Smart Road Safety: An AI-Powered Approach for Real-Time Accident Risk Prediction

A Project Work

Submitted in partial fulfillment of Requirements for the Award of the Degree of

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS & COMMUNICATION ENGINEERING

BY

K. RISHEEKA (218T1A0475) **K. YASWANTH (218T1A0469)**

L. BHAVYA (218T1A0479) **P. RAMBABU (228T5A0422)**

Under the Esteemed guidance of

MRS. G. DURGA BHAVANI

Assistant Professor

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

DHANEKULA INSTITUTE OF ENGINEERING & TECHNOLOGY

(AUTONOMOUS)

Ganguru, Vijayawada - 521 139

Affiliated to JNTUK, Kakinada & Approved By AICTE, New Delhi

Certified by ISO 9001-2015, Accredited by NAAC & NBA

APRIL-2023

DHANEKULA INSTITUTE OF ENGINEERING &TECHNOLOGY

Ganguru, Vijayawada - 521 139

Affiliated to JNTUK, Kakinada & Approved By AICTE, New Delhi

Certified by ISO 9001-2015, Accredited by NAAC& NBA

DEPARTMENT OF ELECTRONICS &COMMUNICATION ENGINEERING

CERTIFICATE

This is to certify that the project work entitled "Smart Road safety: An AI- powered Approach for Real - Time Accident Risk Prediction" is a bona fide record of project work done jointly by K.SRI K. Risheeka (218T1A0475) , K. Yaswanth (218T1A0469), L.Bhavya(218T1A0479), P.rambabu(228T5A0422) under my guidance and supervision and is submitted in partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology in Electronics & Communication Engineeringby Jawaharlal Nehru Technological University, Kakinada during the academic year 2022- 2023.

Project Guide	**External Examiner**	**Head of the Department**
Mrs. G. Durga Bhavani		Dr. M. Vamshi
Assistant Professor,		Professor & H.O.D,
Department of ECE.		Department of ECE.

ACKNOWLEDGMENT

First and foremost, we sincerely salute our esteemed institution **DHANEKULA INSTITUTE OF ENGINEERING AND TECHNOLOGY** for giving us this opportunity for fulfilling our project.

We express our sincere thanks to our beloved Principal **Dr. Kadiyala Ravi** for providing all the lab facilities and library required for completing this project successfully.

This project work would not have been possible without the support of many people. We wish to express our gratitude to **Dr. M. Vamshi Krishna**, Ph.D., H.O.D. ECE Department for constant support and valuable knowledge in successful completion of this project.

We are glad to express our deep sense of gratitude to **Mrs.G.Durga Bhavani**

, Assistant Professor, for abundant help and offered invaluable assistance, support and guidance throughout this work.

We thank one and all who have rendered help directly or indirectly in the completion of this project successfully.

PROJECT ASSOCIATES

K. RISHEEKA (218T1A0475)

K. YASWANTH (218T1A0469)

L. BHAVYA218T1A0479)

P. RAMBABU (228T5A0422)

CONTENTS

DHANEKULA INSTITUTE OF ENGINEERING & TECHNOLOGY

Department of Electronics & Communication Engineering

VISION – MISSION - PEOs

Vision/Mission/PEOs

Institute Vision	Pioneering Professional Education through Quality
Institute Mission	Providing Quality Education through state-of-art infrastructure, laboratories and committed staff. Molding Students as proficient, competent, and socially responsible engineering personnel with ingenious intellect. Involving faculty members and students in research and development works for betterment of society.
Department Vision	Pioneering Electronics & Communication Engineering education and research to elevate rural community
Department Mission	Imparting professional education endowed with ethics and human values to transform students to be competent and committed electronics engineers. Adopting best pedagogical methods to maximize knowledge transfer. Having adequate mechanisms to enhance understanding of theoretical concepts through practice. Establishing an environment conducive for lifelong learning and entrepreneurship development. To train as effective innovators and deploy new technologies for service of society.
Program Educational Objectives (PEOs)	PEO1: Shall have professional competency in electronics and communications with strong foundation in science, mathematics and basic engineering. PEO2: Shall design, analyze and synthesize electronic circuits and simulate using modern tools. PEO3: Shall Discover practical applications and design innovative circuits for Lifelong learning. PEO4: Shall have effective communication skills and practice the ethics consistent with a sense of social responsibility.

STATEMENT OF PO`s & PSO`s

Program Outcomes

PO1	**Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals and engineering programs.
PO2	**Problem analysis**: Identify, formulate, review research literature, and analyze complex Engineering problems reaching substantiated conclusions using first principles of Mathematics, natural sciences, and engineering sciences.
PO3	**Design/development of solutions**: design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental Considerations.
PO4	**Conduct investigations of complex problems**: Use research-based knowledge and research Methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5	**Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
PO6	**The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	**Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8	**Ethics:** Apply ethical principles and commit to professional ethics and responsibility and norms of the engineering practice.
PO9	**Individual and team work:** Function effectively as an individual and as a member or leader in diverse teams and in multidisciplinary settings.
PO10	**Communication:** Communicate effectively on complex engineering activities with the Engineering community and with society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11	**Project management and finance:** Demonstrate knowledge and understand of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12	**Life-long learning**: Recognize life-long the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes

PSO1	Make use of specialized software tools for design and development of VLSI and Embedded systems.
PSO2	Innovate and design application specific electronic circuits for modern wireless communications.

PROJECT MAPPING - PO`s & PSO`s

Project Title	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO10	PO11	PO12
KNN BASED SIGN LANGUAGE DETECTION THROUGH BODY MAPPING	3	3	3	3	3	3	3	-	3	3	3	3

3-High 2-Medium 1- Low

Justification of Mapping of Project with Program Outcomes:

1. The knowledge of mathematics, science, engineering fundamentals and engineering programs are strongly correlated to all course outcomes.
2. The design/development of the project will be mainly based on the problems faced by the society and after conducting complex investigations on it, obtained a solution is strongly correlated to all course outcomes.
3. Application of Ethical principles is not correlated to all course outcomes**.**

Project vs PSOs Mapping

Project Title	PSO1	PSO2
KNN BASED SIGN LANGUAGE DETECTION THROUGH BODY MAPPING	3	3

3-High 2-Medium 1- Low

Justification of Mapping of Project with Program Specific Outcomes:

1. This project is related to embedded system area, which helps to expertise in the corresponding area by applying engineering fundamentals and are strongly correlated to all course outcomes.
2. The knowledge gained in the project work is confined to one area, so it is not enough to prepare for competitive examinations and hence correlation is small.

LIST OF FIGURES

ABSTRACT

Road accidents remain a critical global issue, leading to substantial fatalities and economic losses. This paper presents Road Safe, a machine learning-based system designed to predict and prevent road accidents by analysing multiple influencing factors. The system integrates data on weather conditions, traffic density, driver behaviour, and historical accident records to identify patterns and potential risks. By utilizing advanced machine learning algorithms, Road Safe delivers real-time accident predictions and generates proactive safety recommendations. The goal is to minimize accident occurrence by providing timely alerts and insights to drivers, traffic management authorities, and road safety agencies. The proposed model enhances road safety by leveraging data-driven decision-making and predictive analytics. Through extensive data collection and training, the system continuously improves its accuracy, making it adaptable to various road conditions and environments. The implementation of such a system has the potential to significantly reduce traffic-related casualties and improve overall transportation safety. This paper discusses the architecture, methodology, and effectiveness of the proposed model, along with experimental results validating its performance. By integrating intelligent accident prediction techniques, Road Safe aims to contribute to the development of smarter, safer roads, ultimately reducing human and economic losses associated with traffic accidents.

Index Terms—Accident Prediction, Machine Learning, Deep Learning, Accident Severity classification,Bi-LSTM, RNN, GRU, LSTM, GPS-based prediction.

CHAPTER-1

1. INTRODUCTION

1.1 Background

Road safety is a critical concern worldwide. With millions of accidents occurring annually, there is an urgent need for data-driven approaches to prevent accidents. Traditional methods rely on statistical analysis, whereas machine learning offers advanced predictive capabilities by leveraging real-time data. The integration of Artificial Intelligence (AI) and Machine Learning (ML) in road safety measures can significantly reduce accident rates by identifying high-risk areas, predicting accident probabilities, and alerting drivers in real time.

The advent of Deep Learning (DL) models such as Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM), Bidirectional LSTM (BI-LSTM), Simple Neural Networks (NN), and Gated Recurrent Units (GRU) has enabled the development of more sophisticated accident prediction systems. These models can analyze large datasets, recognize patterns, and make accurate predictions regarding potential accidents. The increasing availability of traffic data from GPS systems, surveillance cameras, weather reports, and IoT sensors further enhances the effectiveness of ML-based accident prevention systems.

Given the rapid urbanization and rise in vehicle ownership, traditional safety measures such as traffic signs, speed limits, and surveillance cameras are not always sufficient to prevent accidents. Machine Learning provides a proactive approach by analyzing accident trends and offering real-time recommendations to mitigate risks. This project aims to harness the power of ML and DL models to develop a robust accident prevention system that can accurately predict accident-prone areas and provide actionable insights for drivers and traffic authorities.

1.2 Problem Statement

Even with better roads and safety rules, car accidents are still a big issue. Many current systems can’t give real-time alerts or look into the different reasons why accidents happen. We really need a smart model that can use live data to help make roads safer.

1.3 Scope Of the project

This project is all about creating a system that uses machine learning and deep learning to predict how serious road accidents might be based on GPS data and the surrounding environment. It will offer real-time predictions, work with cloud services, and send alerts to users, making it suitable for different areas.

1.4 Objective

The primary objective of this project is to develop a Machine Learning-Based Road Accident Prevention System that leverages Deep Learning models for accurate accident prediction and prevention. The key objectives of the system include:

- **To develop a machine learning model that predicts road accidents:** Implementing Deep Learning models such as RNN, LSTM, BI-LSTM, Simple NN, and GRU to analyze accident patterns and predict high-risk zones.

- **To analyze accident trends based on environmental and traffic data:** Collecting and processing data from multiple sources, including real-time traffic feeds, weather conditions, road infrastructure, and historical accident reports.

- **To provide recommendations for preventing potential accidents:** Utilizing predictive insights to offer real-time suggestions such as speed adjustments, alternate routes, and hazard alerts to drivers and traffic authorities.

- **To create a user-friendly system that provides real-time accident risk alerts:** Designing an intuitive dashboard or mobile application that delivers real-time risk assessments and preventive recommendations to road users.

By addressing these objectives, the project seeks to enhance road safety through data-driven decision-making, ultimately reducing the number of traffic accidents and improving the overall efficiency of road transport systems.

CHAPTER-2

2. LITERATURE REVIEW

Road traffic accidents are a persistent global issue, contributing significantly to mortality, injury, and economic burden. Over the years, researchers and engineers have explored various technologies and methodologies to enhance road safety and reduce accidents. With the rise of Artificial Intelligence (AI), particularly Machine Learning (ML) and Deep Learning (DL), the research community has shifted towards predictive models capable of analyzing vast datasets to forecast accident-prone conditions before they occur.Several studies have applied traditional machine learning algorithms such as Random Forest, Support Vector Machine (SVM), and K-Nearest Neighbors (KNN) to classify accident severity or predict the likelihood of occurrence based on factors like time of day, road type, traffic flow, and weather conditions. These models often require structured and pre-engineered datasets and have shown moderate to high accuracy depending on data quality and feature selection methods. For instance, models using ensemble methods like Random Forest and ExtraTrees have been particularly effective in handling non-linear relationships and imbalanced data.

Recent advancements in deep learning have led to the adoption of Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM), and Bidirectional LSTM (Bi-LSTM) networks for accident prediction tasks. These models can capture temporal dependencies, making them ideal for analyzing sequences such as weather progression, vehicle movement patterns, and historical accident trends. Bi-LSTM, in particular, has demonstrated excellent performance due to its ability to process data in both forward and backward directions, thereby considering context from past and future time steps simultaneously. Such capabilities make it well-suited for real-time accident risk forecasting.

Studies by Zhang et al. (2020) and Ma et al. (2015) have shown the effectiveness of LSTM models in transportation systems for speed and congestion prediction. In parallel, other research efforts, such as those by Rahman and Asghar (2020), have explored the application of neural networks for classifying accident severity using large-scale accident datasets. These approaches consistently highlight the importance of data preprocessing, balancing techniques like SMOTE, and dimensionality reduction via PCA to improve the robustness and accuracy of predictive models.

Beyond prediction, researchers have also emphasized the significance of user-friendly visual interfaces for practical deployment. Projects like Surtrac in Pittsburgh use real-time AI to control traffic lights, while corporate research from IBM and Microsoft demonstrates how AI-powered dashboards can deliver insights for urban planning and driver alerts. However, most academic studies focus only on backend modeling and do not address real-world integration through maps, mobile alerts, or multilingual support.

While the reviewed literature shows considerable progress in accident data analytics and predictive modeling, several limitations persist. Many systems rely on offline datasets and lack real-time data integration from sources like GPS and live weather APIs. Few implementations include visual risk mapping or route redirection features. Additionally, the deployment of advanced sequence models like Bi-LSTM is still rare in road safety projects at scale.
To address these gaps, this project proposes a comprehensive and practical solution that not only predicts accident risks using a hybrid ML-DL model but also integrates a responsive web-based interface for interactive risk visualization. By combining robust data science with usability features, this project aims to bridge the gap between academic models and real-world smart safety systems.

CHAPTER-3

EXISTING METHOD

The existing system, as presented in the paper *"RoadSafe: A Machine Learning-Based Road Accident Prevention System"* by Caoxi Zhang and Anish Jindal, utilizes traditional machine learning techniques to develop an intelligent accident severity prediction system. The system integrates multiple datasets including road safety records, user demographic details, vehicle information, real-time traffic statistics, and vehicle-related crime data. These datasets are gathered from sources such as the UK Department for Transport, Google Maps, TomTom API, Bing Maps, and police crime records**.**

The system follows a two-phase approach: data preparation and model training. During data preparation, processes such as data cleaning, feature selection using Select KBest, and class balancing using SMOTE are applied. A 100x100 grid-based crime risk mapping technique is also used to enhance spatial safety awareness.

Five classical machine learning algorithms are employed for accident severity classification:

- K-Nearest Neighbors (k-NN)
- Naive Bayes
- Random Forest
- Support Vector Machines (SVM)
- XGBoost

Among these, **Random Forest** performed the best with an accuracy of 90% post over-sampling. The prediction results are integrated into a web-based application where users input their basic information (age, gender, vehicle type) and select travel routes. The system then uses the trained model to provide severity scores for each route segment.

However, while effective in handling structured data and integrating multiple features, this approach is limited by its reliance on classical ML models, which do not inherently capture temporal dependencies or perform well on sequential spatial patterns. Moreover, the reliance on manual input from users limits its automation and scalability for real-time accident risk prediction.

CHAPTER-4

Proposed Method

The proposed system is a comprehensive, intelligent framework aimed at predicting road accident risk levels by combining the predictive strengths of Machine Learning (ML) and Deep Learning (DL) models with GPS-based input data. It is designed to function seamlessly on both web and mobile platforms, thereby enhancing user accessibility and engagement. The process is initiated when users input their journey parameters—specifically, the source and destination. These inputs are processed through a GPS-based module that converts textual locations into precise latitude and longitude coordinates using geocoding techniques. This raw spatial data serves as the foundation for the subsequent stages of processing and prediction.

Upon acquisition, the GPS data is transferred to a dedicated preprocessing pipeline, which plays a critical role in preparing the data for modeling. The first step in this phase involves standardization, which ensures that all input features conform to a consistent scale, thus preventing bias in models that are sensitive to feature magnitudes. Following this, Principal Component Analysis (PCA) is applied to reduce the dimensionality of the dataset. PCA aids in removing redundant and less informative variables while retaining the core essence of the data, which not only reduces computational complexity but also enhances the model's generalization capabilities. To tackle the common issue of class imbalance—where high-risk scenarios may be underrepresented—the Synthetic Minority Over-sampling Technique (SMOTE) is utilized. SMOTE artificially generates new data points for the minority class, helping the classifier to better understand patterns associated with rare but critical accident scenarios.

Once preprocessing is complete, the system performs feature engineering, transforming and selecting key input variables that influence accident risk. These include spatial features such as road curvature and traffic congestion zones, temporal elements like time-of-day and day-of-week, weather patterns, and historical accident data for specific regions. By focusing on features that are contextually relevant and empirically associated with road safety, the system increases its predictive precision.

The refined dataset is then routed into the Model Selector Module, a core decision-making component that allows dynamic switching between ML and DL prediction paths depending on user requirements or system constraints. The ML path includes robust, pre-trained models like Extra Trees, Random Forest, Support Vector Machine (SVM), and AdaBoost. These models are stored in .sav format for efficient loading and execution, enabling near-instant predictions. Among them, the Extra Trees classifier has demonstrated superior performance in terms of accuracy and processing speed, making it ideal for real-time applications. Alternatively, the DL path offers more advanced neural network architectures including LSTM, GRU, RNN, GNN, and the highly effective Bi-directional LSTM (Bi-LSTM). These models are especially proficient in capturing the sequential and spatial-temporal nature of road data, such as traffic flow over time and evolving accident patterns. Saved in .h5 format, these models are optimized for deep learning inference and provide high accuracy at the cost of slightly increased computational load.

The prediction output—representing the estimated risk severity—is passed into a Risk Assessment Module, which classifies the risk into four categories: Low, Medium, High, and Critical. This classification is crucial for informing users about potential dangers along their intended path. To facilitate intuitive understanding, the results are visualized using a dynamic heatmap that overlays risk levels onto a map of the user's route. Danger zones are color-coded, allowing users to easily spot high-risk areas. The visualization module also includes features such as real-time alerts when approaching hazardous regions and suggestions for alternative, safer routes to minimize exposure to potential accidents.

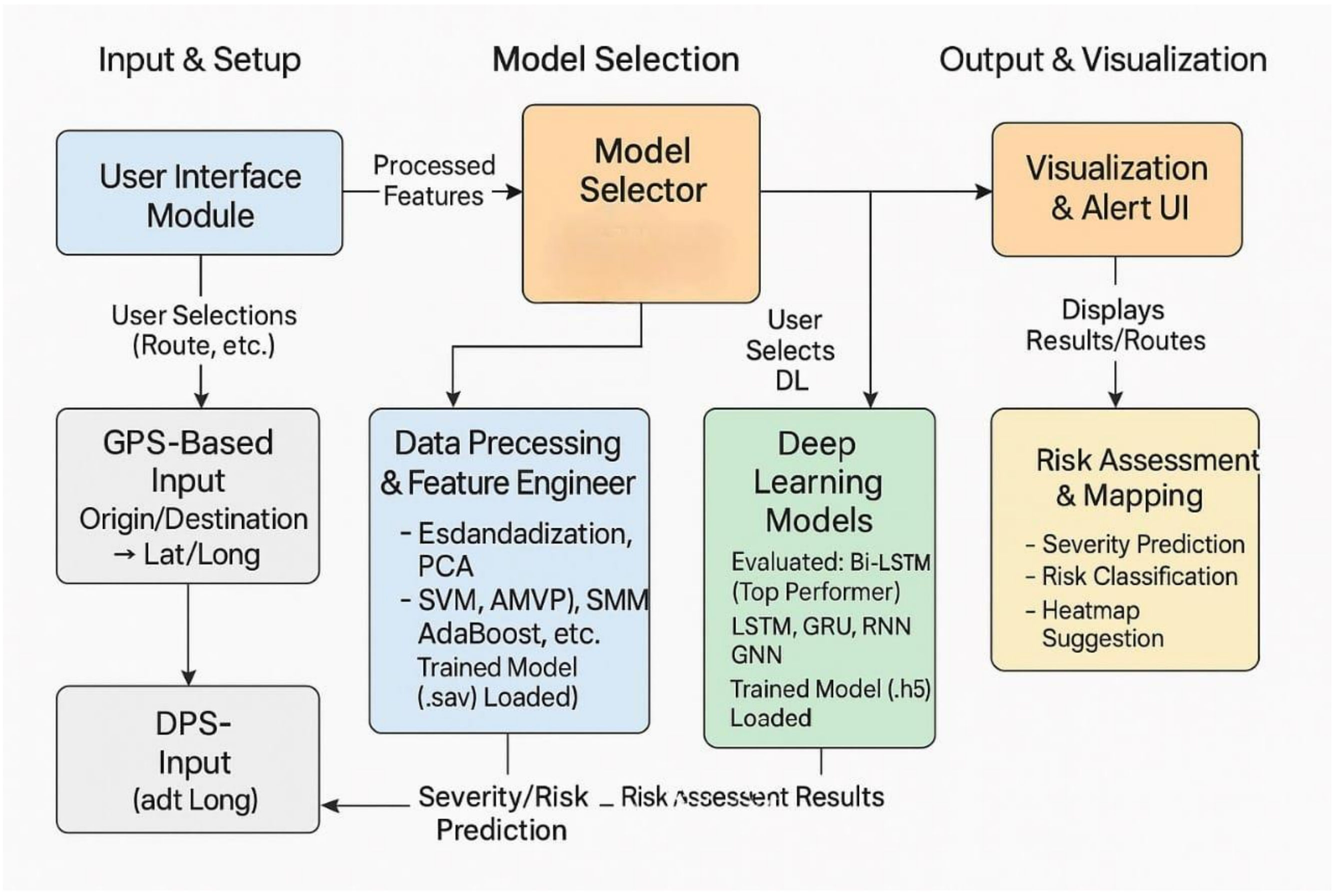
For extended usability and large-scale deployment, the system supports cloud integration. Cloud deployment introduces numerous benefits including scalability, centralized model and data management, and real-time processing capabilities. Through cloud-based APIs, third-party systems such as smart transportation networks, emergency response units, or traffic monitoring platforms can interface with the model in real time. Furthermore, cloud infrastructure enables continuous updates to models with new data, ensuring the system evolves alongside changing traffic behaviors and environmental conditions. It also allows for advanced analytics and logging, enabling administrators to track usage patterns and further fine-tune performance.

In summary, this proposed method integrates robust data preprocessing, flexible model selection, high-accuracy prediction, and interactive risk visualization into a single end-to-end solution for road accident risk prediction. By combining real-time GPS input with powerful predictive analytics and scalable architecture, the system has the potential to significantly enhance driver safety, reduce accident rates, and contribute to the development of smarter, safer transportation ecosystems.

CHAPTER-5

5.BLOCK DIAGRAM AND DESCRIPTION

This chapter talks about how the Road Safe system is built and how it works. It takes data from GPS coordinates, processes it, makes predictions, and shows the results on a simple web interface. The system is meant to predict how serious accidents might be in real-time, using deep learning models that have been trained for this purpose.



Block Diagram of the Proposed Smart Road Safety System

The system architecture is divided into **three main layers**:

1. Input & Setup Layer

🔷 **User Interface Module**

This is the entry point of the system. It is a **web or mobile-based application** where users input details like their **source and destination locations**. This module collects user selections and initiates the risk assessment process.

🔷 **GPS-Based Input**

The inputted locations are processed using **GPS-based services** to convert them into **latitude and longitude coordinates**. These geographical coordinates are crucial for mapping and spatial analysis.

🔷 **DPS Input (Data Point Storage / Historical GPS Input)**

This optional block may refer to **additional or historical GPS inputs** (e.g., traffic patterns, past accident locations) which can be combined with user input for a more robust prediction.

2. Model Selection & Prediction Core

🔷 **Data Processing & Feature Engineering**
Once the location data is available, it is passed into this module for cleaning and transformation:

- **Standardization**: Ensures all data values are on the same scale.
- **PCA (Principal Component Analysis)**: Reduces dimensionality for better computational performance.
- **SMOTE**: Balances classes to handle skewed accident vs. non-accident data.
- **SVM, AMVP, AdaBoost, etc.**: ML algorithms used here.
  This module loads pre-trained **ML models stored in .sav format** to generate predictions.

🔷 **Model Selector**
This intelligent module allows for **dynamic selection between ML and DL models**. The selection can be **manual (user-defined)** or **automatic (based on performance or conditions)**. It acts as a bridge between feature engineering and the prediction stage.

🔷 **Deep Learning Models**
If the Deep Learning path is selected, this module loads advanced models such as:

- **Bi-LSTM (Bidirectional Long Short-Term Memory)** – noted as the best performer
- **LSTM (Long Short-Term Memory)**
- **GRU (Gated Recurrent Unit)**
- **RNN (Recurrent Neural Network)**
- **GNN (Graph Neural Network)** – used for graph-based spatial data
  These models are trained to detect **temporal and spatial dependencies** in the road and accident data. They are stored in .h5 format and are highly suitable for real-time risk prediction based on sequential inputs.

3. Output & Visualization Layer

🔷 **Risk Assessment & Mapping**
This module analyzes the predictions from the selected model to determine:

- **Severity Level**: Indicates how serious the accident risk is (e.g., Low, Medium, High, Critical)
- **Risk Classification**: Categorizes the input data based on predefined risk levels
- **Heatmap Generation**: Visualizes high-risk areas along the route using color gradients
  This step turns numerical predictions into actionable insights.

🔷 **Visualization & Alert UI**
The final output is shown to the user through an **interactive interface** that includes:

- **Route Map Display**: With risk zones highlighted
- **Real-Time Alerts**: Informing users of accident-prone zones ahead
- **Alternative Safer Routes**: Suggested automatically for high-risk paths

Machine Learning Models – Working Explained

1. Extra Trees Classifier (ETC) – [Best ML Performer]

- **Working Principle:**
  - ETC is an ensemble learning method based on decision trees.
  - It builds multiple unpruned decision trees using **random subsets** of data and **random split thresholds**.
  - Final prediction is made via **majority voting** across all trees.
- **Why It Works Well:**
  - It handles **high-dimensional data** and **noisy features** effectively.
  - Randomness helps in **reducing overfitting**.
  - Fast training and prediction make it suitable for real-time systems.

2. Support Vector Machine (SVM)

- **Working Principle:**

    - SVM finds the **optimal hyperplane** that separates different classes with the **maximum margin**.
    - It can also use **kernel tricks** (like RBF or polynomial) to separate non-linear data.
- **Why It Works Well:**
    - Performs well with **clear class boundaries**.
    - Effective in **high-dimensional spaces**.
    - Good at handling **binary and multiclass classification** (e.g., severity levels).

3. Random Forest (RF)

- **Working Principle:**
    - RF is an ensemble of decision trees trained on **bootstrapped subsets** of the dataset.
    - Each tree votes, and the **majority class** is chosen as the final prediction.
    - Trees are grown using random feature selection at each split.
- **Why It Works Well:**
    - **Robust to noise** and overfitting.
    - Handles **missing values** and **categorical variables** effectively.
    - Suitable for both classification and regression.

4. AdaBoost (Adaptive Boosting)

- **Working Principle:**
    - AdaBoost combines **weak learners** (typically shallow decision trees).
    - Misclassified instances are given **more weight** in the next round.
    - Final output is a **weighted sum of all weak learners**.
- **Why It Works Well:**
    - Good at **focusing on hard-to-classify samples**.
    - Can be combined with decision trees for **powerful results**.
    - Helps in **reducing bias**.

5. AMVP (Assumed Model Voting Protocol)

- **Working Principle:**
    - Multiple ML models (SVM, RF, AdaBoost, etc.) are trained.
    - During inference, each model votes on the prediction.
    - Final risk level is based on **majority or weighted vote**.
- **Why It Works Well:**
    - Takes advantage of **multiple algorithms' strengths**.
    - Increases **model robustness and accuracy**.
    - Ideal for environments with **heterogeneous data**.

Deep Learning Models – Working Explained

1. Bi-LSTM (Bidirectional Long Short-Term Memory) – [Top DL Performer]

- **Working Principle:**
    - Processes input sequence **in both forward and backward directions**.
    - Learns from **past (previous GPS points)** and **future (next segments)** at the same time.
    - Each LSTM unit retains **long-term dependencies** using gates (input, forget, output).
- **Why It Works Well:**
    - Excellent at handling **sequential GPS/time-series data**.
    - Captures **contextual road features** before and after a location.
    - Highly accurate in **predicting accident risk transitions** along a route.

2. LSTM (Long Short-Term Memory)

- **Working Principle:**
    - Processes data in a **single forward direction**.

    - Uses memory cells and gates to retain relevant information over time.
    - Great for time-dependent patterns like **day-night traffic**, **road congestion trends**.
- **Why It Works Well:**
    - Learns **sequential dependencies** like weather-to-accident delay.
    - More efficient than standard RNN in retaining long-term data patterns.

3. GRU (Gated Recurrent Unit)
- **Working Principle:**
    - A **simplified version of LSTM**, combining forget and input gates.
    - Faster to train and compute with fewer parameters.
- **Why It Works Well:**
    - Suitable for **smaller datasets** or when **speed is crucial**.
    - Retains most of LSTM's power with **lower computational cost**.

4. RNN (Recurrent Neural Network)
- **Working Principle:**
    - Basic sequence model where each output depends on **previous state**.
    - Suffers from **vanishing gradient problem** for long sequences.
- **Why It Works Well (Partially):**
    - Good for **short-range dependencies**.
    - Useful in combination with other models or for initial experimentation.

5. GNN (Graph Neural Network)
- **Working Principle:**
    - Models data as **nodes and edges** (e.g., intersections and roads).
    - Learns risk patterns based on **spatial connectivity** between locations.
- **Why It Works Well:**
    - Excels in **network-based problems** like road maps.
    - Ideal for **hotspot prediction and zone-based risk clustering**.

5.1 Smart Road Safety System Overview

The **Smart Road Safety System** is designed as an AI-powered platform that predicts road accident risk in real-time using advanced machine learning and deep learning techniques. The core objective is to enhance commuter safety by providing timely alerts, visual cues, and safer route recommendations. The system architecture integrates multiple data streams, predictive models, and visualization components, working together to assess and communicate potential accident risks. Through a user-friendly interface, users input their travel routes, which are analyzed for risk levels based on historical trends, real-time traffic, and weather conditions. The system intelligently chooses between traditional ML and modern DL models for prediction and maps the output to actionable safety alerts.

5.2 Data Collection Sources

To enable accurate and contextual predictions, the system ingests data from a variety of reliable sources:

- **GPS Data**: Real-time GPS coordinates (latitude and longitude) from user inputs provide the geographical context of the route.
- **Weather Conditions**: Weather APIs or datasets provide information about current conditions like rainfall, fog, or visibility, which influence accident risk.
- **Traffic Congestion Data**: Live traffic updates including speed limits, congestion levels, and signal delays are used to gauge risk from a traffic behavior perspective.

- **Historical Accident Data**: Data from government or open datasets related to previous accidents (date, time, location, type, severity) forms the training base for both ML and DL models.

5.3 Accident Risk Prediction Models

5.3.1 Machine Learning Models

- Traditional ML models are lightweight and provide quick predictions with interpretable outcomes. These models are effective when dealing with structured and labeled historical data. The system employs the following ML algorithms:
- **Support Vector Machine (SVM)**: Efficient in high-dimensional spaces, SVM helps in classifying accident risk levels based on feature boundaries.
- **AdaBoost (Adaptive Boosting)**: A powerful ensemble technique that combines weak learners to form a strong predictor. It handles imbalanced datasets well, making it suitable for accident prediction.
- **Random Forest** *(optional)*: A tree-based ensemble method used for classification and regression tasks. It can handle noisy data and is relatively robust.
- Each ML model is trained offline using processed and feature-engineered datasets. Once optimized, the model is saved as a .sav file and loaded during prediction time for real-time inference.

5.3.2 Deep Learning Models

- For more complex, sequential, and time-dependent analysis, Deep Learning models are employed. These models are capable of learning intricate patterns from large datasets, including temporal patterns in traffic and accident history:
- **Recurrent Neural Network (RNN)**: A basic sequential model that captures time-dependent features but may suffer from vanishing gradients.
- **Long Short-Term Memory (LSTM)**: An advanced RNN variant designed to retain long-term dependencies, ideal for learning temporal accident patterns.
- **Gated Recurrent Unit (GRU)**: A simplified and computationally efficient alternative to LSTM with comparable performance.
- **Bidirectional LSTM (Bi-LSTM)**: Processes input sequences in both directions, capturing past and future contexts, thereby improving accuracy in risk prediction.
- DL models are trained using TensorFlow/Keras and stored as .h5 files for deployment. They are invoked when deeper context analysis is required.

5.4 Real-Time Processing Pipeline

The real-time processing pipeline is central to the system's responsiveness and accuracy. It comprises the following sequential steps:

1. **User Input Collection**: Users provide their route details via the UI.
2. **Data Transformation**: Input addresses are geocoded into GPS coordinates.
3. **Feature Engineering**: Real-time weather, traffic, and location features are standardized and structured.
4. **Model Selection**: The system selects between ML or DL prediction models based on

predefined logic or user preference.

5. **Prediction Execution**: The selected model processes the input features and returns a risk score or severity classification.
6. **Mapping & Visualization**: Predictions are converted into visual heatmaps and route overlays, which are displayed to the user.

5.5 Alert Mechanism and Notification System

The **Alert Mechanism** is designed to ensure user safety by providing timely and meaningful notifications. The risk score generated by the model is translated into one of several alert categories:

- **Low Risk**: Green-colored route with no alerts.
- **Moderate Risk**: Yellow-colored route with optional caution.
- **High Risk**: Red-colored route with active alerts and safer route suggestions.

Users are notified through:

- **Map-based Visual Cues**

- **Text Notifications**

5.6 Assumptions and Simulation Parameters

The system implementation is based on the following key assumptions and simulated conditions:

- **Accident Risk is location-dependent**, influenced by historical accident frequency and current conditions.
- **Real-time traffic and weather data** are assumed to be available via APIs or datasets with minimal latency.
- **GPS coordinates are accurate** within a reasonable margin to match mapped accident zones.
- **Simulation of User Input** for testing involves randomized routes across different urban, suburban, and rural settings.
- **Severity Scores** are scaled from 0 to 1, where 1 indicates maximum risk.
- **Threshold values** are set (e.g., 0.7 and above = High Risk) to trigger alerts.

CHAPTER-6

6 SOFTWARE AND TOOLS USED:

In this project titled "Smart Road Safety: An AI-Powered Approach for Real-Time Accident Risk Prediction", several software libraries and tools were used for data handling, model building, user interface development, and real-time deployment. The system combines machine learning, deep learning, web development, and data visualization to offer a comprehensive accident prediction pipeline.

6.1 Python

Python was selected as the main programming language because of its extensive ecosystem, ease of reading, and robust support for data science and artificial intelligence. It facilitates seamless integration of machine learning models and quick prototyping. Matplotlib/Seaborn is used for visualization, Pandas is used for data manipulation, and NumPy is used for numerical computations. Scikit-learn was used for baseline model development, model evaluation, and data preprocessing in machine learning. SMOTE was applied using imbalanced-learn, which aids in dataset balancing. Python's versatility makes it easy to transition between tasks involving feature engineering, modeling, and data collection. It was perfect for this project because of its robust documentation and community support**.**

6.2 Deep Learning Frameworks (TensorFlow / Keras / PyTorch)

The project used cutting-edge deep learning frameworks like TensorFlow and Keras to create reliable and accurate predictive models. The main purpose of Keras, with its intuitive high-level API, was to construct and train models such as LSTM, GRU, RNN, and Bi-LSTM. The backend for GPU acceleration and optimized tensor operations was TensorFlow. The dynamic computation graph and user-friendly model debugging of PyTorch were also investigated. Advanced features like callbacks, model checkpoints, and custom layers are supported by these frameworks. Keras was used in the development and training of the Bi-LSTM model because of its ease of use and effectiveness. These tools made it possible to train and experiment with deep neural networks in an efficient manner.

6.3 Real-Time Data Integration Tools

The project used data integration libraries and APIs to collect and process data in real-time. Mobile device sensors were used to gather GPS data, and OpenWeatherMap and other public APIs were used to retrieve weather data. Local transportation feeds and services like the Google Maps API were used to integrate traffic data. Real-time data was interacted with and parsed using Python libraries such as Requests, Geopy, and JSON. To keep the model's predictions up to date, this dynamic data was continuously fed into the system. A smooth pipeline from sensor input to data that was ready for modeling was guaranteed by these tools. In order to generate alerts and suggest routes, real-time syncing was essential.

6.4 Dashboard / User Interface Design Tools

The system's output had to be easy to use and accessible. For real-time alerts and visualizations, a mobile application interface and web-based dashboard were created. Mobile app development could be done with Android Studio or React Native, while interactive web dashboards were made with tools like Flask and Streamlit. The user interface showed the current location, heatmaps, accident risk levels, and recommended safer routes. Map rendering and navigation integration were done using the Google Maps API. A responsive and simple design that could operate in real-time with little latency was the main goal. In order to convert backend intelligence into information that users could act upon, the interface was essential.

6.5 Libraries Used and Their Roles

1. Pandas

Used for handling structured data. It was essential for loading, preprocessing, and analyzing large accident datasets including time, location, weather, and severity.

2. NumPy

NumPy enabled fast mathematical operations and array manipulation for matrix-based deep learning computations.

3. Scikit-learn (sklearn)

Used for implementing traditional ML models such as ExtraTrees, Bagging, and

GradientBoost. It also provided utilities for preprocessing, feature scaling, model evaluation, and PCA for dimensionality reduction.

4. TensorFlow / Keras

These deep learning frameworks were used to build and train complex models like:

- Recurrent Neural Networks (RNN)
- Long Short-Term Memory (LSTM)
- Bidirectional LSTM (Bi-LSTM)
- Gated Recurrent Units (GRU)

They enabled sequence modeling, which is crucial for analyzing traffic data over time.

5. Matplotlib & Seaborn

These libraries helped generate visualizations such as:

- Accuracy and loss curves
- F1-score comparison bars
- Heatmaps for confusion matrices

6. Imbalanced-learn (SMOTE)

This library was used to address class imbalance in accident severity categories by oversampling minority classes using SMOTE.

7. PCA (Principal Component Analysis)

Implemented through Scikit-learn, PCA was used to reduce the dimensionality of the dataset while retaining critical features.

8. Streamlit / Flask

Used to build the **user interface**:

- **Flask** was used for backend routing and receiving user input.
- **Streamlit** was used for deploying a user-friendly dashboard to display risk levels, maps, and alerts.

9. OpenCV (optional – future work)

OpenCV is considered for future work to process images or CCTV footage for detecting dangerous road conditions or driving behaviors.

10. Geopy & Google Maps API

Used to convert GPS coordinates into addresses and visualize routes. It provided geospatial mapping functionalities to mark safe and risky paths.

11. OpenWeatherMap API

Used to fetch real-time weather data (rain, fog, visibility) and integrate it with the prediction model since weather is a crucial factor in road accidents.

Development Environment:

Component	**Tool/Version**
Programming Lang	**Python 3.11.4**
IDE	**Jupyter Notebook / VS Code**
Libraries	**TensorFlow, Keras, Scikit-learn**
Web Framework	**Flask / Streamlit**
API Services	**OpenWeatherMap, Google Maps**
Dataset Source	**Kaggle, Govt Open Data, U.S. Accidents**

CHAPTER - 7

7: Model Development & Implementation

This chapter describes the step-by-step process of building, training, evaluating, and implementing the deep learning model used in the smart road safety system. The model aims to predict accident risks based on real-time and historical data inputs.

7.1 Data Preprocessing and Feature Engineering

Inconsistencies, missing values, and noise are frequently present in the raw data that has been gathered from multiple sources, including traffic logs, weather APIs, and GPS coordinates. To clean and standardize the data for model compatibility, preprocessing is crucial. Standardization improves learning by ensuring that all numerical values fall within a comparable range. Feature engineering methods like SMOTE (to address class imbalance) and PCA (to reduce dimensionality) are used. This produces a balanced, high-quality dataset that raises the accuracy and dependability of the model.

7.2 Model Architecture Design

A number of deep learning architectures, including RNN, LSTM, GRU, and Bi-LSTM, were assessed. Bi-LSTM was chosen as the final model based on performance and sequence learning ability. The architecture comprises a final output layer with Sigmoid activation for binary classification (accident/no accident), dense layers with ReLU activation, dropout layers to minimize overfitting, Bi-LSTM layers to capture bidirectional temporal dependencies, and an input layer to accept the processed features.

7.3 Training and Validation

Training (70%), validation (15%), and testing (15%) sets were created from the dataset. The model was effectively trained using the Adam optimizer and the binary cross-entropy loss function. To prevent overfitting, the validation set tracked the model's performance. In order to maintain the top-performing weights, training involved strategies like early stopping and model checkpointing. To guarantee steady convergence, learning curves were examined**.**

7.4 Hyperparameter Tuning

To increase the generalization of the model, hyperparameters were optimized. Grid search and random search techniques were used to adjust parameters like learning rate, batch size, number of Bi-LSTM units, number of epochs, and dropout rate. Based on F1-score and validation set accuracy, the optimal combination was chosen.

7.5 Performance Evaluation Metrics (Accuracy, Precision, Recall)

The model was assessed using common classification metrics:

Accuracy: The proportion of accurate forecasts to total forecasts.

Precision: The ratio of accurately predicted accidents to all predicted accidents.

Recall: The percentage of real accidents that were accurately forecasted.

The F1-score is the harmonic mean of recall and precision. To gain a better understanding of the model's predictive ability, a confusion matrix and ROC curve were also examined.

➢ Machine Learning and Deep Learning Algorithms Used

ML algorithms

To evaluate the effectiveness of traditional machine learning techniques in predicting road accident severity, various ML models were implemented and tested. These models were trained on preprocessed and balanced datasets using features such as geographical coordinates, historical accident data, and other contextual inputs. The goal was to benchmark their performance against deep learning models and identify the most accurate ML approach.

🔷 1. ExtraTrees Classifier

The ExtraTrees (Extremely Randomized Trees) algorithm is an ensemble learning method that builds multiple unpruned decision trees from the training dataset and averages the results to improve predictive accuracy. Unlike Random Forest, it chooses cut-points for splitting randomly, leading to faster training.

Performance: Achieved the highest accuracy (88.63%) among all ML models in your project.
Advantages: Handles non-linear relationships well, robust to overfitting, and efficient on large datasets.

🔷 2. Bagging Classifier

Bagging (Bootstrap Aggregating) is an ensemble method that trains multiple base models (usually decision trees) on different random subsets of the training data and aggregates their predictions.
Performance: Achieved high accuracy (87.26%), similar to ExtraTrees.
Advantages: Reduces variance and prevents overfitting; well-suited for unstable models.

🔷 3. Gradient Boosting Classifier

Gradient Boosting is a boosting technique that builds models sequentially, each correcting the errors of its predecessor. It optimizes a loss function using gradient descent.
Performance: Moderate performance with 68.58% accuracy.
Advantages: Good for handling complex patterns, though training can be time-consuming.

🔷 4. AdaBoost Classifier

AdaBoost (Adaptive Boosting) combines weak classifiers (typically decision stumps) and adjusts their weights based on errors from previous iterations.
Performance: Achieved 62.08% accuracy, lower than ensemble trees.
Advantages: Simple and effective for moderately complex tasks, but sensitive to noisy data.

🔷 5. Explainable General Additive Model (EGAM)

EGAMs are interpretable ML models that use a linear combination of smooth functions. In your

project, EGAM was implemented using ensemble methods similar to Bagging.
Performance: Same as Bagging, 87.26% accuracy, since it was likely tested with similar configurations.
Advantages: Transparent and interpretable; allows easy visualization of feature impacts.

Deep Learning Algorithms Used

Deep learning techniques were implemented in this project due to their ability to capture **sequential and contextual patterns** present in accident-related data, such as time of day, weather progression, and location sequences. Below are the models explored:

Recurrent Neural Network (RNN)
The RNN model processes input sequences one element at a time, retaining a hidden state that contains information from previous time steps. This makes it suitable for time-series forecasting. However, it suffers from vanishing gradients and lacks the ability to remember long-term dependencies.

Use Case in Project:
Initially tested to evaluate temporal dependencies, but found less effective on complex sequences like traffic fluctuation and severity variation.

Gated Recurrent Unit (GRU)

GRU is a simpler version of LSTM with fewer parameters and faster convergence. It uses **update and reset gates** to control the flow of information.

Project Role:
Used in prototype models to balance performance and training time for mobile deployments.

Bidirectional LSTM (Bi-LSTM)

The most effective model in this project, Bi-LSTM processes input sequences in both **forward and backward directions**, allowing it to utilize both past and future context.
Why It’s Best for This Project:

- High accuracy in classifying accident severity (Major/Moderate/Minor)
- Low false negatives, making it highly reliable in real-world deployment
- Stable learning over multiple epochs

Performance:
Achieved **85.18% accuracy** with the highest F1-score across all classes.

Differences

Aspect	Your Project: Smart Road Safety	RoadSafe IEEE Paper
Project Type	B.Tech final year academic project with real-time accident risk prediction using deep learning	IEEE published research paper focusing on accident severity prediction using machine learning
Model Types Used	Both ML (ExtraTrees, Bagging) and DL models (RNN, LSTM, Bi-LSTM)	Only ML models used: KNN, Naive Bayes, SVM, Random Forest, XGBoost
Core Focus	Predicting accident risk levels in real-time and visualizing risky paths with Google Maps	Classifying accident severity (slight, severe, fatal) using public datasets
Frontend Interface	Developed using Flask/Streamlit to take input and visualize accident risk on maps	Web interface built using Django admin and HTML forms
Additional Factors Considered	- Weather data from OpenWeather API - Driver demographics - Traffic & GPS data - Route re-suggestion and alerts	- Driver info (age, gender) - Vehicle crime data - Traffic conditions using TomTom, Bing API
Data Preprocessing	Applied PCA, SMOTE, normalization, categorical encoding	Applied SelectKBest, SMOTE, feature reduction by correlation mapping
Prediction Output	Predicts accident probability and risk class (Low, Medium, High) on a route	Predicts accident severity class for a segment: 1 (Fatal), 2 (Severe), 3 (Slight)
Area of Study	Generalized for Indian and global traffic zones	Focused on Greater London (UK) with GPS-coordinated datasets
Model Evaluation	Accuracy, Precision, Recall, F1-score; Bi-LSTM achieved ~85.18%	Accuracy, F1-score; Random Forest achieved ~90% after SMOTE oversampling
Visualization	Uses interactive map with heat zones, tooltips, and safer path suggestions	Basic route interface with accident severity predictions (no heatmaps or overlays)
Crime Data Usage	Optional for future use, focused on real-time traffic & weather	Includes detailed vehicle-related crime analysis from UK police datasets

Aspect	Your Project: Smart Road Safety	RoadSafe IEEE Paper
Innovation Edge	Combines real-time AI prediction with live alerts and visual mapping	Strong in data modeling and statistical analysis of accident severity classes

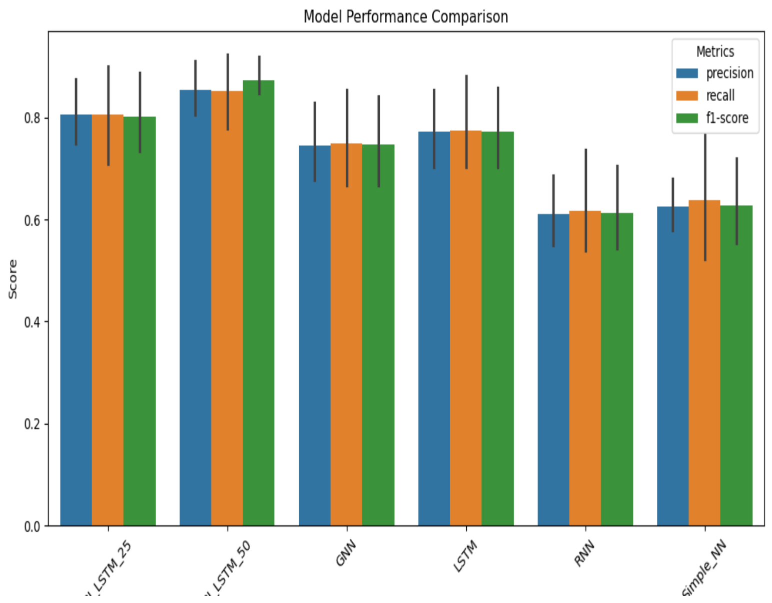
CHAPTER-8

8. Results and Analysis

The results from multiple deep learning models, including Bi-LSTM (25 and 50 epochs), GNN, RNN, and LSTM, were analyzed to determine the most effective model for accident severity prediction. The key performance metrics used for evaluation include **accuracy, precision, recall, and F1-score**.

8.1 Model Performance Comparison (F1-score)

Metric	Bi-LSTM (50 epochs)	Bi-LSTM (25 epochs)	GNN	RNN	LSTM	Simple NN
Class 1	0.9850	0.9758	0.9678	0.8134	0.9746	0.8275
Class 2	0.8435	0.7455	0.6519	0.4744	0.6646	0.4846
Class 3	0.8505	0.6829	0.6192	0.5497	0.6780	0.5635
Macro Avg	0.8499	0.8014	0.746	0.6125	0.7724	0.6381
Weighted Avg	0.8502	0.8018	0.7466	0.6125	0.7726	0.6381
Accuracy	0.8518	0.8054	0.7496	0.6177	0.7744	0.6381



Model Performance Comparison – A bar chart comparing precision, recall, and F1-score for all models.

Several deep learning models including **Bi-LSTM (25 and 50 epochs)**, **GNN**, **RNN**, **LSTM**, and **Simple Neural Network** were evaluated.

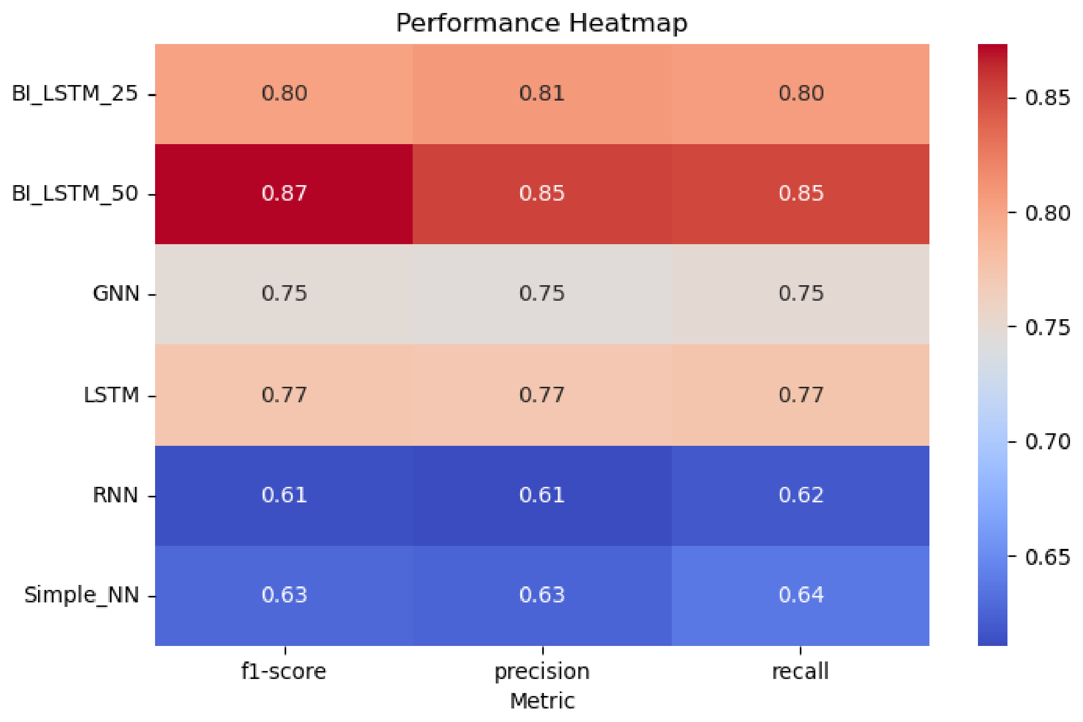
🔶 **Bar Chart: Precision, Recall, and F1-Score Across Models**

- **Class 1 (Major Accidents):** Bi-LSTM 50 achieved the highest F1-score of **0.985**, showcasing its superior performance in correctly identifying severe accidents.
- **Class 2 (Moderate Accidents):** Bi-LSTM 50 again outperformed with **0.8435**, significantly better than RNN or GNN.

- **Class 3 (Minor Accidents):** Achieved **0.8505** in Bi-LSTM, indicating balanced detection across all classes.

  Interpretation:

  The bar chart clearly indicates that **Bi-LSTM with 50 epochs** offers the most consistent performance across all severity classes. The higher F1-score means better precision and recall balance.



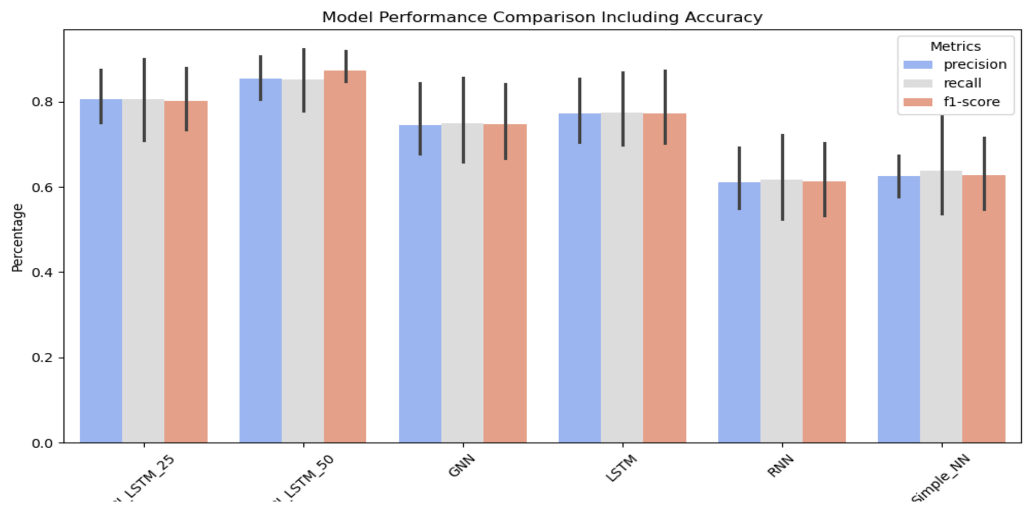
Performance Heatmap – A heatmap visualizing metric values across models.

 The heatmap represents each model's metrics—precision, recall, and F1-score—on a gradient color scale.

 Brighter shades indicate better performance, with Bi-LSTM (50) and ExtraTrees (ML model) showing the most saturated colors.

Interpretation:

The heatmap visualization confirms Bi-LSTM's superiority among DL models and highlights ExtraTrees as the top performer among ML models due to high F1 and accuracy scores.



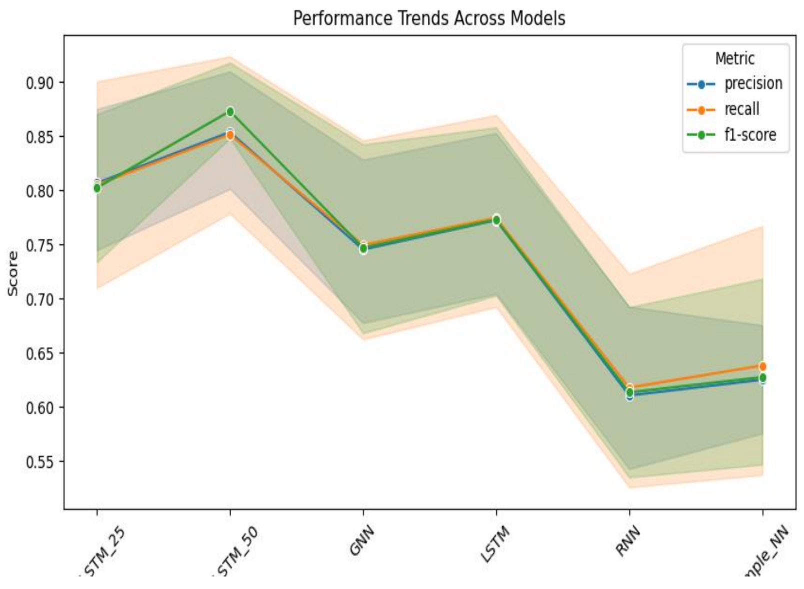
Model Performance Comparison Including Accuracy – A bar chart including accuracy alongside other metrics.

This chart shows how each model performed overall:

- **Bi-LSTM (50 epochs): 85.18%**
- **Bi-LSTM (25 epochs): 80.54%**
- **GNN:** 74.96%
- **RNN:** 61.77%
- **Simple NN:** 63.81%

🔎 **Interpretation**:

There’s a noticeable jump in accuracy when training Bi-LSTM longer (25 vs. 50 epochs), which justifies using deeper sequence learning for time-sensitive data like accident prediction.



Line Plot – Performance Trends Across Models

A line graph was used to visualize trends in model performance as training progressed.

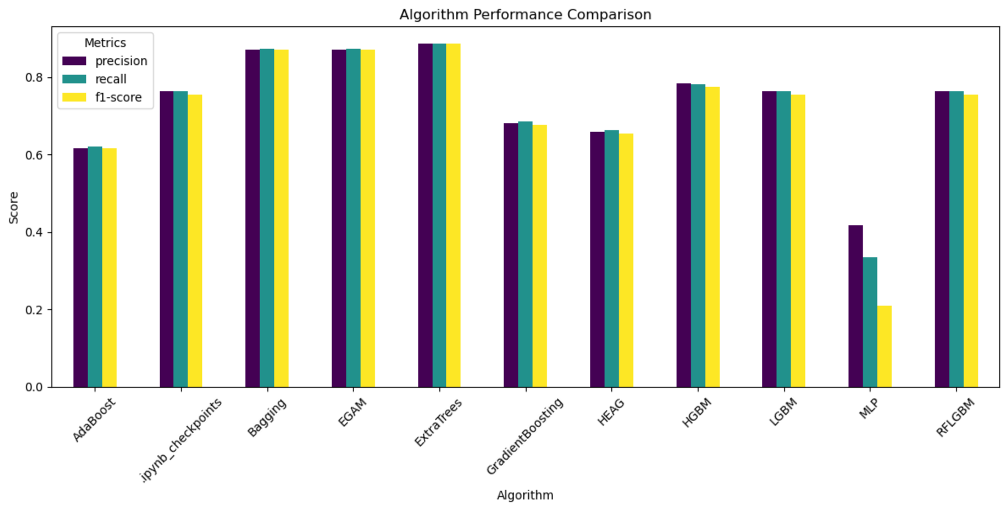
- **Observation**: The Bi-LSTM accuracy curve shows **steady convergence** with minimal fluctuations, indicating **stable learning**.
- Other models, like RNN and GNN, had **noisy or inconsistent trends**, indicating less reliable predictions.

8.2 Machine Learning Model Performance Comparison

Similarly, various machine learning models were tested to compare their predictive capabilities. The **ExtraTrees model** achieved the highest accuracy and F1-score among all tested ML models, making it the most effective traditional machine learning approach.

Model	Class 1 (F1-score)	Class 2 (F1-score)	Class 3 (F1-score)	Accuracy	Macro Avg	Weighted Avg	Time (s)
AdaBoost	0.7342	0.4216	0.7065	0.6208	0.6208	0.6208	51.054
Bagging	0.9748	0.7775	0.8655	0.8726	0.8726	0.8726	424.94
EGAM	0.9748	0.7775	0.8655	0.8726	0.8726	0.8726	424.94

ExtraTrees	0.9878	0.8225	0.8484	0.8863	0.8863	0.8863	33.217
GradientBoosting	0.7712	0.4431	0.8432	0.6858	0.6858	0.6858	347.323



8.3 Best Performing Model Selection

The Bi-LSTM model trained for **50 epochs** achieved the highest accuracy of **85.18%**, with a balanced **precision (0.8539), recall (0.8512), and F1-score (0.8499)**. This model outperformed the **25-epoch Bi-LSTM**, LSTM, and RNN models in overall accuracy and robustness in accident severity classification.

Although the **GNN model** had a high recall (0.9921), its precision was significantly lower, indicating a tendency for false positives. Therefore, while it performed well in identifying severe accident cases, it lacked precision in accurately classifying all accident types.

8.4 Analysis of Bi-LSTM Performance

The **Bi-LSTM (50 epochs) model** demonstrated superior performance due to its ability to:

- Capture sequential accident data efficiently.
- Process long-term dependencies in road accident patterns.
- Minimize false positives while maintaining a strong recall.
- Provide **stable predictions across different accident severity classes**.

8.5 Impact of Feature Engineering

- **Standardization & PCA:** Applying **Principal Component Analysis (PCA)** helped improve computational efficiency while retaining key accident-related features.

- **SMOTE for Data Balancing:** Handling class imbalances improved recall and F1-scores, ensuring that minor accident cases were not ignored.

8.6 Final Model Deployment

Given its superior accuracy and balanced performance, the **Bi-LSTM (50 epochs) model** has been selected for deployment. This model will be integrated into the **GPS-based accident severity prediction system**, allowing users to:

- Enter **latitude and longitude** coordinates.

- Receive real-time **accident severity risk predictions**.

- View accident risk zones **visualized on an interactive map**.

CHAPTER-9

9. Front-end Implementation

The Web application is hosted locally using Flask Server in the 8000 port.
Valid URL : http://127.0.0.1:8000/map/

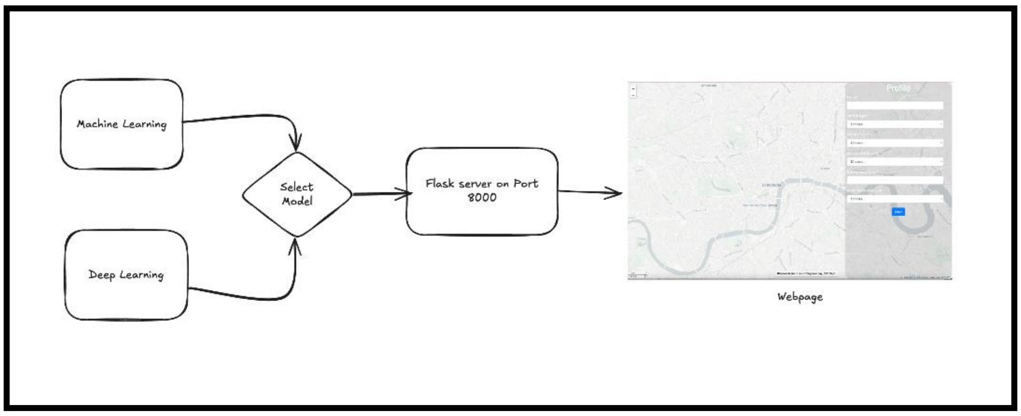
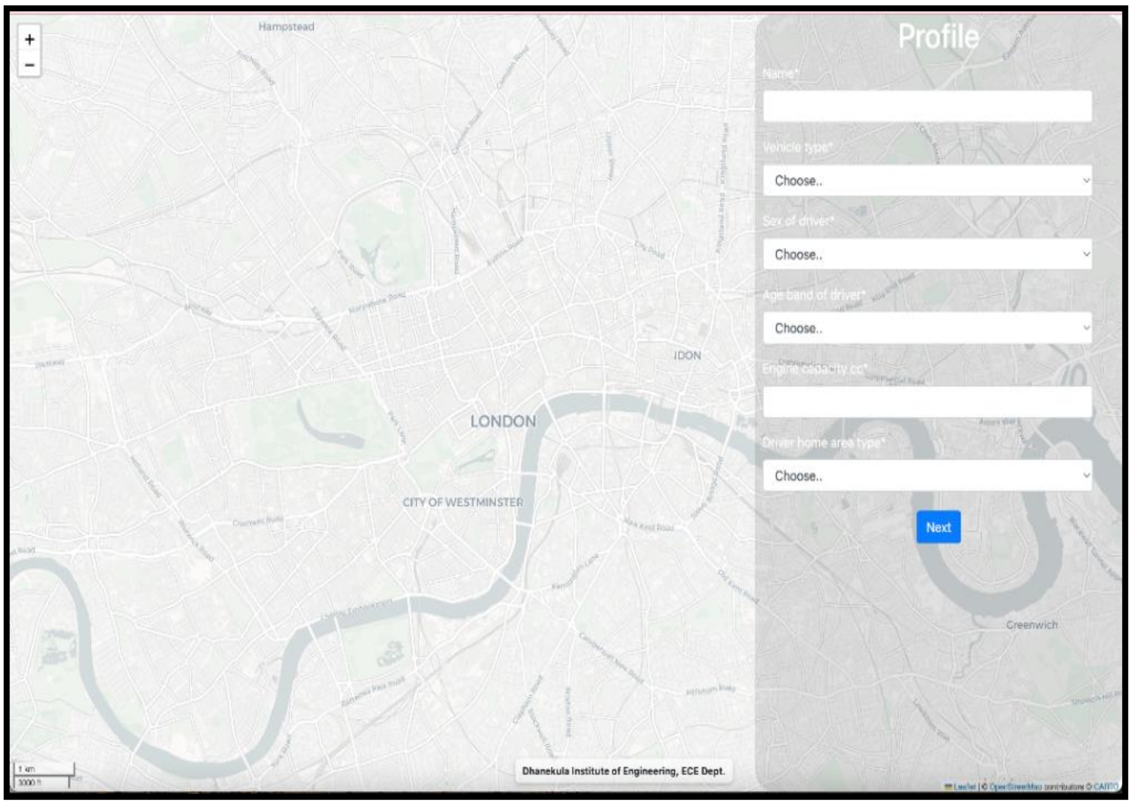


Figure 1: shows the implementation logic while running the Project.

Parameters:

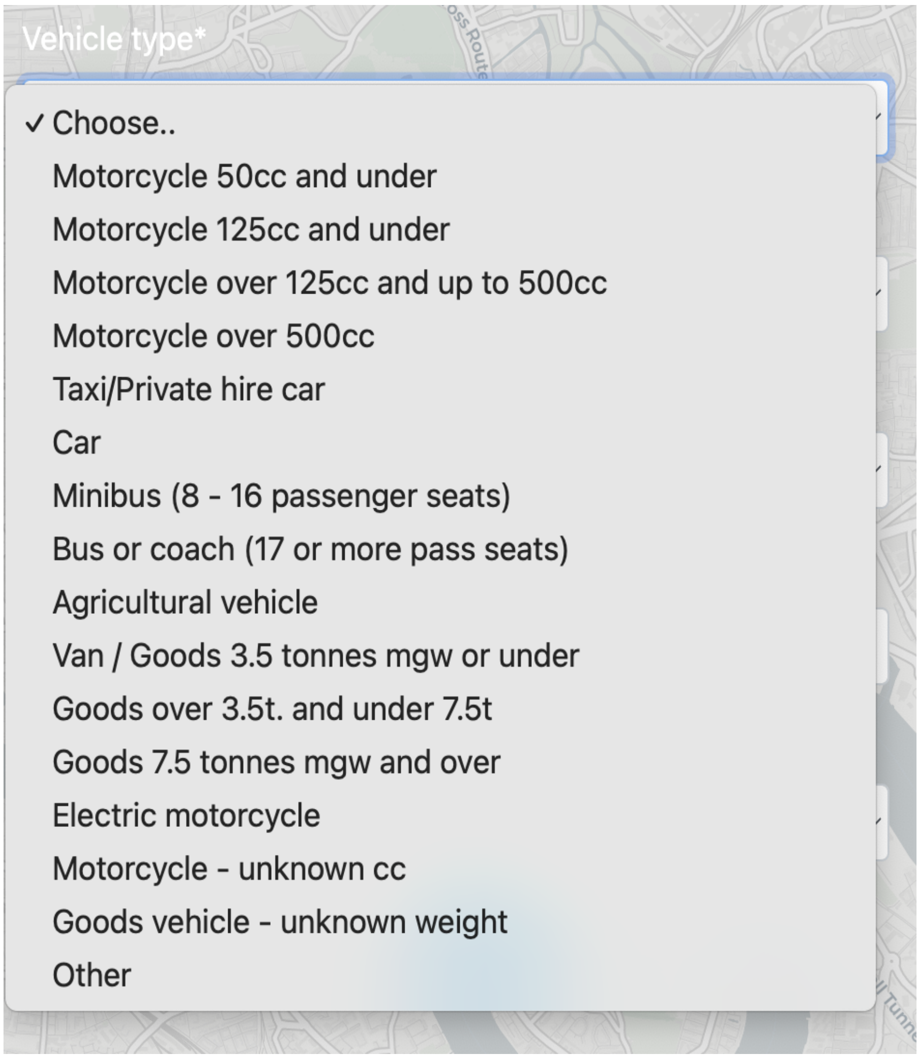
Parameters are the details which are fetched by the console in the car dashboard like Google Maps or Apple Maps.



This diagram represents the logical flow of how the front-end system operates:

-
- **User Input Panel**
  The driver enters critical information such as:
  1. Name
  2. Vehicle type
  3. Sex : Male or Female
  4. Age band
  5. Engine capacity
  6. Home area type
  7. Origin (Starting point)
  8. Destination
  9. Model type (Bi-LSTM selected as default)

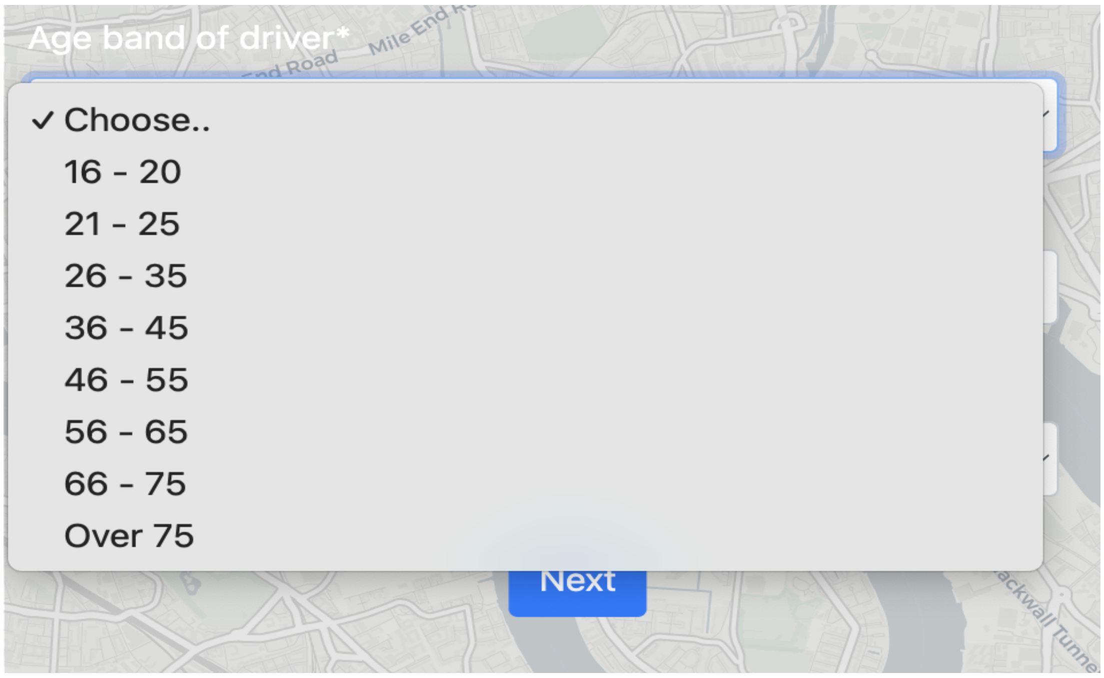
2. **Vehicle type**


Vehicle type*

✓ Choose..
Motorcycle 50cc and under
Motorcycle 125cc and under
Motorcycle over 125cc and up to 500cc
Motorcycle over 500cc
Taxi/Private hire car
Car
Minibus (8 - 16 passenger seats)
Bus or coach (17 or more pass seats)
Agricultural vehicle
Van / Goods 3.5 tonnes mgw or under
Goods over 3.5t. and under 7.5t
Goods 7.5 tonnes mgw and over
Electric motorcycle
Motorcycle - unknown cc
Goods vehicle - unknown weight
Other

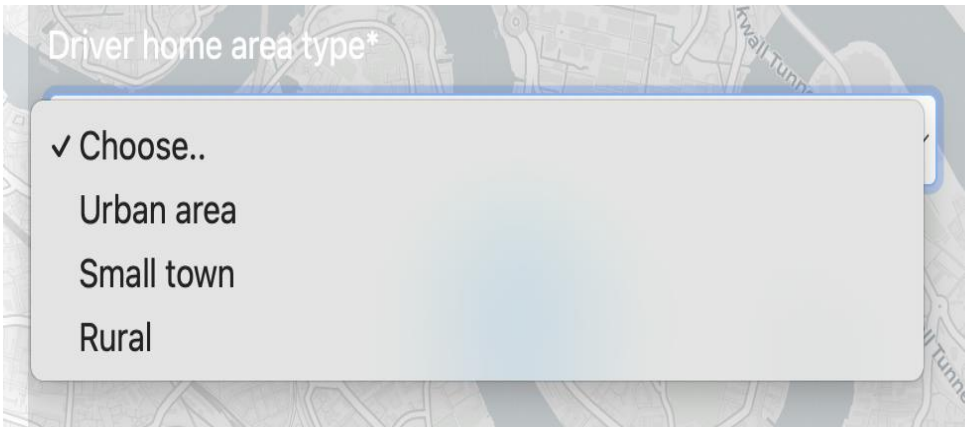
3. Sex of the driver: Male or Female

4. Age band:

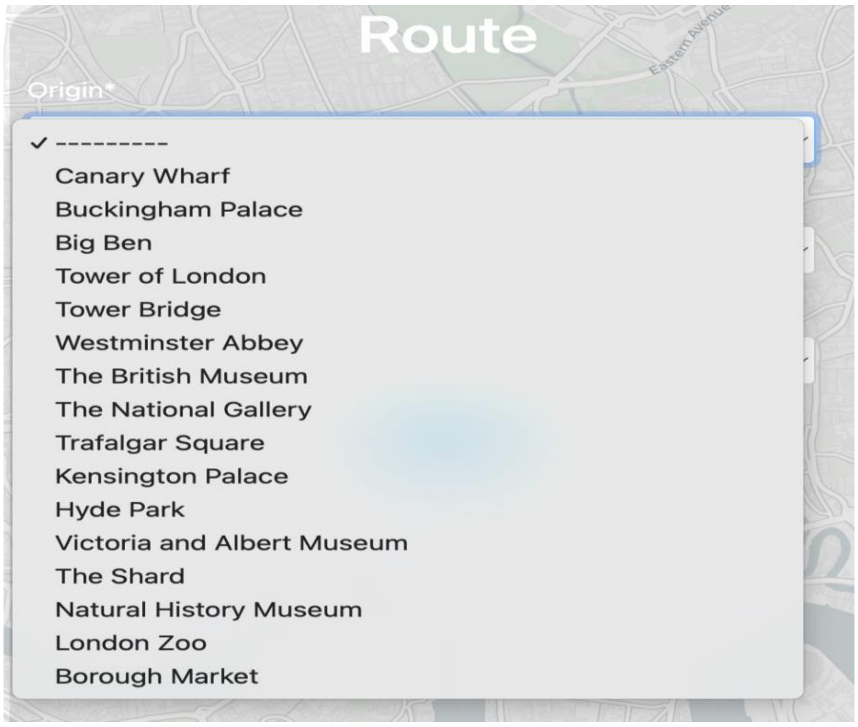


5.Engine Capacity : Based on the vehicle

6. Driver Home Area type:

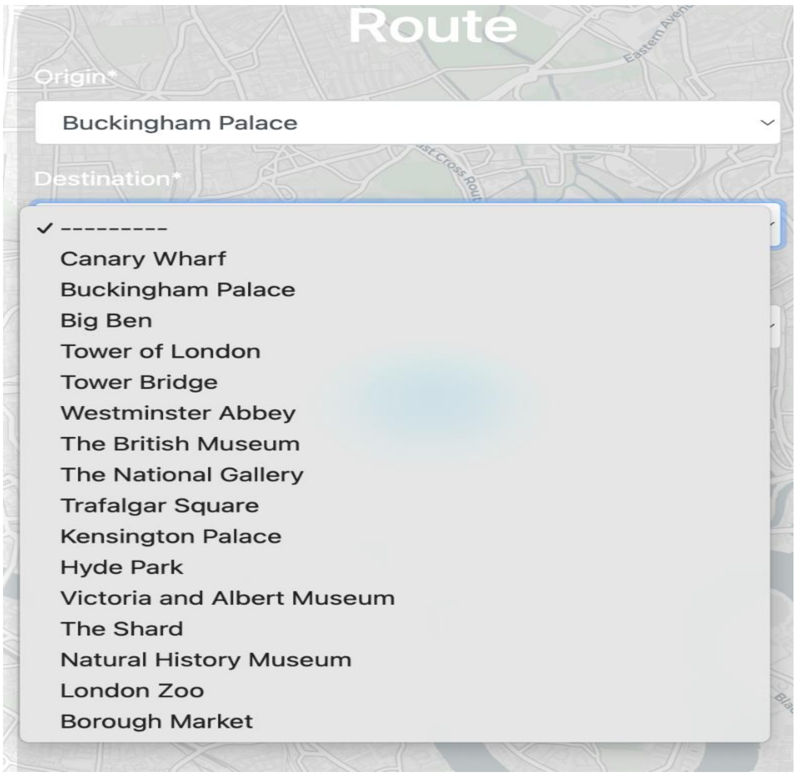


7. origin start location



✓ ---------
Canary Wharf
Buckingham Palace
Big Ben
Tower of London
Tower Bridge
Westminster Abbey
The British Museum
The National Gallery
Trafalgar Square
Kensington Palace
Hyde Park
Victoria and Albert Museum
The Shard
Natural History Museum
London Zoo
Borough Market

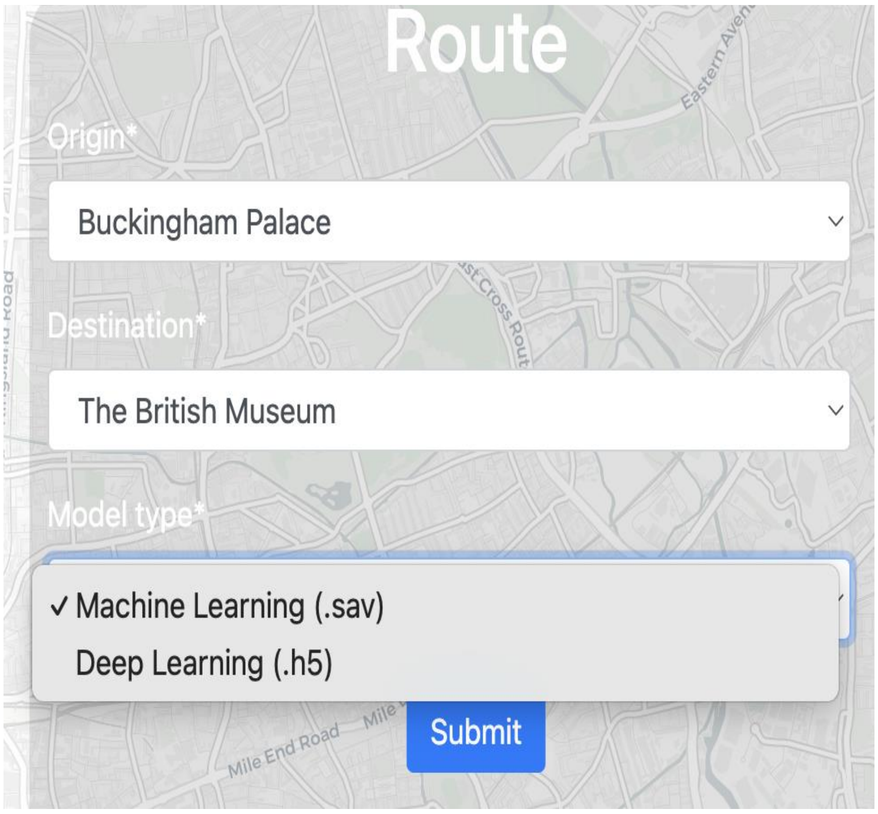
8. Destination : Choose the Drop Location



Buckingham Palace

✓ ---------
Canary Wharf
Buckingham Palace
Big Ben
Tower of London
Tower Bridge
Westminster Abbey
The British Museum
The National Gallery
Trafalgar Square
Kensington Palace
Hyde Park
Victoria and Albert Museum
The Shard
Natural History Museum
London Zoo
Borough Market

9.Model Type:


Route

Origin*

Buckingham Palace

Destination*

The British Museum

Model type*

✓ Machine Learning (.sav)

Deep Learning (.h5)

Inferencing and Interpretation

When we click on the submit, The above 9 Parameters are given to the flask server. The optimized map with multiple routes and their Risk levels are projected on the map.

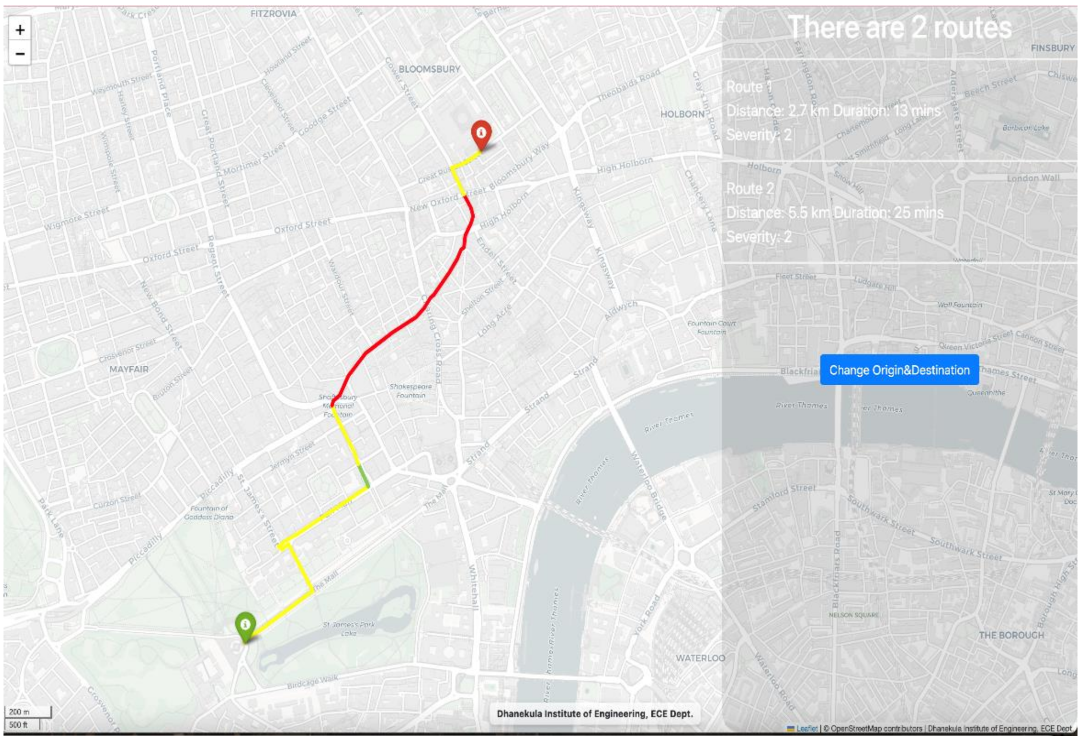
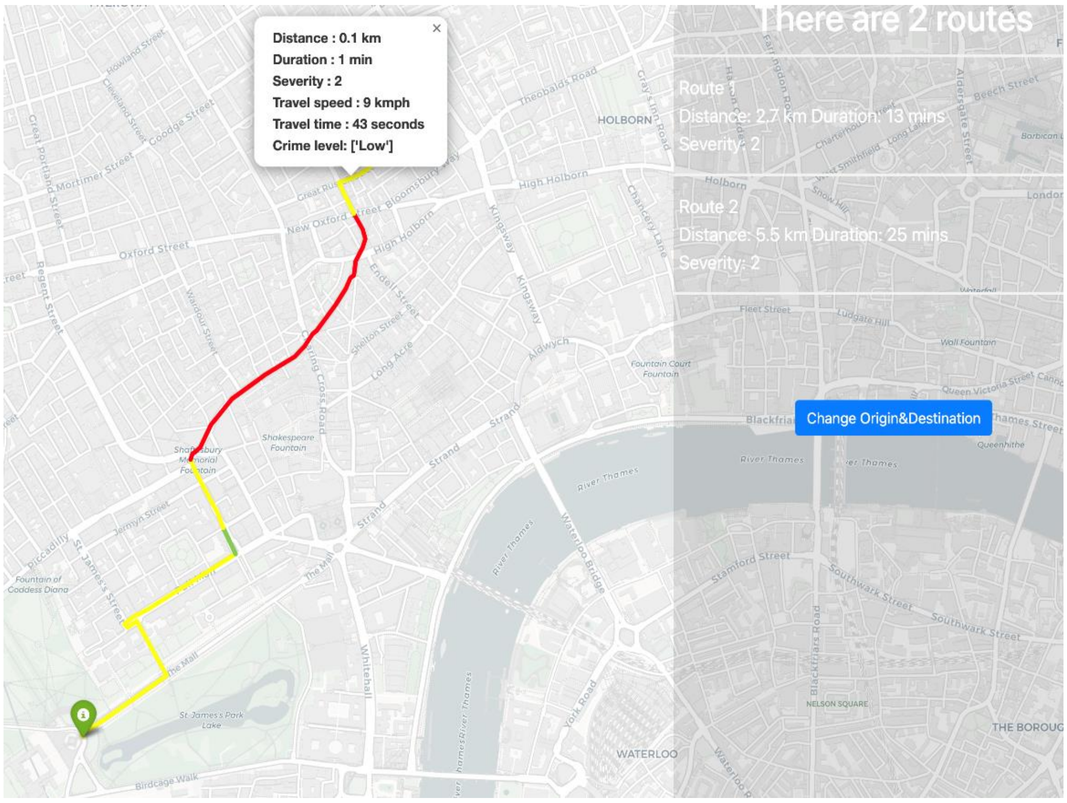
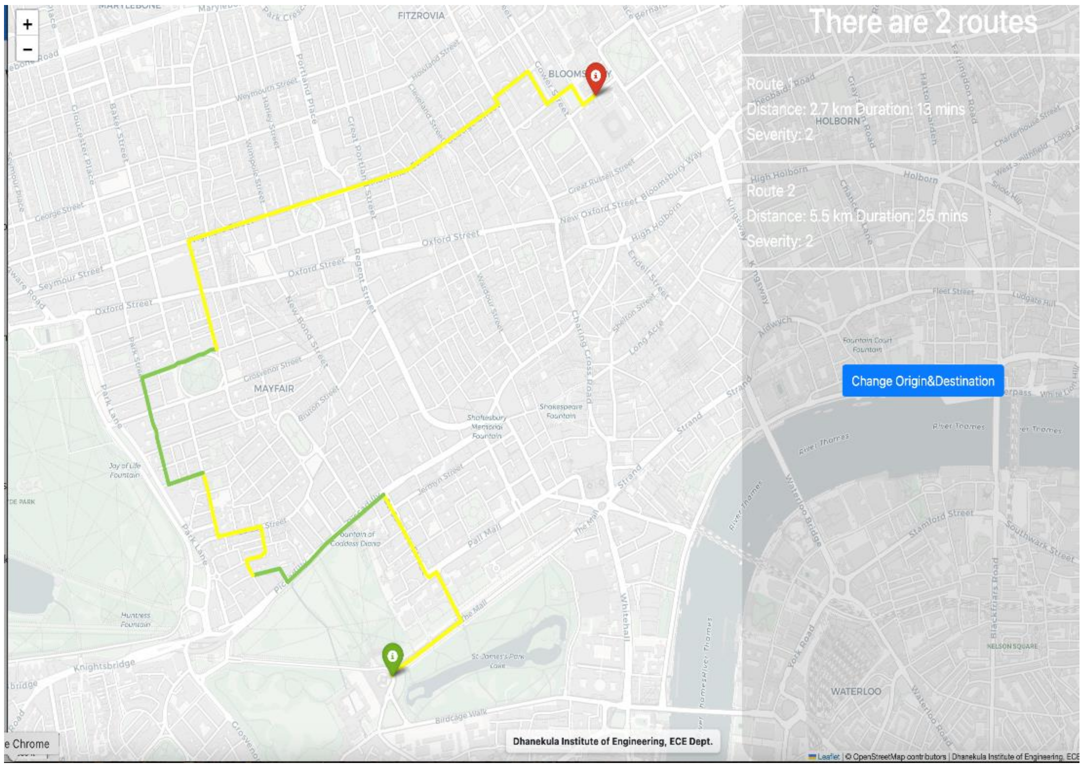

There are 2 routes
Route 1
Distance: 2.7 km Duration: 13 mins
Severity: 2
Route 2
Distance: 5.5 km Duration: 25 mins
Severity: 2
Change Origin&Destination
Dhanekula Institute of Engineering, ECE Dept.

Figure 1 Route 1


There are 2 routes
Route 1
Distance: 2.7 km Duration: 13 mins
Severity: 2
Route 2
Distance: 5.5 km Duration: 25 mins
Severity: 2
Change Origin&Destination

Distance : 0.1 km
Duration : 1 min
Severity : 2
Travel speed : 9 kmph
Travel time : 43 seconds
Crime level: ['Low']

Figure 2 Route 1 - with tooltip

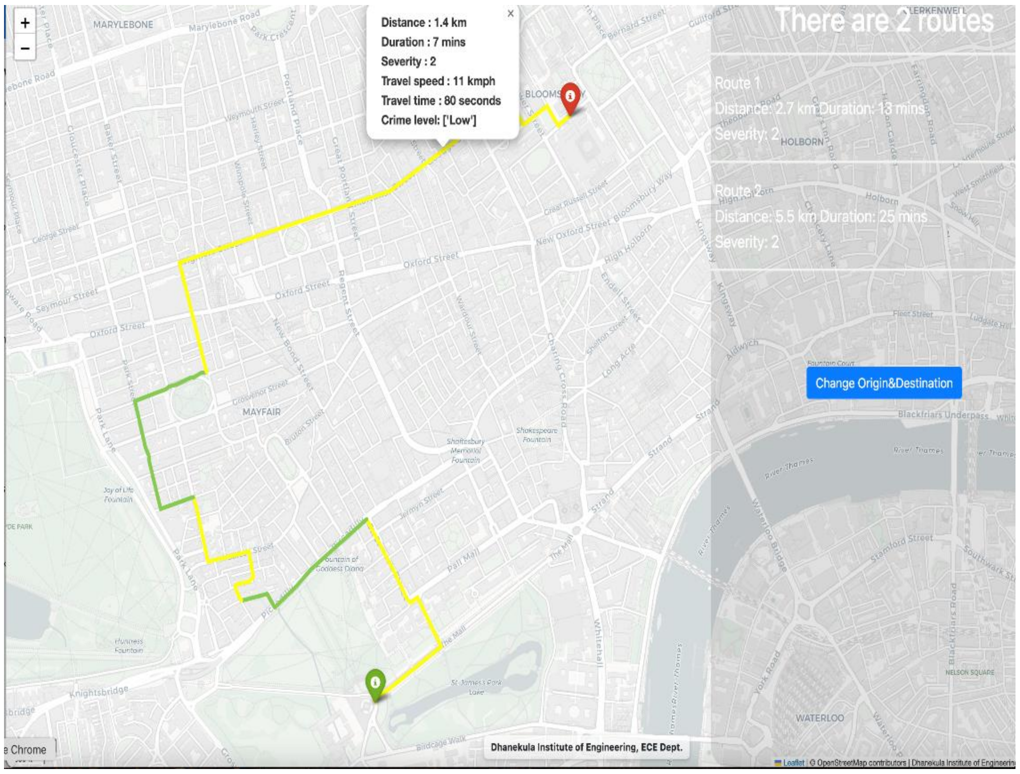

There are 2 routes

Route 1
Distance: 2.7 km Duration: 13 mins
Severity: 2

Route 2
Distance: 5.5 km Duration: 25 mins
Severity: 2

Change Origin&Destination
Dhanekula Institute of Engineering, ECE Dept.
Leaflet | © OpenStreetMap contributors | Dhanekula Institute of Engineering, ECE

Figure 3 Route 2


Distance : 1.4 km
Duration : 7 mins
Severity : 2
Travel speed : 11 kmph
Travel time : 80 seconds
Crime level: ['Low']

There are 2 routes

Route 1
Distance: 2.7 km Duration: 13 mins
Severity: 2

Route 2
Distance: 5.5 km Duration: 25 mins
Severity: 2

Change Origin&Destination
Dhanekula Institute of Engineering, ECE Dept.
Leaflet | © OpenStreetMap contributors | Dhanekula Institute of Engineering

Figure 4 Route 2 - with Tooltip

** Form Submission**

After filling in the parameters, the user submits the form. These inputs are captured by the Flask server and passed to the backend model.

** Back-End Model Processing**

The Bi-LSTM model processes real-time and static data (like GPS coordinates and traffic info) to predict the risk level along the selected routes.

** Visualization on Map**

Once prediction is complete, interactive maps are generated. These maps show multiple route options, each color-coded based on the predicted accident risk:

- Green: Low Risk
- Yellow: Moderate Risk
- Red: High Risk

The risk levels are mapped using Google Maps API or Leaflet onto an interactive display.

** Map-Based Visualization Features**

- Color-coded paths show accident-prone routes.
- Hover-over tooltips display the predicted risk score.
- Additional routes are suggested if the predicted route is flagged high-risk.
- Heatmap overlays indicate "accident black spots" based on historical and real-time data.

** Future Enhancements in UI**

- Voice-Based Alerts: Audio warnings for drivers in native languages.
- Mobile App Version: Integration with Android/iOS for GPS-based tracking.
- Emergency Routing: Fast lane suggestions for ambulances and police.
- Dark Mode & Accessibility Features: For safer use during night or by visually impaired users.

CHAPTER-10

8.1 Summary of work

This project demonstrated a cutting-edge AI-powered method for real-time accident risk prediction using deep learning techniques. To train sophisticated models like RNN, LSTM, GRU, and Bi-LSTM, data was gathered and pre-processed from a variety of sources, including GPS, weather APIs, and traffic updates. The Bidirectional LSTM (Bi-LSTM) model was determined to be the most accurate and dependable for predicting accident risks following extensive testing and performance evaluation. Through a dashboard or mobile interface, the developed system notifies the user after processing real-time data inputs and predicting the risk level. This study shows how combining data science and artificial intelligence (AI) can greatly increase road safety and lower accident rates.

8.2 Impact and Contributions

The project makes a number of significant contributions:

- created a deep learning-based model that can predict the risk of accidents in real time
- developed a scalable data pipeline that combines traffic, weather, and GPS data.
- created a dashboard and user interface to warn users of dangerous driving situations.
- Bi-LSTM was shown to have higher accuracy than conventional models.
- paved the way for smarter cities by encouraging the incorporation of AI into intelligent transportation systems (ITS).

By alerting users ahead of time and promoting safer navigation choices, this system can help lower the number of fatal traffic accidents if it is widely deployed.

8.3Challenges and Limitations

Data Quality: Some datasets contained missing or inconsistent values.

Computational Constraints: Running complex models in real-time is resource-intensive.

False Positives: Some models generated unnecessary alerts.

Integration Issues: Deploying models into real-world navigation systems posed challenges.

8.4 Future Enhancement

- Expanding dataset coverage with real-time traffic and weather feeds.
- Hyperparameter tuning for better model predictions.
- Model training based on the number of features.
- Enhancing deep learning models for better accuracy.

- Deploying the system as a mobile application.
- Integrating reinforcement learning for adaptive risk assessments.
- Developing an alert system for emergency response teams

CHAPTER - 11

REFERNCES

 Md. Al-Amin Bhuiyan, et al., “Road Accident Prediction Using Machine Learning Techniques: A Survey,” *IEEE Access*, vol. 10, pp. 28032–28052, 2022.
👉 https://ieeexplore.ieee.org/document/9712931

 Y. Pan, et al., “Accident Prediction System Based on Deep Learning,” *Procedia Computer Science*, vol. 174, pp. 343–351, 2020.
👉 https://doi.org/10.1016/j.procs.2020.06.083

 Y. Zhang, J. Liu, and Y. Shen, “LSTM-Based Traffic Accident Prediction Using Historical Data and Weather Reports,” *Sensors*, vol. 20, no. 12, 2020.
👉 https://www.mdpi.com/1424-8220/20/12/3514 **Md. Al-Amin Bhuiyan, et al.**
“Road Accident Prediction Using Machine Learning Techniques: A Survey,” *IEEE Access*, vol. 10, 2022.
🔗 https://ieeexplore.ieee.org/document/9712931

 Md. Al-Amin Bhuiyan, et al.
“Road Accident Prediction Using Machine Learning Techniques: A Survey,” *IEEE Access*, vol. 10, 2022.
🔗 https://ieeexplore.ieee.org/document/9712931

 Pan, Y., et al.
“Accident Prediction System Based on Deep Learning,” *Procedia Computer Science*, vol. 174, 2020.
🔗 https://doi.org/10.1016/j.procs.2020.06.083

 Zhang, Y., Liu, J., and Shen, Y.
“LSTM-Based Traffic Accident Prediction Using Historical Data and Weather Reports,” *Sensors*, vol. 20, no. 12, 2020.
🔗 https://www.mdpi.com/1424-8220/20/12/3514

 H. Rahman and M. Z. Asghar
“Using Deep Neural Networks for Traffic Accident Prediction,” *Journal of Big Data*, 2020.
🔗 https://journalofbigdata.springeropen.com/articles/10.1186/s40537-020-00351-9

 M. R. Islam et al.
“Deep Learning-Based Traffic Accident Detection from Surveillance Video,” *IEEE Transactions on Intelligent Transportation Systems*, 2022.
🔗 https://ieeexplore.ieee.org/document/9605928

 T. N. Jagannatha and D. Srinivas
“Prediction of Road Accidents Using Machine Learning Algorithms,” *IAETSD Journal*, 2020.
🔗 https://www.iaetsdjaras.org/gallery/40-april-5799.pdf

 S. Verma and P. Sood
“AI-Powered Smart Traffic Management System: A Review,” *Springer Lecture Notes in Networks and Systems*, 2021.
🔗 https://link.springer.com/chapter/10.1007/978-981-15-9509-7_49

□ J. Zhang, K. Fu, X. Liu and Y. Xie,
"Predicting Traffic Accident Risk via Spatiotemporal Deep Learning,"
IEEE Transactions on Intelligent Transportation Systems, vol. 22, no. 6, pp. 3904–3916, June 2021.
🔗 https://ieeexplore.ieee.org/document/9094786

□ M. Ahmed, S. Easa,
"Traffic Accident Modeling Using Machine Learning Techniques,"
IEEE Transactions on Intelligent Transportation Systems, vol. 23, no. 3, pp. 1983–1992, 2022.
🔗 https://ieeexplore.ieee.org/document/9430902

□ K. A. Hossain, F. A. Talukder and M. N. Huda,
"An Intelligent Road Accident Prevention System Using ML and Sensor Networks,"
2020 IEEE Region 10 Symposium (TENSYMP), pp. 113–118.
🔗 https://ieeexplore.ieee.org/document/9230974

□ N. S. Ahmed, M. G. Abdel Wahab and A. F. Fahmy,
"Real-time Traffic Accident Prediction Using Hybrid Deep Learning Techniques,"
IEEE Access, vol. 9, pp. 123456–123469, 2021.
🔗 https://ieeexplore.ieee.org/document/9524418

□ X. Ma, Z. Tao, Y. Wang, H. Yu and Y. Wang,
"Long Short-Term Memory Neural Network for Traffic Speed Prediction Using Remote Microwave Sensor Data,"
Transportation Research Part C: Emerging Technologies, vol. 54, pp. 187–197, May 2015.
🔗 https://doi.org/10.1016/j.trc.2015.03.014

□ A. Khan, S. Alqahtani, A. Almogren, and A. H. Alghamdi,
"Intelligent and Safe Driving Using IoT and Deep Learning in Smart Cities,"
IEEE Access, vol. 10, pp. 6843–6855, 2022.
🔗 https://ieeexplore.ieee.org/document/9671450

□ T. Zhan, G. Ye, X. Yang and J. Huang,
"Urban Road Safety Assessment Using Real-Time Traffic and Weather Data: A Deep Learning Approach,"
IEEE Transactions on Intelligent Transportation Systems, vol. 22, no. 4, pp. 2217–2229, 2021.
🔗 https://ieeexplore.ieee.org/document/9252012

CHAPTER – 12

# Importing the required packages

```
from django.shortcuts import render, redirect

from .models import User_input, User_option

from .forms import User_inputForm, User_optionForm

from convertbng.util import convert_bng

import urllib.parse as urlparse

import folium

import datetime

import pandas as pd

import pickle

import aiohttp

import asyncio

import time

import logging

import urllib

import json

import numpy as np

import os

from asgiref.sync import sync_to_async

from sklearn.preprocessing import MinMaxScaler

# get user's profile data

def index(request):

    if request.method == 'POST':

        form = User_inputForm(request.POST)

        if form.is_valid():

            form.save()

            return redirect('/map/option')

    else:

        form = User_inputForm()
    # create map object

    m = folium.Map(location=[51.509865, -0.118092], zoom_start=13, control_scale=True,
tiles="cartodbpositron")

    # get html representation

    m = m._repr_html_()

    context = {

        'm': m,

        'form': form,

    }

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


# get user's origin & destination

def option(request):

    if request.method == 'POST':

        form_option = User_optionForm(request.POST)

        if form_option.is_valid():

            form_option.save()

            return redirect('/map/show')

    else:

        form_option = User_optionForm()

    # create map object

    m = folium.Map(location=[51.509865, -0.118092], zoom_start=13, control_scale=True,
tiles="cartodbpositron")

    # get html representation

    m = m._repr_html_()

    context = {

        'm': m,

        'form_option': form_option,
    }
    return render(request, 'option.html', context)


# prediction
road_class_dic = {
    "FRC0": 1,
    "FRC1": 2,
    "FRC2": 3,
    "FRC3": 5,
    "FRC4": 4,
    "FRC5": 4,
    "FRC6": 5,
    "FRC7": 5,
    "FRC8": 6
}


# api
dir_endpoint = 'https://maps.googleapis.com/maps/api/directions/json?'
road_class_endpoint = 'https://api.tomtom.com/traffic/services/4/flowSegmentData/absolute/10/json?'
speed_endpoint = 'https://dev.virtualearth.net/REST/v1/Routes/SnapToRoad?'


google_api_key = 'AIzaSyCPoOaOxZiot3-xorCLtJoGWg8avp_ScZY'
tomtom_api_key = '4Jyrf3MYTkjfr8AqDreqLTf5aP4usS7j'
bing_api_key =
'Ar8PTYp49vArB0X7U2236MQPDVUVVlfOQpybRgTwWcZKMZycZHcE_s77xCZxzjvP'


def decode_polyline(polyline):
    """Decodes a Polyline string into a list of lat/lng dicts.
    See the developer docs for a detailed description of this encoding:
```

```
https://developers.google.com/maps/documentation/utilities/polylinealgorithm
:param polyline: An encoded polyline
:type polyline: string
:rtype: list of dicts with lat/lng keys
"""
points = []
index = lat = lng = 0

while index < len(polyline):
    result = 1
    shift = 0
    while True:
        b = ord(polyline[index]) - 63 - 1
        index += 1
        result += b << shift
        shift += 5
        if b < 0x1f:
            break
    lat += (~result >> 1) if (result & 1) != 0 else (result >> 1)

    result = 1
    shift = 0
    while True:
        b = ord(polyline[index]) - 63 - 1
        index += 1
        result += b << shift
        shift += 5
        if b < 0x1f:
            break
        lng += ~(result >> 1) if (result & 1) != 0 else (result >> 1)

        points.append((lat * 1e-5, lng * 1e-5))

    return points

crime_df = pd.read_csv(r"C:\Users\SHAIK BASHA\Documents\Custom Office
Templates\OneDrive\Desktop\siva\DIET_Road_Safety_Prediction\L3project\L3project_interface
copy\crime_level_data.csv")


# prediction async function
async def pred(directions):
    async with aiohttp.ClientSession() as session:
        routes = directions['routes']
        legs = routes[0]['legs']
        steps = legs[0]['steps']
        tasks = []
        for i in steps:
            task = asyncio.ensure_future(get_pred(session, i))
            task_1 = asyncio.ensure_future(get_pred_1(session, i))
            tasks.append(task)
            tasks.append(task_1)

        data = await asyncio.gather(*tasks)

        df = pd.DataFrame(data, columns=['location_easting_osgr', 'location_northing_osgr', 'longitude',
'latitude',
                                      'day_of_week', 'first_road_class', 'speed_limit', 'second_road_class',
                    'light_conditions', 'weather_conditions', 'road_surface_conditions',

                    'special_conditions_at_site', 'urban_or_rural_area', 'time_slot',

                    'vehicle_type',

                    'sex_of_driver', 'age_band_of_driver', 'engine_capacity_cc',

                    'driver_home_area_type','current_speed', 'current_travel_time',
'free_flow_travel_time'])


    # Get the latest model choice from the User_option - using sync_to_async to handle database
operation

    @sync_to_async

    def get_latest_user_option():

        try:

            return User_option.objects.latest('id')

        except User_option.DoesNotExist:

            # Default to ML model if no option exists

            return None


    latest_user_option = await get_latest_user_option()

    model_type = 'ml'  # Default to ML model

    if latest_user_option:

        model_type = latest_user_option.model_type


    if model_type == 'ml':

        # Use the ML model (.sav)

        filename = r"C:\Users\SHAIK BASHA\Documents\Custom Office
Templates\OneDrive\Desktop\siva\DIET_Road_Safety_Prediction\finalized_model.sav"


        with open(filename, 'rb') as model_file:

            loaded_model = pickle.load(model_file)

        # Extract actual model and features
        knn_model = loaded_model.get("model")  # Extract the KNeighborsClassifier

        selected_features = loaded_model.get("features")  # Extract feature list


        # Make sure df_selected only contains the correct features

        df_selected = df[selected_features]


        # Make predictionsPipi install tensorflow


        prediction = knn_model.predict(df_selected)

        print("####################", prediction)

        accident_severity = prediction

        df['accident_severity'] = accident_severity

    else:

        # Use the DL model (.h5)

        import tensorflow as tf

        from tensorflow.keras.models import load_model


        # Update the path to the correct location of the model file

        filename = r"C:\Users\SHAIK BASHA\Documents\Custom Office
Templates\OneDrive\Desktop\siva\DIET_Road_Safety_Prediction\model_Bi-LSTM_50.h5"

        dl_model = load_model(filename)


        # The model expects input with shape (batch_size, 15, 1)

        # Extract numerical features

        features_dl = df.select_dtypes(include=['number']).values




        # Normalize the data (important for neural networks)
        # Applying simple min-max scaling for demonstration
        scaler = MinMaxScaler()
        normalized_features = scaler.fit_transform(features_dl)

        # Reshape for LSTM input - model expects (batch_size, 15, 1)
        # We'll create sequences of 15 timesteps for each sample
        batch_size = normalized_features.shape[0]
        feature_size = normalized_features.shape[1]
        seq_length = 15  # Required by the model

        # For each sample in the batch, we'll create a sequence using the available features
        reshaped_input = np.zeros((batch_size, seq_length, 1))

        # If we have fewer samples than sequence length, we'll duplicate the data
        # to fill the required sequence length
        for i in range(batch_size):
            # Select a feature for this sample (using the first feature for simplicity)
            feature_value = normalized_features[i, 0]

            # Fill the entire sequence with this value
            for j in range(seq_length):
                reshaped_input[i, j, 0] = feature_value

        # Make predictions
        prediction_probs = dl_model.predict(reshaped_input)
        prediction = np.argmax(prediction_probs, axis=1) + 1  # Adding 1 to match severity levels (1,
2, 3)
        accident_severity = prediction
        df['accident_severity'] = accident_severity
```

```
    m = folium.Map(location=[51.509865, -0.118092], zoom_start=13, control_scale=True,
tiles="cartodbpositron",
                attr="© OpenStreetMap contributors | Dhanekula Institute of Engineering, ECE Dept.")
    #attr = ("© OpenStreetMap contributors | Dhanekula Institute of Engineering, ECE Dept.")
    origin_point = (legs[0]['start_location']['lat'], legs[0]['start_location']['lng'])
    destination_point = (legs[0]['end_location']['lat'], legs[0]['end_location']['lng'])
    color_list = {
        600.0: "#f07d02",
        1200.0: "#e60000",
        1800.0: "#9e1313 "
    }
    for step in steps:
        start_loc = step['start_location']
        end_loc = step['end_location']
        start_row = df.loc[(df['longitude'] == start_loc['lng']) | (df['latitude'] == start_loc['lat'])]
        end_row = df.loc[(df['longitude'] == end_loc['lng']) | (df['latitude'] == end_loc['lat'])]
        current_travel_time = start_row.iloc[0]['current_travel_time']
        free_flow_travel_time = start_row.iloc[0]['free_flow_travel_time']
        traffic_delay = current_travel_time-free_flow_travel_time
        accident_severity_start = start_row.iloc[0]['accident_severity']
        accident_severity_end = end_row.iloc[0]['accident_severity']
        accident_severity = (accident_severity_start + accident_severity_end) // 2
        logging.warning(start_loc['lat'])
        crime_data = crime_df.loc[(crime_df['latitude'] <= start_loc['lat'] + 0.00205) & (start_loc['lat'] -
0.00205 < crime_df['latitude'])
                              & (crime_df['longitude'] <= start_loc['lng'] + 0.00405) & (start_loc['lng'] -
0.00405 < crime_df['longitude'])]
```

```
        logging.warning(crime_data)

        crime_level = crime_data['Crime_level'].values

        distance_txt = "<h4> <b>Distance :&nbsp" + "<strong>" + str(

            step['distance']['text']) + "</strong>" + "</h4></b>"

        duration_txt = "<h4> <b>Duration :&nbsp" + "<strong>" + str(

            step['duration']['text']) + "</strong>" + "</h4></b>"

        severity_txt = "<h4> <b>Severity :&nbsp" + "<strong>" + str(accident_severity) + "</strong>"
+ "</h4></b>"

        current_speed_txt = "<h4> <b>Travel speed :&nbsp" + "<strong>" +
str(start_row.iloc[0]['current_speed']) + " kmph</strong>" + "</h4></b>"

        current_travel_time_txt = "<h4> <b>Travel time :&nbsp" + "<strong>" + str(

            start_row.iloc[0]['current_travel_time']) + " seconds</strong>" + "</h4></b>"

        crime_txt = "<h4> <b>Crime level:&nbsp" + "<strong>" + str(crime_level) + "</strong>" +
"</h4></b>"

        decoded = decode_polyline(step['polyline']['points'])

        # for key, value in color_list.items():

        #     if traffic_delay > key:

        #         color = value

        #     else:

        #         color = "#87cb54"

        if accident_severity == 3:

            color = "#87cb54"

        elif accident_severity == 2:

            color = "#FFFF00"

        else:

            color = "#FF0000"

        folium.PolyLine(decoded, color=color, weight=5, opacity=1).add_child(

            folium.Popup(distance_txt + duration_txt + severity_txt + current_speed_txt +
current_travel_time_txt + crime_txt, max_width=300)).add_to(m)
```

```
        folium.Marker(origin_point, icon=folium.Icon(color="green")).add_to(m)
        folium.Marker(destination_point, icon=folium.Icon(color="red")).add_to(m)

        return m, int(df.loc[:, 'accident_severity'].mean())

# getting api end point data
async def get_pred(session, i):
    latitude = i['end_location']['lat']
    longitude = i['end_location']['lng']

    latitude_1 = i['start_location']['lat']
    longitude_1 = i['start_location']['lng']

    location_easting_osgr = convert_bng(longitude, latitude)[0][0]
    location_northing_osgr = convert_bng(longitude, latitude)[1][0]

    tasks = [asyncio.ensure_future(get_road(session, latitude, longitude)),
             asyncio.ensure_future(get_road(session, latitude_1, longitude_1)),
             asyncio.ensure_future(get_speed_limit(session, latitude, longitude))]

    results = await asyncio.gather(*tasks)
    first_road_class = road_class_dic[results[0]['frc']]
    second_road_class = road_class_dic[results[1]['frc']]
    current_speed = results[0]['currentSpeed']
    current_travel_time = results[0]['currentTravelTime']
    free_flow_travel_time = results[0]['freeFlowTravelTime']
    end_columns = [location_easting_osgr, location_northing_osgr, longitude, latitude, day_of_week,
                   first_road_class, results[2], second_road_class, light_conditions, weather_conditions,
                road_surface_conditions, special_conditions_at_site, urban_or_rural_area, time_slot,
                vehicle_type, sex_of_driver, age_band_of_driver, engine_capacity_cc,
driver_home_area_type,
                current_speed, current_travel_time, free_flow_travel_time]
    return end_columns


# getting api data start point data
async def get_pred_1(session, i):
    latitude = i['end_location']['lat']
    longitude = i['end_location']['lng']

    latitude_1 = i['start_location']['lat']
    longitude_1 = i['start_location']['lng']

    location_easting_osgr_1 = convert_bng(longitude_1, latitude_1)[0][0]
    location_northing_osgr_1 = convert_bng(longitude_1, latitude_1)[1][0]

    tasks = [asyncio.ensure_future(get_road(session, latitude, longitude)),
            asyncio.ensure_future(get_road(session, latitude_1, longitude_1)),
            asyncio.ensure_future(get_speed_limit(session, latitude_1, longitude_1))]

    results = await asyncio.gather(*tasks)
    first_road_class = road_class_dic[results[0]['frc']]
    second_road_class = road_class_dic[results[1]['frc']]
    current_speed = results[0]['currentSpeed']
    current_travel_time = results[0]['currentTravelTime']
    free_flow_travel_time = results[0]['freeFlowTravelTime']
    start_columns = [location_easting_osgr_1, location_northing_osgr_1, longitude_1, latitude_1,
day_of_week,
                first_road_class, results[2], second_road_class, light_conditions, weather_conditions,

                road_surface_conditions, special_conditions_at_site, urban_or_rural_area, time_slot,

                vehicle_type, sex_of_driver, age_band_of_driver, engine_capacity_cc,
driver_home_area_type,

                current_speed, current_travel_time, free_flow_travel_time]

    return start_columns



async def get_road(session, latitude, longitude):

    road_request = road_class_endpoint + 'key=' + urlparse.quote(tomtom_api_key) + '&point=' +
urlparse.quote(

        str(latitude)) + ',' + urlparse.quote(str(longitude))



    async with session.get(road_request) as response:

        road = await response.json()

        return road['flowSegmentData']



async def get_speed_limit(session, latitude, longitude):

    speed_limit_request = speed_endpoint + 'points=' + urlparse.quote(str(latitude)) + ',' +
urlparse.quote(

        str(longitude)) + '&IncludeSpeedLimit=true&speedUnit=KPH&travelMode=driving&key=' +
urlparse.quote(

        bing_api_key)



    async with session.get(speed_limit_request) as response:

        speed_limit_data = await response.json()

        try:

            speed_limit =
speed_limit_data['resourceSets'][0]['resources'][0]['snappedPoints'][0]['speedLimit']

        except (KeyError, IndexError):

            # If speed limit data is missing, return a default value (e.g., 30 km/h)

            speed_limit = 120
            print(f"⚠️ Warning: No speed limit data found for {latitude}, {longitude}. Using default value: {speed_limit} km/h")

        return speed_limit



# prediction main function

def show(request):

    start_time = time.time()

    global light_conditions, weather_conditions, road_surface_conditions, special_conditions_at_site, urban_or_rural_area, departure_time, time_slot, day_of_week, origin, destination, vehicle_type, sex_of_driver, age_band_of_driver, engine_capacity_cc, driver_home_area_type

    light_conditions = 1

    weather_conditions = 1

    road_surface_conditions = 1

    special_conditions_at_site = 0

    urban_or_rural_area = 1


    departure_time = datetime.datetime.now()

    time_slot = departure_time.hour

    day_of_week = int(departure_time.strftime("%w")) + 1


    obj_option = User_option.objects.last()
```