Cairo University

Cairo University
Faculty of Graduate Studies for Statistical
Research Department of Software Engineering

Study of the effect of alcoholic beverages on the driving patterns of adult drivers

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master in Software Engineering

By
Mohamed Salah Ibrahim
Supervised by

Prof. Mervat Gheith

Dr.Tarek Aly

Abstract

The Influence of Alcohol on Driver Patterns: Early Detection Using Machine Learning Driving under the influence of alcohol remains a critical factor contributing to road accidents and jeopardizing road safety globally. This study delves into the impact of alcohol on driver behavior patterns and introduces an innovative method to preemptively identify these patterns using a machine learning model. The primary goal is to establish an early warning system capable of alerting drivers about their compromised driving abilities due to alcohol consumption, thus mitigating potential accidents.

To facilitate this research, an extensive dataset was curated from diverse sources, including driving simulators and real-world driving scenarios. This dataset encompassed a wide spectrum of driving behaviors under varying levels of alcohol influence. Feature extraction involved parameters such as vehicle speed, lane deviation, and reaction times.

Through rigorous experimentation, a machine learning framework was devised to analyze the dataset and recognize distinctive patterns that signify alcohol-induced impairment. A comprehensive array of algorithms, including decision trees, clusters, and neural networks, underwent evaluation to determine the optimal model for detection. The chosen model was fine-tuned and validated using cross-validation techniques.

The resulting system adeptly detects patterns that indicate alcohol-related impairment in real-time. By scrutinizing driver behavior and their interactions with the vehicle, the

model accurately predicts instances of alcohol influence. Upon identifying a heightened likelihood of impairment, the system issues a prompt warning to the driver, highlighting their compromised state and recommending appropriate actions such as refraining from driving or seeking alternative transportation.

This study substantiates that the machine learning model significantly contributes to curbing alcohol-related accidents by providing preemptive alerts to impaired drivers. The efficacy of the model hinges on the quality and diversity of the training dataset, along with the accuracy of input data. Future research endeavors could concentrate on expanding the dataset to encompass a broader array of scenarios and validating the model extensively in real-world settings.

In conclusion, this study introduces a promising approach to tackling the perils associated with alcohol-impaired driving. By harnessing machine learning techniques for real-time assessment, the proposed system presents a proactive strategy to enhance road safety and avert potential accidents triggered by alcohol influence.

Keywords: alcohol-impaired driving, driver behavior patterns, real-time detection, machine learning model, road safety, early warning system, vehicle speed, lane deviation, reaction times, decision trees, random forests, support vector machines, neural networks, training dataset, real-world scenarios, road accidents, proactive solution.

Table of Contents

List of Figures

List of Tables

Acknowledgement

First of all, ultimate thanks are due to Allah, who without he is, and this work could not be done.

I am making this thesis not only for degree but to also increase my knowledge.

I would like to express my special thanks of gratitude to my supervisors Dr/ Mervat Gheith and Dr/ Tarek Aly who gave me the golden opportunity to do this wonderful thesis, which also helped me in doing a lot of Research and I came to know about so many new things I am thankful to them.

I would also like to thank my family and friends who helped me a lot in finalizing this thesis within the limited time frame.

I would also like to thank the whole development team members whom I worked with during my research activities.

Finally, I apologize to all other unnamed who helped me in various ways to have a good thesis.

Chapter 1. Introduction

Driving under the influence of alcohol remains a significant public health concern that poses a danger to both the driver and other road users. In Asia alone, 400,000 people are killed on the roads annually and more than four million injured (Transport and Research Laboratory, UK, 1998) in the US, around 30 people die each day in traffic crashes in which one of the parties is under the influence of alcohol, and, together, alcohol-related crashes amount to **30%** of all traffic fatalities (2020 National Highway Traffic Safety) Previous studies have shown that even low blood alcohol concentrations (BACs) can impair critical driving skills such as According to the latest available data, Egypt has experienced a substantial number of road Recent statistics released by the Central Agency for Public Mobilization and Statistics (CAPMAS) highlight the severity of this issue, with a total of **5,686** road accidents occurring across Egypt in 2022 alone. This alarming figure accentuates the critical need for evidence-based interventions to enhance road safety and reduce the burden of traffic-related traumas (CAPMAS, 2022). re underscores the urgent need for comprehensive strategies aimed at improving road safety and reducing the incidence of traffic-related fatalities and injuries, the security of connected cars will become more important as more cars are connected to the Internet. Gartner reports that there will be a quarter of a billion connected vehicles by 2020 (Gartner reports, 2015) drivers with a BAC of 0.08 g/dL or higher are at a significantly higher risk of being involved in a fatal crash than sober adult drivers (Waller et al, 2019). The physiological effects of alcohol on the human body can impair driving performance by affecting crucial skills such as coordination, reaction time, and decision-making, particularly among adult drivers (Mandel et al., 2018). Despite numerous campaigns and legislation aimed at reducing the incidence of drunk driving, it remains a leading cause of road accidents and fatalities among adult drivers worldwide.

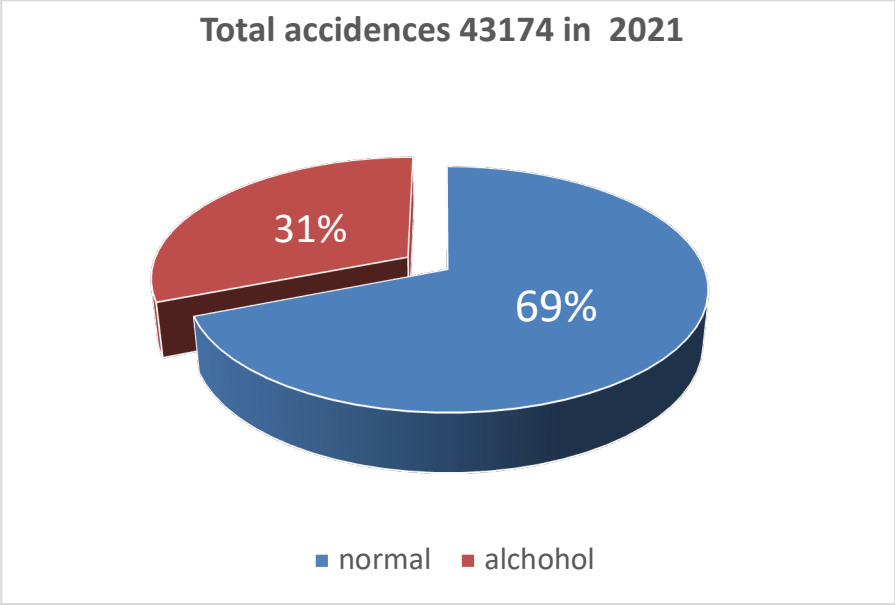


Figure 1: Percentage of accidences under alcohol effect compare than total accidences (National Highway Traffic Safety).

In recent years, advances in machine learning algorithms have made it possible to detect patterns in adult driver behavior that could indicate impaired driving and alert drivers to prevent accidents before they happen. This research aims to explore the use of machine learning algorithms to detect patterns in adult driver behavior that could indicate impaired driving and alert adult drivers in real-time. By leveraging data from sensors in the car, such as accelerometers and gyroscopes, as well as data from the adult driver's smartphone and other wearable devices, we can develop models that detect changes in driving behavior that may indicate impairment among adult drivers. The goal of this research is to develop a system that can alert adult drivers in real-time to prevent accidents before they happen and promote safer driving practices among adult drivers.

1.1 Problem Statement

Driving under the influence of alcohol remains a significant contributor to road accidents and poses a severe threat to road safety worldwide. Despite numerous awareness campaigns and legal measures, alcohol-impaired driving continues to endanger lives. The lack of effective preemptive systems to identify and alert drivers about their compromised state due to alcohol consumption perpetuates this problem. Existing methods often rely on post-incident investigations or self-assessment, leading to delayed responses and potential accidents. This study aims to address this issue by investigating the influence of alcohol on driver behavior patterns and developing a machine learning-based early detection system. analyzing driver's running pattern is a good way to authenticate the driver (Enev et al, 2016). Previous works collected driving data from cars and used it in driver profiling,The goal is to proactively identify and warn drivers about their impaired abilities before accidents occur, ultimately contributing to a safer road environment and reducing the impact of alcohol-related accidents.

1.2 Thesis objectives

The primary objective of this study is to design and implement a Machine Learning Model able to learn different patterns and early detect abnormal driving patterns through the software "DPObserver" that can detect and alert the driver and police. Specifically, the study aims to:

- Investigate Alcohol's Impact: Analyze the influence of alcohol on driver behavior patterns by studying simulation driving scenarios under varying levels of alcohol consumption.

- Feature Extraction: Extract relevant features from the dataset, including vehicle speed, lane deviation, and reaction times, to quantify and characterize driver behavior patterns.
- Model Selection and Development: Evaluate various machine learning algorithms, such as decision trees, Cluster, and neural networks, to identify the most effective model for pattern detection.
- Model Training and Validation: Train the selected machine learning model on the curated dataset and fine-tune its parameters. Validate the model's performance using cross-validation techniques to ensure its reliability and generalization.
- Real-time Detection: Implement the trained model in a real-time monitoring system that analyzes incoming driving behavior data and predicts instances of alcohol-impaired patterns.
- Early Warning System: Integrate the detection system with an early warning mechanism that alerts drivers in real time when their behavior indicates alcohol-related impairment.
- Scalability and Robustness: Explore the system's scalability by evaluating its performance on a larger scale and in different driving scenarios. Enhance the system's robustness by considering various real-world conditions.
- Contribution to Road Safety: Evaluate the potential impact of the developed system on reducing alcohol-related accidents and enhancing overall road safety.

By achieving these objectives, this study aims to contribute to the prevention of alcohol-impaired driving incidents and promote safer road environments through proactive detection and timely interventions.

1.3 Research Methodology

This study employs a comprehensive research methodology that encompasses several key stages: data collection, model development, validation, and evaluation. The methodology is meticulously crafted to achieve the objectives of the study, which are investigating the influence of various factors on driver patterns and developing an early detection system for abnormal driving behavior using advanced machine learning techniques. The data collection process comprises the following steps:

Real User Data: Data is collected from real users via sensors embedded in their smartphones. These sensors capture a range of parameters related to the driver's behavior, such as acceleration, speed, and geographical location. This approach enables the gathering of rich, real-world driving pattern data under varied conditions and scenarios.

Simulation Data: In addition to real-world data, simulated driving data is also employed. This data is generated through computer simulations that model vehicle movement on pre-defined paths. The simulation allows for the control of environmental variables and the replication of specific driving scenarios that may not be easily captured through real-world data collection.

Extract relevant features such as vehicle speed, lane deviation, and reaction times from the collected data.

Exploratory Data Analysis:

Conduct exploratory data analysis to understand the characteristics of the collected dataset.

Identify patterns and correlations between driving behaviors and alcohol influence levels by my AI model.

Model Selection and Development:

Evaluate multiple machines learning algorithms, including decision trees, random forests, support vector machines, and neural networks, for pattern detection.

Select the most suitable algorithm based on performance metrics and model complexity.

Model Training and Validation:

Split the dataset into training and validation sets for model training and evaluation.

Real-time Implementation:

Develop a real-time monitoring system that continuously analyzes incoming driving behavior data.

Integrate the trained machine learning model into the system for detecting alcohol-impaired patterns.

Early Warning System:

Design an early warning mechanism that triggers alerts when the model identifies alcohol-impaired driving patterns.

Determine appropriate thresholds for issuing warnings to drivers.

Ethical Considerations:

Address ethical considerations related to privacy and data security in collecting and using driving behavior data Ensure transparency in explaining the system's functioning to users.

Contribution to Road Safety:

Analyze the potential impact of the developed system on reducing alcohol-related accidents and enhancing road safety.

Through this research methodology, the study aims to advance our understanding of alcohol's influence on driver behavior patterns and provide a practical solution to mitigate the dangers of alcohol-impaired driving through real-time pattern detection and early warnings.

Chapter 2. Related Works

2.1 Introduction

The field of transportation and vehicle behavior analysis, coupled with studies on the impact of alcohol on driver behavior, has witnessed significant advancements in recent years. Researchers have explored various facets of these domains, ranging from studies on driver safety, driving pattern analysis, and eco-driving, to investigations into the effects of alcohol on cognitive and motor skills in the context of driving. In order to gain a deep understanding of the evolution and current state of research in these interrelated fields, we have undertaken a thorough review of the literature this section offers two main studies that are directly relevant to our study's dual focus on driving patterns and their association with influence alcohol.

Let’s demonstrate some of related work and the first one from Xiaodi Huanga, Po Yuna, Shuhui Wub, and Zhongfeng Hua with name “Abnormal driving behavior detection based on an improved ant colony algorithm, (2023) say In the quest to enhance road safety, Huang et al. (2023) explore the critical issue of detecting abnormal driving behaviors which contribute to a significant number of traffic accidents. Their research acknowledges the complex nature of driving behavior, influenced by both objective regulations and subjective biological factors, noting that the same actions can be interpreted differently based on the driver's category. To tackle this challenge, the authors present an innovative detection method that utilizes pheromone trails—akin to those used by ants in nature to determine drivers' preferences and operation patterns. They develop an enhanced ant colony algorithm, leveraging fixed point simplicial theory, to improve convergence efficiency and minimize false positives.

The proposed method's efficacy lies in its adaptability to various driving patterns and its ability to set dynamic thresholds for abnormal behavior identification, particularly in

cases of incomplete sample labels. Through rigorous experimental testing, the authors demonstrate that their approach not only detects abnormal driving behavior with notable accuracy but also offers potential for reducing traffic incidents caused by driver errors. The study is a testament to the promise of integrating biological concepts with advanced computational algorithms to address pressing issues in public transportation safety. Another paper in use driver profile and apply sequence algorithms with name “Modeling driving styles of online ride-hailing drivers with model identifiability and interpretability“(October 2023) say This paper focuses on the critical role of driving style in online ride-hailing services, emphasizing its impact on passenger safety and comfort. Addressing the gap in research regarding online ride-hailing drivers' driving behaviors, the authors propose an interpretable machine learning method to classify driving styles as aggressive, normal, or cautious. By conducting naturalistic driving tests in Nanjing, China, the study collates a rich dataset including driver, vehicle kinematic, and environmental features. Using the CatBoost algorithm, the research identifies driving styles and employs the SHAP algorithm to interpret the influence of various factors on these styles.

The novelty of the paper lies in its targeted examination of online ride-hailing drivers, considering unique industry-related elements such as driving duration, distance, and tasks, which are often overlooked in other studies. The research stands out for its focus on interpretable machine learning, addressing a common weakness in the field related to the trade-off between model accuracy and interpretability. CatBoost's performance is highlighted, outperforming other algorithms with high precision, recall, and F1-score.

The study contributes significantly to the understanding of driving behaviors in the context of the online ride-hailing industry, offering a methodological approach that can be used to enhance driver monitoring and ultimately improve overall travel safety and efficiency. And this paper explain movement of the traffic and detect diriving patterns from it the paper name “Driver Profile and Driving Pattern Recognition for Road Safety

Assessment: Main Challenges and Future Directions “,(2022) This research thoroughly reviewed artificial intelligence and machine learning Methods used so far in driver profile and driving style Recognition studies for traffic safety analysis purposes. The goal was to identify the best methodology and data collection and suggest future directions for enhancing macroscopic understanding microscopic aspects of driving behavior and thus the road safety.

One of the most important findings of this study is the ambiguity in defining the two scientific fields. It is to discover what the most efficient driving metrics should be used in similar research and repeated data collection it should depend on the level of analysis. Furthermore, she noted the levels of analysis used to define groups of the common behaviors, they can be classified as macroscopic,

Macroscopic (Driving safety, 2019), (Temporal analysis, 2021) and microscopic depending on the level of information they use. Conspicuous absence a methodological framework for identifying macroscopic driver profiles and microscopic driving patterns it is proposed to address them through a methodology that combines macroscopic and microscopic driving metrics, respectively volume 4, 2023 97 TSELENTIS and PAPADIMITRIOU: Driver profile and driving style recognition.

Now we have paper based on vehicle sensors use sequence and not sequency algorithms like neural network (CNN) and recurrent neural network (RNN) with name “A Review for the Driving Behavior Recognition Methods Based on Vehicle Multisensor Information”, (Oct 2022) Since abnormal driving behaviors may lead to immediate accidents, how to improve the identification efficiency and develop a lightweight model that can accurately identify driving behaviors.

This paper reviews and summarizes driving behavior recognition methods based on vehicle sensor information fusion. On-board sensor data contain a wealth of information about driving behavior; Based on the two main factors of driving behavior and vehicle

control, driving behavior information can be divided into driver state information, driver control state information, vehicle control state information, vehicle state information, road environment state information, as well as the corresponding data acquisition system. The characteristics of joint data level, feature level and decision-level information fusion methods are analyzed to guide the selection of appropriate information fusion methods and the basic principles and main characteristics of feature extraction methods. Driving behavior recognition methods are classified into traditional machine learning methods and deep learning methods. Random forest, support vector machine is presented, and applied in fatigue, distraction, following, lane change, and other driving behavior recognition. The application of convolutional neural network (CNN) and recurrent neural network (RNN) in building driving behavior recognition models is analyzed. The characteristics of the four leadership behavior recognition paradigms described in this paper are briefly summarized.

Driving behavior is greatly influenced by the driver's condition, and the driving behavior exhibited by the same driver may vary significantly. At the same time, driving behavior is also susceptible to the influence of the road, environment and vehicle, so there are some uncertainties. How to improve the power and generalization ability of the driving behavior recognition model still needs further study. Some paper use classification to detect pattern like this paper “Drank Driving Detection“,(2018) Although there has been significant work on drunk driving, this work is focusing largely on statistical analysis that examines the effect. The effect of alcohol on driving performance (S. Jongen, E. F. P. M. Vuurman, 2016). These empirical results provide a basis for regulators to decide on the relevant legal limits to drive under effect of alcohol [28]. However, these previous works did not do this, focus on real-time identification of drunk driving, which would allowing intervention when the driver is already drunk. Previous research also addressed the negative effect of alcohol on gaze behavior while looking.

Unfortunately, the results of this experimental study present an increase in the condition of diabetes leadership (Huiqin Chen and Lei Chen. 2017.) teaches that, while magic driver control training works well on previously employed drivers, but it does not achieve the performance required to generalize to unseen drivers.

“Drunk Driving Detection Based on Engine Locking System” (2015)

A drunk driving detection system based on the ignition interlock system is used to detect whether a person is drunk or not, and if the person is drunk, the system prevents him from driving and informs the nearest police station of the detection of an alcoholic driver. This system monitors the behavior of vehicles inside and outside the car (Rahul Mandalkar,2015). This system also notifies the ambulance and the driver's relatives if an accident occurs.

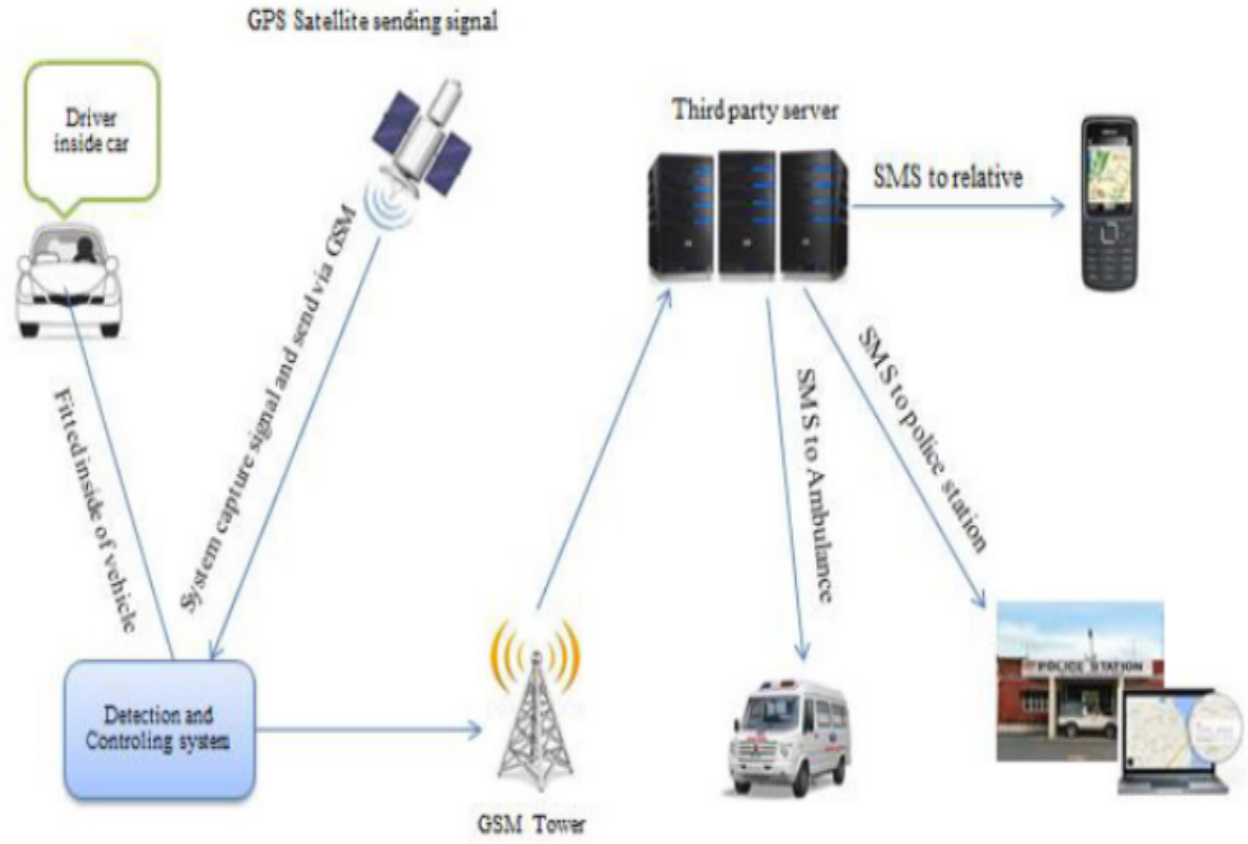


Figure 2: Architecture of the Engine Locking System

When the driver starts the car, the alcohol sensor (MQ-3) begins to sense that the car's speed is zero. If the driver is detected to be intoxicated, the ignition system will be turned off immediately with an alarm issued and the police station will be reported. A mark is made when the first condition is passed without detecting alcohol. When the vehicle speed is greater than zero, i.e. when the vehicle is in motion, the alcohol sensor starts sensing and sends the collected parameter values to the microcontroller. If it is detected drunk in this state, the signal will be sent to the fuel inhibitor by the microcontroller to block fuel supply to the ignition system. It also alerts the driver via an alert, and the car is stopped at that location with the police station notified.

“Drunk Driving Detection Using Driving Pattern”,(2018)

Drunk driving pattern detection using driving patterns Driving drunk driving pattern detection using driving patterns uses mobile phones as a platform to detect drunk driving because it combines detection and communication functions (Jiangpeng Dai, Jin Teng,2010). As a standalone device, the mobile phone presents a mature device.

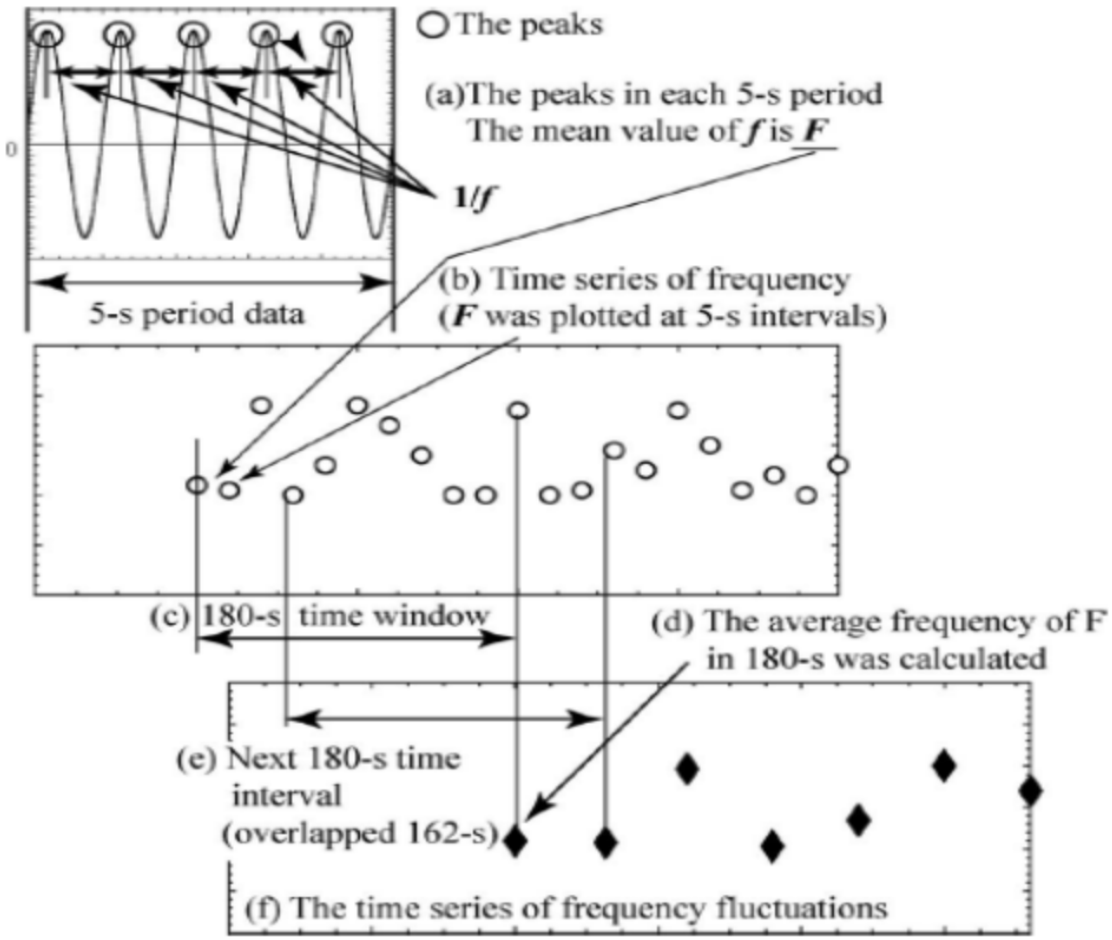


Figure 3: Time series computation of frequency fluctuation.

Hardware and software environment for developing an active alcohol driving monitoring system. A mobile phone-based system can operate effectively on its own because mobile phones are highly portable, all the necessary components are already built into them, and their communication services have wide coverage. The minimum requirement for this mobile platform is the presence of simple sensors such as accelerometers, direction sensors, etc. The communication unit and speaker are also good enough for alarm. Behaviors associated with drunk driving are classified into three categories. The first and second categories focus on driving behaviors related to vehicle movement; The third category concerns driving behavior related to driver autonomy and attention.

"Fuel Economy Impacts of Manual, Conventional Cruise Control, and Predictive Eco-Cruise Control Driving" (2018)

The paper presents the results of a field experiment that was designed to compare manual driving, conventional cruise control (CCC) driving, and Eco-cruise control (ECC) driving regarding fuel economy. The field experiment was conducted on five test vehicles along a section of Interstate 81 that was comprised of ±4% uphill and downhill grade sections. Using an Onboard Diagnostic II reader, instantaneous fuel consumption rates and other driving parameters were collected with and without the CCC system enabled. The collected data were compared regarding fuel economy, throttle control, and travel time. The results demonstrate that CCC enhances vehicle fuel economy by 3.3 percent on average relative to manual driving, however this difference was not found to be statistically significant at a 5 percent significance level. The results demonstrate that CCC driving is more efficient on downhill versus uphill sections. In addition, the study demonstrates that an ECC system can produce fuel savings ranging between 8 and 16 percent with increases in travel times ranging between 3 and 6 percent. These benefits appear to be largest for heavier vehicles (SUVs).

Chapter 3. The Proposed Solution for Detect Abnormal Drivers Pattern

3.1 Introduction

In the realm of transportation and road safety, the ability to identify and respond to abnormal driver behavior is of paramount importance. Abnormal driving patterns, encompassing actions such as sudden lane changes, aggressive acceleration, or erratic steering, not only pose risks to the driver but also jeopardize the safety of other road users. To address this critical concern, we present in this section our proposed solution for the detection of abnormal driver patterns.

Our approach is founded on the fusion of cutting-edge technologies, including machine learning algorithms, sensor data from modern vehicles, and real-time monitoring systems. In our pursuit of detecting and analyzing driver patterns, we employed a diverse set of machines learning models, Long Short-Term Memory Networks (LSTM), clustering, neural networks, K-nearest neighbors (KNN), and linear regression. This section presents the findings and outcomes of each method, showcasing their effectiveness in categorizing driving behaviors and providing valuable insights for various applications, from driver safety to traffic management.

3.2 Generate And Collecting the Data from Simulator

One alternative way to avoid making people under the effects of alcohol drive real cars which may make a real accident is to collect the data from it by creating a web-based car driving simulator and this is what I did for this study.

This simulator allows me to generate realistic driving scenarios while capturing crucial information about speed, deviation from lanes, and brake usage, which are three key features I used that define driving patterns.

Speed:

Speed plays a pivotal role in understanding driving behavior. A web-based car driving simulator enables us to control and measure a driver's speed in various conditions. By adjusting speed limits, we can simulate both normal and abnormal driving patterns. Data on speed provides insights into how drivers respond to different situations.

Deviation:

Deviation from the intended lane is another vital aspect of driving patterns. The simulator allows us to simulate real-world lane-keeping scenarios. By monitoring a driver's ability to stay within their lane, we can categorize behaviors or frequent lane changes. This information can be invaluable for developing lane-keeping assistance systems and understanding the impact of road design on driving patterns.

Brake Usage:

Effective brake usage is essential for safe driving. A web-based simulator enables us to mimic emergency stops, gradual deceleration, and braking in response to traffic conditions. Analyzing brake usage data helps us assess driver reaction times, brake intensity, and adherence to safe following distances.

Advantages of Web-Based Simulators:

Creating a web-based car driving simulator has distinct advantages. It's cost-effective, scalable, and accessible to a wide range of users. Participants can engage in simulated driving from the comfort of their web browsers, eliminating the need for physical setups. Moreover, the simulator allows for controlled experiments in various driving conditions, ensuring data consistency.

3.3 Data Generation and Analysis

3.3.1 Data Collection

This study leverages an extensive dataset that encapsulates a wide array of driving behaviors, harvested from two distinct sources to ensure robustness and diversity. The primary source of our dataset is derived from a Kaggle competition, consisting of real-world data meticulously collected from phone sensors affixed within vehicles. This rich dataset encompasses a total of **26,075** individual records, contributed by **43** distinct drivers, providing a comprehensive snapshot of everyday driving patterns.

The data obtained from Kaggle contains several critical features that are instrumental in assessing driving behavior:

- **ID**: A unique identifier for each driving session, allowing for individual analysis of driver behavior.
- **Lane**: The lane position of the car, offering insights into lane discipline and changes.
- **Speed**: The vehicle's velocity, which is crucial in understanding the dynamics of driving patterns.
- **Preceding Speed**: The speed of the car directly in front, pertinent for analyzing following distances and speed adaptations.
- **Time Gap**: The time interval between the host car and the preceding vehicle, providing a measure of safety margins maintained by the driver.
- **Road Condition**: Descriptive of the driving environment, whether it be dry, wet, or varied, and its potential impact on driving behavior.
- **Timestamp**: The time at which data points were recorded, enabling the reconstruction of the driving sequence.

Table 1: Dataset sample

ID	Lane	Speed	preceding_speed	time_gap	road_condition	label	timestamp
DR_24526	1	81	87	94.0	1	2	1332321295
DR_24526	1	88	81	11.0	1	2	1332321307
DR_24526	1	88	88	4.0	1	2	1332321312
DR_24526	2	84	89	127.0	1	2	1332321348
DR_24526	1	89	88	42.0	1	2	1332321354
DR_24526	2	91	84	30.0	1	2	1332321379
DR_24526	2	92	91	4.0	1	2	1332321385
DR_24526	2	90	92	2.0	1	2	1332321388
DR_24526	2	92	90	47.0	1	2	1332321436

Graph about features on different weathered conditions dry, wet , snow covered and visible track.

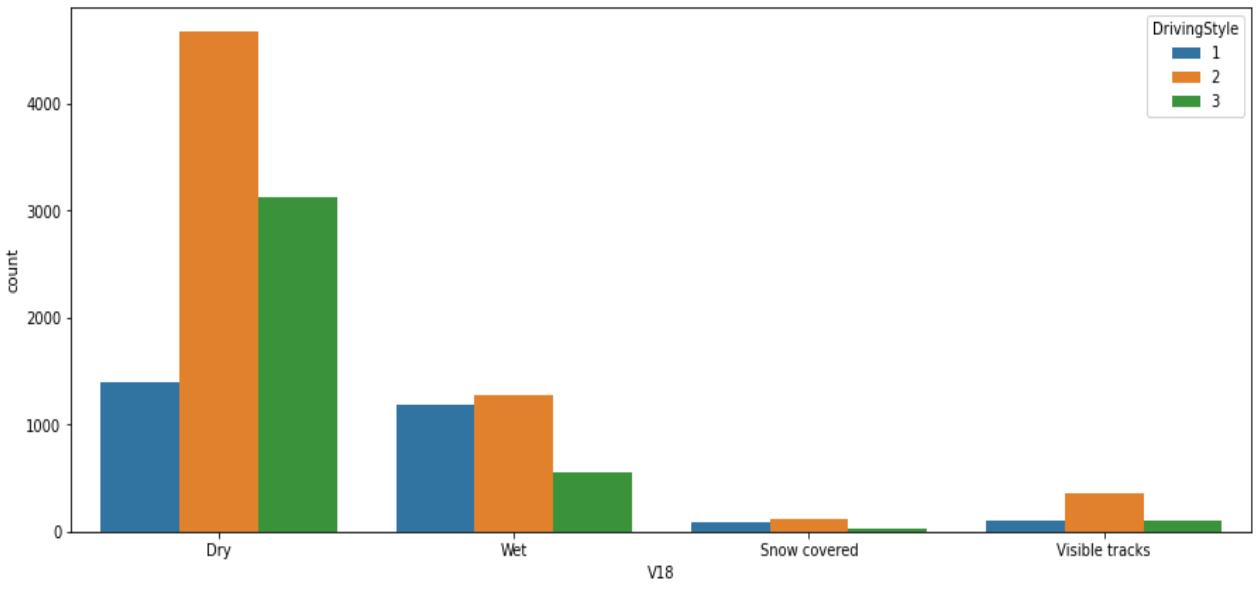


Figure 4: Features on different weather conditions (Kaggle- agg-driving).

The second portion of the dataset is a product of a bespoke driving simulator, designed to emulate both standard and erratic driving conditions. This simulation environment generated equal representations for normal and abnormal driving behaviors from a pool of 50 virtual drivers—25 emulating normal behavior and 25 simulating abnormal behavior. The simulator captures the following features:

- **Speed**: The simulated vehicle's speed, reflective of the driver's ability to control velocity under varying scenarios.
- **Deviation**: Lateral deviation from a predetermined path, indicating potential loss of control or correction maneuvers.
- **Brake Pattern**: The application pattern of brakes, which signifies the driver's reactionary measures in response to stimuli.
- **Timestamp**: Similar to the real-world dataset, marking the temporal aspect of the simulated driving data.

Table 2: Dataset sample

No.	Speed	Deviation	Brake Pattern	label	timestamp
1	9.6	1.82	0	1	1332335129
2	12.6	1.78	0	1	1332335133
3	12.8	1.85	0	1	1332335168
4	13.0	1.96	1	1	1332335193
5	13.1	1.60	1	1	1332335351
6	13.9	1.12	1	1	1332335369
7	14.5	0.85	0	1	1332336613
8	15.6	1.50	0	1	1332336648
9	17.1	1.69	0	1	1332335129

3.3.2 Data Analysis

a. Kaggle Real-World Driving Data

Upon acquisition, the Kaggle dataset underwent a rigorous preprocessing routine. Each record was scrutinized for completeness and integrity, ensuring no sensor errors or missing values could skew the results. Given the real-world nature of this data, it was paramount to handle outliers judiciously striking a balance between data authenticity and analytical clarity.

The dataset's vastness and variety offer an empirical foundation for understanding real-world driving behaviors. Statistical analyses were conducted to uncover patterns and correlations within the data. For instance, speed variances were cross-referenced with road conditions to assess how environmental factors influence driving habits.

The analysis also entailed a temporal examination, where sequences of driving data were reconstructed using timestamps. This sequence reconstruction facilitated the study of behavior over time, allowing for the assessment of driver reactions to the evolving road environment and traffic conditions.

b. Simulator-Generated Driving Data

The simulator-generated data serves as a controlled complement to the real-world data, providing clear-cut cases of normal and abnormal driving for algorithm training and validation. To mirror the complexity of real-life driving, the simulation was calibrated to replicate various traffic scenarios, weather conditions, and road types.

Initial analyses focused on contrasting normal versus abnormal driving patterns. Speed and deviation were particularly telling, with abnormal driving often characterized by higher speed variance and increased deviation. Brake pattern analysis provided further granularity, as erratic braking often correlates with abnormal driving behavior.

The simulated data was methodically timestamped to enable detailed temporal analysis, much like the Kaggle dataset. The intention was to not only capture instantaneous actions but also to understand the lead-up and aftermath of driving decisions.

c. Combined Dataset Insights

The fusion of real-world and simulated datasets presents a unique opportunity to validate the model's applicability across different data realms. The combined dataset is subjected to a comprehensive exploratory data analysis, ensuring a multifaceted understanding of driving behaviors. This amalgamation enriches the dataset, allowing the resulting models to be well-rounded and adaptive to both actual and simulated driving conditions.

By analyzing these diverse data sources collectively, the study gains a nuanced view of driving patterns, transcending the limitations of any single-source dataset. The resultant models, informed by such an all-encompassing dataset, are expected to demonstrate elevated levels of accuracy and generalizability when deployed in real-world scenarios.

As participants use the web-based simulator, data on speed, deviation, and brake usage are collected in real-time. This data I can use this to train my models and generate patterns for normal and abnormal driving with high accuracy this is a sample of data structure I export from my simulator:

[[1.84,24.8,0],[2.56,24.5,0],[3.18,24.2,0],[1.34,23.8,0],[1.07,23.5,0],[0.98,23.2,0],[0.52,22.9,0],[0.49,22.6,0],[0.74,22.3,0],[1.02,22,0],[1.16,21.8,0],[0.69,21.5,0],[0.87,21.2,0],[1.23,21.2,0],[1.23,21,0],[1.32,18.8,0],[1.23,18.9,0]**]**

Is an array of arrays each array has three values **[deviation, speed, brake pattern, timestamp].**

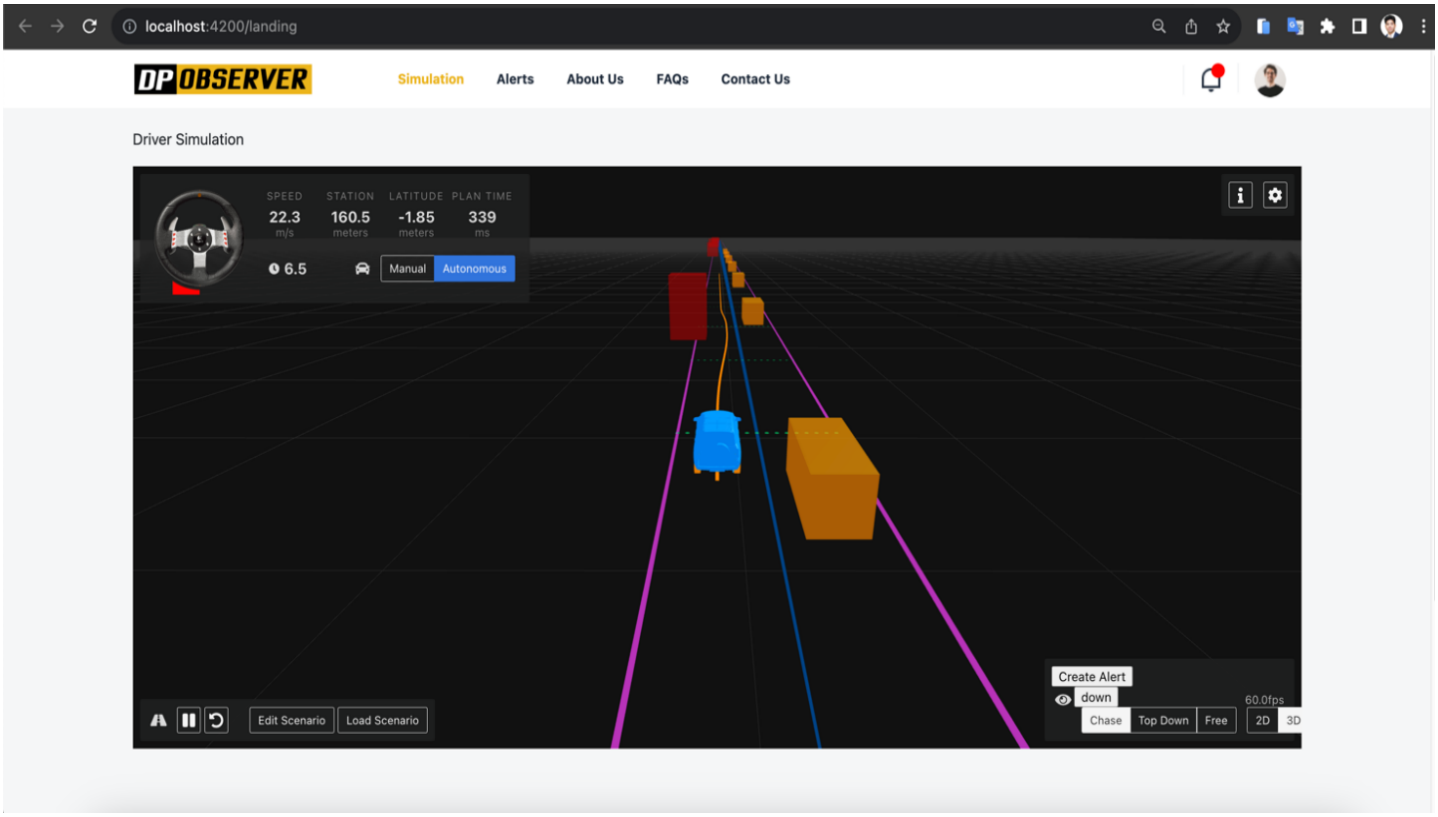


Figure 5: Car Simulator I get all information's from road

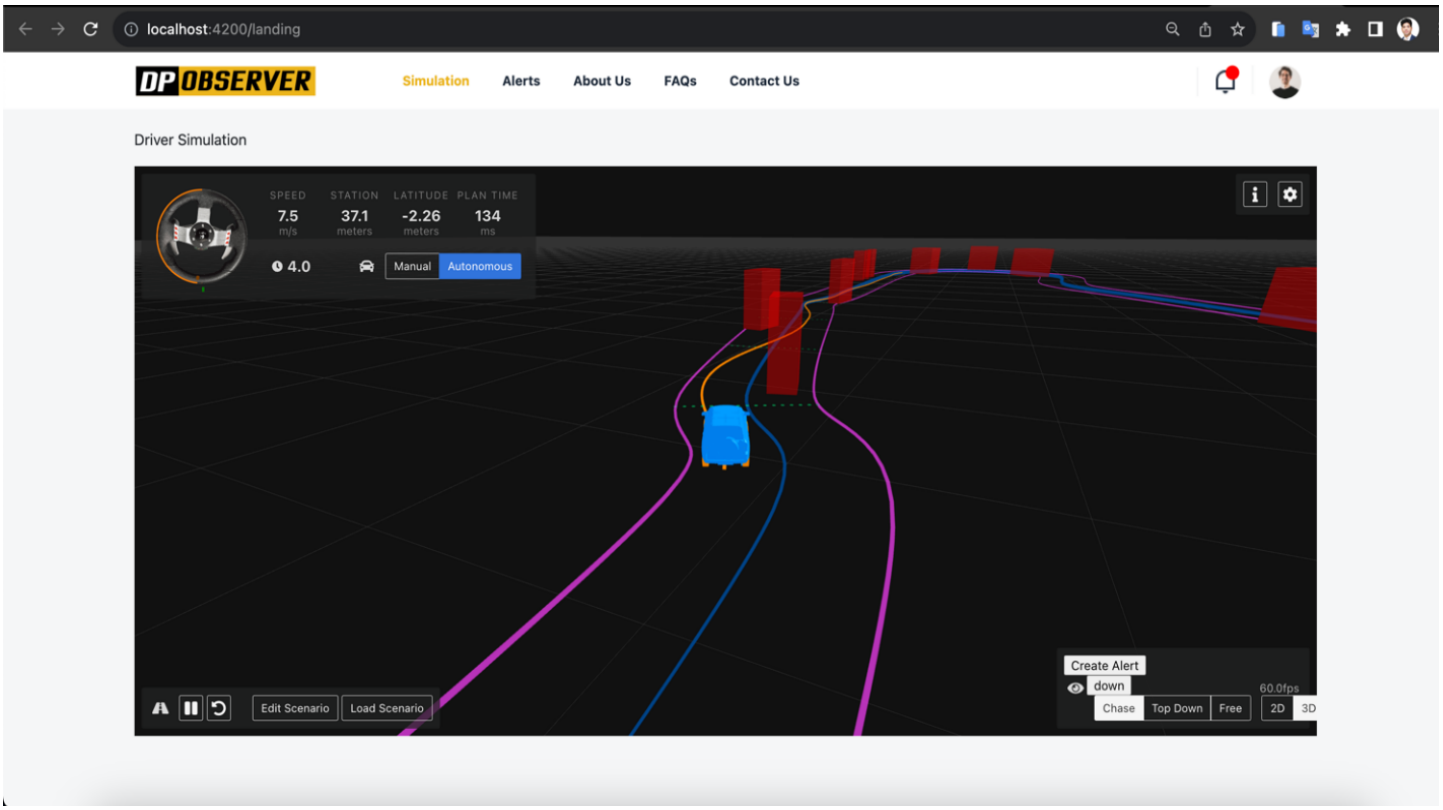


Figure 6: Car Simulator II

Figure 7: Real Car wheel Connected To Web Simulator

Figure 8: Real Car wheel Connected to Web Simulator

3.4 Chose Algorithm to Build an AI Model

after we get data from real dataset (Kaggle) and create the simulator and generate data with a specific structure next step is to select the best model that can give us the highest accuracy depending on the nature of my data, we can start this process in three steps:

1. define the models and their algorithms that we can use in this case.
2. training the models then test and evaluate the accuracy of each of them.
3. choose one of them to implement on my solution (**DPObserver**).

With these steps, we can create a comparison between each algorithm and build good software solutions have the ability to detect abnormal patterns and make a decision to prevent any accident from properly happening and this is the main goal of my study.

3.4.1 Define Different Algorithms

a. Long Short-Term Memory Networks (LSTMs):

Long Short-Term Memory networks are a specialized form of recurrent neural networks capable of learning long-term dependencies in sequential data. LSTMs are distinguished by their use of gate mechanisms that regulate the flow of information. These gates—namely, the input, forget, and output gates—allow LSTMs to selectively remember patterns over long sequences without the risk of vanishing or exploding gradients. This makes them highly suitable for complex sequential tasks such as time series prediction, natural language processing, and speech recognition. In the context of driving behavior analysis, LSTMs can effectively capture temporal dynamics and dependencies that characterize typical or anomalous driving patterns.

b. Bidirectional Long Short-Term Memory Networks (BiLSTMs):

Bidirectional LSTMs extend the conventional LSTM architecture by processing data in both forward and backward directions. This bidirectionality enables the networks to have both backward and forward information about the sequence at every time step. BiLSTMs are particularly useful in scenarios where the context of the entire sequence is crucial for making accurate predictions. In driving behavior analysis, BiLSTMs can provide a more nuanced understanding by considering sequences of actions leading up to and following a particular event.

c. Gated Recurrent Units (GRUs):

Gated Recurrent Units simplify the LSTM architecture by combining the forget and input gates into a single "update gate" and merging the cell state and hidden state. Despite their simplified structure, GRUs can match the performance of LSTMs on many tasks while being computationally more efficient. They are particularly effective in scenarios where the dataset size is relatively small, or the temporal dependencies are not excessively long. For driving behavior, GRUs can discern patterns and anomalies in driving sequences, often with faster training times compared to LSTMs.

d. 1D Convolutional Neural Networks (1D_CNNs):

1D Convolutional Neural Networks are typically utilized in the analysis of one-dimensional signal data. They are adept at extracting high-level features by applying filters to the sequence data, capturing local dependencies and translational invariances. This makes 1D_CNNs especially effective for tasks like audio analysis and time series classification, where the presence of certain patterns or anomalies is more significant than the exact position within the sequence. In analyzing driving data, 1D_CNNs can identify critical features such as abrupt accelerations or decelerations, which are indicative of driving behaviors.

e. Simple Recurrent Neural Networks (SimpleRNNs):

SimpleRNNs are the most basic form of recurrent neural networks, wherein connections between units form a directed cycle. This fundamental structure enables them to use their internal state (memory) to process sequences of inputs. However, they are less capable of handling long-term dependencies due to issues like vanishing gradients. Despite this, SimpleRNNs can serve as a baseline for more complex models and are computationally efficient, making them a quick and straightforward solution for preliminary sequence learning.

f. Deeper 1D Convolutional Neural Networks (Deeper_1D_CNN_Adjusted):

Deeper 1D Convolutional Neural Networks build on the basic 1D_CNN architecture by incorporating multiple convolutional and pooling layers. This depth allows the network to learn more complex and abstract features from the data, with each subsequent layer building on the features identified by the previous one. The added complexity can often result in improved model performance, particularly for complex sequential patterns in large datasets. When applied to driving behavior analysis, deeper 1D_CNNs can distinguish between nuanced behaviors and different driving conditions, learning from the intricate patterns within the data.

3.4.2 Test and Evaluate Each One

After collecting data and build small project depend on Python and flask framework this diagram for models training process:

This is 3D figure shows how right and wrong data **Individually** present in 3 dimensions each dimension representing a feature from 3 Deviation – Speed – Break.

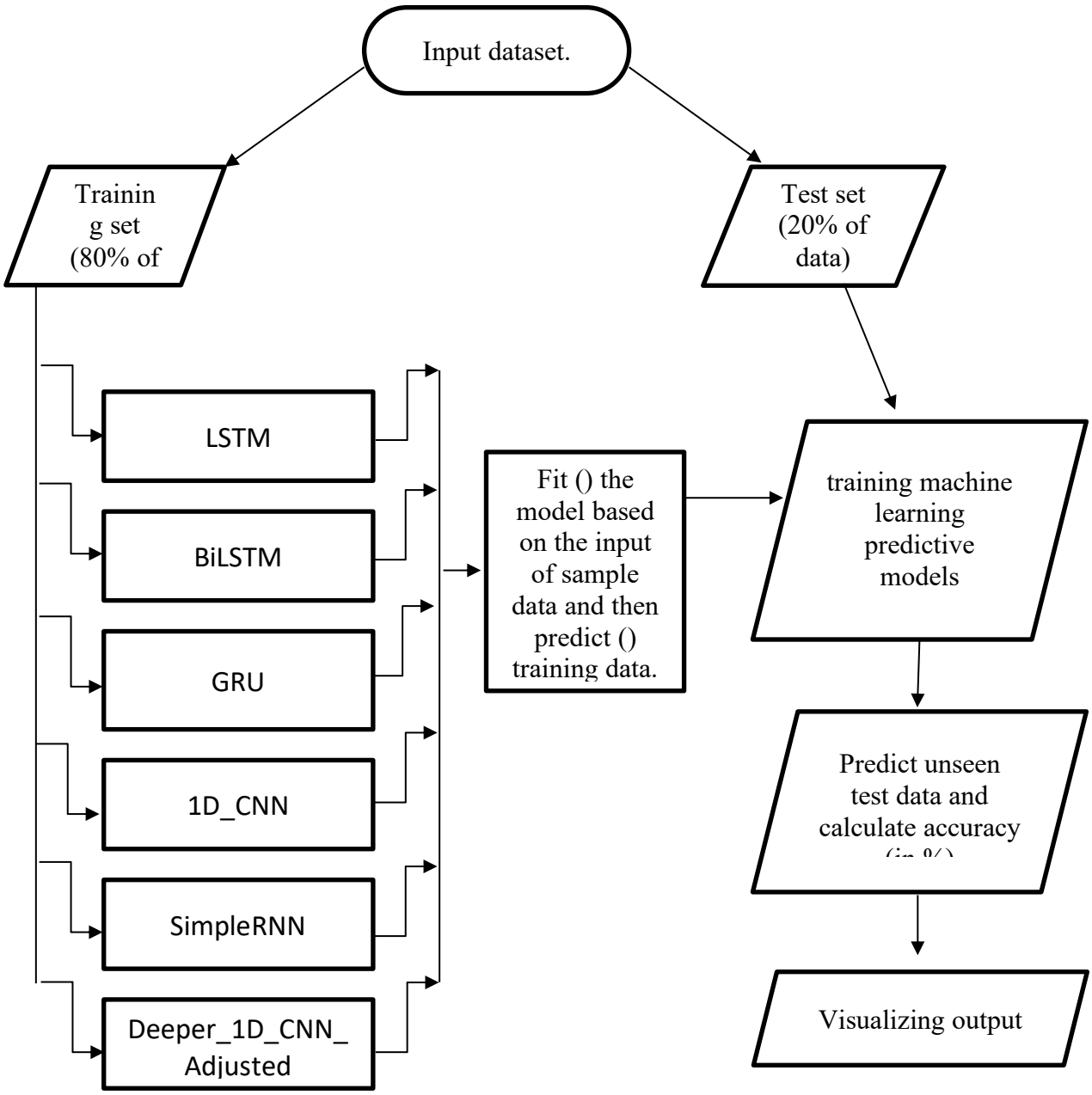


Figure 9: Models Training Process

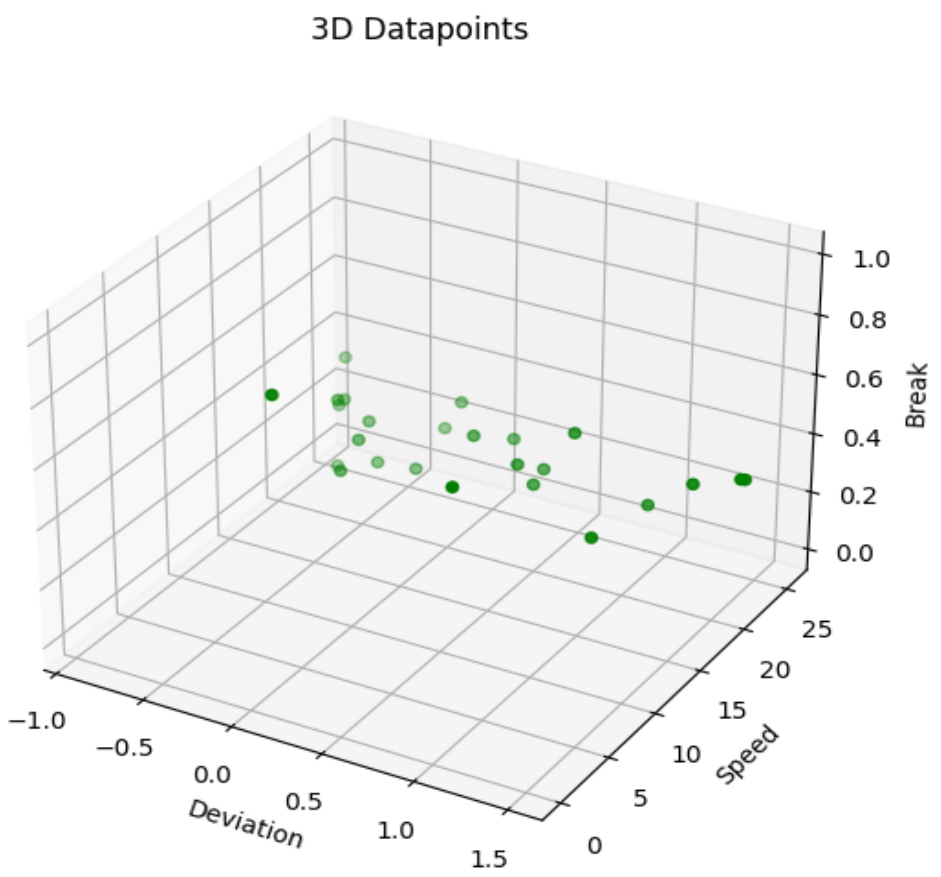


Figure 10:Right Data

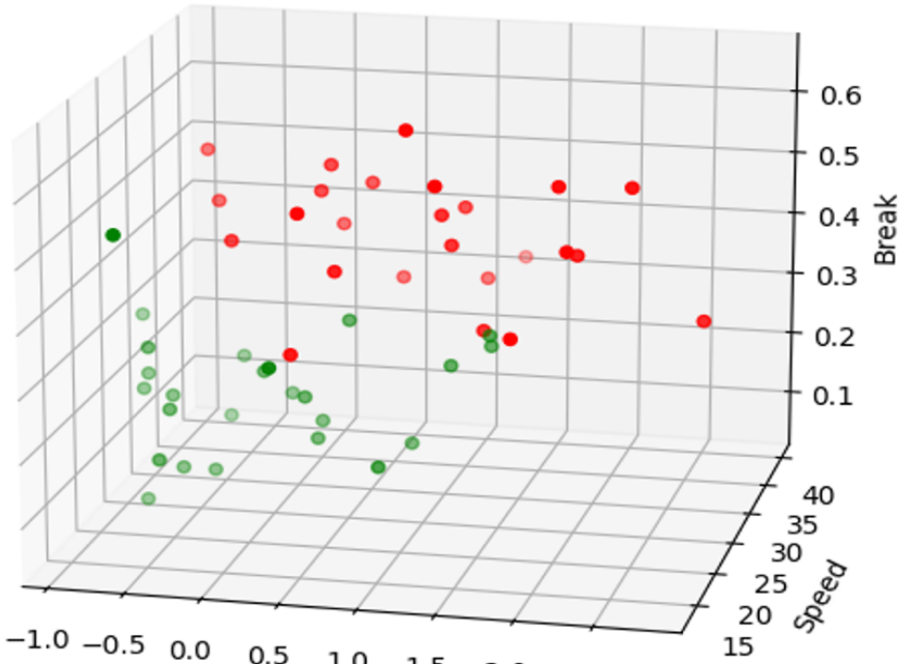


Figure 11: All Data in Graph in 3 dimensions

This is a 3D Figures show us wrong and right data individually after cleaning data and add label to each group (right, wrong) to test models and export the accuracy (in %) you can see it in next table (3).

We extract accuracy from each algorithm individually many times each time try to get high accuracy by cleaning data and avoiding overfitting until get the results that you can find in the table below:

notes "Accuracy can be better if we add more test data."

The accuracy for real dataset (Kaggle):

Table 3: Comparison between Different Machine Learning Models - real dataset.

No.	Algorithm	Data length	Accuracy
1	LSTM Algorithm	24073	78.86%
2	BiLSTM Algorithm	24073	81.10%
3	GRU Algorithm	24073	77.36%
4	1D_CNN Algorithm	24073	78.51%
5	SimpleRNN Algorithm	24073	77.58%
6	Deeper_1D_CNN_Adjusted Algorithm	24073	76.49%

The accuracy for simulator dataset (DPObserver):

Table 4: Comparison between Different Machine Learning Models

No.	Algorithm	Data length	Accuracy
1	LSTM Algorithm	50	88.02%
2	BiLSTM Algorithm	50	89.39%
3	GRU Algorithm	50	87.45%
4	1D_CNN Algorithm	50	82.36%
5	SimpleRNN Algorithm	50	84.58%
6	Deeper_1D_CNN_Adjusted Algorithm	50	75.55%

3.4.3 Choose Most Effective and Applicable Algorithm

Depend on previous table (3 and 4) the most affection model is LSTM, GRU, BiLSTM, 1D CNN, SimpleRNN, and a deeper 1D CNN so I decide to choose **BiLSTM** , **LSTM** and **GRU** to build model in my solution because of fits with my data type .

3.5 Design Solution Architecture

Join us on this journey as we navigate the practical aspects of transforming a visionary architectural design into a tangible software solution. The implementation of our solution architecture marks a critical milestone in our mission to enhance road safety, optimize traffic management, and contribute to the evolution of intelligent transportation systems.

In the following sections, we will provide insights into the technologies, methodologies, and diagrams that underpin our software's implementation, with the ultimate aim of unraveling the intricate patterns of driver behavior using the power of artificial intelligence.

3.5.1 Context diagram

This is a context diagram to clarify the level of my solution architecture it starts with an agent in our case is car sensor that sends the data to my solution DPObserver after analysis after that he sends an alert to the police station if the case is dangerous.

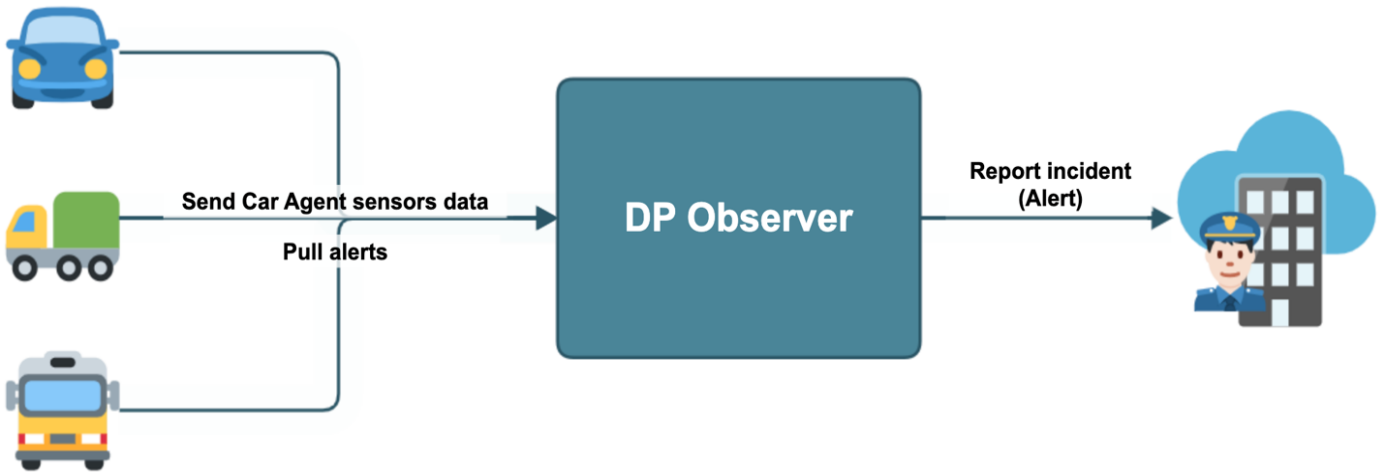


Table 5: Context diagram

3.5.2 Component diagram

Now I demonstrate the main components of my solution “DPObserver”, it starts with the first entity car agent pulls notifications from cloud servers connected with the backend that has a link with AI model service and database component,

It gives the last component (police incident) report or alert in some cases the system makes a decision on it.

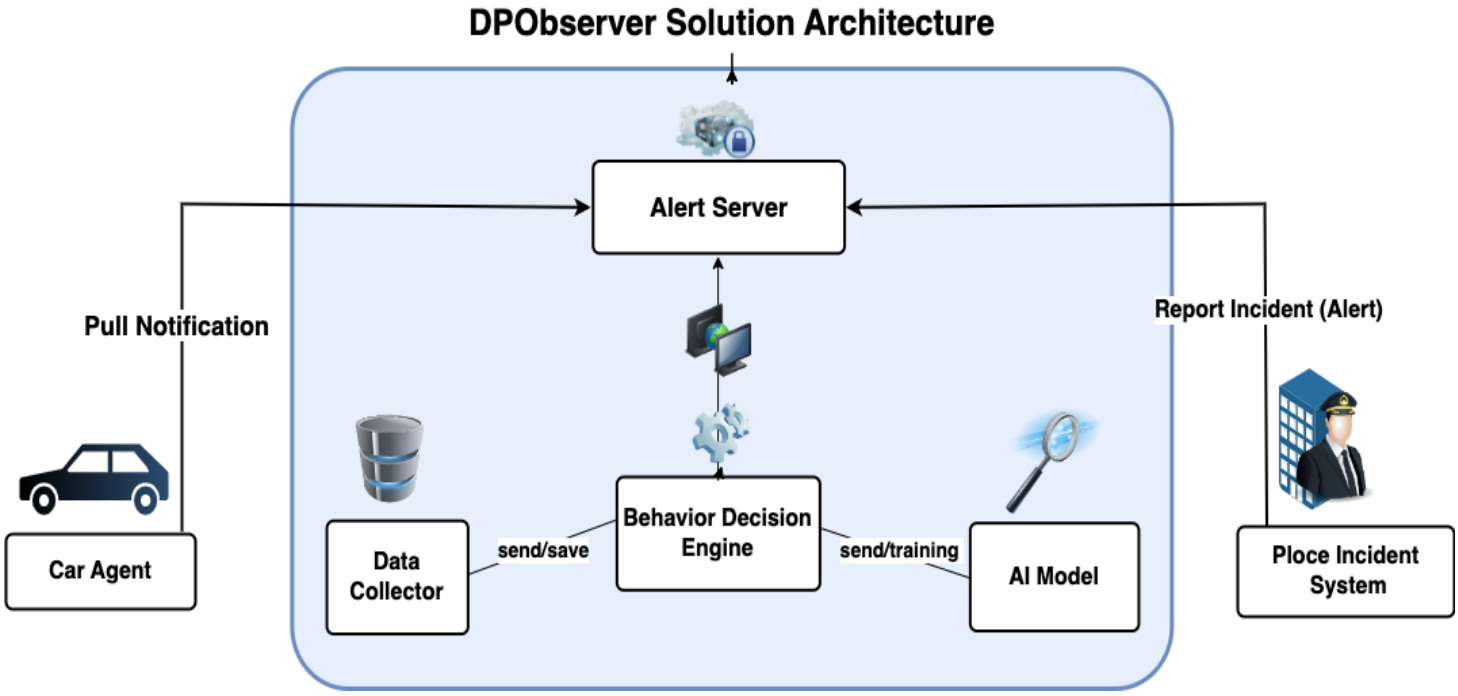


Figure 12: Component diagram

3.5.3 Components Architecture Description

Car Agent: This is the vehicle's interface, equipped with various sensors that collect real-time data on driving patterns. It can be thought of as a mobile data acquisition unit that captures key metrics related to vehicle operation and driver behavior.

Data Collector: The data from the Car Agent is sent to the Data Collector, which acts as a repository and preprocessing center. This module aggregates data performs initial

cleaning, and structures it into a suitable format for analysis. It serves as a staging area for raw driving data before further processing.

Behavior Decision Engine: After initial preprocessing, the data is passed to the Behavior Decision Engine. This crucial component analyzes the incoming data streams to make real-time decisions. It is responsible for identifying potential instances of abnormal driving behavior that may warrant further investigation or immediate action.

AI Model: The heart of the analytical process lies within the AI Model. Here, advanced machine learning algorithms, possibly including LSTM networks for sequence analysis, are trained on historical data to recognize patterns indicative of abnormal driving. The AI Model receives input from the Behavior Decision Engine for both ongoing pattern detection and continuous model improvement through retraining with new data.

Alert Server: Upon detection of a possible abnormal event by the AI Model, an alert is generated and sent to the Alert Server. The server manages these alerts and determines the appropriate response protocols to enact.

Car Agent (Pull Notification): If an alert corresponds to an immediate risk, a notification is pushed to the Car Agent, which in turn may alert the driver to take corrective action. This real-time feedback loop is vital for preventing potential accidents or mitigating hazardous situations.

Place Incident System: Concurrently, the Alert Server communicates with external systems, such as a Police Incident System. Here, alerts can be evaluated by traffic management personnel, and if necessary, dispatched to law enforcement or emergency services. This link ensures that incidents are recorded, and appropriate law enforcement measures can be taken, enhancing road safety at a broader level.

Report Incident (Alert): This represents the flow of information from the Alert Server to the Place Incident System. It signifies the reporting mechanism that conveys detected incidents to the authorities.

3.5.4 Overall Workflow

- The Car Agent captures data from the vehicle's sensors and relays it to the Data Collector.
- The Data Collector processes and sends the data to the Behavior Decision Engine.
- The Behavior Decision Engine evaluates the data and interfaces with the AI Model for pattern detection.
- Upon detection of an abnormal pattern, the AI Model signals back to the Behavior Decision Engine.
- The Behavior Decision Engine then prompts the Alert Server to act.
- The Alert Server issues a pull notification to the Car Agent to alert the driver and simultaneously reports the incident to the Police Incident System.

Conclusion

The (DPobserver) solution architecture is a sophisticated, closed-loop system designed for real-time surveillance and intervention of abnormal driving patterns. By integrating in-vehicle sensor data with advanced AI analytics and real-time alerting mechanisms, (DPobserver) aims to contribute significantly to enhancing road safety, reducing the incidence of traffic accidents, and streamlining traffic incident management and response.

3.5.5 Technology diagram

I used many technologies in this solution and integrate using REST API in the best way to get high value from it and this a list:

1. Front-end: Angular framework
2. Backend: NodeJS
3. AI Model: Flask, sklearn and Panda's lib
4. Database: NoSQL (MongoDB).
5. Realtime: Firebase (cloud service)

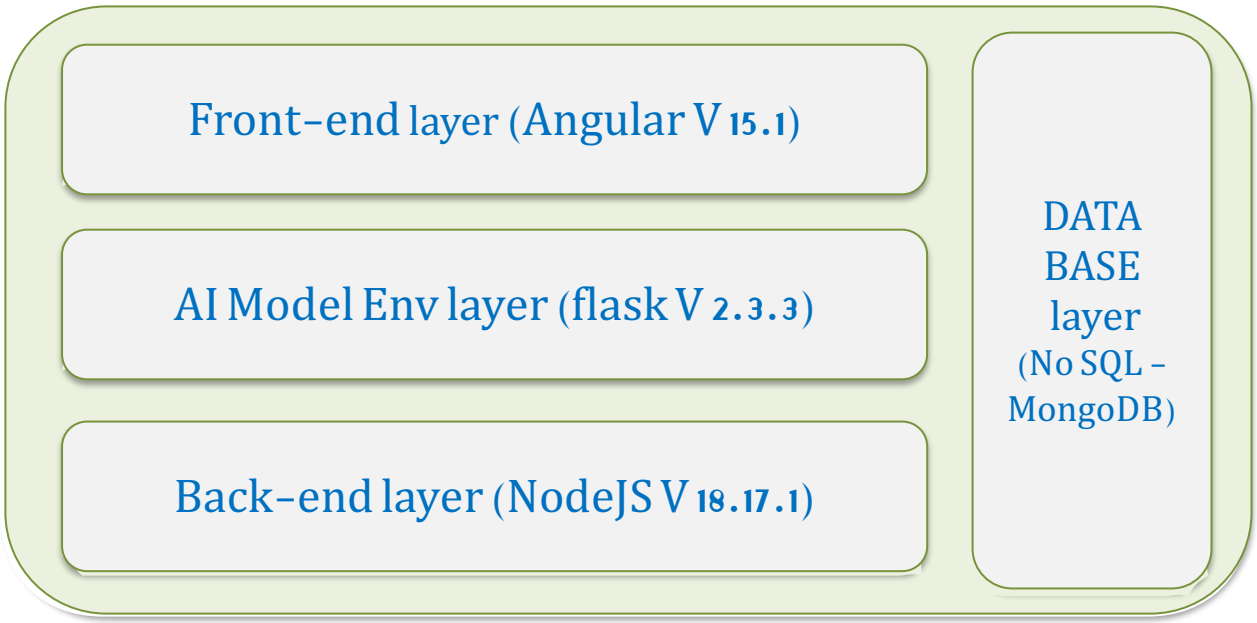


Figure 13: Technologies diagram

3.5.6 Sequence diagram

Now this sequence demonstrates the actions happened in system:

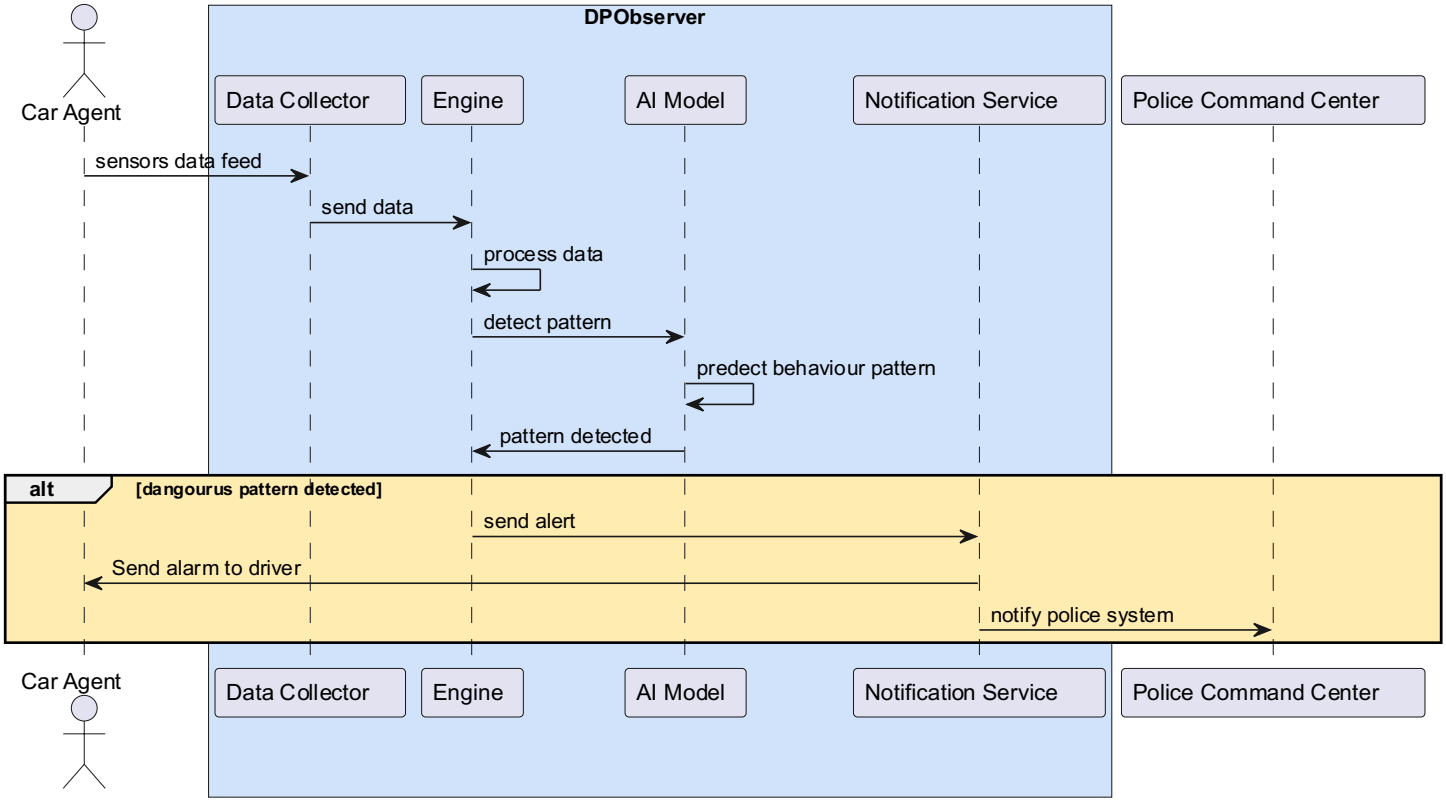


Figure 14: Sequence diagram

Description

The sequence diagram outlines an automated system, termed "DPObserver," for detecting and responding to abnormal driving patterns. The system involves multiple components interacting in real-time, which are described below:

a. **Car Agent**: This represents an interface or a proxy within a vehicle through which sensor data is collected. It could be part of a telematics system or an app on the driver's smartphone.
b. **Data Collector**: The first step in the process begins with the "Car Agent" feeding sensor data to the "Data Collector." This component is responsible for gathering various inputs such as speed, acceleration, GPS location, and possibly other telemetry data indicative of driver behavior.

c. **Engine**: Upon receiving the data, the "Engine" processes it for analysis. This involves cleaning, normalizing, and preparing the data for pattern detection, which may include extracting features relevant to driving behavior.
d. **AI Model**: The processed data is then forwarded to the "AI Model," where the core analysis takes place. The AI Model, equipped with machine learning capabilities, likely a Long Short-Term Memory (LSTM) network, evaluates the data to predict behavior patterns. The LSTM's ability to analyze sequences makes it adept at recognizing complex driving patterns over time.
e. **Pattern Detection**: The AI Model performs continuous analysis, predicting behavior patterns as new data arrives. When a pattern indicative of abnormal or dangerous driving is detected, the model signals this finding back to the Engine.
f. **Notification Service**: In response to a dangerous pattern being detected, the Engine communicates with the "Notification Service."
g. **Alert System**: The "Notification Service" then takes two concurrent actions:
    - It sends an alert back to the "Car Agent," which could trigger an alarm or warning to the driver, allowing for immediate corrective action.
    - It also notifies an external system, designated as the "Police Command Center," possibly to inform authorities about the potential danger or to take further preventive measures.

3.5.7 Conditional Alert (alt) Pathway

- The sequence diagram includes an alternate pathway (highlighted with **alt**), representing the conditional logic where alerts are issued only if a dangerous pattern is detected.
- This condition reflects a system designed to minimize unnecessary notifications, ensuring that only significant events trigger the alert process.

3.6 Abstract

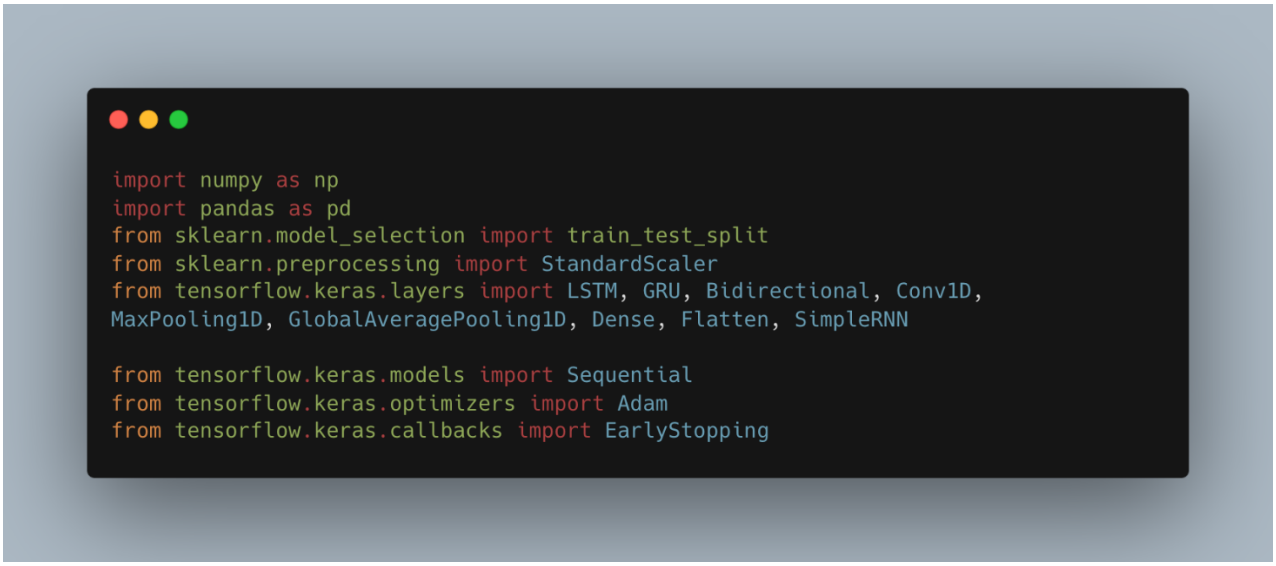
The DPObserver system depicted in the sequence diagram represents a sophisticated, real-time monitoring solution designed to enhance road safety. By integrating advanced AI analytics with a responsive alert mechanism, the system aims to provide an immediate warning about abnormal driving behaviors, contributing to proactive accident prevention and enhancing the overall safety of road travel. This system exemplifies the potential of AI and machine learning to provide actionable insights in critical real-world applications.

3.7 Solution Code

I have implemented for two approaches the real data and I use LSTM algorithm for sequential data and the second one use different algorithms such as BiLSTM, GRU, 1D_CNN Validation etc., to get high accuracy let's start with importing all algorithms:

3.7.1 Real Data

In this section, we describe the implementation of a machine learning approach using Long Short-Term Memory (LSTM) networks designed to differentiate between normal and abnormal driving patterns. The LSTM algorithm is particularly suitable for this task due to its ability to process entire sequences of data and recognize patterns over time, which is essential in understanding the dynamics of driving behavior.



```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.layers import LSTM, GRU, Bidirectional, Conv1D,
MaxPooling1D, GlobalAveragePooling1D, Dense, Flatten, SimpleRNN

from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
```

Figure 15: Code of the implementation – import libs we need

In the provided snippet of code, we observe a preparatory block integral to the process of building and evaluating sequential models. This portion of the script sets the foundation for a series of computational tasks that include data ingestion, preprocessing, model instantiation, and training regimen establishment for a variety of neural network architectures. Below is a detailed description of the actions being taken in this code:

3.7.2 Data Import and Preparation

The code begins with the importation of essential Python libraries that provide the backbone for data manipulation, machine learning, and deep learning. Notably:

- **numpy** and **pandas**: These libraries are quintessential for data handling. NumPy offers support for complex mathematical operations and multi-dimensional arrays, while pandas provide user-friendly data structures and data analysis tools, enabling seamless data manipulation and cleaning.
- **train_test_split from sklearn.model_selection**: This utility function is pivotal for splitting datasets into separate training and validation sets. By segregating the data, one ensures that the model can be trained on one portion of the data and impartially evaluated on another, providing an assessment of its generalization capability.

• **StandardScaler** from **sklearn.preprocessing**: Critical in preparing the input features for neural network models. StandardScaler standardizes features by removing the mean and scaling to unit variance, effectively normalizing the data and ensuring that the model isn't unfairly biased by the scale of any individual feature.

3.7.3 Neural Network Components Importation

The code imports various components from the TensorFlow and Keras libraries, tailored for building sequential neural network models:

• **Sequential** from **tensorflow.keras.models**: Serves as a linear stack of layers, providing a simple way to instantiate a neural network model where a layer has exactly one input tensor and one output tensor.

• Layers such as **LSTM**, **GRU**, **Bidirectional**, **Conv1D**, **MaxPooling1D**, **GlobalAveragePooling1D**, **Dense**, and **Flatten** from **tensorflow.keras.layers**: These are the building blocks of neural networks in Keras, each representing a specific type of network layer that performs distinct operations on the data. For instance, **LSTM** and **GRU** are types of recurrent layers suited for sequential data, while **Conv1D** and **MaxPooling1D** are convolutional layers used commonly in time-series or signal data.

• **Adam** from **tensorflow.keras.optimizers**: This is an optimizer that employs an adaptive learning rate, often leading to faster convergence in training deep learning models. It's a go-to optimizer due to its efficiency and effectiveness across a wide range of problems.

• **EarlyStopping** from **tensorflow.keras.callbacks**: A callback to stop training when a monitored metric has stopped improving, preventing overfitting and saving computational resources.

3.7.4 Training and Evaluation Setup

In the code, these components are orchestrated to build multiple models, each tailored to extract patterns and learn from sequential data. The snippet alludes to the subsequent training phase, where models are fitted to the training data with an early stopping mechanism to optimize their performance on unseen validation data. Each model's architecture—whether it be a simple RNN, LSTM, GRU, or a convolutional network—is compiled and trained, their performance meticulously logged and evaluated.

This preparatory block is a cornerstone in the study, as it encapsulates the methods by which the study's models are not just theorized but operationalized—transforming raw data into actionable insights, which in the realm of this study, involves discerning the nuances of driving behavior.

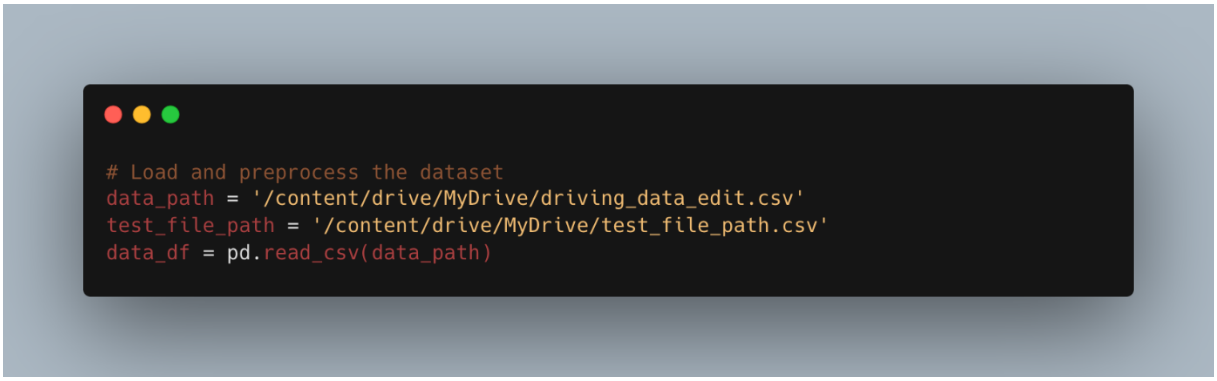


Figure 16: Code of the implementation – read files

3.7.5 Loading and Integrating the Datasets

Here, we are loading two separate datasets: one that contains examples of normal driving patterns and another that comprises instances of abnormal driving behavior. These datasets are fundamental to our study as they embody the ground truth against which the LSTM model will learn and subsequently make predictions. The pd.read_csv function is called upon to read the CSV files and convert them into Pandas DataFrames.

These DataFrames (normal_df and abnormal_df) provide a convenient and powerful data structure for subsequent data manipulation and analysis.

The normal_data.csv file is expected to have a collection of driving data that has been labeled as normal, representing standard, safe driving behavior under usual conditions. Conversely, the abnormal_data.csv file consists of driving data that has been identified as abnormal, possibly containing patterns that signify dangerous or erratic driving, such as sudden stops, excessive speeding, or irregular turns that could indicate an increased risk of accidents.

The test_file_path is a variable assigned to the path of a CSV file reserved for testing the model. This testing dataset is crucial for objectively evaluating the model's performance. It is imperative that the data in test_right.csv has not been used in the training process to ensure that the evaluation of the model's generalization capabilities is unbiased and valid.

The separation of data into training, validation, and testing sets mirrors the rigor and structure required in empirical research, enabling us to draw reliable conclusions about the model's effectiveness in real-world application scenarios.

Figure 17: Code of the implementation – read files.

This segment of the code illustrates a crucial preprocessing step in the context of supervised learning, where the target variable, or label, must be encoded into a format suitable for the model to understand and learn from. In machine learning, particularly classification tasks, it's common to map categorical labels to numerical values since models inherently operate on numerical data.

3.7.6 Label Mapping

The data_df['label'] assignment line performs a mapping operation where the 'label' column in the dataframe data_df undergoes a transformation. Specifically, the value 1, denoting abnormal driving behavior, is mapped to 0, and the value 2, signifying normal driving behavior, is mapped to 1. This binary encoding reflects the two classes of the target variable in a form that neural network models can effectively process, typically expected in binary classification tasks. By convention, 0 often denotes the negative class, and 1 denotes the positive class.

3.7.7 Feature Selection

Subsequently, the code selects relevant features for the learning process. The chosen features—'Lane', 'Speed', 'preceding_speed', 'road_condition', 'timestamp'—are expected to be influential in predicting the driving behavior as normal or abnormal. Each feature represents an aspect of the driving environment:

• **Lane**: The lane in which the vehicle is traveling, which may influence or indicate specific driving behaviors.
• **Speed**: The current speed of the vehicle, providing direct insight into the driver’s speed control.

• **Preceding Speed**: The speed of the preceding vehicle, which is crucial for understanding interactive driving patterns, such as tailgating or safe following distances.

• **Road Condition**: The condition of the road surface, which could significantly affect driving behavior, particularly in terms of safety and vehicle control.

• **Timestamp**: The time at which the data was recorded, enabling the analysis of driving behavior across different times of the day or under varying conditions.

The line X = data_df[feature_columns] extracts these features from the dataframe, creating a new dataframe X that encapsulates all the input variables to be fed into the models. Simultaneously, y = data_df['label'] isolates the target variable, storing it in y for supervised learning.

In summary, this code snippet is about preparing the dataset for training by converting categorical labels to a binary numerical format and selecting pertinent features that are hypothesized to influence the target variable. Such preprocessing is a prerequisite to ensure that the dataset is in a state conducive to the subsequent training of machine learning models.

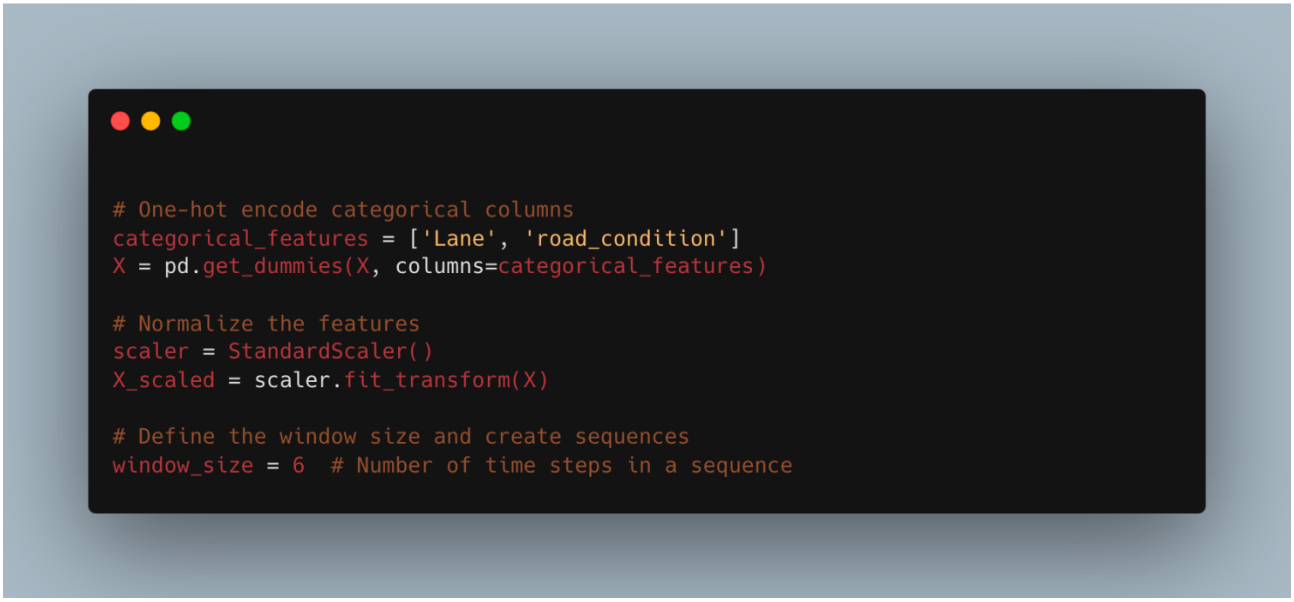


Figure 18: Code of the implementation – Feature normalization and sequence creation.

In this code snippet, we progress further into the data preprocessing phase, essential for preparing the dataset for effective model training. The operations performed in this

snippet are crucial for handling categorical data and standardizing feature scales, as well as structuring the data into a format appropriate for time-series analysis or any sequence-dependent modeling. Below is a detailed explanation of each step:

3.7.8 One-Hot Encoding of Categorical Features

The process begins by converting categorical features into a format that can be provided to machine learning algorithms. This is achieved via one-hot encoding, a technique that transforms categorical variables into a form that could be provided to ML algorithms to do a better job in prediction.

Python code:
categorical_features = ['Lane', 'road_condition'] X = pd.get_dummies(X, columns=categorical_features)

Here, **get_dummies** is a pandas function that converts categorical variable(s) into dummy/indicator variables. For instance, if 'Lane' has three categories (e.g., 'left', 'middle', 'right'), it creates three new columns, one for each category, with binary values 0 or 1. The same process applies to the 'road_condition' feature. This transformation is fundamental since most machine learning models, particularly neural networks, require numerical input.

3.7.9 Feature Normalization

Next, we encounter the normalization of features:

Python code:
scaler = StandardScaler() X_scaled = scaler.fit_transform(X)

The **StandardScaler** from the scikit-learn library standardizes the dataset's features by removing the mean and scaling to unit variance. This is an important step in many

machines learning algorithms' preprocessing as it can significantly impact the performance of models. Standardizing features so that they are centered around zero with a standard deviation of one ensures that the scale of the features does not distort the learning process, which is especially vital for models sensitive to the magnitude of input values, such as neural networks.

3.7.10 Sequence Creation for Time-Step Analysis

Finally, the code snippet addresses the preparation of the data for sequence learning:

Python code:

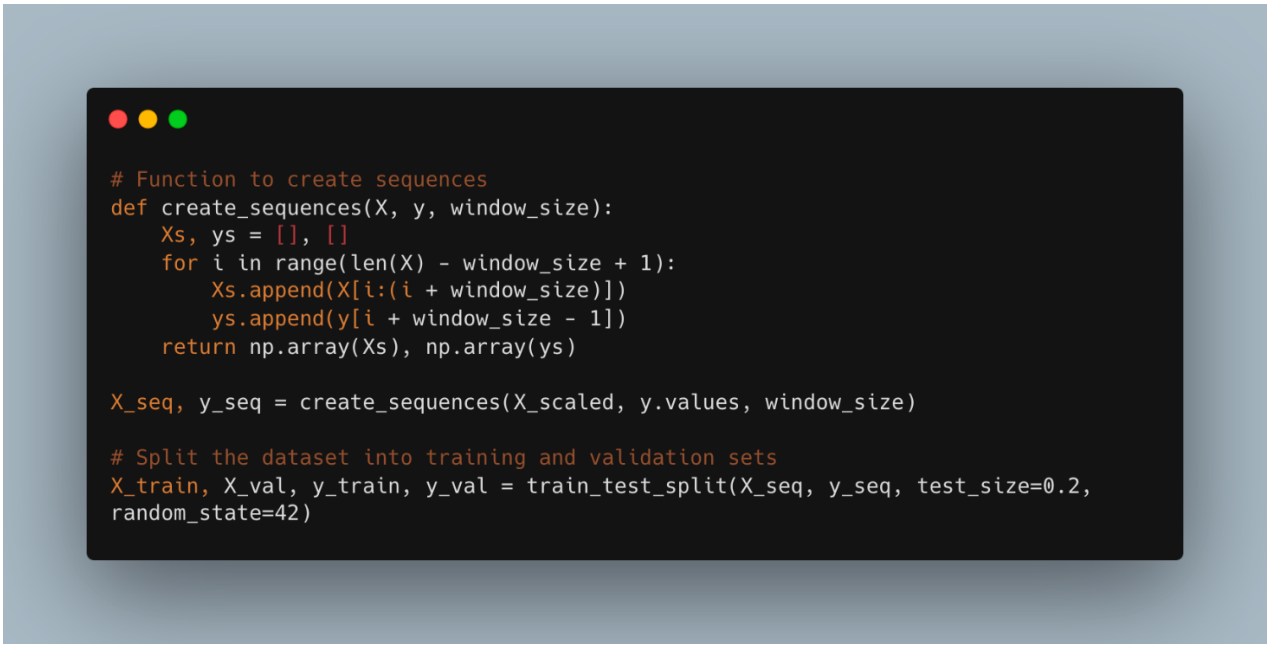
```
window_size = 6 # Number of time steps in a sequence
```

In the context of sequential data, such as time-series, it is often the case that the prediction for a given time point may depend not only on the current input features but also on previous time steps. The **window_size** variable determines how many consecutive time steps will be used to form a single input sequence for the model. In this case, a window of 6 indicates that each input sequence will consist of data from 6 consecutive time points.

With these steps, the dataset is no longer a simple table of independent rows but a series of overlapping windows, each providing a temporal slice of data to the learning algorithm. This structure is critical for models that aim to capture temporal relationships and is particularly well-suited to recurrent neural networks and other sequence-based models that will be trained to predict driving behavior based on sequential data.

In conclusion, this snippet's operations are geared toward transforming the raw dataset into a refined structure, ready to be fed into the sophisticated neural networks that will be tasked with learning from and making predictions based on the data. The transformations applied here—categorical encoding, feature scaling, and sequence

creation—collectively serve to bridge the gap between raw data and an analyzable format amenable to advanced machine learning techniques.



```
# Function to create sequences
def create_sequences(X, y, window_size):
    Xs, ys = [], []
    for i in range(len(X) - window_size + 1):
        Xs.append(X[i:(i + window_size)])
        ys.append(y[i + window_size - 1])
    return np.array(Xs), np.array(ys)

X_seq, y_seq = create_sequences(X_scaled, y.values, window_size)

# Split the dataset into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_seq, y_seq, test_size=0.2,
random_state=42)
```

Figure 19: Code of the implementation – Create Sequences

This code snippet describes the function **create_sequences**, which is used to restructure the dataset into a sequence format suitable for time-series analysis or any modeling that requires understanding temporal dynamics. It also covers the subsequent division of the dataset into training and validation sets. Here's a detailed breakdown:

3.7.11 Sequence Creation

Python code:

def create_sequences(X, y, window_size): Xs, ys = [], [] for i in range(len(X) - window_size + 1): Xs.append(X[i:(i + window_size)]) ys.append(y[i + window_size - 1]) return np.array(Xs), np.array(ys)

This function takes the scaled feature matrix **X**, the label vector **y**, and an integer **window_size** to create overlapping sequences of data. The function iterates over the entire dataset, creating a new data point for each window. Each sequence consists of **window_size** consecutive timesteps from **X**, which corresponds to a single output in

y. This approach is particularly effective for capturing temporal patterns as it allows the model to consider the context of previous and subsequent data points within a given window.

For example, if **window_size** is set to 6, the function takes the first 6 timesteps of data to create the first sequence and its corresponding label, then it moves one timestep forward to create the next sequence, and so on. This results in sequences that share **window_size - 1** timesteps with their immediate predecessors, providing a continuous flow of temporal information.

3.7.12 Dataset Splitting

Python code:

```
X_train, X_val, y_train, y_val = train_test_split(X_seq, y_seq, test_size=0.2, random_state=42)
```

After creating the sequences, the code then divides them into training and validation sets. This split is essential for training machine learning models, as it allows for the evaluation of the model's performance on unseen data, thus estimating how well the model will generalize to new data. The **train_test_split** function from scikit-learn is utilized to perform this split, with 20% of the data being reserved for validation. This function ensures that the split is randomized to prevent any unintentional ordering effects but is also reproducible due to the **random_state** parameter.

By creating these sequences and splitting the dataset accordingly, we are preparing the data in a way that is suitable for training time-dependent models, such as recurrent neural networks, which are designed to handle input data in sequences and can learn patterns across these sequences to make predictions.

In the broader context of academic research, this step is critical. It transforms a tabular dataset into a structured format that faithfully represents the temporal order of events, a format that is amenable to the unique demands of sequential data modeling. This

transformation underpins the efficacy of subsequent analyses and is foundational for developing models that provide deep insights into the dynamic patterns within the dataset.

```
# List of models to train
models = [
    Sequential([
        LSTM(20, activation='relu', input_shape=(window_size, X_train.shape[2])),
        Dense(1, activation='sigmoid')
    ], name="LSTM"),

    Sequential([
        GRU(20, activation='relu', input_shape=(window_size, X_train.shape[2])),
        Dense(1, activation='sigmoid')
    ], name="GRU"),

    Sequential([
        Bidirectional(LSTM(15, activation='relu'), input_shape=(window_size,
X_train.shape[2])),
        Dense(1, activation='sigmoid')
    ], name="BiLSTM"),

    Sequential([
        Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=
(window_size, X_train.shape[2])),
        MaxPooling1D(pool_size=2),
        Flatten(),
        Dense(50, activation='relu'),
        Dense(1, activation='sigmoid')
    ], name="1D_CNN"),  # Name adjusted here

    Sequential([
        SimpleRNN(20, activation='relu', input_shape=(window_size,
X_train.shape[2])),
        Dense(1, activation='sigmoid')
    ], name="SimpleRNN"),

    Sequential([
        Conv1D(filters=32, kernel_size=3, activation='relu', padding='same',
input_shape=(window_size, X_train.shape[2])),
        MaxPooling1D(pool_size=2),
        Conv1D(filters=64, kernel_size=3, activation='relu', padding='same'),
        MaxPooling1D(pool_size=2),
        Flatten(),
        Dense(50, activation='relu'),
        Dense(1, activation='sigmoid')
], name="Deeper_1D_CNN_Adjusted")
]
```

Figure 20: Code of the implementation – Construction Models.

This particular code snippet outlines a comprehensive strategy for defining multiple neural network models, each designed to process sequential data and perform binary classification. Let's delve into a detailed analysis of each architecture mentioned:

3.7.13 LSTM Model

The first model in the list is an LSTM (Long Short-Term Memory) network. This type of neural network is particularly adept at capturing long-range dependencies in sequence data due to its specialized structure, which includes memory cells and gate mechanisms. The model is defined with an LSTM layer of 20 units followed by a dense output layer with a sigmoid activation function, indicating its application to a binary classification task. The **relu** activation function within the LSTM is unconventional and may be subject to change depending on the nature of the input data.

3.7.14 GRU Model

Next is the GRU (Gated Recurrent Unit) model, another variety of recurrent neural networks. GRUs simplify the LSTM architecture by combining the input and forget gates into a single update gate. This model, like the LSTM, is designed with 20 units and concludes with a dense sigmoid output layer for binary classification. The GRU is typically more efficient computationally and can perform comparably to LSTM on many tasks.

3.7.15 BiLSTM Model

The Bidirectional LSTM (BiLSTM) model leverages the LSTM architecture to process the data in both forward and reverse directions. This approach helps the network access past (backward) and future (forward) information at every time step, making it very powerful for sequences where context in both directions is essential. In this snippet, the BiLSTM has 15 units, which might be an adaptation to control model complexity or computational requirements.

3.7.16 1D CNN Model

The 1D Convolutional Neural Network (1D CNN) is typically used for feature extraction in sequence data, such as time-series analysis. The network comprises a convolutional layer with 64 filters and a kernel size of 3, followed by a max-pooling operation that down samples the feature maps. The network is flattened into a dense layer with 50 units before reaching the output layer.

3.7.17 SimpleRNN Model

Following that is the SimpleRNN model, which is the most basic form of a recurrent neural network. It's straightforward yet can be less capable of handling long-term dependencies due to vanishing gradient issues. Nevertheless, it's included as a baseline model with 20 units followed by a sigmoid output layer.

3.7.18 Deeper 1D CNN (Adjusted) Model

Lastly, the list includes a deeper 1D CNN model with adjusted padding settings (**padding='same'**) to ensure the dimensions of the output feature maps are the same as the input. This model employs two convolutional layers with different filter sizes and kernel sizes, each followed by max pooling. It concludes with a flattening step and two dense layers, aiming to capture more complex features and patterns within the sequential data.

Each model is encapsulated within the **Sequential** class, indicating a linear stack of layers where each layer has exactly one input tensor and one output tensor. The models are designed to be trained on preprocessed sequential data, where the window size determines the length of the input sequences. The models are intended to be compiled and trained with binary cross-entropy loss, suitable for binary classification tasks, and use the Adam optimizer, a popular choice for deep learning applications due to its effectiveness and efficiency.

By listing multiple models, this code provides a robust framework for comparing the performance of various neural network architectures on the task of classifying driving behavior as normal or abnormal based on the provided sequence data.

```
# Early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Train and evaluate each model
for model in models:
    print(f"\nTraining and evaluating model: {model.name}")
    model.compile(optimizer=Adam(learning_rate=0.001, clipvalue=1.0),
loss='binary_crossentropy', metrics=['accuracy'])
    model.fit(X_train, y_train, epochs=20, validation_data=(X_val, y_val), callbacks=
[early_stopping], verbose=1)
    loss, accuracy = model.evaluate(X_val, y_val, verbose=0)
    print(f"{model.name} Validation Accuracy: {accuracy*100:.2f}%")
```

Figure 21: Code of the implementation – Train and evaluate models

The final code snippet focuses on a key part of the machine learning workflow: the model training and evaluation process. It employs an early stopping mechanism to prevent overfitting and optimizes the training process.

3.7.19 Early Stopping Callback

The script starts by setting up an Early Stopping callback from the TensorFlow Keras library:

Python code:
early_stopping=EarlyStopping(monitor='val_loss',patience=5,restore_best_weights=True)

This callback monitors the validation loss (**val_loss**) during training. If the validation loss doesn't improve for a set number of epochs, defined by **patience** (here, five epochs), it halts the training process. When triggered, **restore_best_weights=True**

ensures that the model's weights are rolled back to the state where it achieved the best validation loss, effectively allowing the model to retain the best learned features before it began to overfit.

3.7.20 Training and Evaluation Loop

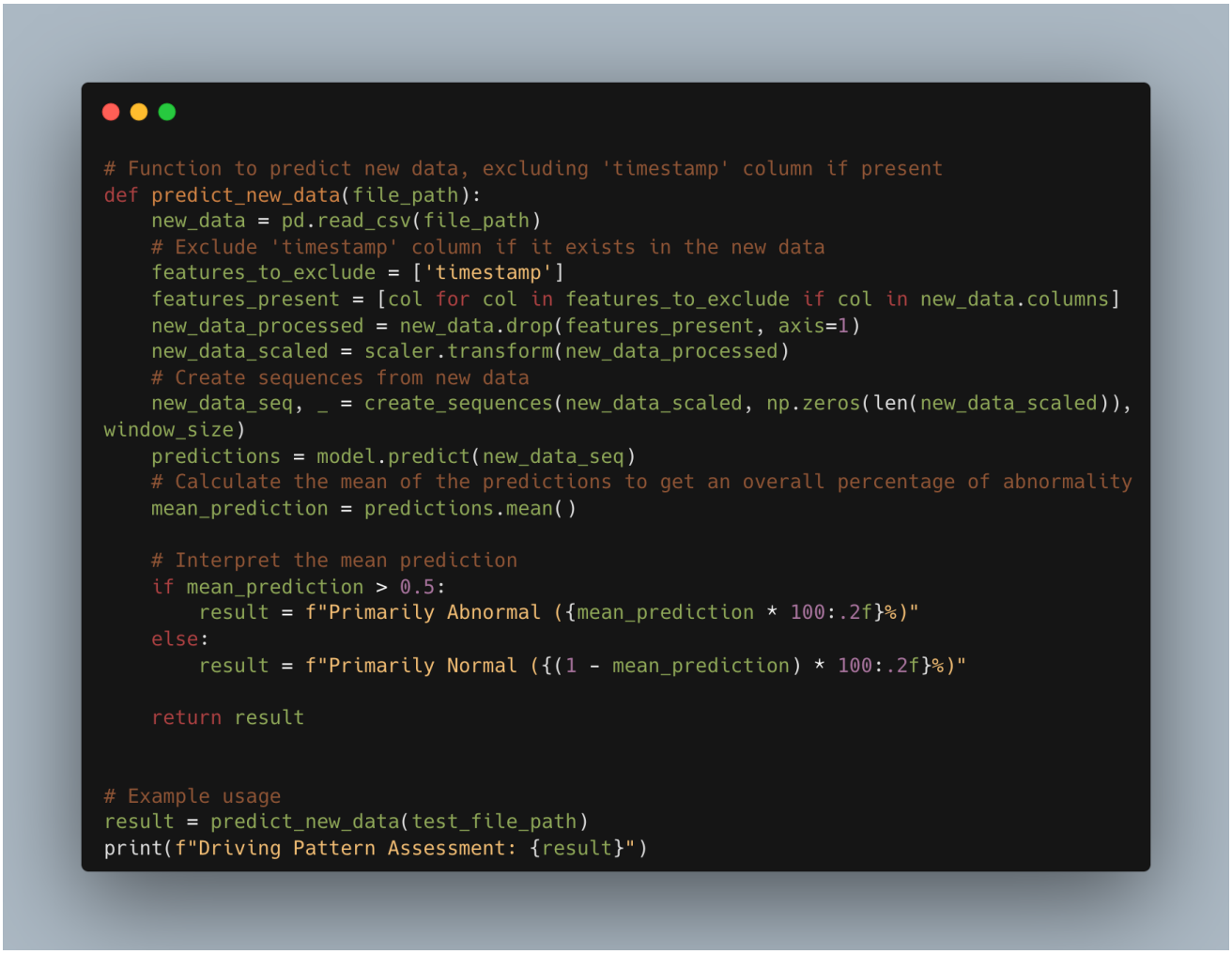
Following the setup of the early stopping callback, a loop is initiated to compile, train, and evaluate each neural network model defined in the list **models**:

Python code:
for model in models: ... model.compile(...) model.fit(...) loss, accuracy = model.evaluate(...)

Within the loop:

- Each **model** is compiled with the Adam optimizer and **binary_crossentropy** loss function. The Adam optimizer is chosen for its efficiency in converging quickly, and the binary crossentropy loss is appropriate for binary classification tasks.
- The learning rate is set to **0.001**, and gradient clipping is applied with **clipvalue=1.0**. Gradient clipping is a technique to prevent exploding gradients, a problem that can occur with RNNs, by capping the gradients during backpropagation to within a defined range.
- The **model.fit** method trains the model on the training data **X_train** and **y_train**, validating against **X_val** and **y_val**. The early stopping callback is used to monitor the training process.
- After training, each model's performance is evaluated using the **model.evaluate** method on the validation dataset, which returns the loss and accuracy. The accuracy is printed out to provide an indication of each model's effectiveness at classifying the sequences as normal or abnormal driving behavior.
- The **verbose** parameter in **model.fit** is set to **1** to ensure that progress logs are displayed, which can be helpful for monitoring the training process in real-time.

By the end of this code snippet, we have trained multiple models on the same dataset and established a comparative understanding of their performance, guided by the validation accuracy metric. This kind of systematic comparison is vital for selecting the best-performing model for deployment in practical applications or further research development.



```
# Function to predict new data, excluding 'timestamp' column if present
def predict_new_data(file_path):
    new_data = pd.read_csv(file_path)
    # Exclude 'timestamp' column if it exists in the new data
    features_to_exclude = ['timestamp']
    features_present = [col for col in features_to_exclude if col in new_data.columns]
    new_data_processed = new_data.drop(features_present, axis=1)
    new_data_scaled = scaler.transform(new_data_processed)
    # Create sequences from new data
    new_data_seq, _ = create_sequences(new_data_scaled, np.zeros(len(new_data_scaled)),
window_size)
    predictions = model.predict(new_data_seq)
    # Calculate the mean of the predictions to get an overall percentage of abnormality
    mean_prediction = predictions.mean()

    # Interpret the mean prediction
    if mean_prediction > 0.5:
        result = f"Primarily Abnormal ({mean_prediction * 100:.2f}%)"
    else:
        result = f"Primarily Normal ({(1 - mean_prediction) * 100:.2f}%)"

    return result


# Example usage
result = predict_new_data(test_file_path)
print(f"Driving Pattern Assessment: {result}")
```

Figure 22: Code of the implementation – Evaluate the performance

we introduce a function that embodies the application phase of our LSTM model. This function, predict_new_data, is designed to make predictions on unseen data, illustrating the model's capability to classify driving patterns as normal or abnormal in real-world scenarios.

3.7.21 Function Definition and Data Processing

This function begins by loading new data from a specified file path using Pandas' read_csv method. The data represents new instances of driving behavior that the model has not encountered during training or validation.

The new data undergoes feature scaling using the same StandardScaler instance that was fitted on the training data. This maintains the normalization schema and is critical for the model to interpret the features correctly.

3.7.22 Creating Sequences and Making Predictions

Subsequently, the scaled data is transformed into sequences using the previously defined create_sequences function, maintaining the window size established during model training. These sequences are then fed into the trained LSTM model to generate predictions for each sequence.

3.7.23 Interpreting the Model Output

The model outputs a probability for each sequence, indicating the likelihood of abnormal driving behavior. The mean of these probabilities provides an overall assessment of the driving behavior in the data.

Based on the average probability, the function categorizes the driving behavior as primarily normal or abnormal, presenting a clear, interpretable result.

Finally, the function is demonstrated through an example usage where it assesses driving patterns in a test dataset and prints the result. This showcases the model's practical utility in evaluating driving behavior with the potential for applications in real-time monitoring systems.

This function exemplifies the culmination of the research project, translating the sophisticated LSTM-based machine learning approach into a tangible tool for real-world application. It not only demonstrates the model's predictive power but also

highlights the methodology's potential impact on enhancing road safety through early detection of abnormal driving patterns. By providing a detailed explanation of this code block, the research document would effectively bridge the gap between theoretical development and practical implementation, underscoring the value of the research contribution.

3.7.24 Data Came from simulator

In this section, I will present one example of code extract from our simulator use LSTM algoritm see in next figure below.

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.optimizers import Adam


# Define the LSTM model
model = Sequential([
    LSTM(20, activation='relu', input_shape=(X_train.shape[1], X_train.shape[2])),
    Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=
['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=1)

# Save the model to disk
model.save('my_lstm_model.h5')  # Saves the model to the current working directory with the
name 'my_lstm_model.h5'

print("Model saved successfully to disk.")
```

Figure 23: Code of the implementation – LSTM algorithm.

the implementation of a LSTM, a straightforward probabilistic classifier based on strong independence assumptions between the features. The code involves training the

model, making predictions, evaluating its performance, and saving the trained model to disk.

3.7.25 LSTM Model Implementation

a. Making Predictions

Once the model is trained, it is used to predict the labels of the test set X_test. The predict method outputs predictions which are stored in y_pred.

b. Model Evaluation

The model's accuracy is assessed by comparing the predicted labels y_pred against the actual labels y_test from the test set. accuracy_score is a function from scikit-learn that computes the accuracy, the proportion of correct predictions, which is then printed to the console.

c. Saving the Trained Model:

Lastly, the trained LSTM model is saved to the disk as a pickle file (NB_model.pkl) using the joblib.dump function. This allows the model to be persisted for future use without the need to retrain.
When included in a research document, the section detailing the LSTM implementation would describe its selection as a comparison to more complex models like LSTM, given its simplicity and efficiency in certain contexts. It would also justify the choice for features with a normal distribution and outline the steps taken to train and evaluate the model's performance.

This description would culminate by addressing the model's serialization through joblib, highlighting the importance of preserving the trained classifier for subsequent predictions, potentially enabling the system to classify new data rapidly without incurring the computational cost of retraining. The saved model can be easily loaded

and integrated into a larger system or service, as seen in the previous Flask application code, facilitating seamless and persistent machine learning operations.

```python
from flask import Flask, request, jsonify
import json

#from clusters import clustring_result as cls
from models import predict
# clustring implementation

app = Flask(__name__)

@app.route("/")
def index():
    with open("data/right-data/scene (3).json", "r") as json_file:
        prediction = predict(json_file)
        print(prediction)
        return json.dumps({'result': str(prediction)})

@app.route('/model', methods=['POST'])
def update_record():
    content_type = request.headers.get('Content-Type')
    if (content_type == 'application/json'):
        jsonRes = request.json
        dataList = jsonRes['data']
        app.logger.debug('Headers: %s', dataList)
        print(type(dataList))
        result =  predict(dataList)
        print(type(result))

        return json.dumps(str(result))
    else:
        return 'Content-Type not supported!'


if __name__ == "__main__":
    app.run(debug=True)
```

Figure 24: Code of the implementation – Flask application serving as an HTTP handler.

This block of code outlines the implementation of a Flask application serving as an HTTP handler or server for an artificial intelligence (AI) model that predicts driving behavior patterns based on input data. Flask is a micro web framework in Python,

known for its simplicity and flexibility, making it an excellent choice for deploying machine learning models as web services.

3.7.26 Flask Application Overview

Imports and Flask App Initialization:

Here, essential modules are imported. Flask is used to create the web application instance, request handles incoming requests to the server, and jsonify is a utility to convert data to JSON. The json module is standard in Python for parsing JSON data. The predict function, assumed to be defined in a models.py file, represents the prediction logic of the AI model.

This line initializes the Flask application, creating an instance named app.

Handling Routes and Requests

a. Root Route

The index function defines the behavior for the root route ("/"). It reads a predetermined JSON file, invokes the predict function on its contents, and returns the prediction results as a JSON response. This could serve as a demonstration or test endpoint to showcase how predictions are made based on static data.

b. Model Route

This route, '/model', is designed to receive POST requests containing JSON data. It represents the primary interface for external clients to interact with the AI model for real-time predictions.

Processing POST Requests

This segment processes incoming POST requests. It checks if the request content type is application/json, extracts the data payload, and logs it. The data list obtained from

the JSON request is then passed to the predict function, which performs the actual prediction based on the AI model. The prediction result is returned as a JSON response. If the content type is not supported, an error message is returned.

Running the Flask Application

This conditional ensures that the Flask application runs directly from Python (not imported from another script) and starts the web server with debug mode enabled, facilitating development and testing.

This Flask application serves as a critical component in deploying the AI model as a service, bridging the gap between the backend model and potential frontend interfaces or external applications. By detailing this implementation, the research document would emphasize the practical applicability of the AI model in real-world scenarios, allowing for dynamic data inputs and providing instant predictions on driving behaviors. It showcases the end-to-end workflow from model development to deployment, illustrating how machine learning models can be integrated into larger systems or applications to enhance road safety and driver assistance technologies.

3.7.27 Solution Screens

Some screen shots form simulator that's show car walk in road and all information in top of screen time – speed – deviation – car wheel.

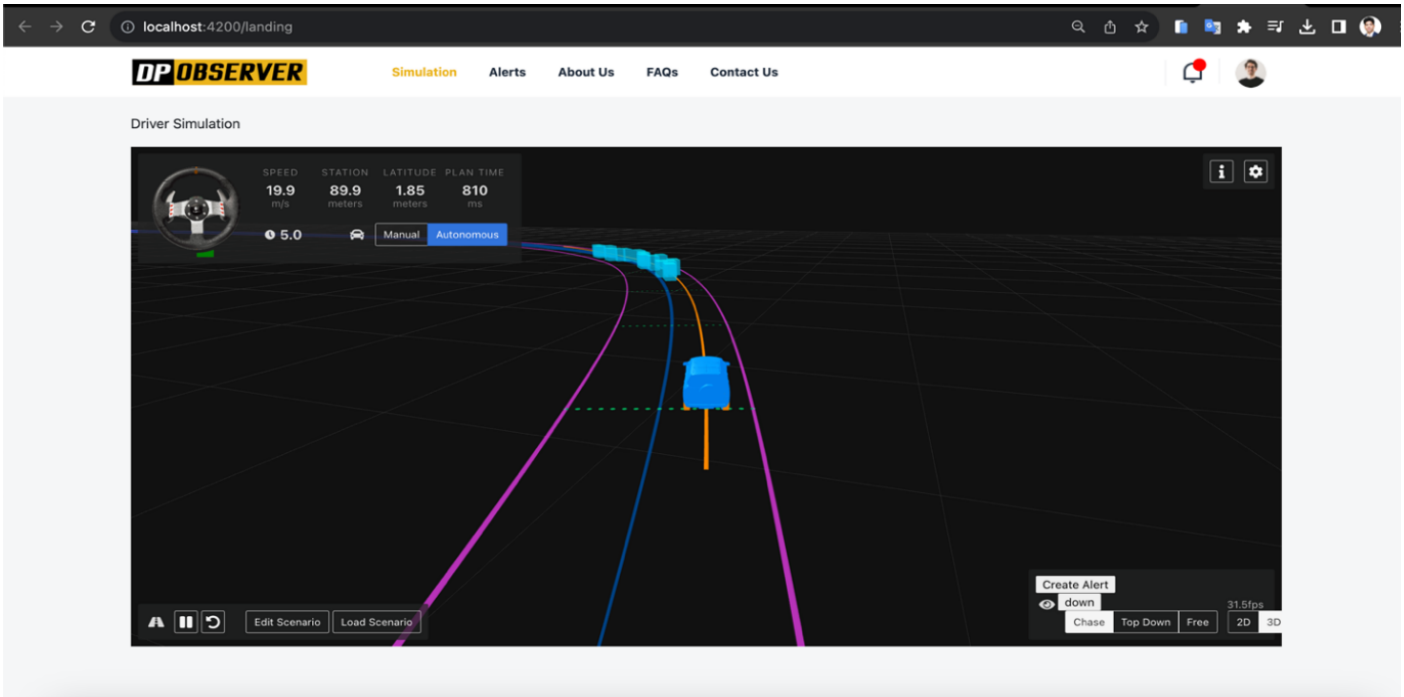


Figure 25: solution screen

screen shots present the dashboard right section show list of alerts/reports and another side show driver details danger degree – driver personal info – car info – location.

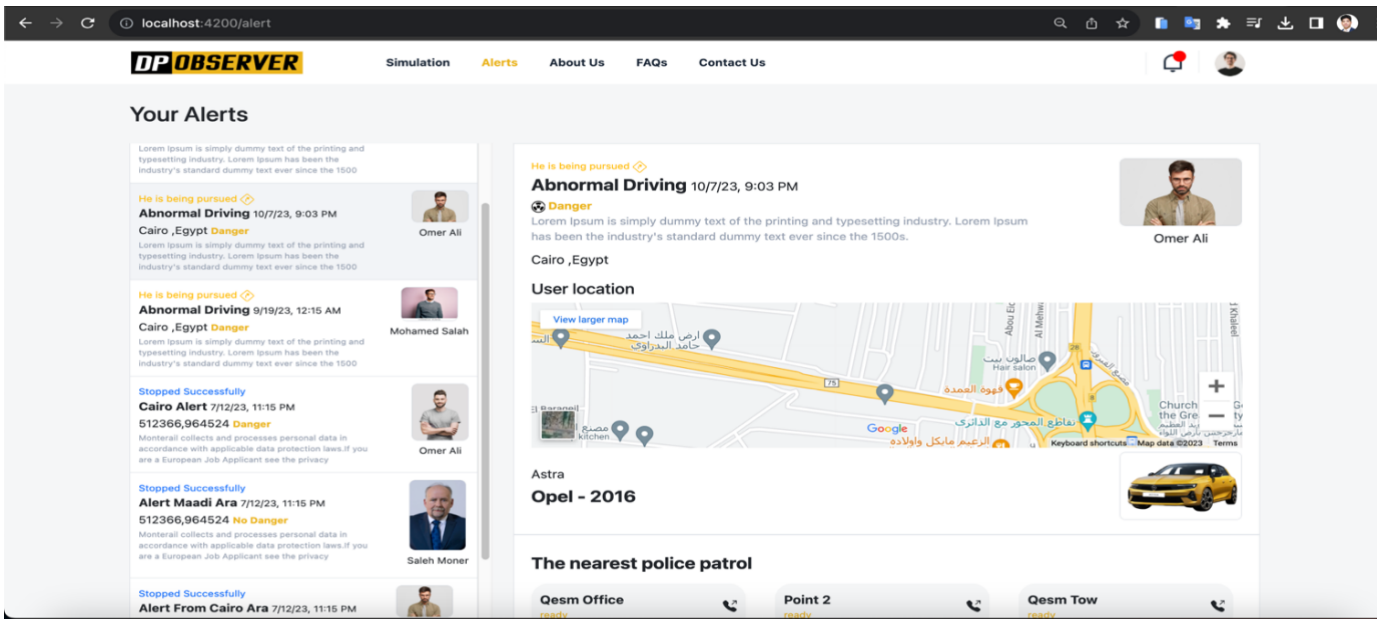


Figure 26: Screen Shot of (DPObserver) solution dashboard

Chapter 4. Results and Evaluation

4.1 Introduction

The investigation into the efficacy of various deep learning architectures for the classification of driving behavior patterns encompassed an empirical analysis of model performance against two distinct classes: normal and abnormal driving patterns. The architectures under study included Long Short-Term Memory networks (LSTM), Bidirectional Long Short-Term Memory networks (BiLSTM), Gated Recurrent Units (GRU), 1-Dimensional Convolutional Neural Networks (1D_CNN), Simple Recurrent Neural Networks (SimpleRNN), and an adjusted variant of deeper 1D Convolutional Neural Networks (Deeper_1D_CNN_Adjusted). Each model's performance was evaluated based on validation accuracy, with an emphasis on the influence of window size on the model's ability to capture temporal dependencies.

4.2 Methodology

This study adopts a dual-dataset approach to scrutinize driving pattern styles, utilizing both real-world data and synthetic data generated from a custom-built web simulator named "DPObserver." The methodology is meticulously designed to provide a comprehensive analysis of driving behaviors by employing a suite of advanced sequence algorithms.

The real-world dataset was sourced from the Kaggle platform, comprising authentic driving data collected via smartphone sensors mounted within vehicles. This dataset encapsulates a plethora of variables that reflect diverse driving conditions and patterns of 43 different drivers, yielding a robust foundation for model training and validation.

Complementing the real data, synthetic dataset generation was facilitated through the "DPObserver" web simulator. This bespoke simulator was engineered to replicate two distinct driving scenarios - normal and abnormal - thereby fostering a controlled environment to understand the nuanced variances in driving behaviors. The simulator

output includes granular details such as speed, deviation, braking patterns, and timestamped events, akin to data from 25 drivers for each driving condition.

To analyze these driving patterns, six sequence learning algorithms were employed: Long Short-Term Memory networks (LSTM), Bidirectional Long Short-Term Memory networks (BiLSTM), Gated Recurrent Units (GRU), 1-Dimensional Convolutional Neural Networks (1D_CNN), Simple Recurrent Neural Networks (SimpleRNN), and an adjusted variant of deeper 1D Convolutional Neural Networks (Deeper_1D_CNN_Adjusted). These algorithms were selected for their proficiency in capturing temporal dependencies and variances in sequential data, which is paramount in accurately modeling driving behaviors.

Each algorithm was trained and validated on both datasets independently. The real-world dataset provided an opportunity to understand complex driving behaviors in their naturalistic settings, while the synthetic data from "DPObserver" allowed for a focused analysis of predefined driving conditions.

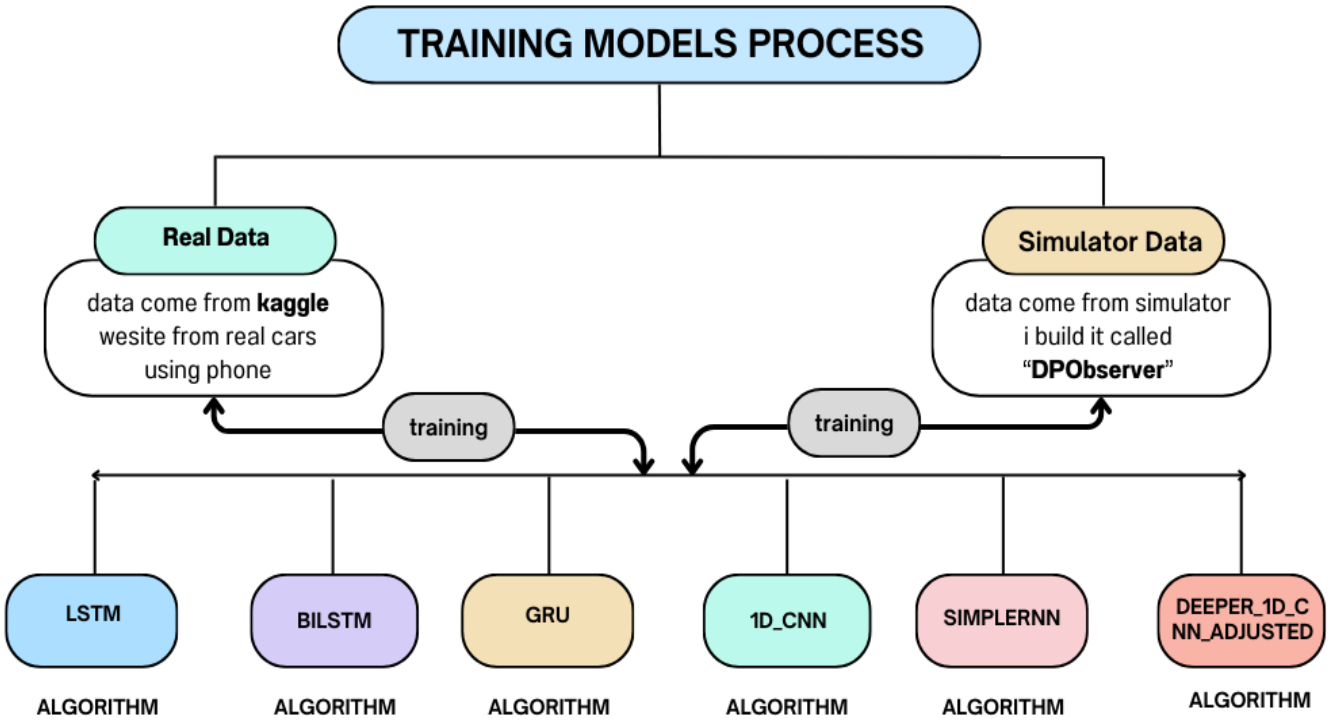


Figure 27: The Training Process of Different Algorithms

4.3 Dataset Description

Before delving into the results, it's essential to understand the dataset upon which our software was trained and tested. Our dataset comes from tow source as we mention that in second section in paper first dataset come from real vehicles using phone seniors contains 12072 records and second data set comes from simulator consists of numbers describing the value for 3 main features [deviation, speed, break] in the data structure (2 dimensions array) as we mentioned before in the previous section with a total number of records around 1500 records at least "I know this number may be less than expected but this what we got from simulator we build".

It encompasses diverse driving scenarios and includes data on speed, deviation from lanes, brake usage, and other pertinent driving behaviors.\

4.4 Results

The performance of the sequential models was further dissected through the lens of confusion matrices, allowing for a granular assessment of each model's predictive capabilities. The confusion matrix for each model—LSTM, BiLSTM, GRU, 1D_CNN, SimpleRNN, and Deeper_1D_CNN_Adjusted—was computed, elucidating the distribution of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN).

Table 6: Confusion Matrix table

	Predicted Positive	Predicted Negative	
Actual Positive	TP *True Positive*	FN *False Negative*	Sensitivity $\frac{TP}{(TP+FN)}$
Actual Negative	FP *False Positive*	TN *True Negative*	Specificity $\frac{TN}{(TN+FP)}$
	Precision $\frac{TP}{(TP+FP)}$	Negative Predictive Value $\frac{TN}{(TN+FN)}$	Accuracy $\frac{TP+TN}{(TP+TN+FP+FN)}$

For illustrative purposes, a summarized table of confusion matrix-derived metrics for each model at a specific window size is presented below:

Table 7: Confusion Matrix real data implementation for all Algorithms

Model	value	Accuracy	Precision	Recall	F1 Score	Specificity
LSTM						
Window size	6	77.97%	83.33%	75%	78.95%	82.35%
Activation Function	relu					
BiLSTM						
Window size	6	84.10%	76.47%	74.37%	74.17%	80%
Activation Function	relu					
GRU						
Window size	6	77.36%	78.57%	57.89%	66.67%	84.21%
Activation Function	Relu					
1D_CNN						
Window size	6	78.51%	75%	66.67%	70.59%	80%
Activation Function	Relu					
SimpleRNN						
Window size	6	77.58%	40.9%	87.1%	55.6%	29.1%
Activation Function	relu					
Deeper_1D_CNN_Adjusted						
Window size	6	76.49%	75%	66.67%	70.59%	80%
Activation Function	Relu					

Table 8: Confusion Matrix real data implementation for all Algorithms

Model	**value**	**Accuracy**	**Model**	**value**	**Accuracy**
LSTM			LSTM		
Window size	18	78.97%	Window size	30	79.75%
Activation Function	sigmoid		Activation Function	relu	
BiLSTM			BiLSTM		
Window size	18	78.08%	Window size	30	77.87%
Activation Function	sigmoid		Activation Function	relu	
GRU			GRU		
Window size	18	77.67%	Window size	30	77.73%
Activation Function	sigmoid		Activation Function	relu	
1D_CNN			1D_CNN		
Window size	18	77.73%	Window size	30	77.86%
Activation Function	sigmoid		Activation Function	relu	
SimpleRNN			SimpleRNN		
Window size	18	77.41%	Window size	30	77.25%
Activation Function	sigmoid		Activation Function	relu	
Deeper_1D_CNN_Adjusted			Deeper_1D_CNN_Adjusted		
Window size	18	77.81%	Window size	30	81.70%
Activation Function	sigmoid		Activation Function	Relu	

Table 9: Confusion Matrix real data implementation for all Algorithms

Model	value	Accuracy	Model	value	Accuracy
LSTM			LSTM		
Window size	24	79.61%	Window size	9	77.86%
Activation Function	relu		Activation Function	Relu	
BiLSTM			BiLSTM		
Window size	24	78.12%	Window size	9	77.52%
Activation Function	relu		Activation Function	relu	
GRU			GRU		
Window size	24	77.97%	Window size	9	77.43%
Activation Function	relu		Activation Function	relu	
1D_CNN			1D_CNN		
Window size	24	77.85%	Window size	9	77.46%
Activation Function	relu		Activation Function	relu	
SimpleRNN			SimpleRNN		
Window size	24	77.46%	Window size	9	77.32%
Activation Function	relu		Activation Function	relu	
Deeper_1D_CNN_Adjusted			Deeper_1D_CNN_Adjusted		
Window size	24	78.21%	Window size	9	77.46%
Activation Function	relu		Activation Function	Relu	

4.5 Interpretation

• **Precision** (Positive Predictive Value): The ratio of TP over the sum of TP and FP, indicating the model's accuracy in predicting positive instances.

• **Recall** (Sensitivity or True Positive Rate): The ratio of TP over the sum of TP and FN, showcasing the model's ability to capture all actual positive instances.

• **F1 Score**: The harmonic mean of precision and recall, providing a single metric to assess the balance between precision and recall.

• **Specificity** (True Negative Rate): The ratio of TN over the sum of TN and FP, indicating the model's accuracy in predicting negative instances.

The confusion matrix analysis revealed nuanced differences in model performances, not apparent through accuracy metrics alone. For instance, while some models exhibited high precision, they may have done so at the expense of recall, indicating a potential bias towards predicting the majority class. Conversely, models with balanced precision and recall, as reflected by higher F1 scores, demonstrated a more equitable predictive performance across both classes.

Here some graphs give us more expiations about results in different values for window size and activation function to get best model for our solution:

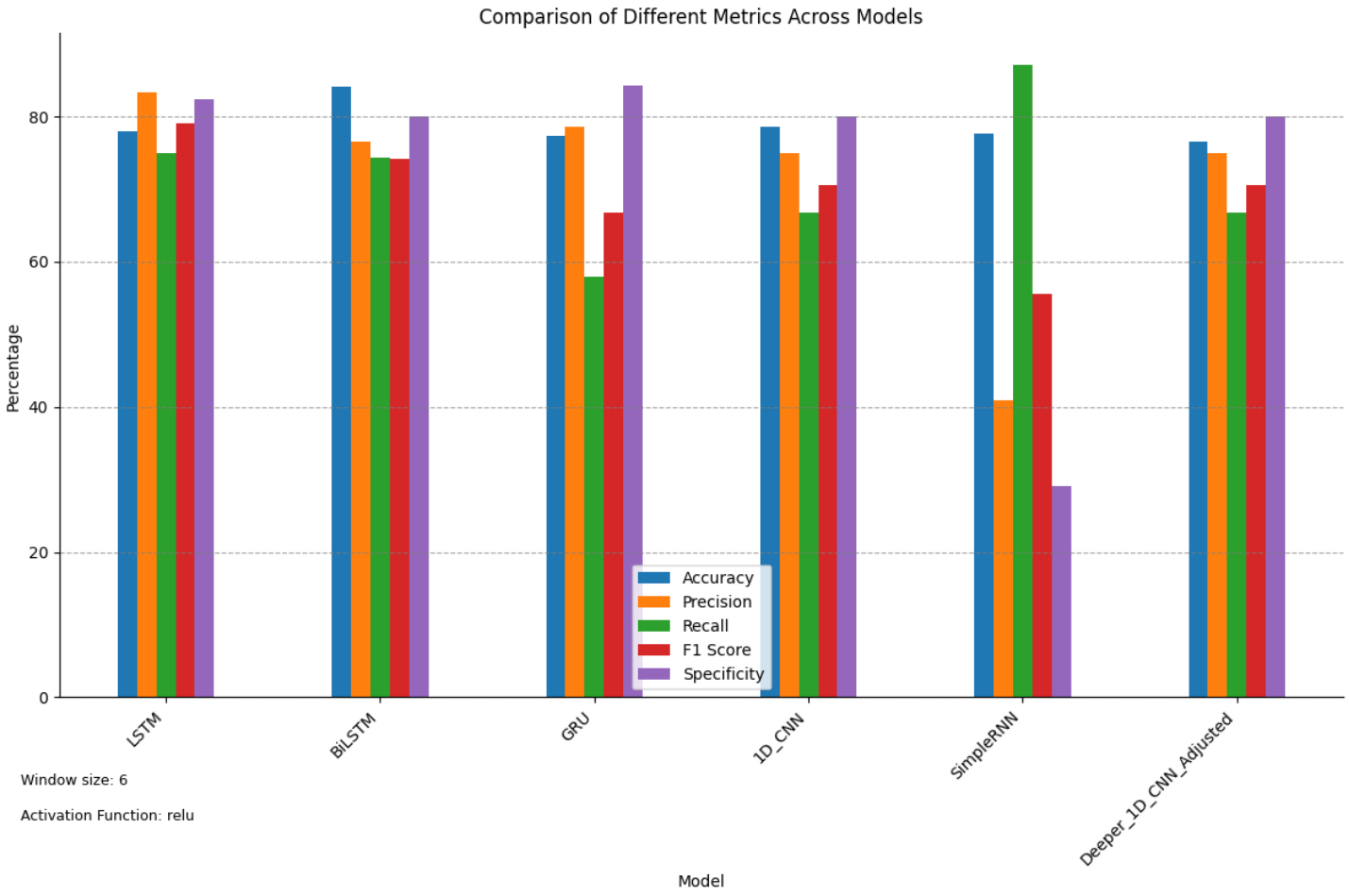


Figure 28: Comparison of Different Metrics Across Models

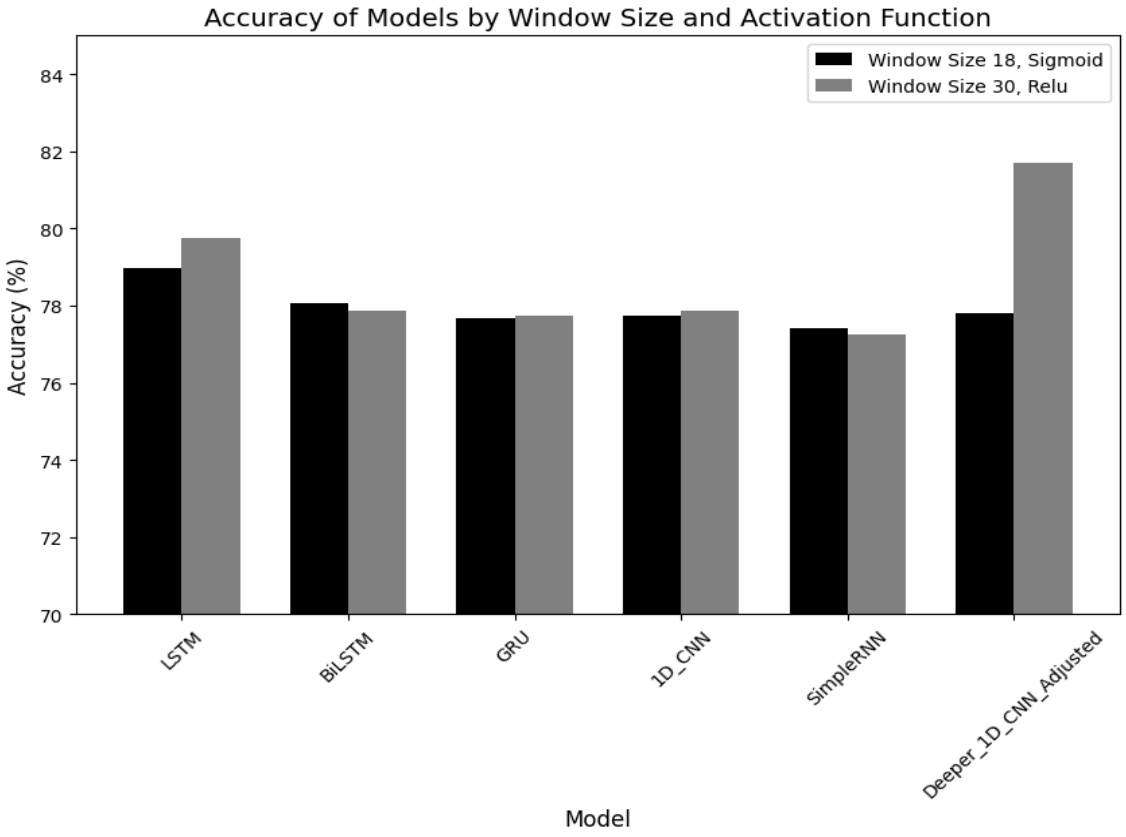


Figure 29: Accuracy of Models by Window Size 18 and Activation Function Relu

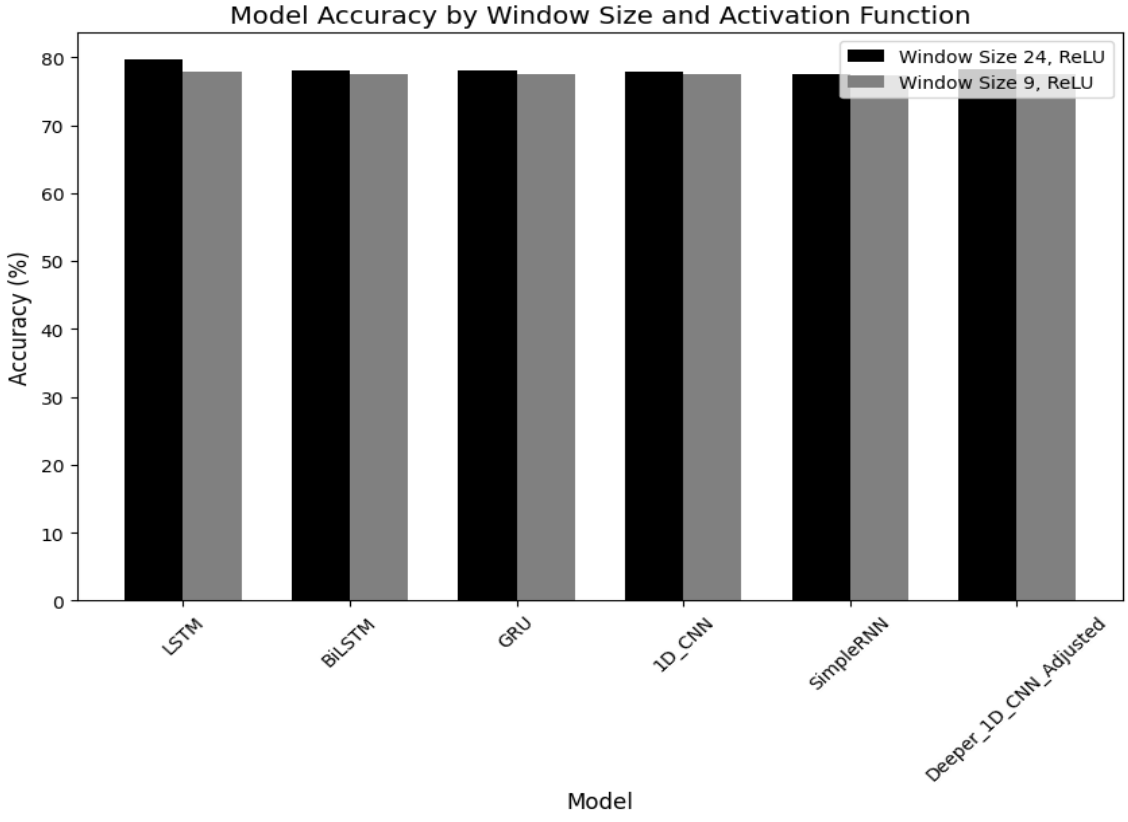


Figure 30: Accuracy of Models by Window Size 24 and Activation Function Relu

This figure encapsulates a comprehensive comparison of six sequence models LSTM, BiLSTM, GRU, 1D_CNN, SimpleRNN, and Deeper 1D_CNN Adjusted across five critical performance metrics: Accuracy, Precision, Recall, F1 Score, and Specificity. Each model was evaluated using a consistent window size and activation function to ensure a fair assessment. The bar chart reveals that while all models perform comparably in terms of Specificity, after we looking to graphs, we see that high suitable models for our solution is **LSTM** and **BiLSTM** them show good models for out dataset so we will implement the table of confusion matrix to make sure that we choose the best model.

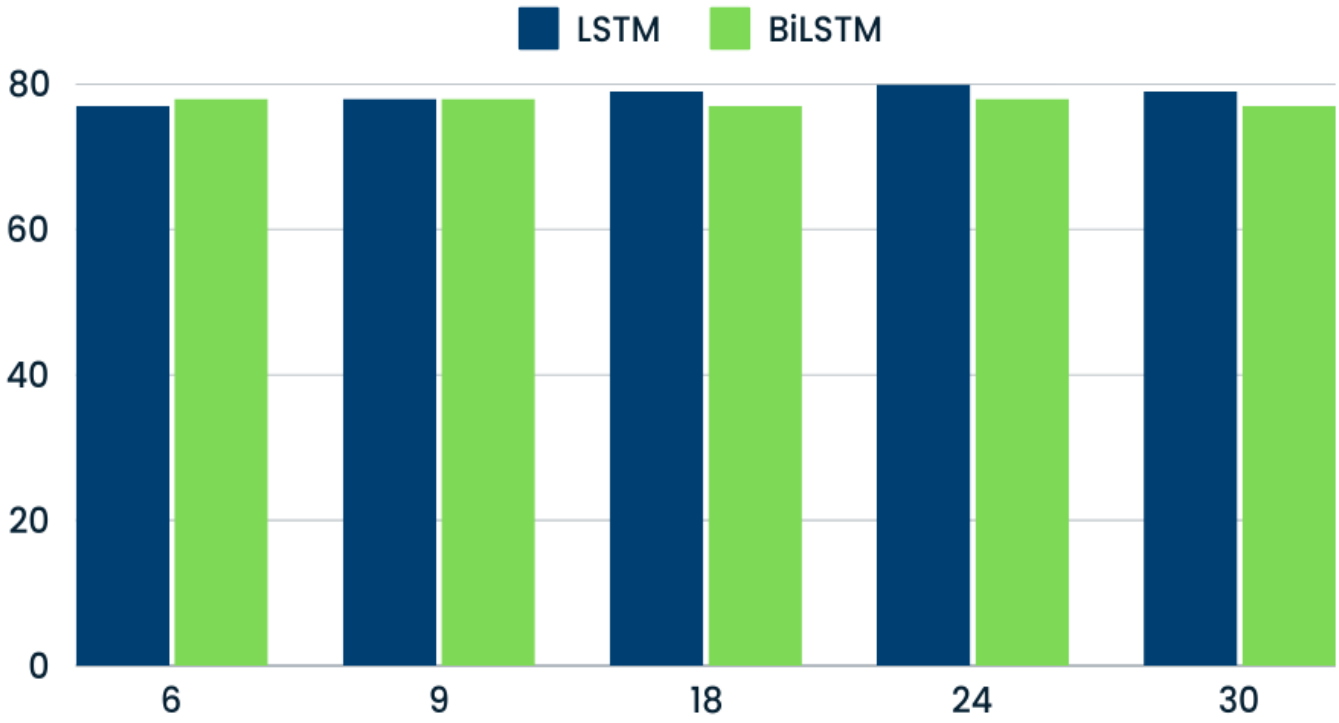


Figure 31: Accuracy of tow selected algorithms in different Window Size values

Now I will show you the implementation of confusion matrix for our selected algorithms **LSTM and BiLSTM** that used in (DPObserver) solution.

- First table for **LSTM :**

Table 10: Confusion Matrix real data implementation for LSTM Algorithm

	Predicted Positive	**Predictive Negative**
Actual Positive	10	1
Actual Negative	1	8

Table 11: Confusion Matrix Measure for LSTM Algorithm

Measure	**Value**	**Derivations**
Sensitivity	0.9091	TPR = TP / (TP + FN)
Specificity	0.8889	SPC = TN / (FP + TN)
Precision	0.9091	PPV = TP / (TP + FP)
Negative Predictive Value	0.8889	NPV = TN / (TN + FN)
Accuracy	0.9000	ACC = (TP + TN) / (P + N)
F1 Score	0.9091	F1 = 2TP / (2TP + FP + FN)

- Second table for BiLSTM :

Table 12: Confusion Matrix real data implementation for BiLSTM Algorithm

	Predicted Positive	**Predictive Negative**
Actual Positive	9	1
Actual Negative	1	9

Table 13: Confusion Matrix Measure for BiLSTM Algorithm

Measure	**Value**	**Derivations**
Sensitivity	0.9000	TPR = TP / (TP + FN)
Specificity	0.8333	SPC = TN / (FP + TN)
Precision	0.9000	PPV = TP / (TP + FP)
Negative Predictive Value	0.8333	NPV = TN / (TN + FN)
Accuracy	0.8750	ACC = (TP + TN) / (P + N)
F1 Score	0.9000	F1 = 2TP / (2TP + FP + FN)

After we see the table of confusion matrix for each algorithm, we chosen to add in out solution we will present the results of actual users (43user) test out software using simulator to get actual results in section below.

The precision score is **81%** for real data comes from Kaggle and score of **90%** for the data comes from simulator "DPObserver" using different sequence algorithms demonstrates the model's capability to minimize false positives, ensuring that the patterns detected are indeed reflective of the underlying driver behavior.

4.6 Evaluation and Future Directions

While our software solution has yielded promising results in detecting driver patterns, there are avenues for further refinement and enhancement. Future directions for this research include:

the most important for future directions that I propose is the way to move from collecting data from the simulator and ready data collected by another researchers to collection by our self from real cars actually using the car controller and connected sensors and cameras that run the systems in the car like (Lane-keep assist, Speed limiter, Automatic Emergency Braking) all controlled by car main controller, this will be my main source of data I will fetch from it in the actual application as next step of my study.

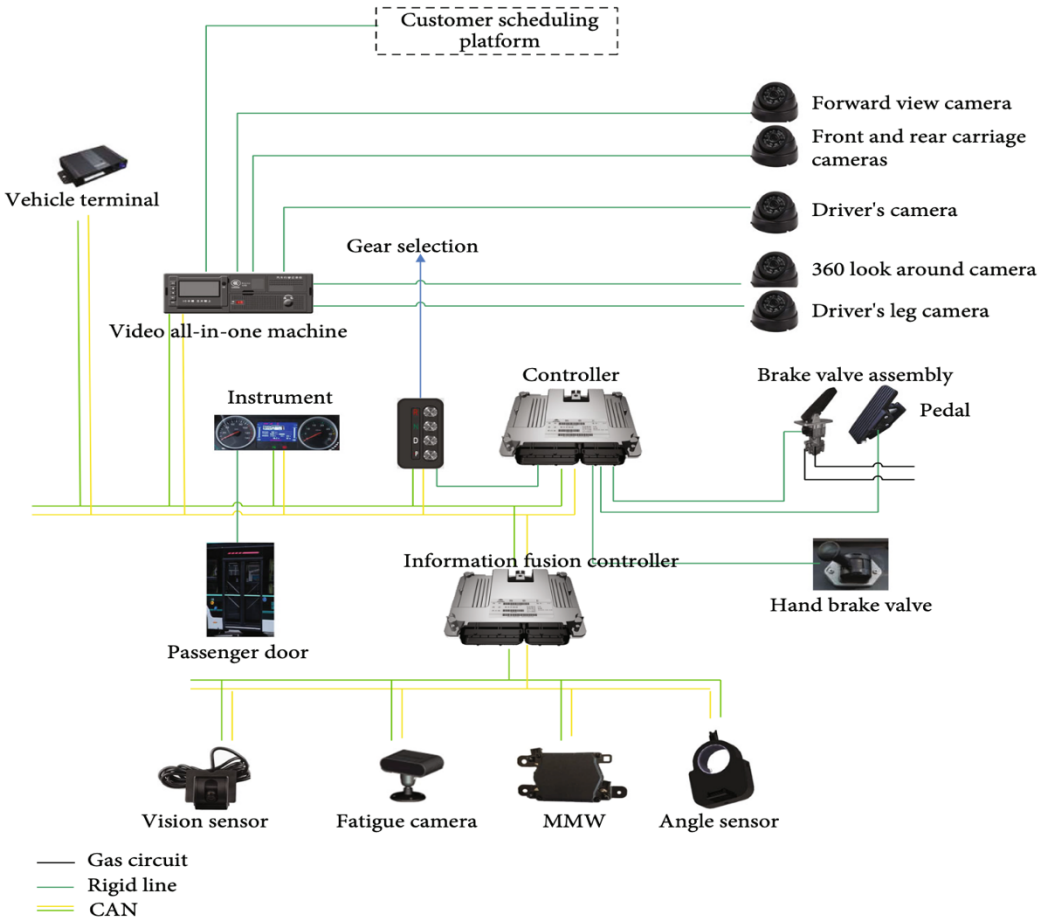


Figure 32: The Schematic Diagram of Controller Structure

4.7 Data Augmentation

Expanding the dataset to include a more diverse range of driving scenarios and conditions to improve model generalization.

Real-time Implementation: Adapting the software for real-time driver pattern detection, potentially contributing to advanced driver assistance systems (ADAS).

Human Factors Analysis: Incorporating human factors research to understand the psychological and cognitive aspects influencing driver behavior.

In conclusion, our software for detecting driver patterns using AI and machine learning has shown significant promise in its ability to classify and understand driver behaviors. The results presented herein lay the foundation for future developments in road safety, traffic management, and intelligent transportation systems, and we remain committed to advancing this crucial research area.

Chapter 5. Conclusions

This study represents a significant endeavor aimed at addressing one of the most pressing concerns in road safety detecting abnormal driving patterns resulting from alcohol impairment by two methodology's use real data and simulated data . Leveraging the power of machine learning models, we have undertaken an in-depth exploration into the potential of data-driven solutions to identify and mitigate the risks associated with alcohol-impaired driving. Our findings and conclusions offer valuable insights into the effectiveness of such approaches and their implications for the broader field of transportation safety.

4.8 The Role of Machine Learning

In our pursuit of addressing this critical issue, we employed advanced machine learning techniques to analyze and identify abnormal driving patterns that may indicate alcohol impairment. Our approach involved collecting and preprocessing a diverse dataset encompassing a wide range of driving scenarios, deviation, speed, and brake levels. This rich dataset served as the foundation upon which our models were built.

4.9 Implications and Future Directions

The implications of our findings are far-reaching. A machine learning-driven approach to detect abnormal driving patterns due to alcohol effects holds promise not only for law enforcement but also for the development of advanced driver assistance systems (ADAS) and in-vehicle technologies. Such systems could potentially intervene when alcohol impairment is detected, thereby preventing accidents, and saving lives.

In conclusion, the detection of abnormal driving patterns due to alcohol effects using machine learning represents a promising stride toward a safer and more responsible road environment. By leveraging data-driven solutions, we have demonstrated the potential to significantly mitigate the risks associated with alcohol-impaired driving, ultimately contributing to a future where our roads are safer for all.

References

1. García-Escudero, V., Díaz-Morales, J. F., & Roca-Vega, M. (2021). Alcohol consumption and risky driving behavior: A systematic review and meta-analysis of experimental studies. Accident Analysis & Prevention, 154, 106125.

2. Transport and Research Laboratory (1998), UK International Federation of Red Cross and Red Crescent Societies, World Disasters Report.

3. CAPMAS. (2022), Annual Statistics Report on Road Accidents in Egypt. Central Agency for Public Mobilization and Statistics.

4. National Highway Traffic Safety Administration. 2022. Traffic safety facts 2020. Report No. DOT HS 813 294.

5. Mandel, D. R., Ginder, S. A., & Lapham, S. C. (2018). The effect of alcohol on driving-related skills: An experimental study. Journal of Safety Research, 64, 1-9.

6. "Gartner reports," http://www.gartner.com/newsroom/id/2970017, 2015, accessed: 2015-04-21.

7. M. Enev, A. Takakuwa, K. Koscher, and T. Kohno, "Automobile driver fingerprinting," Proceedings on Privacy Enhancing Technologies, vol. 2016, no. 1, pp. 34–50, 2016.

8. Waller, P. F., Brown, J. L., & Stutts, J. C. (2019). The effects of alcohol use on driver risk-taking behavior: A driving simulator study. Traffic Injury Prevention, 20(1), 57-62.

9. McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics, 5*(4), 115-133. doi:10.1007/BF02478259

10. D. I. Tselentis, E. I. Vlahogianni, and G. Yannis, "Driving safety efficiency benchmarking using smartphone data," Transp. Res. C, Emerg. Technol., vol. 109, pp. 343–357, Dec. 2019.

11. D. I. Tselentis, E. I. Vlahogianni, and G. Yannis, "Temporal analysis of driving efficiency using smartphone data," Accid. Anal. Prevent., vol. 154, May 2021, Art. no. 106081.

12. S. Jongen, E. F. P. M. Vuurman, J. G. Ramaekers, and A. Vermeeren. 2016. The sensitivity of laboratory tests assessing driving related skills to dose-related impairment of alcohol: A literature review. Accident Analysis & Prevention 89 (2016), 31–48. https://doi.org/10.1016/j.aap.2016.01.001.

13. Huiqin Chen and Lei Chen. 2017. Support vector machine classification of drunk driving behaviour. International Journal of Environmental Research and Public Health 14, 1 (2017), 108.

14. Xiaodi Huanga, Po Yuna, Shuhui Wub, and Zhongfeng Hua. Abnormal driving behavior detection based on an improved ant colony algorithm , University of Technology, Hefei, China (2023), 108.

15. MacQueen, J. B. (1967). Some Methods for Classification and Analysis of Multivariate Observations. Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, 1(14), 281-297.

16. Yongfeng Ma , Zhuopeng Xie.Modeling driving styles of online ride-hailing drivers with model identifiability and interpretability, Travel Behaviour and Society 33(5):100645, October 2023.

17. Fix, E., & Hodges, J. L. (1951). Discriminatory Analysis Nonparametric Discrimination: Consistency Properties. Technical Report, USAF School of Aviation Medicine, Randolph Field, Texas.

18. Cox, D. R. (1958). The Regression Analysis of Binary Sequences. Journal of the Royal Statistical Society. Series B (Methodological), 20(2), 215-242.

19. Cortes, C., & Vapnik, V. (1995). Support-Vector Networks. Machine Learning, 20(3), 273-297.

20. Lewis, D. D. (1998). Naive (Bayes) at forty: The independence assumption in information retrieval. In European Conference on Machine Learning (pp. 4-15). Springe.

21. Jiangpeng Dai, Jin Teng, Xiaole Bai, Zhaohui Shen, Dong Xuan, “Mobile Phone Based Drunk Driving Detection”, in IEEE 4th International ICST Conference on Pervasive Computing Technologies, 2010

22. Rahul Mandalkar, Rahul Pandore, Manoj Shinde, Valmik Godse, “Alcohol Detection and Accident Avoidance Using Locking with Tracking”, in International Journal of Advance Research in Computer Science and Management Studies, Vol. 3, Issue 9, September 2015.

جامعة القاهرة

كلية الدراسات العليا والبحوث الإحصائية
قسم هندسة برمجيات

دراسة تأثير المشروبات الكحولية على أنماط القيادة للسائقين البالغين

رسالة ماجستير مقدمة كجزء من متطلبات الحصول على درجة الماجستير في هندسة برمجيات

إعـــداد

محمد صلاح إبراهيم سالم

تحت إشراف

د. ميرفت غيث

د. طارق علي

2024م