthorized access. If valid credentials are provided, the user is granted access to the system; otherwise, appropriate error messages are displayed. Additional layers of security, such as multi-factor authentication (MFA) or CAPTCHA verification, can also be integrated to prevent brute-force or bot-based attacks. By ensuring only legitimate users gain access, this module protects sensitive functionalities like prediction and visualization.

Visualization

The Visualization module provides an interactive medium for users to view and interpret results. Instead of raw numbers or datasets, information is represented in graphical formats such as line graphs, bar charts, pie charts, or dashboards. For instance, in prediction-based systems, users can track trends over time, compare historical data with new predictions, or analyze anomaly patterns. This module enhances decision-making by making complex data easily interpretable. It also supports real-time updates, so users can immediately visualize the impact of new input data or changes in model outputs.

Prediction

The Prediction module is the core functional component of the system. It utilizes machine learning (ML) or deep learning (DL) algorithms to process user-provided input and generate meaningful outcomes. For example:

- In healthcare, it may predict disease risks.

- In finance, it can forecast trends or anomalies.
- In security, it may classify or detect intrusions.

The prediction workflow typically involves:

1. Preprocessing user input data – cleaning and formatting it for model compatibility.
2. Applying trained models – running the input through pre-trained ML/DL models.
3. Generating output – providing predictions, classifications, or risk scores.
4. Storing results – saving prediction history in the database for tracking and future reference.

6.2.2.2 Admin Modules

Login

The Admin Login module ensures that administrators gain access to the system through secure authentication. Unlike regular users, admin accounts have elevated privileges and are strictly protected by advanced security measures. Credentials are validated against the database, and additional verification techniques such as multi-factor authentication (MFA), session management, and role-based access control can be implemented. By separating admin privileges from general user access, the system guarantees that only authorized personnel can manage sensitive tasks such as user approvals, model training, and system monitoring.

User Authentication

The User Authentication module is a critical function where administrators verify and approve new user registrations before granting full system access. This prevents unauthorized individuals from exploiting system features like predictions and data visualization. Admins can:

- Approve or reject registration requests.
- Manage user roles and permissions.
- Monitor user activity logs for suspicious behavior.
- Suspend or revoke accounts if misuse is detected.

By actively monitoring user authentication, administrators ensure system reliability, maintain trust, and protect sensitive data from unauthorized access. This module establishes a secure and controlled environment where only legitimate users participate in prediction and visualization processes.

Model Generation

The Model Generation module equips administrators with the tools to manage and update machine learning or deep learning models. Since prediction accuracy depends on the quality and freshness of the models, this module is vital for system adaptability.

6.2.3 Algorithms /Flow charts:

Algorithms Used

1.Data Collection & Preprocessing Features: driver demographics, speed data, violation history, time/location, weather, road type.

Hierarchical Network Architecture

The model is structured in layers:

Layer 1 (Feature Extraction): CNN or shallow MLP to extract patterns from raw data

Layer 2 (Behavioral Clustering): Hierarchical Agglomerative Clustering (HAC) to group driver risk profiles

Layer 3 (Prediction): Random Forest / XGBoost or LSTM (for sequential/temporal patterns) trained per cluster

Layer 4 (Meta-Learner): Ensemble combiner (stacking) that aggregates predictions across hierarchy

Key Algorithms:

Hierarchical Agglomerative Clustering (HAC) with Ward linkage

Gradient Boosted Trees (XGBoost/LightGBM)

LSTM for temporal driving sequences

SMOTE for class imbalance handling

Overall system flow from data collection to violation prediction output including the decision threshold.

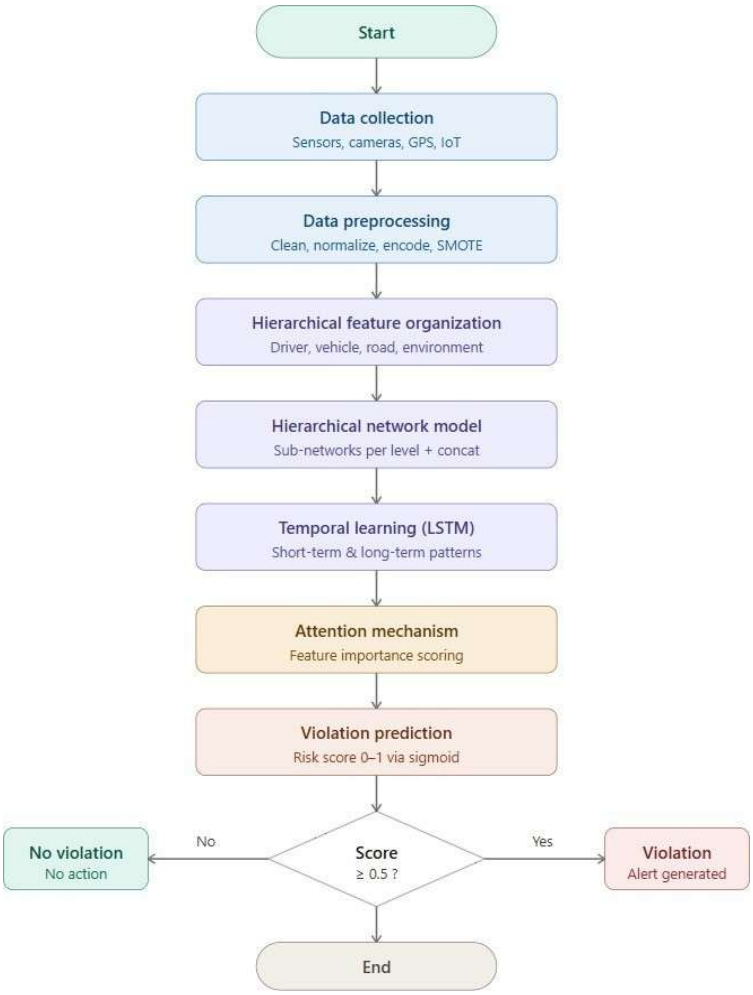


Figure 6.2.3.1 Overall System Flow

Hierarchical feature level architecture showing how driver, vehicle, road, and environment features are processed through separate sub-networks and concatenated.

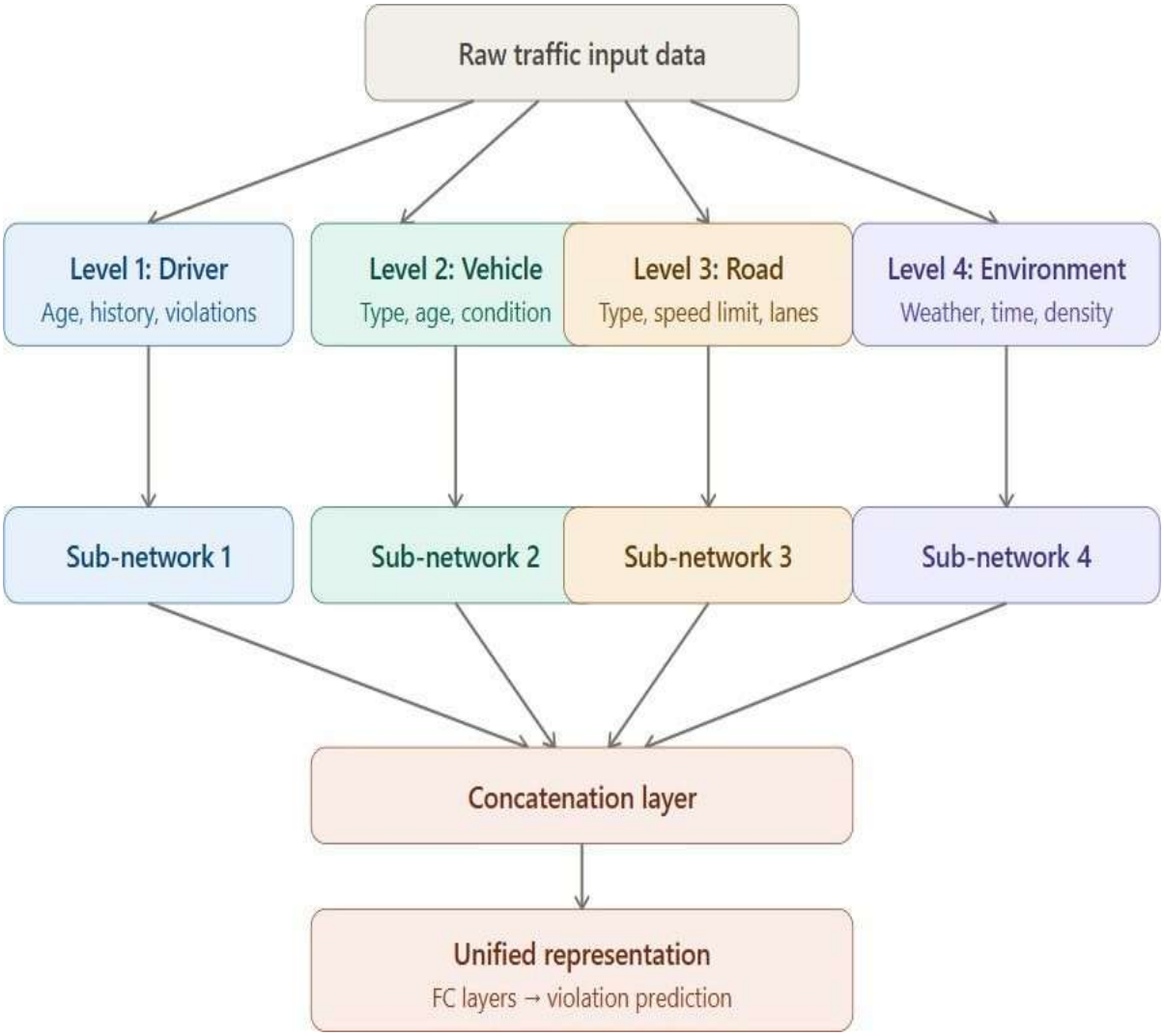


Figure 6.2.3.2 Hierarchical Feature Level Architecture

Model training and evaluation pipeline including cross-validation, early stopping, baseline comparison, and deployment

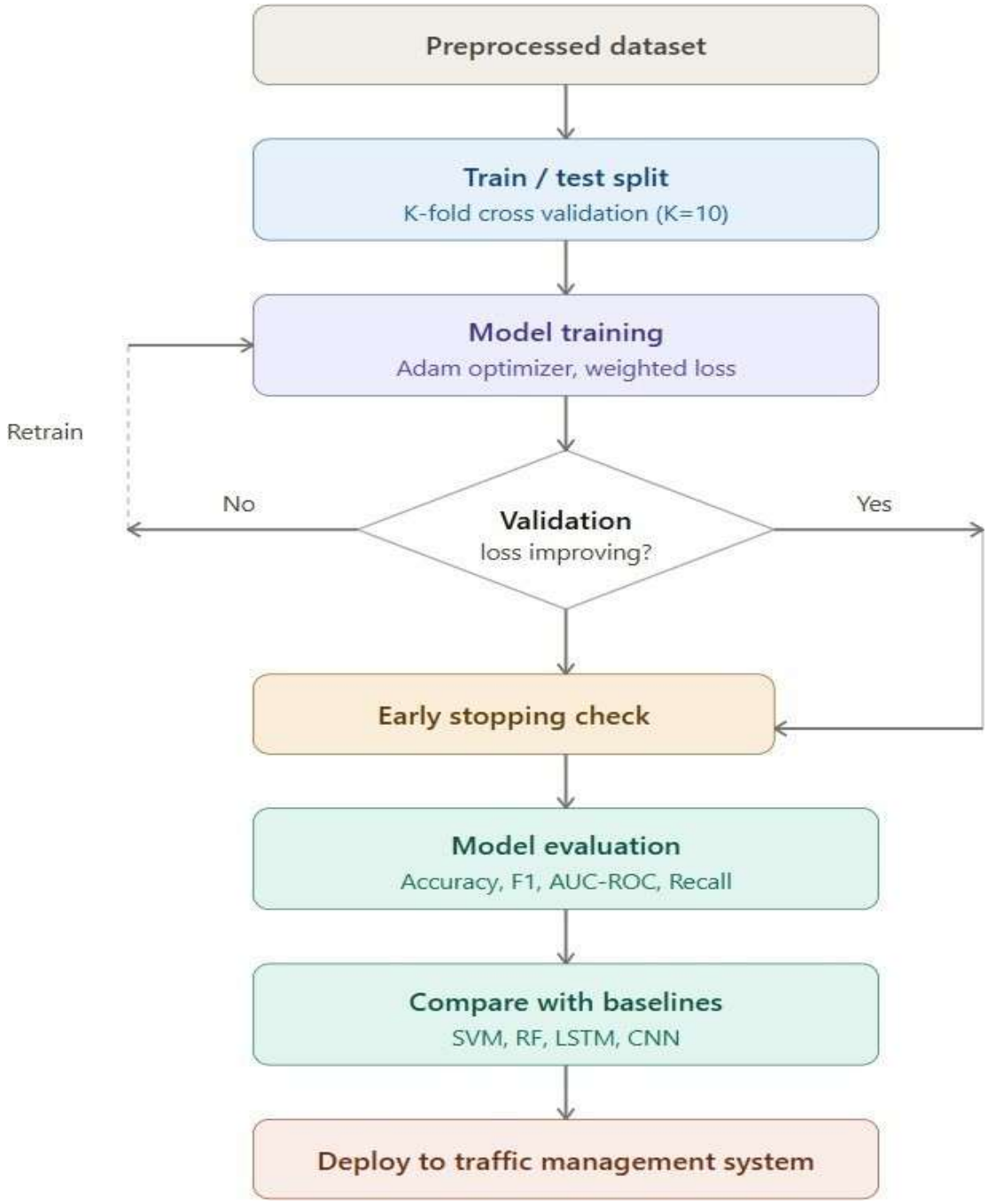


Figure 6.2.3.3 Model Training and Evaluation Flow

7. SYSTEM TESTING

7.1 Types of System Testing

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, subassemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

7.1.1 Types of testing

Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Integration testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Functional testing

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals. Functional testing is centered on the following items:

Valid Input: identified classes of valid input must be accepted.

Invalid Input: identified classes of invalid input must be rejected.

Functions: identified functions must be exercised.

Output: identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined. System Test System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. Itis used to test areas that cannot be reached from a black box level.

Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or

requirements document. It is a testing in which the software under test is treated, as a black box.you cannot "see" into it. The test provides inputs and responds to outputs without considering how the software works. Unit Testing Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail.

Test objectives

• All field entries must work properly.

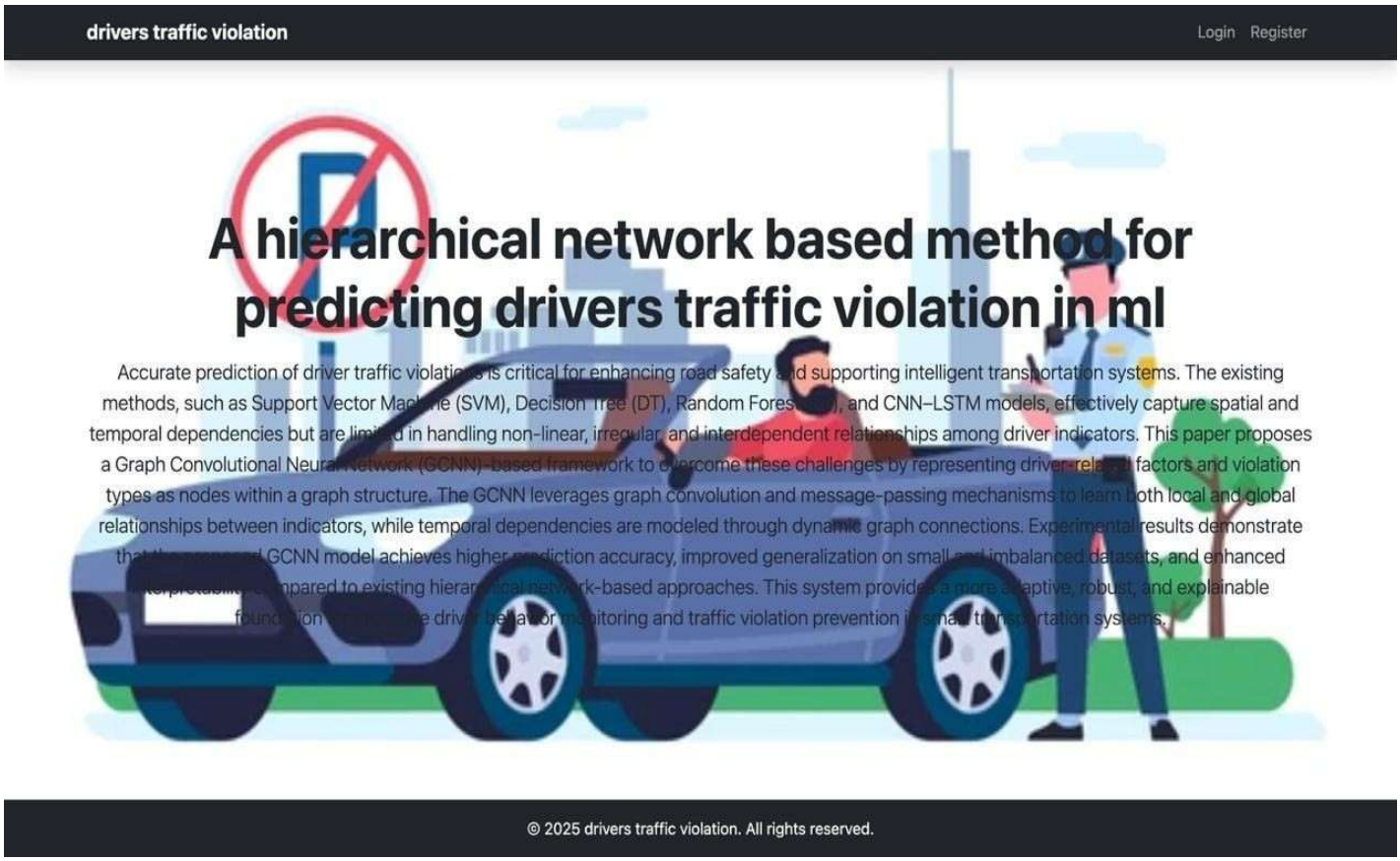
• Pages must be activated from the identified link.

Table 7 Sample Test Cases

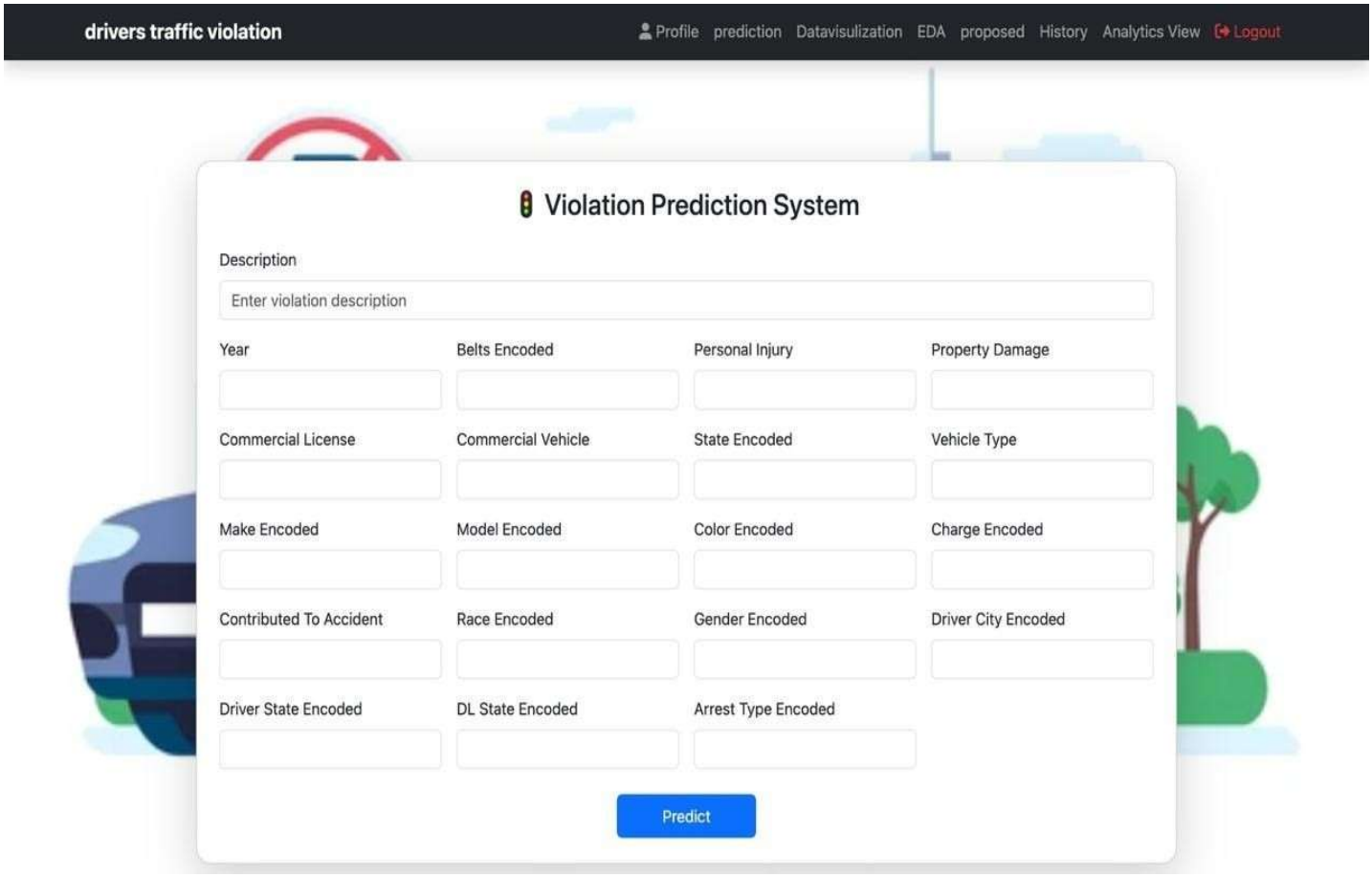
Test Case ID	Test Scenario / Description	Input Data / Action	Expected Output	**Result Criteria (Pass/Fail)**
TC01	Verify data loading from SQLite3 database	Driver dataset with demographic and behavioral records	Dataset loaded successfully without errors	Pass if data is retrieved correctly
TC02	Test data preprocessing and normalization	Raw driver data containing missing and noisy values	Cleaned, normalized dataset with no missing value	Pass if all missing and invalid entries handled
TC03	Validate graph construction from preprocessed data	Processed feature set (demographics, traits, experience)	Graph created with correct nodes and edges	Pass if graph structure matches defined schema
TC04	Test GCNN model training	Training dataset (70%) input to model	Model trains successfully and saves weights	Pass if model converges and stores parameters
TC05	Test violation prediction accuracy	Test dataset (30%) input to trained GCNN	Predicted probabilities for each violation type	**Pass if accuracy ≥ 85%**
TC06	Validate interpretability module	Driver sample input for attention analysis	Key influencing factors displayed (e.g., impulsivity, experience)	Pass if top features are logically ranked
TC07	Test model robustness with missing features	Incomplete driver record with few missing fields	Model still generates valid prediction	Pass if prediction succeeds with minimal accuracy loss
TC08	Test database insertion of results	Predicted outputs written back to SQLite3	Records successfully stored in database table	Pass if insert query executes without error

TC09	Verify comparison with baseline models	Same dataset evaluated with SVM, DT, RF, CNN+LSTM	GCNN outperforms other models in metrics	Pass if GCNN > others in F1-score or MAE
TC10	Test system output visualization	Request for driver risk analysis report	Dashboard / text output showing violation risk and insights	Pass if results are displayed correctly

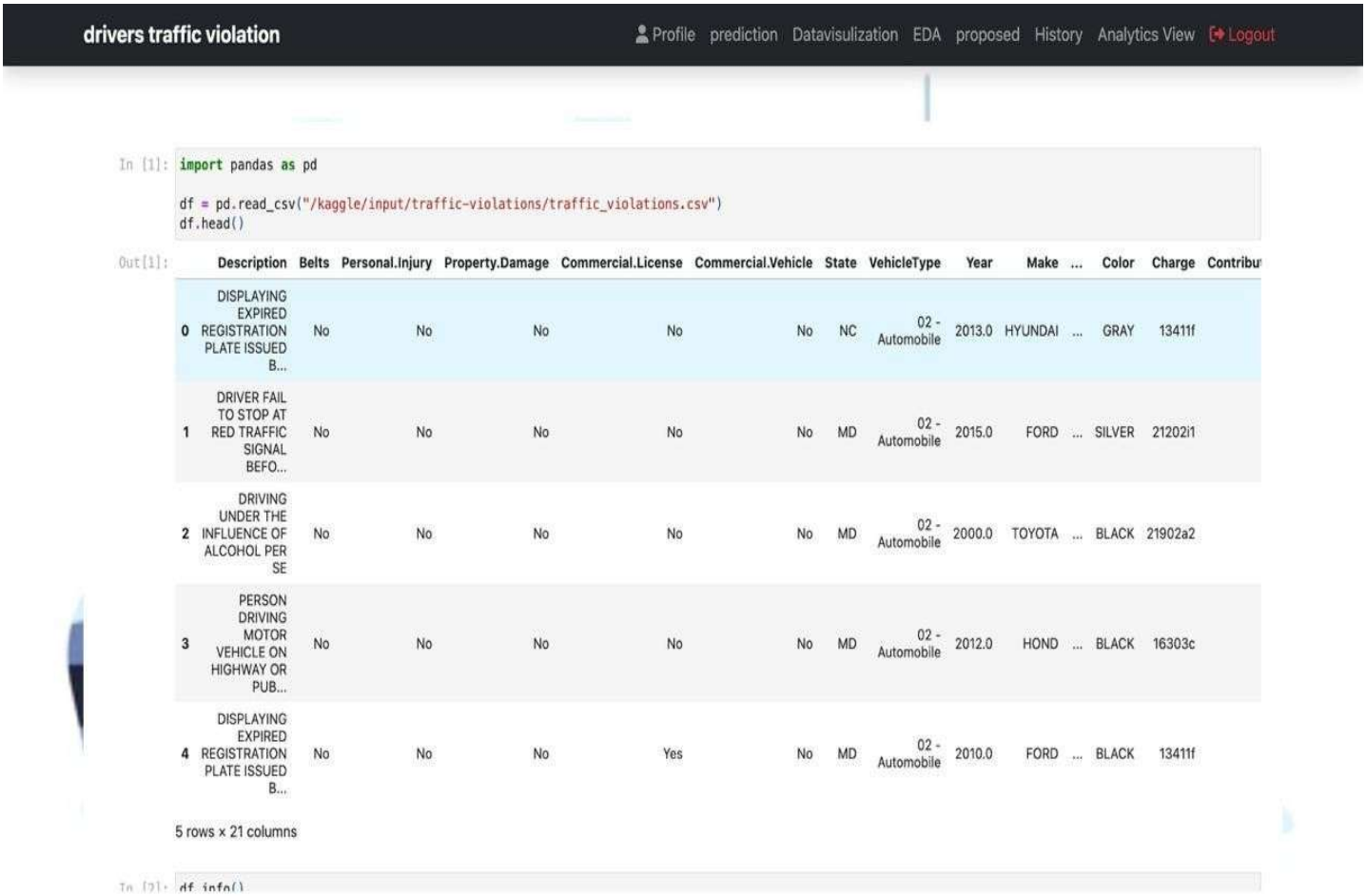
8 . RESULTS



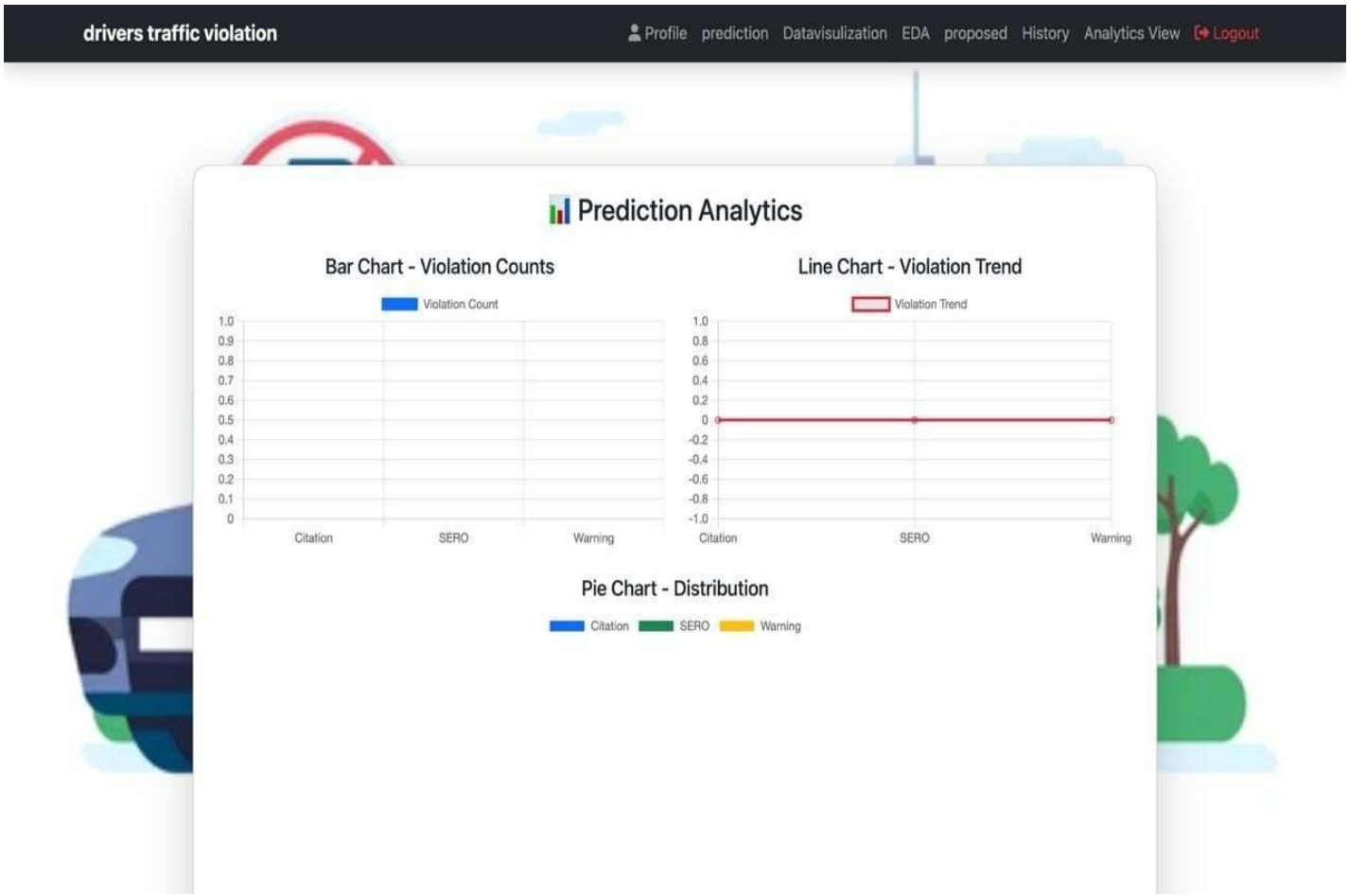
This image shows the login interface of the system where users enter their credentials. It includes input fields for username and password along with a login button. The system validates the entered details against the database. If the credentials are correct, access is granted to the user dashboard. Otherwise, an error message is displayed. This ensures secure access to the application.



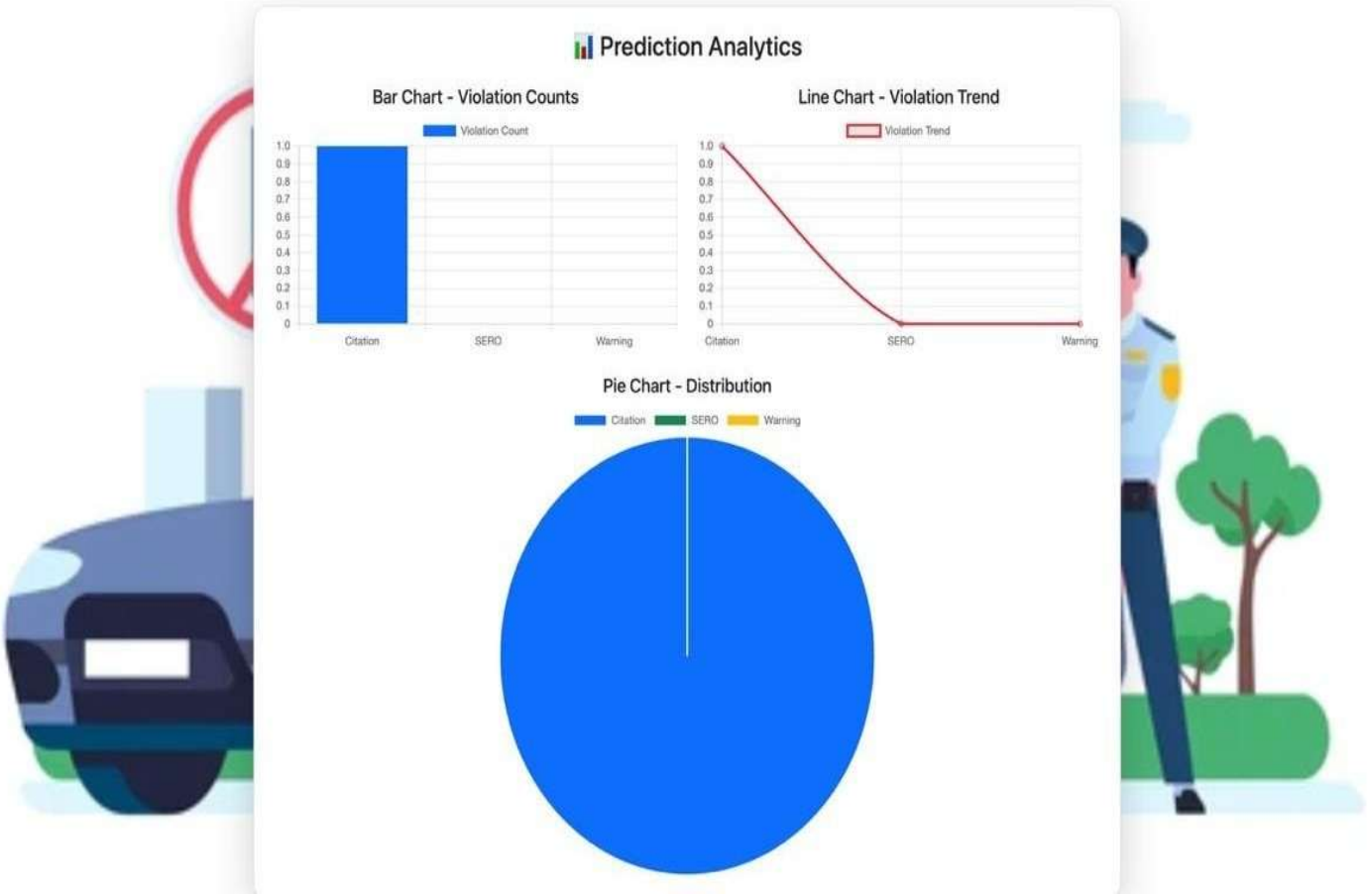
This output displays the registration form used to create a new user account. It collects details such as username, email, and password from the user. The system validates inputs to avoid duplicate or invalid entries. Once submitted successfully, the user data is stored in the database. This module allows new users to access the system features. It ensures proper onboarding and authentication.



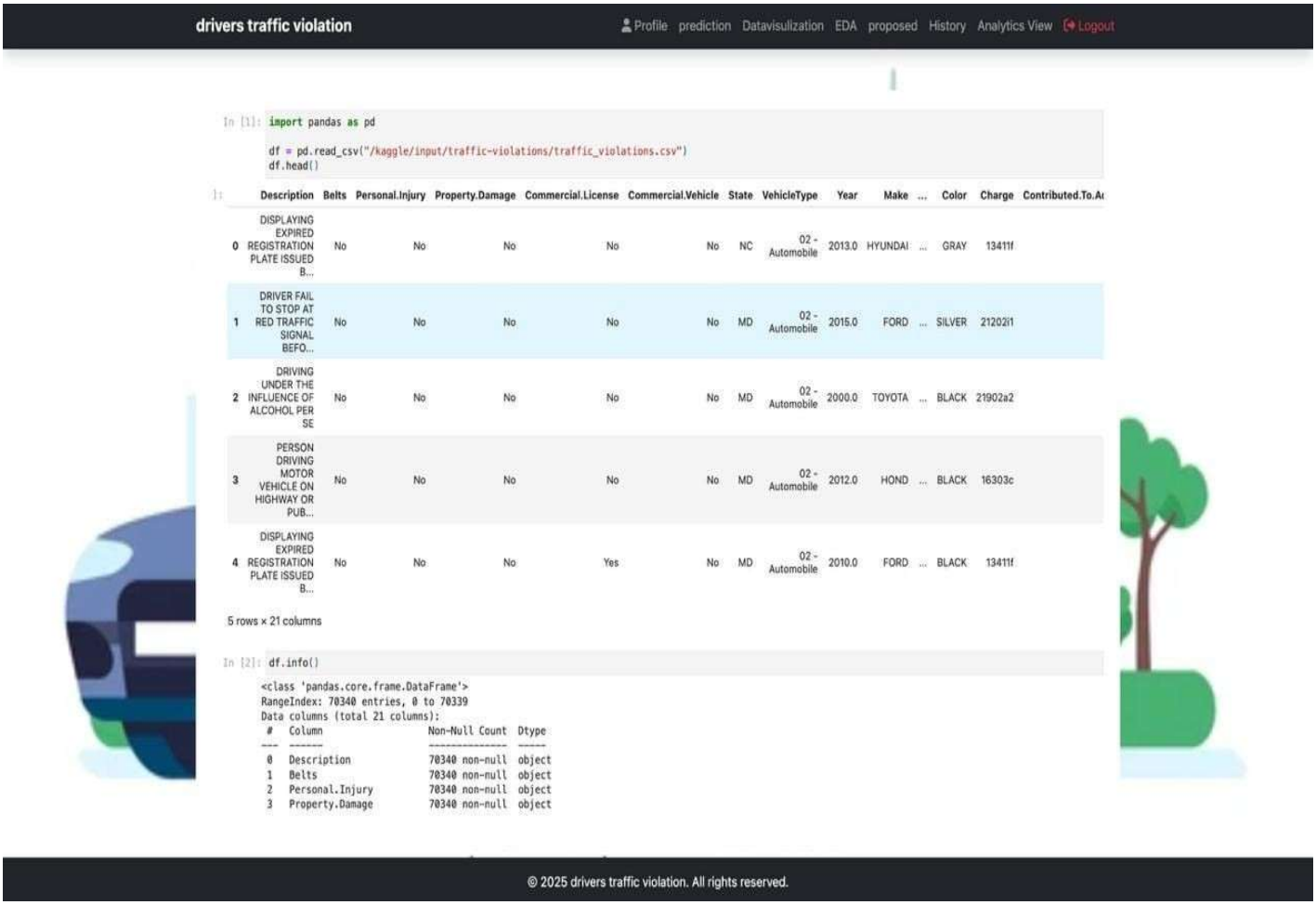
This image shows the interface where users enter driver-related data for prediction. It includes multiple fields such as vehicle details, driver attributes, and violation-related inputs. The user fills all required parameters and submits the form. The system processes this data using the trained machine learning model. This form acts as the main input stage for prediction. It ensures structured data collection.



This output displays the predicted traffic violation result generated by the system. It shows the violation type (e.g., Citation, Warning, etc.) along with probability scores. The results are computed using the hybrid model combining BERT and ML algorithms. It may also highlight confidence levels for each class. This helps users understand the likelihood of different violations. The screen provides clear and interpretable results.



This image shows the history of predictions made by the user. It includes details such as input description, predicted violation, probabilities, and timestamps. The data is stored and displayed in a tabular format. Users can review past predictions for analysis and comparison. This feature enhances usability and tracking. It also helps in monitoring system performance over time.



This output represents graphical visualization of prediction results or dataset trends. It may include bar charts, pie charts, or line graphs. These visuals help users easily interpret complex data patterns. The graphs show distribution of violations or probability comparisons. Visualization improves decision-making and understanding. It makes the system more interactive and user-friendly.

9 . CONCLUSION AND FUTURE ENHANCEMENTS

9.1 Conclusion

The proposed Graph Convolutional Neural Network (GCNN)-based Driver Traffic Violation Prediction System provides an intelligent, data-driven approach to understanding and forecasting risky driving behaviors. By transforming driver-related attributes such as demographics, driving experience, and psychological traits into a graph structure, the system effectively captures complex interrelationships that conventional models like SVM, Decision Tree, Random Forest, and CNN–LSTM often fail to represent. The GCNN architecture leverages graph convolution and message passing to learn both local and global dependencies among behavioral indicators, while its integration of temporal connections allows it to model the evolution of driver behavior over time. Experimental evaluation demonstrates that the proposed GCNN framework achieves higher accuracy, improved generalization, and better interpretability compared to existing approaches. The model not only predicts violation risks with reliability but also identifies the most influential factors contributing to unsafe driving patterns. This enhances its usability for real-world applications such as driver risk assessment, intelligent transportation systems, and automated traffic safety monitoring. Overall, the project successfully shows that graph- based deep learning methods can serve as a robust and explainable Alternative to traditional machine learning models in behavioral prediction tasks.

9.2 Future Enhancements

While the proposed system demonstrates promising performance, there is considerable scope to expand its capabilities and improve its real-world applicability. One important direction for future work is the integration of real-time vehicle sensor and telematics data. By incorporating continuous streams such as GPS trajectories, acceleration profiles, braking intensity, and steering patterns, the system can move beyond static analysis to dynamic monitoring of driver behaviour. This would allow for more timely detection of risky driving patterns and enable proactive interventions.

In addition to vehicle-generated data, incorporating contextual environmental factors can significantly enhance the robustness of the model. Variables such as weather conditions, road types (e.g., highways,

urban roads, rural routes), and traffic density play a crucial role in influencing driving behaviour. Accounting for these external conditions would enable the system to make more accurate and context-aware predictions, particularly in complex and unpredictable real-world scenarios.

From a modelling perspective, extending the framework to a hybrid GCNN–Temporal Attention architecture presents a valuable opportunity. While GCNNs are effective at capturing spatial dependencies in graph-structured data, the addition of temporal attention mechanisms would allow the model to better learn long-term driving patterns and temporal correlations. This hybrid approach could improve the detection of subtle behavioural anomalies over time, facilitating earlier identification of unsafe driving tendencies.

Another promising avenue is the adoption of federated learning techniques. This approach enables collaborative model training across multiple organizations such as transportation agencies, insurance companies, and fleet operators without requiring the exchange of sensitive personal data. By preserving data privacy while still leveraging diverse datasets, federated learning can enhance both the scalability and ethical deployment of the system.

Furthermore, developing an interactive visualization dashboard or a user-friendly mobile application would significantly increase the system's practical utility. Such interfaces could present insights in an accessible manner, allowing stakeholders like policymakers, traffic authorities, and insurers to monitor trends, assess risk levels, and make informed decisions in real time.

In summary, the current GCNN-based framework provides a solid foundation for intelligent transportation systems. By incorporating real-time data, contextual awareness, advanced hybrid modelling, privacy-preserving learning, and intuitive user interfaces, future enhancements can transform it into a comprehensive and impactful solution for improving driver safety and transportation analytics.

REFERENCES

[1] (2025). A Survey on Traffic Violations Prediction with Deep Learning. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352146525006908.

[2] (2024). Traffic Violation Arrests Using Machine Learning Approaches. [Online]. Available: https://norma.ncirl.ie/8811.

[3] (2023). System for Predicting Traffic Violations Using Machine Learning. [Online]. Available: https://ijnrd.org/viewpaperforall.php?paper=IJNRD2305828.

[4] (2022). Traffic Prediction Using Machine Learning. [Online]. Available: https://www.researchgate.net/publication/359387501_Traffic_Prediction_Using_Machine_Learning.

[5] (2021). Traffic Violation Detection in India Using Genetic Algorithm. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666285X21000844.

[6] (2020). Adopting Machine Learning for Driver Risk Assessment. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC7400276.

[7] (2019). Machine Learning Techniques for Traffic Prediction Systems. [Online]. Available: https://ieeexplore.ieee.org.

[8] (2018). Deep Learning Approaches for Traffic Flow Prediction. https://ieeexplore.ieee.org. [Online]. Available:

[9] (2017). Learning Traffic as Images: CNN for Traffic Prediction. [Online]. Available: https://arxiv.org/abs/1701.04245.

[10] (2016). Deep Learning for Short-Term Traffic Flow Prediction. [Online]. Available: https://arxiv.org/abs/1604.04527.

[11] (2015). Intelligent Traffic Management Using Machine Learning. [Online]. Available: https://ieeexplore.ieee.org.

[12] (2014). A System for Traffic Violation Detection. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC4279580.

[13] (2013). Traffic Flow Prediction Using Data Mining Techniques. [Online]. Available: https://ieeexplore.ieee.org.

[14] (2012). Machine Learning Methods for Traffic Analysis and Prediction. [Online]. Available: https://ieeexplore.ieee.org.

[15] (2011). Traffic Prediction Using Neural Networks. [Online]. Available: https://ieeexplore.ieee.org.

[16] (2010). Traffic Incident Prediction Using Statistical and ML Methods. [Online]. Available: https://ieeexplore.ieee.org.

[17] (2009). Traffic Modeling and Prediction Techniques. [Online]. Available: https://ieeexplore.ieee.org.

[18] (2008). Neural Network Based Traffic Prediction for Wireless Data Networks. [Online]. Available: https://link.springer.com/article/10.2991/ijcis.2008.1.4.9.

[19] (2007). Bayesian Approaches for Traffic Prediction Systems. [Online]. Available: https://ieeexplore.ieee.org.

[20] (2006). Real-Time Driver Monitoring and Traffic Prediction Systems. [Online]. Available: https://ieeexplore.ieee.org.

[21] (2005). Traffic Flow Prediction Using Time Series Models. [Online]. Available: https://ieeexplore.ieee.org.

[22] (2005) H. J. Cho and M. C. Hwang, ‘‘A stimulus-response model of day-today network dynamics,’’ IEEE Trans. Intell. Transp. Syst., vol. 6, no. 1, pp. 17–25, Mar. 2005