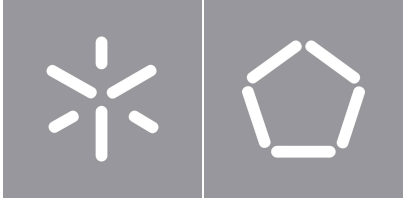
Universidade do Minho
Escola de Engenharia

Bruno Daniel Mestre Viana Ribeiro

Enhancing Road Safety for VRUs: Automatic Collision Prediction using V2X Data

May 2024

Enhancing Road Safety for VRUs: Automatic Collision Prediction using V2X Data
Bruno Ribeiro
UMinho | 2024



University of Minho
School of Engineering

Bruno Daniel Mestre Viana Ribeiro

Enhancing Road Safety for VRUs: Automatic Collision Prediction using V2X Data

PhD Thesis
PhD in Informatics

Thesis supervised by
Professor Alexandre Santos
Professor Maria João Nicolau

May 2024

Copyright and Terms of Use for Third Party Work

License granted to users of this work:

Acknowledgements

This research work would not have been possible without the help of many people, for their support and valuable contributions. I would like to express my appreciation and sincere thanks to my supervisors from University of Minho, Professor Alexandre Santos and Professor Maria João Nicolau. I am very grateful for their great effort in giving me useful advices and guidance throughout the development of my thesis. I would also like to thank my parents and remaining family for their inexhaustible encouragement throughout my life. Finally, I must express my gratitude to Raquel for her immense patience and support during the most stressful times.

This work is supported by: European Structural and Investment Funds in the FEDER component, through the Operational Competitiveness and Internationalization Programme (COMPETE 2020) [Project nº 179491; Funding Reference: SIFN-01-9999-FN-179491].

Statement of Integrity

I hereby declare having conducted this academic work with integrity.

I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

University of Minho, Braga, May 2024

Bruno Daniel Mestre Viana Ribeiro

Abstract

Intelligent Transportation Systems (ITS) are systems that consist on a complex set of technologies that are applied to road agents, aiming to provide a more efficient and safe usage of the roads. The safety aspect of **ITS** is particularly important for **Vulnerable Road Users (VRUs)**, as they are typically more exposed to dangerous situations. The fact that these users typically possess poorer safety mechanisms (comparing to regular vehicles), together with their high agility and hard to anticipate behavior, makes the implementation of automatic safety systems challenging. However, if road entities are equipped with communication technologies, they are able to frequently disseminate high amounts of ego and environmental traffic information. That huge collection of **Vehicle to Anything (V2X)** data can be leveraged by **Machine Learning (ML)** mechanisms to implement such automatic safety systems for **VRUs**.

Hence, this Ph.D. work aims to evaluate the feasibility of leveraging **V2X** communications data when using **ML** to predict collisions related to **VRUs** (in particular, motorcycles). This thesis presents a system architecture (based on *Fog Computing*) that gives support to the heavy requirements that are required by the system - particularly regarding computation resources and large amounts of storage that the **ML** models require for training and testing. Two different safety systems are presented and evaluated: one for the detection of collisions and one for the prediction of future collisions. The datasets that were used for the **ML** models training, validation and testing were synthesized from simulation scenarios based on **European Telecommunications Standards Institute (ETSI)** standards (implemented using the **Vehicles in Network Simulation (VEINS)** framework). A simulation prototype was also developed to evaluate the collision prediction system (running the models *on-line* on the simulation framework). The results indicate that the proposed systems are able to detect and predict the collisions accurately. Furthermore, regarding the collision prediction system, most predictions are also achieved in a timely manner - as predictions are made early enough, the vehicles have enough time to receive warning messages and perform emergency brakes, thus being able to prevent the collisions.

Keywords Intelligent Transportation Systems, Vulnerable Road Users, V2X, Machine Learning

Resumo

ITS são sistemas que consistem num conjunto complexo de tecnologias que são aplicados aos agentes rodoviários, com o objetivo de proporcionar um uso mais eficiente e seguro das vias. O aspeto da segurança em **ITS** é particularmente importante para os **VRUs**, pois estes estão tipicamente mais expostos a situações perigosas. O facto desses utilizadores possuírem normalmente sistemas de segurança menos sofisticados (quando comparados com os veículos normais), juntamente com a sua elevada agilidade e comportamento difícil de antecipar, torna a implementação de sistemas automáticos de segurança num desafio. No entanto, se as entidades rodoviárias estiverem equipadas com tecnologias de comunicações, elas podem disseminar com frequência grandes quantidades de informação individual e também ambiental. Essa grande quantidade de dados **V2X** pode ser aproveitada por mecanismos de **ML** para implementar tais sistemas automáticos de segurança para **VRUs**.

Assim, este trabalho de doutoramento visa avaliar a viabilidade de aproveitar os dados de comunicação **V2X** em técnicas de **ML** para prever colisões relacionadas com **VRUs** (em particular, motociclos). Esta tese apresenta uma arquitetura baseada em *Fog Computing*, que dá suporte aos requisitos pesados que são exigidos pelo sistema - particularmente em relação aos recursos de computação e à grande capacidade de armazenamento que os modelos de **ML** exigem para treino e teste. São propostos e avaliados dois sistemas de segurança diferentes: um para o deteção de colisões e outro para previsão de colisões futuras. Os conjuntos de dados que foram usados para o treino, validação e teste dos modelos de **ML** foram sintetizados a partir de cenários de simulação baseados em standards da **ETSI** (implementados usando a framework de simulação **VEINS**). Foi também desenvolvido um protótipo de simulação para avaliar o sistema de previsão de colisões, que executa os modelos *on-line* diretamente na framework de simulação. Os resultados indicam que os sistemas são capazes de detetar e prever as colisões com precisão. Além disso, em relação ao sistema de previsão de colisões, a maioria das previsões é realizada em tempo útil - como as previsões são feitas com antecedência suficiente, os veículos têm tempo suficiente para receber mensagens de alerta e realizar travagens de emergência, podendo assim evitar as colisões.

Palavras-chave Intelligent Transportation Systems, Vulnerable Road Users, V2X, Machine Learning

Contents

List of Figures

List of Tables

Acronyms

AMT Alternating Model Tree.

ANN Artificial Neural Networks.

API Application Programming Interface.

APT Average Prediction Time.

ARIB Association of Radio Industries and Businesses.

BN Bayesian Network.

BSM Basic Safety Messages.

BSS Basic Service Set.

BTP Basic Transport Protocol.

CAM Cooperative Awareness Message.

CanuMobiSim CANU Mobility Simulation Environment.

CDP Correct Decision Percentage.

CEN European Committee for Standardization.

CPP Collision Prediction Percentage.

CPU Central Processing Unit.

CSV Comma Separated Value.

DBN Deep Belief Network.

DENM Decentralized Environmental Notification Message.

DRNN Deep Recurrent Neural Network.

DSRC Dedicated Short-Range Communications.

DT Decision Tree.

ENN Elman Recurrent Neural Network.

EPD EtherType Protocol Discrimination.

ETSI European Telecommunications Standards Institute.

FN False Negative.

FNN Feedforward Neural Network.

FP False Positive.

GNB Gaussian Naive Bayes.

GPS Global Positioning System.

GPU Graphics Processing Unit.

GRU Gated Recurrent Units.

GTNetS Georgia Tech Network Simulator.

GUI Graphical User Interface.

HMI Human-Machine Interface.

IDE Integrated Development Environment.

IEEE Institute of Electrical and Electronics Engineers.

IMM Interacting Multiple Model.

IoV Internet of Vehicles.

IP Internet Protocol.

ISO International Organization for Standardization.

iTETRIS Integrated Wireless and Traffic Platform for Real-Time Road Traffic Management Solutions.

ITS Intelligent Transportation Systems.

JiST Java in Simulation Time.

KNN k-Nearest Neighbor.

LDM Local Dynamic Map.

LLC Logical Link Control.

LOS Line of Sight.

LR Logistic Regression.

LSTM Long Short-Term Memory.

LTE Long-Term Evolution.

LTE-A Long Term Evolution - Advanced.

LTE-V Long Term Evolution - Vehicle.

MAC Medium Access Control.

MEC Multi-access Edge Computing.

MIB Management Information Base.

MiXiM Mixed Simulator.

ML Machine Learning.

MLP Multilayer Perceptron.

MOVE MObility model generator for VEhicular networks.

NB Naive Bayes.

NED Network Description.

NN Nearest Neighbour.

ns-2 Network Simulator 2.

ns-3 Network Simulator 3.

OBU On-Board Unit.

OFDM Orthogonal Frequency Division Multiplexing.

OID Object Identifier.

OMNeT++ Objective Modular Network Testbed in C++.

OS Operating System.

OSM OpenStreetMap.

RAM Random-access Memory.

RF Random Forest.

RFE Recursive Feature Elimination.

RIPPER2 Repeated Incremental Pruning to Produce Error Reduction 2.

RNN Recurrent Neural Network.

RSS Received Signal Strength.

RSU Roadside Unit.

RT Regression Tree.

SAE Society of Automotive Engineers.

SDN Software-Defined Networking.

SFS Sequential Feature Selection.

sLDA supervised Latent Dirichlet Allocation.

SMO Sequential Minimal Optimization.

SUMO Simulation of Urban MObility.

SVM Support Vector Machines.

SWANS Scalable Wireless Ad-Hoc Network Simulator.

TCP Transmission Control Protocol.

TraCI Traffic Control Interface.

TSIS-CORSIM Traffic Software Integrated System - Corridor Simulation.

TTC Time to Collision.

V2I Vehicle to Infrastructure.

V2P Vehicle to Pedestrian.

V2V Vehicle to Vehicle.

V2X Vehicle to Anything.

VANETs Vehicular Ad Hoc Networks.

VEINS Vehicles in Network Simulation.

VISSIM Verkehr In Stadten SIMulationsmodell.

VLC Visible Light Communication.

VM Virtual Machine.

VRU Vulnerable Road User.

VSimRTI V2X Simulation Runtime Infrastructure.

WAVE Wireless Access in Vehicular Environments.

WLAN Wireless Local Area Network.

WSA WAVE Service Advertisement.

WSL Windows Subsystem for Linux.

WSM WAVE Short Message.

WSMP WAVE Short Message Protocol.

Chapter 1

Introduction

Intelligent Transportation Systems (ITS) are systems that consist on an intricate set of technologies applied to road agents (e.g. *vehicles, pedestrians, infrastructures*) that aim to provide a more efficient and safe usage of the roads - allowing, for example, to control traffic operations or influence drivers behavior. Furthermore, **ITS** has the potential to improve aspects such as productivity or even levels of pollution. These systems enable the implementation of several applications that, relying on information that is exchanged between the road agents, allow entities (e.g. drivers or controllers) to make smarter choices - either manually or automatically. These applications can range from simple *day-1* use cases (e.g. *Emergency Vehicle Warning*) to more advanced solutions, such as *Advanced Collisions Prediction Systems*.

The improvement of road safety is particularly important for **Vulnerable Road Users (VRUs)**, since (as the name suggests) they are the most vulnerable road agents existent and have a high casualty rate. The **VRUs** group, as defined by the European Commission [1], consists of *pedestrians, cyclists, motor-cyclists and persons with disabilities or reduced mobility and orientation*. Typically, these users do not have a protective external 'shell' and possess poorer safety mechanisms when, for instance, compared to normal passenger cars or trucks. Most **VRUs** are typically very agile (e.g. pedestrians, motorcyclists) and their movement and behavior is hard to anticipate (sometimes not even in compliance with traffic rules).

In order to protect these users, there are more and more system whose goal is to improve their road safety. Passive safety methods are essentially used to reduce the severity of injuries during/after collisions involving **VRUs** (e.g. establish a call to an emergency center after a collision). Given that these safety mechanisms are normally located on regular vehicles, they should preserve both the passengers of the vehicles and the **VRUs**, who have less probability of possessing safety equipment.

On the other hand, active safety methods are, for instance, capable of predicting and avoid collisions. These systems typically resort to sensors to obtain environment knowledge (e.g. cameras, *RADAR*,

LIDAR) and activate active safety measures (such as *emergency breaking, automatic steering* or *airbag deployment*). However, this type of solutions still present poor efficiency in situations where **Line of Sight (LOS)** is non-existent (e.g. pedestrian is in a blind spot, behind a traffic signal or a large vehicle), despite the great investment and effort to overcome limitations such as rough weather, low light environments, among others.

In contrast, wireless communications between **VRUs** and vehicles, who do not possess such limitations, can have a great impact on collision prevention and in the general safety of road agents. Road agents equipped with communication capabilities are able to exchange important knowledge that can help saving lives by preventing/predicting collisions, improve traffic flow, etc. Naturally, given the increasing number of equipped devices on **ITS** environments that are exchanging such information (e.g. about the vehicles, the infrastructures, the road/traffic conditions), there is an huge amount of data that is generated with high frequency.

According to some studies, the **Vehicle to Anything (V2X)** market is expected to largely grow in the next upcoming years: *Research and Markets* [2, 3] projects the growth of global automotive **V2X** market from 2.6 billion USD in 2022 to 19.5 billion by 2028; ABIResearch [4, 5] suggests that, by 2025, more than 10 million vehicles will be capable of using short-range **V2X** communications (364 million vehicles if including cellular connectivity). An example of the integration of such technologies is the *Cadillac CTS* since 2017 [6]. Naturally, the use of these devices, together with the *IEEE WAVE/DSRC* (American) [7] and *ETSI ITS-G5* (European) [8] standard communication stacks, requires the dissemination of cooperative messages periodically: **Basic Safety Messages (BSM)** [9] and **Cooperative Awareness Messages (CAMs)** [10], respectively. These messages contain updated information on the node's direction, position, speed, acceleration, among others. Additionally, there are some other standard messages, such as the **Decentralized Environmental Notification Messages (DENMs)** from European standards, that allow the notification of events and incidents related to the environment/traffic [11].

As stated before, predicting the behavior of **VRUs** is a very difficult task. However, **Machine Learning (ML)** techniques can be used to implement more advanced systems that try to predict such actions and prevent collisions. The application of **ML** techniques on vehicular environments data has the potential to predict **VRUs** movement, detect/locate them, and even compute probabilities of collision with them. These solutions may enable the avoidance of collisions and thus also achieve a more efficient and safer traffic flow. This is particularly important, since these agents have an increased risk of injury in cases of incidents, and represent a large portion of fatalities and injuries on roads.

Naturally, this implementation is only possible if these users posses communication capabilities that

allow communication between themselves and other road agents. Even though some **VRUs** may possess communication capabilities (e.g. pedestrians using smart-phones with cellular technologies), it is unlikely that they are able to communicate with other road agents in a direct way, as vehicles are typically equipped with different technologies (such as IEEE 802.11p). Nonetheless, from the **VRUs** group, motorcycles are fairly simple to equip with communication devices that are similar to regular vehicles (using similar **On-Board Units (OBUs)**). Hence, motorcycles are the most appropriate subjects to survey the usage of automatic solutions for **VRUs** collisions prediction resorting to ML, applied to data that comes from **V2X** communications.

This thesis main goal is to evaluate the feasibility of leveraging **V2X** communications data when using **ML** mechanisms to predict collisions related to **VRUs** (in particular, motorcycles).

To achieve a concept proof, the following tasks were achieved: a system architecture was developed; simulation scenarios based on standards were established; from the simulation scenarios, several datasets were synthesized (based on **V2X** data); **ML** models were trained and tested, to predict collisions based on the simulation datasets; Finally, a simulation prototype was developed to evaluate the solution (running the models *on-line*).

The next section presents the methodology followed to fulfill the established objectives.

1.1 Methodology

This thesis intends to investigate if it is possible to predict collisions that involve **VRUs** by analyzing the data that is exchanged by road agents using **V2X** communications. This document proposes, tests, and evaluates a system's architecture that aims to improve the safety of **VRUs**. The system aims to predict motorcycle collisions on an intersection resorting to **ML** models (which leverage **V2X** data), and notify the road users with warning messages.

The methodology to achieve a final prototype for proof of concept is described next.

Study ITS characteristics **ITS** are the underlying framework for the proposed system's architecture. Hence, **ITS** should be surveyed, to identify its main characteristics in terms of communications (communication modes and technologies) and also the most important standards. This should include an analysis of standard messages and information that is typically exchanged in vehicular environments.

Survey ML solutions applied to ITS **ML** solutions have a strong potential to help to solve many open problems in the **ITS** field, as they may be able to leverage the huge amount of **V2X** data to per-

form classification and prediction of several different issues. In that sense, it is important to survey the usefulness of **ML** learning in this context - both in **ITS** generally, but also related to the detection/prediction of incidents. This study should also gather related work on incidents that are particularly related to **VRUs**.

Research and design the system's architecture As the proposed system resorts to the usage of multiple types of road agents in different hierarchical levels, a proper system's architecture should be defined. The architecture should take into consideration the different levels of requirements that are needed by the different users (e.g. in terms of computation, storage and latency).

Collect datasets related to VRU collisions and V2X data **ML** models typically require very large amounts of data to be properly trained, validated and tested. Hence, it is important to survey the existence of datasets related to the proposed use case. If they are non-existent (or non-available), new datasets should be synthesized, resorting to simulation. In that case, it is important to: survey tools and frameworks that allow to simulate road agents mobility and communication (simultaneously); study, define and implement simulation scenarios on the selected tools; collect **V2X** data from the defined scenarios in order to finally compile the *datasets*.

Evaluate ML models for the prediction of collisions The proposed system should rely on **ML** algorithms to perform the prediction of the collisions. So, first, it is important to research adequate **ML** solutions for the proposed problem - predict collisions related to **VRUs**. The selected models should be trained, validated, tested and eventually optimized (parameterized) to try to achieve the optimal solution.

Develop and implement a prototype A prototype should be implemented for proof of concept. The resulting **ML** model should be deployed and tested in a simulation tool or framework, as realistic as possible. Thus, it is important to survey **ITS** simulation tools (if not achieved on the previous steps).

Evaluate the Methodology The main goal is to test the proposed methodology and evaluate the system's capacity for improving traffic safety for **VRUs**. It is important to evaluate the system's accuracy in terms of predicted collisions, the number of false positives and similar metrics. Additionally, it is also important to verify the timeliness of the predictions (in other terms, if the predictions allow for the users to prevent the collision).

1.2 Scientific Contributions

The research that was achieved for this thesis was aligned with two different Research and Development (R&D) projects, in a collaboration between the *University of Minho* (Portugal) and *Bosch Car Multimedia* (Portugal) - *EasyRide: Experience is everything (V2X Communications)* and *AURORA: Experience Mobility*.

Both projects had similar research focus - **V2X** communications, with particular focus on the development of safety applications for **VRUs**. This research included the following topics: survey and utilization of simulation tools for **ITS** mobility and communications (focused on **VRUs**); compile simulation datasets for the train and test of ITS safety applications; study of intelligent **ML** mechanisms to apply for **VRUs** safety, in terms of predicting collisions; specification and development of intelligent applications for **VRUs** safety (safety warnings).

This tasks resulted in the development of two application prototypes: use of **Vehicle to Pedestrian (V2P)** communications to notify vehicles of the detection of pedestrians crossing a crosswalk without vision, in a real and controlled environment; Use of **V2X** communications for predicting collisions involving **VRUs** (motorcycles) and passengers on intersections, through automatic learning mechanisms, in a simulation environment.

Additionally, this research also resulted in multiple scientific publications, which are listed next.

As first author:

Leveraging Vehicular Communications in Automatic VRUs Accidents Detection [12]
Bruno Ribeiro, Maria João Nicolau, and Alexandre Santos. 2022 thirteenth international conference on ubiquitous and future networks (ICUFN). IEEE, 2022.

[Poster] Machine Learning for VRUs Accidents Prediction Using V2X Data [13]
Bruno Ribeiro, Maria João Nicolau, and Alexandre Santos. Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing. 2023.

Using Machine Learning on V2X Communications Data for VRU Collision Prediction [14]
Bruno Ribeiro, Maria João Nicolau, and Alexandre Santos. Sensors 23.3 (2023): 1260.

Evaluation of a Collision Prediction System for VRUs Using V2X and Machine Learning: Intersection Collision Avoidance for Motorcycles [15]

Bruno Ribeiro, Maria João Nicolau, and Alexandre Santos. 28th IEEE Symposium on Computers and Communications (ISCC). 2023.

As co-author:

Enhancing VRUs Safety with V2P Communications: an Experiment with Hidden Pedestrians on a Crosswalk. [16]

Fábio Gonçalves, Bruno Ribeiro, João Santos, Oscar Gama, Francisco Castro, João Fernandes, António Costa, Bruno Dias, Maria João Nicolau, Joaquim Macedo, and Alexandre Santos. 2022 14th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT). IEEE, 2022.

Design and Evaluation of an Adaptive Virtual Traffic Light System for VANETs. [17]

Oscar Gama, António Costa, Maria João Nicolau, Alexandre Santos, Joaquim Macedo, Bruno Dias, Fábio Gonçalves, and Bruno Ribeiro. 14th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT). IEEE, 2022.

Agnostic Middleware for VANETs: Specification Implementation and Testing. [18]

Fábio Gonçalves, Bruno Ribeiro, Oscar Gama, Maria João Nicolau, Bruno Dias, António Costa, Alexandre Santos, and Joaquim Macedo. WINSYS - 19th International Conference on Wireless Networks and Mobile Systems, 2022.

Synthesizing Datasets with Security Threats for Vehicular Ad-hoc Networks. [19]

Fábio Gonçalves, Bruno Ribeiro, Oscar Gama, João Santos, António Costa, Bruno Dias, Maria João Nicolau, Joaquim Macedo, and Alexandre Santos. GLOBECOM 2020-2020 IEEE Global Communications Conference. IEEE, 2020.

Evaluation of Push and Pull Communication Models on a VANET with Virtual Traffic Lights. [20]

Oscar Gama, Alexandre Santos, António Costa, Maria João Nicolau, Bruno Dias, Joaquim Macedo, Bruno Ribeiro, Fábio Gonçalves, and João Simões. Information 11.11 (2020): 510.

Evaluation of Broadcast Storm Mitigation Techniques on Vehicular Networks Enabled by WAVE or NDN. [21]

Oscar Gama, Alexandre Santos, António Costa, Bruno Dias, Joaquim Macedo, Maria João Nicolau, Bruno Ribeiro, and Fábio Gonçalves. Intelligent Transport Systems. From Research and Development to the Market Uptake: Third EAI International Conference, INTSYS 2019, Braga, Portugal, December 4–6, 2019. Springer International Publishing, 2020.

A Systematic Review on Intelligent Intrusion Detection Systems for VANETs. [22]

Fábio Gonçalves, Bruno Ribeiro, Oscar Gama, Alexandre Santos, António Costa, Bruno Dias, Joaquim Macedo, and Maria João Nicolau. 2019 11th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT). IEEE, 2019.

1.3 Document Structure

This first chapter of the document presents the motivation of the thesis work, along with its main objectives. It also presents the methodology to achieve such goals and the scientific contributions that were achieved. The remaining of this document is organized in six main chapters.

Chapter 2 presents the related work. Section 2.1 introduces an overview on the background and basis of this thesis work, namely by presenting the most relevant and prominent **ITS** standards (both in Europe and America) and **V2X** communications characteristics (communication modes and technologies). Section 2.2 presents the most relevant works that rely on the usage of **ML** techniques for **ITS** solutions. The works are organized from the most generic to the most alike ones (works that are related to the prediction/detection of **VRUs** incidents).

The proposed work aims to take advantage of **ML** capabilities to improve the safety of **VRUs**, using **V2X** communications data. Such **ML** mechanisms usually require large sets of data to perform both training and testing. As discussed further ahead, no relevant datasets were found on the related works (datasets related to **VRUs** incidents that contain **ITS** standard messages exchanges). For that reason,

there is a need to resort to simulation tools to synthesize datasets. Hence, Chapter 3 is focused on **ITS** simulation. Section 3.1 surveys the most common **ITS** simulation tools (in terms of mobility simulators, communications simulators and also coupled simulators) and presents the main requirements for the simulation framework. Section 3.2 presents some preliminary tests, in order to perform a comparison of the tools identified in the previous section. Section 3.3 finally presents the complete development environment.

The proposed method of using **ML** techniques for the safety of **VRUs** requires heavy computation resources and large amounts of storage, particularly for the training and testing processes. Additionally, the use case of predicting collisions and sending warnings on such predictions also requires very low latencies (in terms of information exchange and also the real-time analysis of **V2X** data), so that road users have enough time to safely actuate. Hence, Chapter 4 starts by presenting the system's architecture in section 4.1, based on a *Fog Computing* approach, which organizes the entities and functionalities of the system in hierarchical layers (according to their characteristics in terms of computation requirements, timeliness, and so on). Next, section 4.2 describes the established use case based on **European Telecommunications Standards Institute (ETSI)** standards, where a passenger vehicle oversees an approaching motorcycle when turning on an intersection, which results in a possible collision. Section 4.3 discusses the implementation of the simulation scenario on **Vehicles in Network Simulation (VEINS)**, including the details regarding the mobility on **Simulation of Urban MObility (SUMO)** and the communications setup of **Objective Modular Network Testbed in C++ (OMNeT++)**. Also, it also presents how (and what kind of) data is collected inside **VEINS** to synthesize the datasets that are to be used in **ML**.

Chapter 5 presents a discussion on **ML** techniques for the development of the **VRUs** safety systems. Section 5.1 presents some preliminary **ML** tests - where different models were trained and tested to classify the vehicle's *type* based on the generated **V2X** messages. Then, an initial work for the detection of collisions (using the same traditional **ML** models) is achieved on section 5.1.2. This exploratory analysis led to the conclusion that traditional models are unsuitable for the problem that is to be solved, as it possesses some strong temporal concerns that is important for the learning process. Hence, section 5.2 discusses *time series forecasting*, including a discussion on the main characteristics of **Long Short-Term Memorys (LSTMs)** and **Multilayer Perceptrons (MLPs)**.

Chapter 6 presents the collision detection system, starting with a discussion focused on how the data was collected and processed for the training and testing procedures (section 6.1). Section 6.2 presents the main results for the collision detection system evaluation (by *aggregation time*).

Chapter 7 discusses the collision prediction system. It highlights the implementation differences to the previous developed safety system (focusing on multi-step forecasting strategies on section 7.1) and also presents several other important considerations are presented in section 7.2 (such as *feature selection*, *imbalanced data*, etc.). Section 7.3 presents the evaluation results for the collision prediction system, focusing on the metrics: Number of False Positives, Predicted Collisions, Collisions Prediction Percentage, Correct Decision Percentage and Average Prediction Time. Finally, section 7.4 evaluates the system in terms of collision avoidance capabilities and the timeliness of the collision predictions - if the prediction gives enough time for drivers to actuate in order to avoid the collision.

Finally, Chapter 8 discusses the main conclusions of this thesis work. This chapter also discusses some of the identified limitations and the future work possibilities that can complement this thesis (section 8.2).

Chapter 2

Related Work

This chapter starts by providing an overview on the background of the work proposal and its context. An introduction to **ITS** is presented, together with a description of the main European and American standards and how the communications work on these systems (regarding communication modes and technologies). Furthermore, it also presents some relevant related works, particularly focused on **ML** solutions applied to **ITS** - both generic and incident detection related solutions (and even more particularly for **VRUs**).

2.1 ITS Background

ITS is defined by the European Parliament as *systems in which information and communication technologies are applied in the field of road transport, including infrastructure, vehicles and users, traffic management and mobility management, as well as for interfaces with other modes of transport* [23]. There are a series of **ITS** standards that are defined in order to establish and enable interoperability, by specifying the communications and the type of messages that are meant to be exchanged, which communication media and protocols should be used, etc.

The definition of these standard differs from region to region of the globe. For that reason, there are several organizations in charge of developing such standards: **European Committee for Standardization (CEN)**, **ETSI** and **International Organization for Standardization (ISO)** in Europe, **Institute of Electrical and Electronics Engineers (IEEE)** and **Society of Automotive Engineers (SAE)** in the USA, **Association of Radio Industries and Businesses (ARIB)** in Japan, among others.

The next subsections focus on European and American standards, as they are the most relevant to this work.

2.1.1 ITS Standards in Europe

Figure 2.1 shows the ITS station (e.g. vehicles, road-side units or pedestrians) reference architecture, taken from the *EN 302 665* European standard in [24].

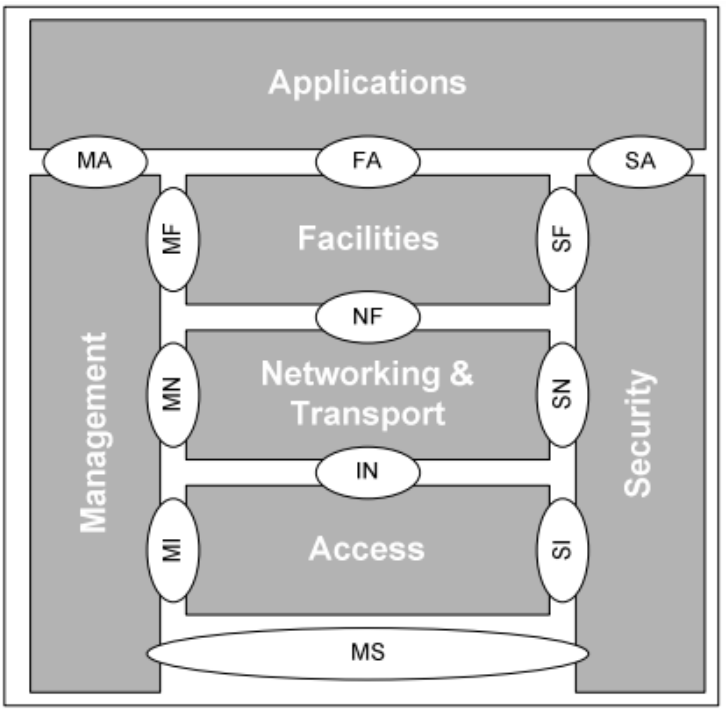


Figure 2.1: **ITS** *Station Reference Architecture* [24]

The main aspects related to the layers are briefly presented below (not focusing on security or management layers, which are transversal to all the other layers):

Application Layer This layer is where the **ITS** applications are meant to be deployed, for instance the ones from the *Basic Set of Applications* [25] (e.g. Road Safety applications - e.g. Emergency vehicle warning, slow vehicle - or traffic efficiency applications - e.g. speed limits notifications, enhanced route guidance). These applications rely on services provided by the *Facilities* layer.

Facilities Layer The *Facilities* layer is designed to provide support to the applications on the top layer, since **ITS** stations exchange information that may be useful to several applications. Some of the most important facilities are **CAM**, **DENM** and **Local Dynamic Map (LDM)**. The **CAM** service [10] allow the exchange of messages that possess status information of ITS stations, such as position, speed, moving direction, etc. **DENM** [11] messages deliver information on road traffic events, such as traffic collisions, road work, etc. **LDM** [26] provides a representation of vehicle's surroundings (managing the vehicle sensor and map data), with information regarding all static and dynamic safety elements.

Network and Transport Layer This layer aggregates all the networking protocols that allow the dissemination of messages. Regarding networking, there are two essential standards that are allowed in **ITS**: **Internet Protocol (IP)** and *Geonetworking*. *Geonetworking* [27] is a protocol for the delivery of information to a given geographical destination region. It can work on the following routing schemes: *Geobroadcast* (messages are sent to all nodes in an area), *Geounicast* (to a node), *Geomulticast* (to a set of nodes) or *GeoAnycast* (to any node in an area). This protocol also allows the building of a neighbor location table, using that for forwarding decisions. This layer also allows the use of **IP** (both IPv4 and IPv6), to enable interchangeability between **ITS** and non-**ITS** sectors.
Regarding transport, it is important to mention the **Basic Transport Protocol (BTP)**, which can be described as an **User Datagram Protocol (UDP)**-like protocol for *Geonetworking*, providing an end-to-end, connection-less transport service [28].

Access Layer This layer includes several access technologies (e.g. Infrared, cellular, microwave, among others) that can be used for communication with other stations, even simultaneously, depending on the purpose of the station. Still, ITS-G5 typically resorts to the use of *IEEE 802.11p* (**Medium Access Control (MAC)** and PHY), similarly to the United States set of standards, but over the European spectrum [29]. The *IEEE 802.11p* amendment introduced features to the **Wireless Local Area Network (WLAN)** family of standards, in order to enable its use on **V2X** communication - it was specifically built for vehicular environments, supporting applications that are very strict in terms of communication requirements.

2.1.2 ITS Standards in America

Wireless Access in Vehicular Environments (WAVE) system is a radio communication system that provides seamless, inter-operable services to users of a transportation system. **WAVE** aims to provided communications between vehicles and infrastructure, and also communications amongst vehicles themselves. Figure 2.2 illustrates the communication stack of the **WAVE**/**Dedicated Short-Range Communications (DSRC)** architecture (published full-use **WAVE** standards), as defined in the IEEE 1609.0 standard [30].

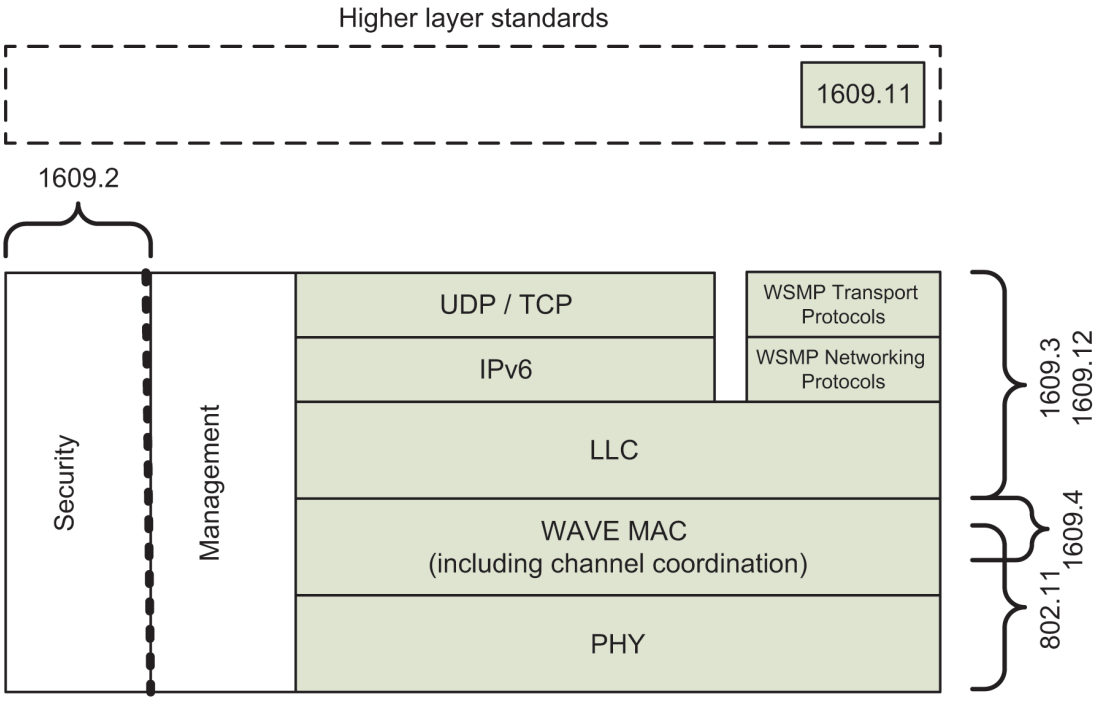


Figure 2.2: Full-use **WAVE** Standards (IEEE 1609 Standards)

In summary, the stack consists of standards from the IEEE 1609 family and IEEE 802.11p: IEEE 802.11p defines Physical and **MAC** layers while the upper layers are defined by IEEE 1609. The layers are briefly briefly discussed below (not focusing on security or management layers, which are transversal to all the other layers - similarly to the European stack):

Application Layer In the first trial-use standards, the IEEE 1609.1 [31] standard defined the usage of a *Resource Manager*, which was useful for the application to communicate to components or some other devices supported by **OBUs**. However, it was found unnecessary for **WAVE** standardization, hence being withdrawn (without a planned revision). At this point, IEEE 1609.11 [32] is the only application level standard and specifies a payment protocol for **ITS**, referencing **ISO** standards.

Transport & Network Layer The IEEE 1609.3 [33] standard aims to give support to WAVE data exchange, by defining transport and network layer services, including how addressing and routing should work. It also defines **WAVE Short Message (WSM)** (**WAVE**-specific alternative to **IP**), which can be directly used by applications. It includes features such as: **WAVE Service Advertisement (WSA)** transmission and monitoring; channel access assignment; **WAVE Short Message Protocol (WSMP)**; usage of **Logical Link Control (LLC)** and **EtherType Protocol Discrimination (EPD)**; IPv6 (including streamlined configuration); **Management Information Base (MIB)** maintenance; and others. IEEE 1609.12 [34] records the allocations of some identifiers used by the **WAVE** standards, including **Object Identifier (OID)**, EtherType, and Manage-

ment ID.

Media Access Control Layer IEEE 1609.4 [35] introduces enhancements to IEEE 802.11 **MAC** to support **WAVE** and services that support multi-channel for the upper layers. It includes features such as: channel coordination and routing; multi-channel synchronization; usage of IEEE 802.11 facilities (e.g., channel access) outside the context of a **Basic Service Set (BSS)**; usage of IEEE 802.11 timing advertisement frames; **MAC**-layer readdressing in support of pseudonymity; **MIB** maintenance; among others.

Physical Layer IEEE 802.11p [36] is an amendment to the 802.11 family of standards which introduced new characteristics to support vehicular networks: longer ranges of operation (up to 1000 meters), mechanisms to deal with the high speed of the vehicles and the nature of the automotive applications (e.g. having reliable broadcast), among others.

2.1.3 V2X Communications

Vehicle to Anything (V2X) is a term that is typically used to describe a new generation of information and communication technologies that allow vehicles to connect with any entity belonging to a vehicular environment (and vice-versa). Connecting these vehicular entities (e.g. vehicles, pedestrians, infrastructures, backbones, etc.) through the use of **OBUs** and **Roadside Units (RSUs)**, allows them to work together: by sharing information with each other in a cooperative way (e.g. safety warnings), they are able to address **ITS** issues more effectively than the traditional way where vehicles try to solve problems by themselves.

Communication systems in **ITS** are very characteristic: vehicles tend to move at high speeds, which may lead to loss of connections, packet losses and frequent topology changes. Furthermore, if there are too many nodes communicating within a given area, it is frequent to have high channel loads. For these reasons, communicating in **ITS** may be challenging at times.

The most common communication modes are described below:

Vehicle to Vehicle (V2V) This mode is used to describe direct communication between vehicles. It is typically used to support cooperative applications (mostly aimed at preventing collisions) that send messages with information to nearby vehicles (regarding position, speed and such). Due to the highly dynamic nature of the vehicular traffic and its surroundings, technologies with native broadcasting capabilities are the most suitable for these mode. Still, it is also possible to use this

mode in a *multi-hop* fashion, where vehicles relay messages towards the intended destination (if the destination is beyond the direct reach of the source node).

Vehicle to Infrastructure (V2I) This mode allows vehicles to communicate with the infrastructure (and vice-versa). It is useful in both local broadcast and bidirectional modes. In local broadcast, vehicles typically receive messages from the infrastructure **RSUs** (e.g. warning about poor road conditions). The bidirectional mode is usually required by a lot of mobility and comfort applications, such as *navigation assistance*, *Internet access* or *media download*. Furthermore, this mode can also be used to broadcast messages from a vehicle to other vehicles via the infrastructure servers.

Communication Technologies

Connecting all the entities on a vehicular environment requires a very efficient vehicular communication system, primarily due to the short delay requirements of many applications, particularly those related to safety. This section briefly describes the main characteristics of the most commonly used (wireless) technologies in vehicular environments to support these applications.

Bluetooth *Bluetooth* [37] is a short-range wireless communications technology, which is intended to replace the cable(s) connecting portable and/or fixed electronic devices. This technology is characterized for being very low power consuming and having short range, while operating on the unlicensed 2.4 GHz frequency band. Although it is not very suitable for **V2V** or **V2I**, it may be useful in some use cases - e.g. connecting the driver's mobile phone to the vehicle or pedestrians to nearby **RSUs**. This technology is adequate for audio or simple file transfers, and enables things such as perform hands-free calls on a vehicle, download travel information, etc. Frank et al. [38] study the applicability of *Bluetooth* on vehicular environments, analysing its performance on **Vehicular Ad Hoc Networks (VANETs)** in terms of range, signal quality or latency.

Visible Light Communications **Visible Light Communication (VLC)** is a bidirectional, high speed wireless communication technology. In simple terms, **VLC** works in a similar way to *Wi-Fi*, but instead of using radio signals, it relies on light waves to transmit data. The main difference between them is the amount of bandwidth available - **VLC** is much more plentiful when compared to radio frequencies. IEEE introduced the IEEE 802.15.7 standard [39], which defines *PHY* and *MAC* layers for short-range optical wireless communications. **VLC** systems can be deployed in vehicular environments through the use of existing infrastructures to allow **V2I** or by, for instance, using the vehicle head and back lights to allow **V2V** [40].

IEEE 802.11p *IEEE 802.11p* (sometimes also referred to as **WAVE**), is an amendment to the traditional IEEE 802.11 standard, introducing wireless access in vehicular environments [41]. It defines improvements that are required to support **ITS** applications, allowing data exchange between high-speed nodes (vehicles) and vehicles and road-side infrastructures. *IEEE 802.11p* uses **Orthogonal Frequency Division Multiplexing (OFDM)** applied to seven 10 MHz channels and supports high data rates (from 6 to 27 Mb/s) with a maximum theoretical communication range of 1km. This technology enables ad-hoc communication and direct information exchange between vehicles and also road-side infrastructures, making it suitable for most of the applications on **ITS**.

Cellular Networks *Cellular network* technologies, such as the traditional **Long-Term Evolution (LTE)** standard, are nowadays used by manufacturers to provide vehicles with applications and services (e.g. *Remote Diagnostics, Passenger Entertainment*, etc.). Due to its centralized architecture, **LTE** provides good **V2I** communications for its high data, penetration rate and large coverage. However, being centralized, it is challenging to implement **V2V** communications. For these reasons, a standard was proposed in *Release 14* of **LTE**: **Long Term Evolution - Vehicle (LTE-V)** [42]. This release introduces two new communication modes: *Mode 3* allows the selection and management of the radio resources that are used by the vehicles in **V2V** communications; *Mode 4* allows the vehicles to select themselves the radio resources for **V2V** communications, which mean that they do not depend on cellular coverage - particularly useful for safety applications. *Mode 4* is considered an alternative to the use of *IEEE 802.11p*. The *3GPP* organization (responsible for the standards development) proposed *5G V2X* enhancements under *Release 15*. *5G* aims to significantly increase cellular performance in terms of latency, throughput and reliability - while also supportive a huge number of connected devices. On [43] it is possible to find a series of services that shall be supported by *5G* from the following areas: non-safety (e.g. *entertainment, mobile office, digital map updates*), safety (*autonomous Driving, platooning*) and services in multiple Radio Access Technologies and networks environment (e.g. interoperability with other *V2X* technology such as *ITS-G5* or **DSRC**).

2.2 Machine Learning in the ITS Field

Machine Learning (ML) is a term that is used to describe computational methods that aim to improve performance or make predictions using past experience - past data that is available for analysis [44]. **ML** consists on algorithms that are designed to be efficient and to make accurate predictions,

resorting to *data mining* methods in order to infer knowledge from data - in other words, to find patterns on data.

The success of using **ML** techniques is highly dependent on the data available, since **ML** is inherently related to statistics and analysis of data. These learning techniques combine methods from computer science with statistics, optimization and probabilities.

Depending on the learning methods, **ML** techniques can be classified into several categories [45, 46]. Some of the most important are briefly described below:

Supervised Learning In these methods, the data that is studied is labeled (e.g. *True* or *False*) and a classifier is the used to predict the label of the test data. The classifier is tuned and parametrized in order to achieve a good level of accuracy. *Decision Trees*, *Neural Networks*, *Support Vector Machines*, *Fuzzy Logic* and *Genetic* algorithms are examples of supervised learning.

Unsupervised learning In this case, the input/training data is not labeled. These classification methods can find patterns/clusters in unstructured data. Some examples of unsupervised learning are *K-Means*, *K-mediods* and *Bayesian clustering*.

Semi-Supervised Learning Semi-supervised methods are hybrid, where that data can be either labeled and unlabeled.

Reinforcement Learning These methods aim to take actions that would maximize rewards, or minimize risks. This kind of algorithms learn from the environment in an iterative trial-and-error fashion, training to find the most rewarding actions. Some examples are *Q-Learning*, *Temporal Difference* and *Deep Adversarial Networks*.

ML can be applied to several areas and has a very broad set of uses: in text/document classification (e.g. assigning topics to documents), speech processing (e.g. speech recognition), computer vision (e.g. object and faces detection), among others. Naturally, **ML** techniques can also be applied on **ITS** environments, as discussed next. The related works are presented from the most generic to the most alike works. First, some generic **ML** solutions in the **ITS** field are presented, then solutions related to the prediction/detection of incidents on the roads and finally such solutions related to **VRUs**.

2.2.1 Generic Machine Learning Solutions for ITS

Applying **ML** techniques to the aforementioned information that is received by vehicles and **VRUs** allows the collection and examination of complete *datasets*, instead of the traditional method of sampling

and extrapolation [47]. Together with data that is collected through the use of sensors, cameras, and so on, it is possible to obtain useful information in order to improve traffic [48]. For instance, the collection of this data allows the later use of prediction models to foresee traffic jams and compute new routes for vehicles, based on parameters such as consumptions or desired speed, and so on [49].

Collecting cooperative information allows the development of advanced solutions to predict traffic. For instance, Terroso-Saenz et al. [50] show that it is possible to detect different levels of traffic through the study of messages together with environmental data from external sources (e.g. weather conditions), resorting to an event-driven architecture - through complex event processing on the exchanged beacons (both achieved on vehicles and road infrastructures). Since the traffic density calculation relies on the number of slow vehicles detected, the results show that the penetration rate of the system naturally has direct affect on its performance and reliability.

The usage of **ML** techniques and *data mining* on these vehicular environments has the potential to also solve some **ITS** inherent problems. A very large part of the research in this area is related to communication issues (e.g. routing, link failure detection, malicious behaviors). For instance, Lai et al. [51] propose a **Machine Learning-assisted Route Selection (MARS)** system for routing protocols, where **RSUs** use **ML** on the road information to predict vehicles movement and the transmission capacity of routing paths, choosing then the path that seems more suitable. Furthermore, **MARS** can also chose to forward messages through other **RSUs**, according to the predicted destination location and the estimated transmission delays.

Grover et al. [52] present a security framework for **VANETs** communications based on a **ML** mechanism to classify misbehaviors (the misbehaviors are implemented by the authors by tampering the information of the packets). These misbehaviors are classified based on features such as **Received Signal Strength (RSS)**, number of packets delivered, etc.

Although not particularly focused on vehicular communications, Bhutani [53] surveys the application of **ML** prediction solutions to issues such as routing, handover latency, link duration, and so on, on wireless networks. Naturally, some of these problems are also present on vehicular communications - thus, they may also be applied to these environments.

Furthermore, managing and optimizing traffic is also a very important research subject. For example, Gregoire et al. [54] propose the use of **ML** to optimize traffic lights control - controllers typically use pre-timed stages, not considering dynamic changes in the traffic flow. The authors use a **ML** algorithm that attributes rewards to the operating agents according to the results of their selected actions, aiming to obtain an optimal control policy.

Al Najada and Mahgoub [55] study the impact of human behavior in traffic and road safety decisions. By applying techniques of feature selection to find the most important predictors, the authors analyzed a big vehicular *dataset* containing examples of casualties and predicted things such as the driver's age, sex and the collision severity from their behavior.

Finally, a very hot topic when talking about **ML** combined with vehicular environments is the detection/prediction of collisions. The following subsection describes in more detail some of the main work achieved in this area of research, also particularly focusing on **VRUs**.

2.2.2 Detection/Prediction of Traffic Incidents

A very interesting area of research when talking about **ML** in vehicular environments is the detection and prediction of incidents (a term used to describe events on the road) and collisions on traffic. Typically, these works focus on applying machine learning to data collected in vehicular environments via cameras, sensors, **Global Positioning System (GPS)** and similar, but also from **V2V** communications. The following works are some of the most interesting related to generic incidents detection.

Júnior et al. [56] present an evaluation of the performances of four different machine learning algorithms (**Artificial Neural Networks (ANN)**, **Support Vector Machines (SVM)**, **Random Forest (RF)**, and **Bayesian Network (BN)**) in order to detect different driving event types (aggressive breaking, acceleration, turns and lane change, and other non-aggressive events) using data collected from smartphone sensors (accelerometer, magnetometer, linear acceleration and gyroscope) in real-world experiments. The data gathering was achieved by driving a real car (and performing different manoeuvres) while collecting data from several sensors. The main goal of the work was to assess which combination of sensors and methods achieves higher performance regarding their classification, aiming to find the best combination match to each class of driving behavior. The processing steps are illustrated on Figure 2.3. Although not being related to collisions, these predictions can still be useful to prevent them. The results show that, in summary, the gyroscope and the accelerometer sensors are the best to detect the driving events, and that **RF** is the best performing algorithms, followed by **MLP** (a simple form of an **ANN**).

Dogru and Subasi [57] focus on finding a way to reduce the frequency and severity of traffic accidents. To achieve such goal, the authors present an accident detection system that resorts to **V2V** communications, in which vehicles exchange their information (speed and coordinates) and send traffic alerts, improving safety and mobility. Furthermore, it also shows how **ML** techniques can leverage this data to detect accidents - each communication node that possesses information from other vehicles is able to determine the traffic condition. Resorting to simulation, the authors analyse the **ANN**, **RF** and

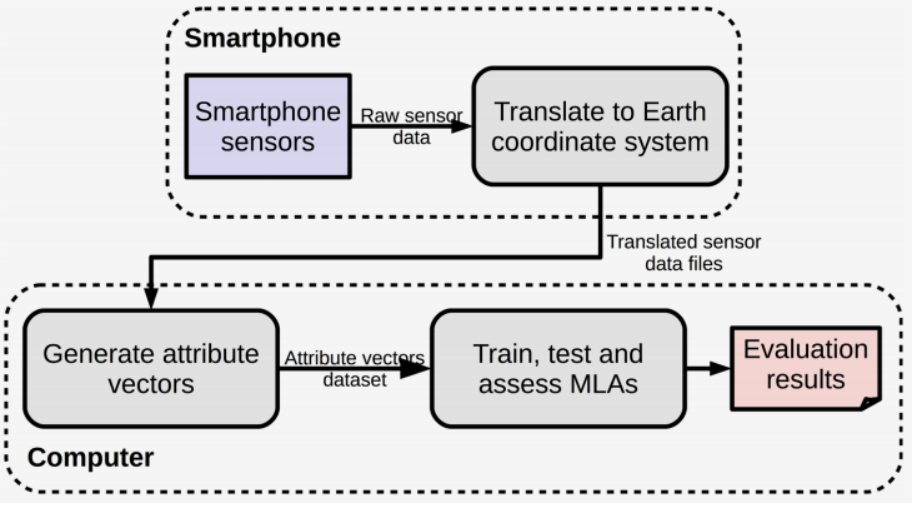


Figure 2.3: *High level view of our evaluation pipeline showing processing steps from raw sensor data sampling to training, testing, and assessing MLAs.* [56]

SVM algorithms to evaluate their performance. Using **SUMO** to simulate traffic and generate both position and velocity data, this information is then used as input for the algorithms in each vehicle. Results show that incidents can be regarded as outliers in data and, for this reason, **ML** techniques can be used to detect them, allowing later actuation - e.g. warning other vehicles about the incident, resulting in route redirection. From the simulations, the **RF** algorithm was found to perform better (in terms of accuracy) when compared to the **ANN** and **SVM**.

Ozbayoglu et al. [58] propose an accident-detection system based on computational intelligence techniques where traffic-flow data is processed in real-time. The real-time traffic monitoring system enables the detection of accidents immediately after an incident, allowing to take precautions preventively. The traffic-flow data was collected using several sensor locations in Istanbul - the sensors collected data related to the number of vehicles passing, their average speed, the average occupancy of the lane and information related to time/date. This work analyses three different models for accident detection: **Nearest Neighbour (NN)**, **Regression Tree (RT)**, and **Feedforward Neural Network (FNN)**. The inputs for the models were obtained using big data techniques (since they allow real-time and scalability capabilities). The raw original data goes through an *Extract-Transform-Load Hadoop Distributed File System* process and *Apache Spark*. The raw data is stored in *SQLServer* and imported to *Hadoop* using *Sqoop*. The extracted features are then fed into the models and the probability of an accident is computed as output. The results indicate that the number of false alarms is greater than the accident cases. Still, the system provides useful information that can be used for status analysis and enable early reactions, potentially saving lives and time/resources. Despite the large number of false positives, the overall accuracy of the models are mostly over 99%.

Zhang et al. [59] employ deep learning in order to detect a traffic accident from social media data. In a first step, the authors investigate over 3 million tweet contents related to traffic accidents in Northern Virginia and New York City (with explicit indications of traffic accidents - not including traffic congestion, construction works, and so on). From the tweets, *token* features were found, both individual and paired, that may indicate the event of a traffic accident. Results show that paired tokens can capture the association rules inherent in the accident-related tweets and can increase the accuracy of the detection (in particular when the number of individual token is more limited). Then, the **Deep Belief Network (DBN)** and **LSTM** deep learning methods were implemented in the extracted tokens. Results show that **DBN** can obtain an overall accuracy of 85%, outperforming **SVM** and **supervised Latent Dirichlet Allocation (sLDA)**. Finally, the accident-related tweets were compared with the traffic accident log on highways and traffic data on local roads from 15,000 loop detectors - 66% of the tweets can be located by the accident log and around 80% of them can be compared to nearby abnormal traffic.

Although not exactly related to incidents detection, [60] presents a mechanism that focuses on the automatic detection of street elements (traffic lights, street crossings and roundabouts), combining outlier detection with deep learning and classification from **GPS** data gathered while driving. The speed and acceleration were derived from the collected **GPS** data. The data collection journey included two urban areas and a connecting motorway. The *dataset* was generated by a driver which drove three different vehicles while a mobile device recorded the **GPS** traces. The outlier detection algorithm is used first in order to detect abnormal driving locations. Window-segments around these locations were classified as outliers and used to train a **DBN**, followed by a classifier in order to detect them. Using deep learning, speed and acceleration patterns were analysed at each outlier, enabling the extraction of features which are then classified into a traffic light, street crossing, roundabout or other element. The mechanism is validated using two different *datasets*: one with several drives following the same circuit by a single driver and a second in which ten drivers follow a particular circuit each. The results are acceptable (combined recall of 0.89 and a combined precision of 0.88 for classification) if the recall is accepted to be low, but the performance worsens when the number of detected elements increases.

In addiction to the previously discussed works, there are some other interesting researches related to communications and sensors/probe data, which can be found in [61–63]. Still, most of the works regarding **ML** are typically applied to video data, as exemplified by [64–67].

Finally, the application of **ML** techniques to **VRUs** is introduced next.

Vulnerable Road Users

Goldhammer et al. [68] present a set of **VRUs** movement models based on **ML** techniques, aiming to classify **VRUs** current motion state and to predict the upcoming trajectory. The *dataset* consists of over a thousand pedestrian and near five hundred cyclist scenes acquired at an urban intersection. The data was collected using a mix of cameras and laser scanners. The recognition of the motion state and the trajectory prediction is then tested using a method of polynomial approximation combined with **MLPs** - called *PolyMLP*. The overview of this method is illustrated on Figure 2.4.

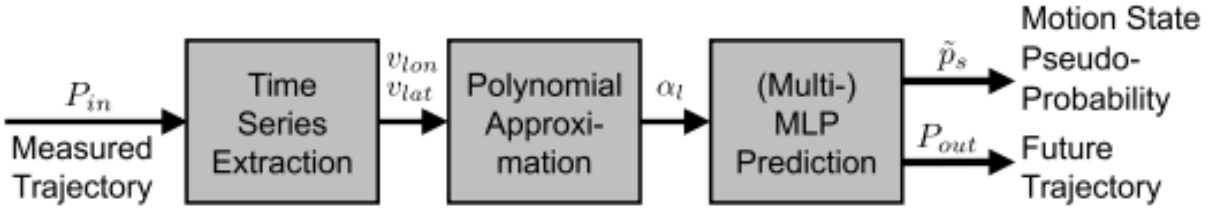


Figure 2.4: *Overview of the basic processing steps of the method PolyMLP* [68]

The results show higher classification values and the system is able to recognize motion state changes earlier, compared to **Interacting Multiple Model (IMM)** *Kalman Filter*.

Parada et al. [69] introduce a **VRU** trajectory prediction service, using regression algorithms on *Cartesian* coordinates data. The authors compared different regression methods in terms of prediction time window, position accuracy and processing time, resorting to the *Weka* tool. The system overview is illustrated on Figure 2.5.

The system considers two types of users: vehicles and **VRUs**. A 5G cellular base station is connected to a **Multi-access Edge Computing (MEC)** server (which possess resources in terms of storage and computation). It is assumed that both kind of users transmit **CAMs** through 5G (using **OBUs** and *smartphones*), which the **MEC** is able to receive. The *Collision Avoidance Service* application (running on the **MEC**) collects the position of the road users and then feeds them to a regression algorithm that tries to predict the trajectories and, ultimately, the probability of collision between them. Using an **Alternating Model Tree (AMT)**, the next position is predicted with an error of less than 3.2 centimeters, increasing up to 1 meter when predicting the next 5 positions (1s between consecutive positions). As future works, the authors plan to use this service to estimate the collision probability.

Komol et al. [70] compare the use of different **ML** algorithms in the identification of crash severity factors of different **VRUs** Groups (pedestrian, bicyclist and motorcyclist), using real data from Queensland, Australia (from 2013 to 2019). Seventeen road crash parameters were considered as input features to train models (**k-Nearest Neighbor (KNN)**, **SVM** and **RF**) - related to road user characteristics, weather

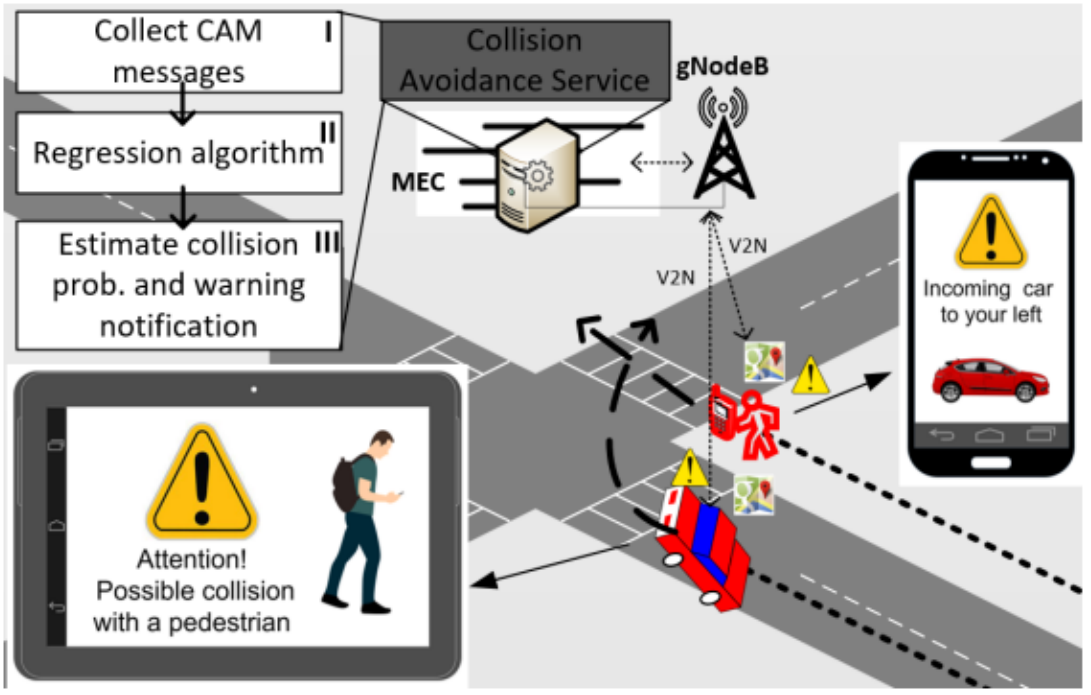


Figure 2.5: *Collision avoidance system overview* [69]

and environment, vehicle and driver condition, period, road characteristics and regions, traffic, and speed jurisdiction. **RF** classification models performed more robustly in test accuracy: (motorcyclist: 72.30%, bicyclist: 64.45%, pedestrian: 67.23% and unified **VRU**: 68.57%).

Vilaça et al. [71] propose a model that identifies risk factors for **VRUs** that can affect their injury severity when involved in an accident. Records from **VRUs** related crash data from Portugal (both imbalanced and balanced datasets) were used to compare the performance of two **ML** classifiers - **Decision Tree (DT)** and **Logistic Regression (LR)** -, also considering three different *resampling* techniques (*undersampling*, *oversampling* and *synthetic oversampling*). The results indicated that **DT** outperformed **LR**, since the coefficient correlations were not considered and all the variables were taken into account - the model revealed to be more accurate considering the crash severity data under evaluation. Nevertheless, both methods could correctly classify the classes with relatively high accuracy. The analysis of the results also allowed to conclude that road markings, road conditions and luminosity greatly affected the severity of pedestrian's injuries. Regarding the cyclists injuries, age group and temporal variables (month, weekday and time period) were the most significant risk factors.

The work in [72] analyses injury severity of three-wheeled motorized rickshaws, resorting to several algorithms (**Decision Jungle (DJ)**, **RF**, and **DT**) and data from the city of Rawalpindi, Pakistan. Results showed that **DJ** outperformed the other solutions with an overall accuracy of 83.7%. The analysis also showed that features such as the lighting condition, younger drivers, high-speed facilities, weekdays, off-

peak, and shiny weather conditions were more likely to worsen injury severity of the crashes.

Völz et al. [73] study the behavior of pedestrians at cross-walks through **ML**, by training a model to predict a pedestrian intention to cross the road. The database was build from real pedestrian trajectories (from a city in Germany) using a *Velodyne* laser scanner, since cameras usually have a limited field of view. The collected data contains a large set of possible features that could be suitable (e.g. distance to the curb, distance to the cross-walk, distance to nearby cars, etc.). These features were there evaluated using a **Recursive Feature Elimination (RFE)** algorithm (for relevance determination) and multiple **SVMs**. Results show that the most relevant features are only related to the pedestrians trajectories and the local map - the features related to the nearby road users are of less importance.

Jahn et al. [74] describe a system for detection of pedestrians crossing a curb, based on wireless communications (e.g. **WLAN** or *5G/LTE*) and a *context filter* that identifies **VRUs** at risk. The *context filter* identifies **VRUs** in potentially dangerous situations based on several information such as position, movement direction or accelerations. The chosen scenario consists of the situation when a pedestrian steps onto the road, investigating the performance of detecting this particular activity. The data is collected from smart-phones sensors, particularly accelerometers and gyroscopes. The activity detection was performed using several **ML** algorithms (**KNN**, *C4.5*, **Repeated Incremental Pruning to Produce Error Reduction 2 (RIPPER2)**, **Naive Bayes (NB)**, and **Sequential Minimal Optimization (SMO)**) - being that **KNN** performed best. These results indicate that the use case can support the overall *context filter* premise, in order to identify **VRUs** in potential danger.

Bieshaar et al. [75] present an holistic approach for detecting **VRUs** intentions (*pedestrians* and *bicyclists*) by cooperative methods, consisting of basic movement prediction (e.g. standing, moving), and future trajectory. Intelligent vehicles (equipped with sensors, data processing and communication capabilities) should acquire and also maintain a local model of their surrounding traffic, which allows the perception gained from individual agents (beyond their own sensor capabilities), enabling a more extensive prediction. Some local inconsistencies may also be resolved by the cooperative intelligence of these intelligent agents. All agents have the ability to locally detect **VRUs** and their intention, as well as the ability to exchange information via communications. The global concept is shown in Figure 2.6.

The intention detection consists of two stages: basic movement and trajectory forecast. The basic movement detection starts in each agent separately: the goal is to early detect/predict movement transitions (e.g. *standing* to *walking*) of **VRUs**. These transitions serve as indicator of the future behavior of the **VRUs**. They should be detected using, for instance, **SVM**, trained on labeled data. The trajectory prediction goal is to provide situational awareness, enabling vehicles to adapt and take action if necessary

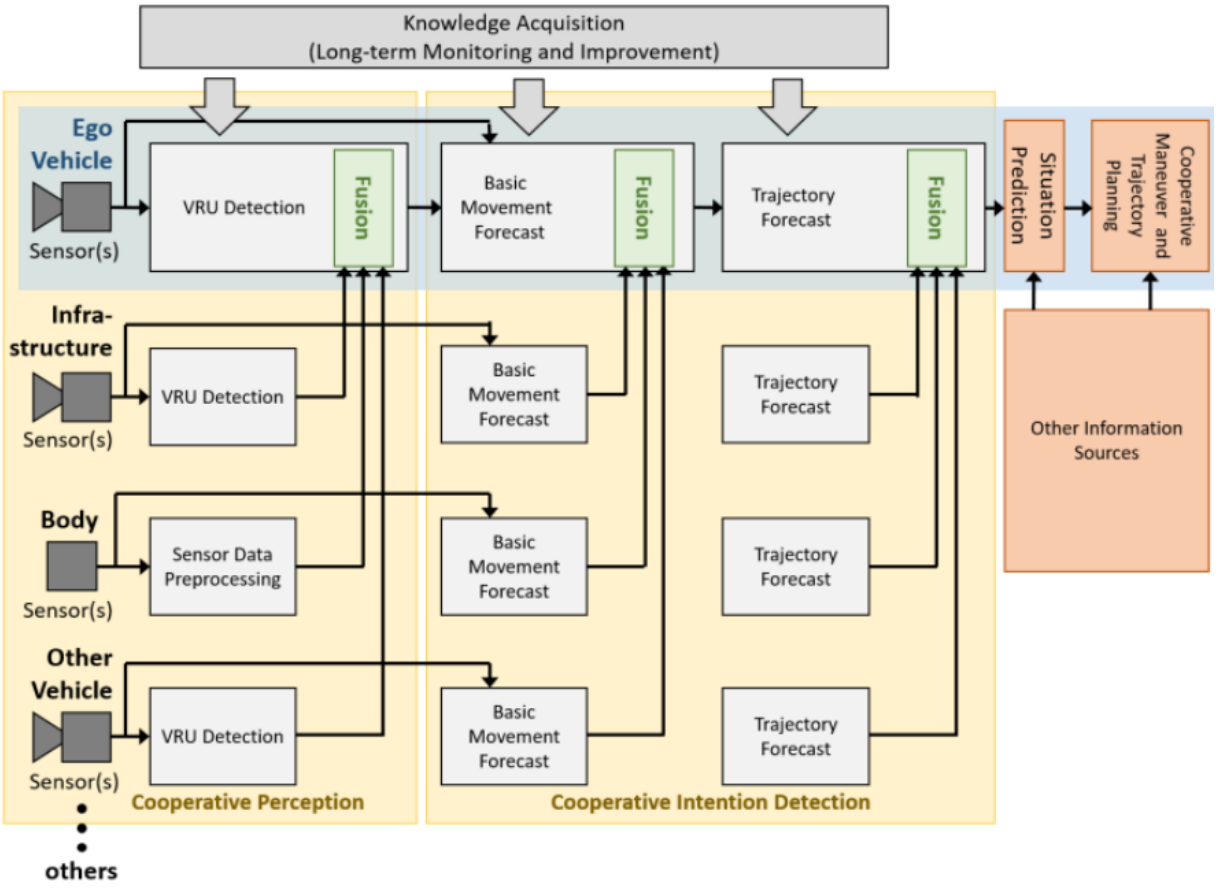


Figure 2.6: *System architecture for cooperative VRU perception and intention detection* [75]

- e.g. perform an emergency brake. The approach is based on supervised techniques, such as **ANN** and kernel based regression algorithms, as well as on analytical models. The combination should take place at both feature (different information sources are complemented) and decision levels (predictions from different users are combined).

2.3 Summary

This chapter provided an overview on **ITS**, focusing on the main communication standards and technologies. Additionally, it presented a survey on some of the most interesting works regarding the application of **ML** methods in **ITS**.

First, some generic **ML** solutions for **ITS** are presented. A very large part of the research in this area is related to managing and optimizing traffic (e.g. traffic lights control), although there are some works that also focus on communication issues (e.g. routing protocols, security).

Then, the use of **ML** in the prediction of incidents and collisions is discussed. Typically, when talking about applying **ML** techniques for this kind of prediction, most works tend to focus on data collected in

vehicular environments via sensors, **GPS**, cameras or similar.

Finally the state of the art work particularly focused on **VRUs** is presented. In this case, the data collection and the models testing tends to be similar to generic solutions. Still, most works tend to focus on pedestrians and bicycles, and not particularly in reduced mobility persons or even motorcycles.

Chapter 3

ITS Simulation

The development of advanced **ITS** systems requires a proper evaluation of their performance. However, performing fields tests in vehicular environments is challenging. The large number of nodes and traffic scenarios makes it very difficult to collect data in real experiments. Also, developing and testing real prototypes is a very expensive task, both in terms of money and time (preparation and processing time).

On other hand, within the research community, simulators are a very popular choice for evaluating communication and transportation solutions for **ITS** due to their capacity to conduct large-scale assessments. Researchers have the advantage of having a much more controlled environment when using simulators: these tools offer the ability to parameterize and configure settings, control input data, and facilitate easy deployment and repetition of tests, among other benefits. Furthermore, and since these tools are widely used by the community, researchers can focus only on the development of their solutions, as individual models are already considered as validated. Still, simulation typically resorts to simpler individual models, which may impact the results veracity, as the implementation's accuracy may reduce the system's realism.

In summary, using simulation applied to **ITS** brings benefits in terms of cost, repeatability, scale and practical requirements. For these reasons, it is a very popular choice among researches when compared to real-world field testing. The increasing popularity of **ITS** lead researchers towards the development of accurate and realistic simulation tools, which are used to evaluate the performance of whole systems, applications, communications solutions, among others.

At this point, it has not been found a *dataset* related to **VRUs** collisions that contains **ITS** standard messages exchanges on the related works or projects (between vehicles/infrastructures and **VRUs**). For that reason, there is a need to resort to simulation tools to construct one.

Hence, this chapter is focused on **ITS** simulation. First, **ITS** simulation tools are surveyed and presented on Section 3.1. Additionally, the section also presents the established requirements for the simulation framework. Section 3.2 discusses some initial tests on simulators, to enable a comparison

between the identified tools. Finally, section 3.3 presents the development environment.

3.1 ITS Simulation Tools

In order to perform a proper simulation of vehicular networks, both a traffic and a network simulator are required (they can work either in a dependent or independent way). This section identifies and describes some of the most widely used tools/frameworks on the literature for the simulation of **ITS**. First, some mobility simulators are presented, followed by some network simulators. Finally, some important frameworks that integrate and interconnect both kinds of simulators are also presented.

3.1.1 Traffic Mobility Simulation

Traffic mobility simulation tools aim to replicate traffic flows within transportation networks. These simulator tools are typically classified as *microscopic*, *macroscopic*, and *mesoscopic*. *Microscopic* tools simulate traffic on a large scale considering each entity (e.g. vehicles) in an individually way. *Macroscopic* tools, on the other hand, model traffic as a fluid without focusing on individual nodes. *Mesoscopic* simulation tools combine aspects of both previous models, representing traffic through aggregated traffic platoons using simplified models, avoiding the time and complexity issues of microscopic solutions.

However, proper **ITS** simulation scenarios require and depend upon accurate models of communication between nodes and their exact position, which makes *microscopic* tools more robust and reliable solutions when in comparison to both *macroscopic* and *mesoscopic*, since they offer less detail by comparison.

SUMO [76] is an open source *microscopic* simulation tool that is widely used in **ITS** research to simulate driving and traffic management strategies [77]. Some of its most important features include the ability to simulate large networks of roads, different vehicle types, individual routing, multi-lane streets, and so on. **SUMO** also includes a graphic interface that illustrates the road network topology and the movement of the nodes during the simulation runtime. Furthermore, it is possible to combine **SUMO** with other tools such as **OpenStreetMap (OSM)** [78] to generate real road networks from any location in the world and import them into the simulation environment. However, since **SUMO** is purely a mobility simulator, the tool cannot be directly used to simulate communications. Still, **SUMO** provides the **Traffic Control Interface (TraCI) Application Programming Interface (API)**, which allows **SUMO** to act as a server and to connect the tool to external applications through a **Transmission Control Protocol (TCP)** socket.

MObility model generator for VEhicular networks (MOVE) [79] (built on top of **SUMO**) is a generator of realistic mobility models: **MOVE** generates mobility traces with data from realistic vehicles movements. These movement can later be imported into network simulators (e.g. **Network Simulator 2 (ns-2)** communications simulator).

Although **SUMO** is indeed the most widely used tool when simulating traffic, there are also some other important tools that have similar characteristics.

Verkehr In Stadten SIMulationsmodell (VISSIM) [80] is a microscopic, time-step and behaviour-based traffic simulation tool. This framework provides users with the ability to define maps and scenarios and to observe traffic conditions in a very high level of detail. **VISSIM** allows the simulation of important road entities: vehicles, pedestrians, buses, bicycles, motorcycles, trams and even rickshaws. The **VISSIM** tool contains a traffic model that takes into consideration driver's psychological aspects and also provides a pedestrian mobility model.

VanetMobiSim [81] is an extension to **CANU Mobility Simulation Environment (CanuMobiSim)**, which is a framework that is used for modelling mobility through the generation of trace files. *VanetMobiSim* provides vehicular mobility at both microscopic and macroscopic levels. At microscopic level, *VanetMobiSim* implements **V2V** and **V2I** models (that enable, for instance, vehicles to regulate speed and obey traffic signs). *VanetMobiSim* mobility models have been validated by the **Traffic Software Integrated System - Corridor Simulation (TSIS-CORSIM)** [82] traffic generator.

3.1.2 Network Simulation

Network simulation is one of the most prominent evaluation methods in computer networks, modelling the behaviour of networks by calculating the interaction between the nodes. Naturally, the same methods can also be applied in the field of vehicular communications.

Network Simulator 3 (ns-3) [83] is a very well-known discrete event simulator for communication networks, which evolved from **ns-2** [84]. The **ns-3** simulator introduced several improvements, in particular related to scalability and memory efficiency, incorporating architectural concepts from **Georgia Tech Network Simulator (GTNetS)** [85]. It supports realistic models of wireless communications for cooperative **ITS** applications, providing models to emulate radio propagation effects and functionalities and protocols for all layers of the *ITS-G5/ WAVE* stacks.

The **Objective Modular Network Testbed in C++ (OMNeT++)** [86] tool is a discrete event network simulator. The **OMNeT++** tool is not a concrete simulator by itself. Instead, **OMNeT++** provides a component architecture for models that can be developed for several domains: communication

networks, protocol modeling, modeling of queuing networks, among others. Generically, modules are programmed in C++ and assembled into larger components and models using a high-level language (**Network Description (NED)**). In that sense, the network simulation models are available as several external frameworks, that are developed in an independent way by its user community. **OMNeT++** is built upon two main frameworks that together provide a specially designed propagation model for **V2X**: **Mixed Simulator (MiXiM)** [87] and *INET Framework* [88].

Java in Simulation Time (JiST) [89] is a discrete event simulation tool that allows the simulation of *Java* applications. Although this tool allows generic network simulation, it is frequently used together with **Scalable Wireless Ad-Hoc Network Simulator (SWANS)** [89], which is a highly scalable mobile ad hoc networks simulator, that supports thousands of nodes. However, and despite its good characteristics, **JiST** is no longer maintained, and the latest available version does not include models suitable for **ITS**.

SimuLTE [90] is an open-source simulator for **LTE** and **Long Term Evolution - Advanced (LTE-A)** networks based on **OMNeT++**. Being open source, it allows the development of new modules containing new algorithms and protocols. Additionally, since it is built on top of **OMNeT++**, it can be easily integrated with all its modules (e.g. *INET*), enabling good simulation for applications that rely on heterogeneous technologies (e.g. **LTE** and IEEE 802.11p scenarios).

3.1.3 Coupled Simulation

The simulators presented earlier may work in an independent way, but there are some other additional tools that enable them to interconnect and work in conjunction. The combined utilization of these tools can be characterized as follows [91]:

- **One-way decoupled** Solutions where the mobility tool generates a trace file from the intended scenario which is later processed by the network simulator.
- **Two-way loosely coupled** Solutions where simulators communicate bidirectionally. For instance, the network simulator controls the traffic simulator through commands (e.g. using *TraCI* from **SUMO**) and the mobility simulator reports back information about the nodes (e.g. vehicle's speed and position).
- **Two-way tightly coupled** A single simulation tool handles both traffic and network simulation simultaneously, operating in a coordinated and parallel way.

The relationship between the simulators is illustrated on Figure 3.1.

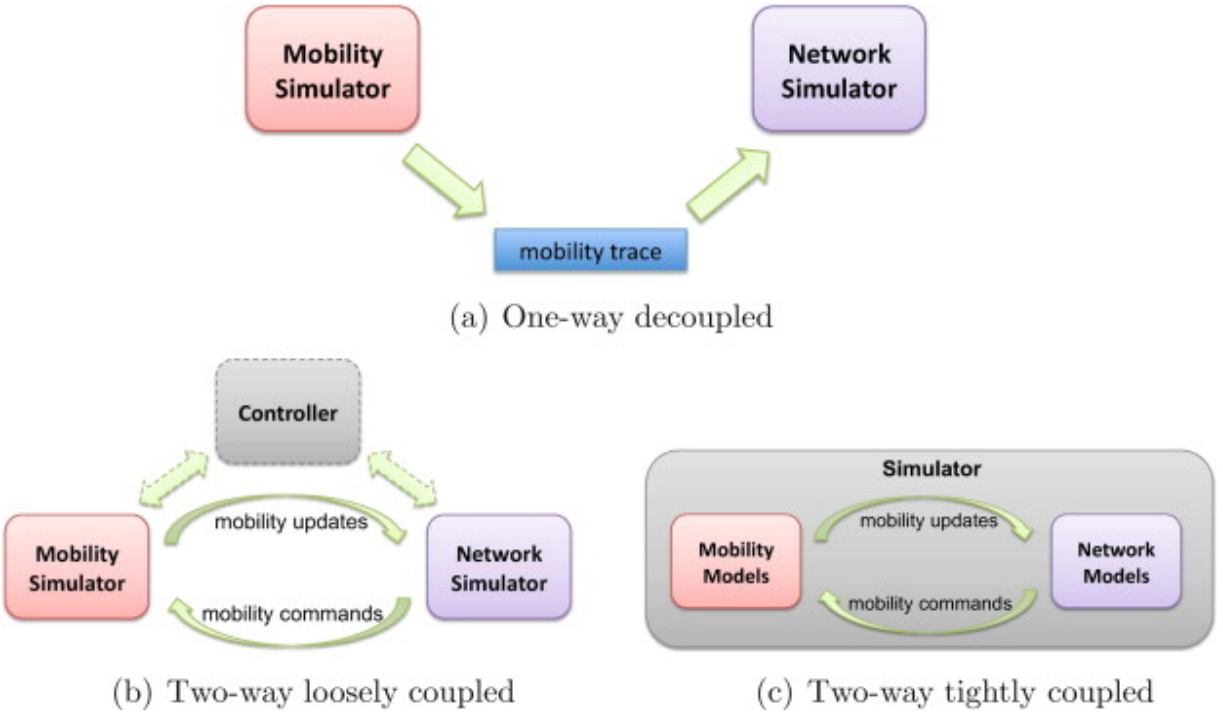


Figure 3.1: *Relationships between mobility and network simulators* [91]

Although *one-way decoupled* simulation may be appropriate for some **ITS** applications, when testing systems where communications have direct impact on traffic (and vice-versa), two-way coupled tools are the most interesting kind of simulators. For example, a safety warning coming from an infrastructure node that has detected an imminent collision may lead vehicles to break or change lane. For this reason, researchers tend to use frameworks that couple these powerful simulators in order to obtain results as accurate as possible. Some of these *two-way coupled* tools are presented next on Table 3.1.

One of the main tasks of this thesis is the simulation of scenarios of collisions related to **VRUs**. In this case, the message exchanges between the road entities (and the collision events) may have impact on the mobility configuration. Therefore, as the traffic dynamics may change abruptly, this thesis work will use the same strategy (the mobility and network simulation tools must be linked).

The simulation framework should support the following requirements:

- Realistic vehicular mobility simulation.
- Realistic **V2X** communications simulation (realistic implementations of IEEE 802.11p).
- Support for modeling of vehicular domains such as vehicles, **RSUs**, collision events, and similar.
- Support for the deployment of individual applications for different road entities (e.g. passenger vehicles, **VRUs** and **RSUs**).

- Ability to import realistic map scenarios (or the ability to design them from scratch).
- Support for a **Graphical User Interface (GUI)** showcasing the collisions (in terms of mobility).
- Support for the collection of simulation statistics (e.g. messages statistics, vehicles data, collision events, etc.).
- Support for the parameterization of both mobility (road entities parameterization) and communications (IEEE 802.11p specifics).

Simulator	Open Source	Active Development	Language	Mobility Simulator	Network Simulator
Artery	Yes	Yes	C++	SUMO	OMNet++
CAVENET	No	Yes	Matlab	Built-in	ns-2
Eclipse Mosaic (VSimRTI)	Yes	Yes	C++ (Java on VSimRTI)	Support for SUMO, PHABMACS, and others	Support for ns-3, SNS, OMNet++, and others
EstiNET	No	Yes	Java	Built-in	EstiNET
GrooveNet	Yes	No	C/C++	Built-in	Built-in
iTETRIS	Yes	No	C++	SUMO	ns-3
NCTUns	Yes	No	C++	Built-in	Built-in
NetSim	No	Yes		SUMO	NetSim
SWANS++	Yes	No	Java	JiST/SWANS	SWANS
TraNS	Yes	No	Java/C++	SUMO	ns-2
VANETSim	Yes	No	Java	Maps imported from OSM	Built-in
VEINS	Yes	Yes	C++	SUMO	OMNet++
VENTOS	Yes	No	C++	SUMO	OMNet++

Table 3.1: Simulation Tools - Coupled Simulators for **ITS**

Additionally, it is also important that the selected framework is currently in development and also open source (free to use and modify). Furthermore, they should be easy to setup and use and possesses rich documentation, both for the setup of scenarios and for its parameterization.

The first elimination factor was to consider only open-source and actively in development frameworks. Hence, **VEINS**, *Artery* and *Eclipse MOSAIC* are the most interesting solutions amongst the identified frameworks. For that reason, they are discussed next.

Eclipse MOSAIC [92] (formerly known as **V2X Simulation Runtime Infrastructure (VSimRTI)**) is a tool for the development and evaluation of **ITS** systems that allows the coupling of different network and mobility simulators. The *Eclipse MOSAIC* modules handle the management tasks, such as synchronization, communication, and data exchange. As shown on Figure 3.2, *Eclipse MOSAIC* uses the concept of *Federates* and *Ambassadors* to couple the simulation tools: each simulator is wrapped into a *Federate* object and linked to an *Ambassador* to enable the connection with the runtime infrastructure.

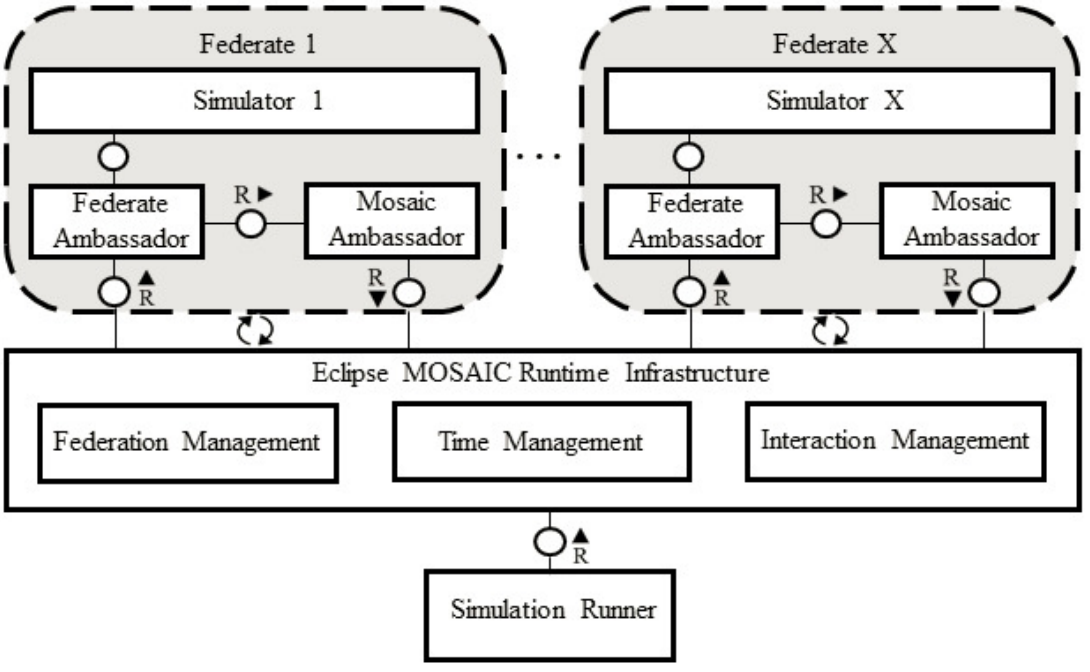


Figure 3.2: *Eclipse MOSAIC Concept* [93]

This tool provides coupling for *Eclipse SUMO*, **OMNeT++** and **ns-3**, as well as other proprietary network and cellular communication simulators.

Vehicles in Network Simulation (VEINS) [94] is yet another open source integrated framework, providing an interface between **SUMO** and **OMNeT++**. **VEINS** framework development was based on **MiXiM**. This framework resorts to **SUMO**'s **TraCI API** to enable the communication between the network and mobility simulators - the **OMNeT++** tools is extended with the ability to control and command vehicles directly (e.g. set their speed). The **VEINS** framework also contains modules that implement the **DSRC** stack (together with the lower *IEEE 802.11p standard* layers). Figure 3.3 illustrates the **VEINS** architecture.

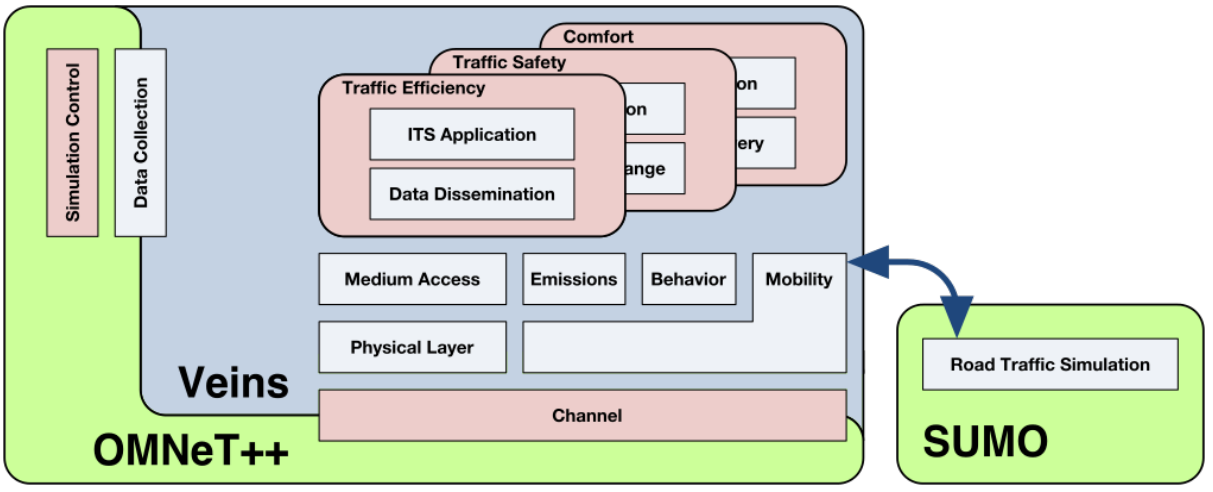


Figure 3.3: *VEINS Architecture* [95]

Artery is an open-source simulation tool that first appeared as an extension to **VEINS** [96], introducing the implementation of the European *ETSI ITS-G5* **V2X** protocol stack and the feature of being able to support multiple applications per simulation setup. However, current releases of *Artery* are independent from **VEINS** - instead, *Artery* now incorporates **VEINS** within the framework as an option for the radio model [97]. The integration of the components that is achieved by *Artery* is illustrated on Figure 3.4.

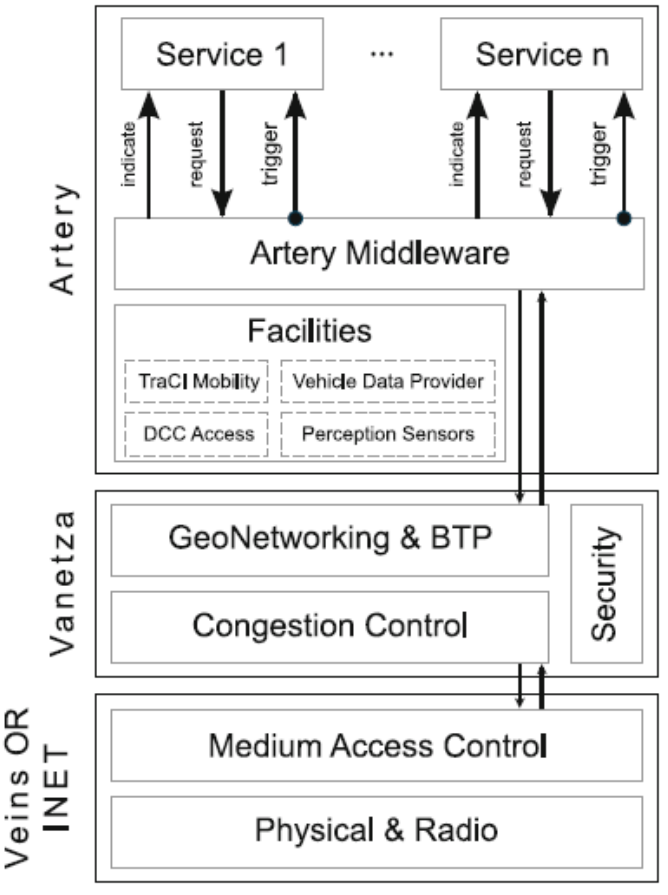


Figure 3.4: *Artery Architecture* [97]

Artery makes use of **VEINS**, **OMNeT++** and *Vanetza* to implement the lower layers of the architecture. Within the *Facilities* layer, the local vehicle data is gathered from the **SUMO** simulator tool. Additionally, it also provides means for **SUMO** and **OMNeT++** to interact (via **SUMO**'s **TraCI**) in order to handle the mobility of the nodes. The stack also enables the support for other communication technologies (e.g. Cellular **V2X**) - *Artery* allows the integration with *SimuLTE* to simulate joint cellular **V2X** scenarios.

3.2 Preliminary Tests on the Simulators

As stated on the previous section, using coupled simulators (**VEINS**, *Artery* or *Eclipse MOSAIC*) is the most interesting solutions for the development of simulation scenarios that involve **V2X** communications.

At this point, and due to the previously acquired experience on other R&D projects, the **VEINS** and *Eclipse MOSAIC* (formerly **VSimRTI**) were already some very well know tools. Some different types of vehicular applications have been previously developed on said frameworks and one is able to conclude that they offer similar functionalities (with similar level of implementation difficulties).

Still, the *Artery* framework is particularly interesting for the fact that it implements the European *ETSI ITS-G5* **V2X** protocol stack through the use of *VANETZA* (unlike **VEINS**, which does not possess *ITS-G5* integration). Hence, it was also important to carry out some preliminary tests on the *Artery* framework to better understand its functionalities. This allows a better comparison with the other two platforms, in order to establish the most suitable framework for the work to be developed.

The first step towards the framework testing was the development of a complete mobility scenario resorting to the *osmWebWizard.py* script that is available in **SUMO**. This script runs on a web browser and allows the selection of a real geographic region of the map to be converted into a **SUMO** scenario. The map is obtained from **OSM**, which provides free real geographic data that is maintained by a community. The database that is kept by **OSM** includes information about roads, points of interest, buildings and other interesting data such as road speed limits or turn restrictions. Using the mentioned script, one is able to select a real-world map and easily generate traffic demand for a series of configurable entities.

In this case, as illustrated on Figure 3.5, a map from the area near the *campus* of Gualtar from the University of Minho (Braga, Portugal) was selected, generating traffic for cars, buses, bikes, motorcycles and also pedestrians using the default parameters.

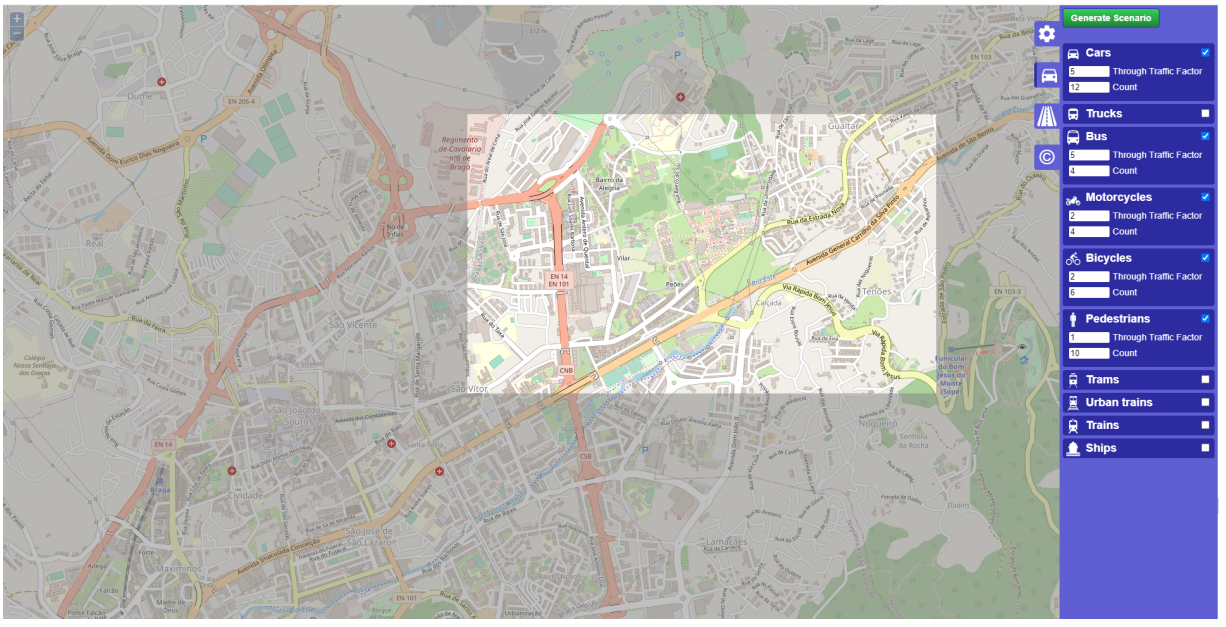


Figure 3.5: Importing Real World Maps and Generating Traffic using the *osmWebWizard*

The script generated the necessary configuration files, which resulted in the complete **SUMO** scenario that is illustrated in Figure 3.6.

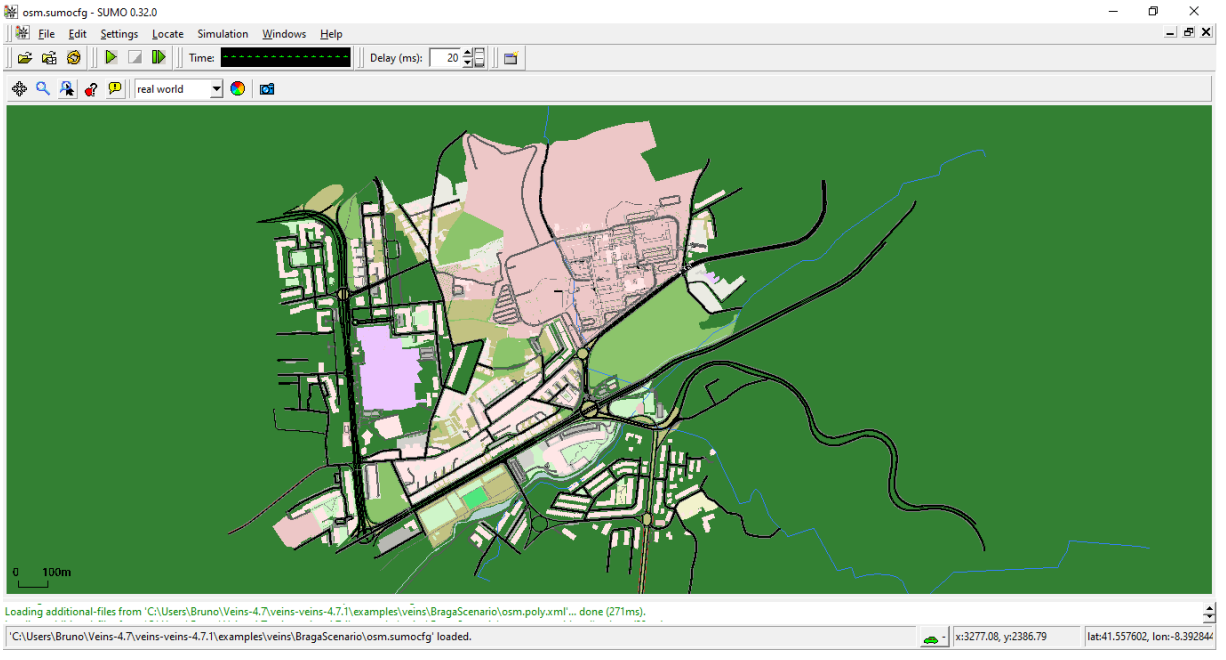


Figure 3.6: SUMO Scenario - University of Minho (Braga)

These configuration files were imported to the *Artery* framework, in order to test a very simple *echo* application on top of the generated map and traffic. The *out-of-the-box* application consisted on nodes simply echoing basic messages that were received on their communication interface.

The scenario was working as intended in a first instance where only regular vehicles were communicating. However, some difficulties soon appeared when configuring nodes positioning (particularly for

the **RSUs** case positioning) and interfaces, mostly because of the lack of documentation available for the tool, which could not be overcame. For that reason, the *Artery* framework was eventually excluded from the list of possible usable tools.

Taken into consideration the previous discussion and the experimental tests, the simulation tools overview (in terms of requirements fulfillment) is presented on Table 3.2.

	VEINS	**Artery**	**Eclipse Mosaic (VSimRTI)**
Realistic Vehicle Mobility	Yes	Yes	Yes
Realistic V2X Communications	Yes	Yes	Yes
Different Road Entities	Yes	Yes	Yes
Support for Multiple Applications	Yes	Yes	Yes
Realistic Scenarios	Yes	Yes	Yes
Real-Time GUI	Yes	Yes	Yes
Simulation Parameterization	Yes	Yes	Yes
Setup and usage	Medium	Hard	Easy
Documentation	Medium	Bad	Good

Table 3.2: Simulation Tools - Requirements Overview

Hence, considering the *Artery's* lack of documentation and that both remaining frameworks have very similar characteristics, the **VEINS** tool was selected to be used, as it was a matter of personal preference. However, at this point, it is important to highlight that the communications stack is the American one (**WAVE**/**DSRC**), which comes by default with **VEINS**. Additionally, **VEINS** does not possess out of the box integration with pedestrians/persons, which limits the establishment of a scenario that is related to **VRUs**.

3.3 Development Environment

Considering the discussion on the previous section, the *Instant Veins 5.0-i1* version was selected to be installed, since that release allows a quick use of the framework, containing all the necessary packages and software already installed:

- Simulation Modules
    - Veins 5.0
    - INET Framework 4.1.1
    - SimuLTE 1.0.1
    - Veins_INET (included with Veins 5.0)

- Software
    - OMNeT++ 5.5.1
    - SUMO 1.2.0

This **VEINS** software package was installed inside a *Docker container* on *Ubuntu 18.04 LTS*. Docker [98] is a platform that allows to package applications with their dependencies (including libraries, system tools, code, and runtime) into standardized units called *containers*, and run them in lightweight isolated environments. Unlike traditional **Virtual Machines (VMs)**, these *containers* run on top of the host **Operating System (OS)** - instead of booting up their own **OS**. These *docker containers* allow an easy deployment of applications across different environments in a consistent way.

At first, this *Docker container* was being used on a *laptop* with limited computation and storage resources. As it was soon perceived that the machine had performance limitations, the *Docker container* was shipped to a more capable *desktop*, with the characteristics identified on Table 3.3:

OS	Windows 10
Central Processing Unit (CPU)	AMD Ryzen 5 5600x
Graphics Processing Unit (GPU)	NVIDIA GeForce GTX 1060 6GB
Random-access Memory (RAM)	32 GB DDR4 1200 MHz
Storage	Samsung SSD 860 EVO 250GB

Table 3.3: Desktop Technical Specifications

Moving the container to a *Windows* machine brought some issues, as the established *Docker container* was prepared for the *Ubuntu* **OS**. At this point, *Docker Desktop* for *Windows* did not allow to run a *Ubuntu Docker container* directly. Hence, three different solutions were identified to overcome this issue:

- Using a *dual-boot* environment (installing *Ubuntu* alongside *Windows* and choose one to initialize at boot);
- Installing an *Ubuntu* **VM** in *Windows* (resorting to software such as *VM VirtualBox*);
- Using the **Windows Subsystem for Linux (WSL)** (version 2.x), which is a *Windows* feature that consists in a full *Linux kernel*, allowing to run *Linux* distributions (*Ubuntu* in this case) directly from *Windows*, without the need for **VMs** or *dual-booting*.

Table 3.4 presents a comparison for some key factors to consider when choosing between **WSL**, **VMs**, and *dual-booting*.

	WSL	**VMs**	**Dual-booting**
Performance	Good performance due to lightweight virtualization and shared kernel with *Windows*	Performance depends on allocated resources (and **VM** configuration)	Native performance, direct hardware access to the operating system
Resource Efficiency	Shares host **OS** resources. Consumes fewer system resources when compared to **VM**	Requires dedicated resources; Depends on the **VM** configuration and allocated resources	Requires dedicated resources - separate partition and storage resources dedicated to each **OS**
OSs Integration	Seamless integration between *Windows* and *Linux*.	Limited integration when compared to **WSL**; Guest **OS** more isolated from host **OS**;	No direct integration; Requires rebooting to switch between **OSs**
File Sharing	Direct and easy access to host files	Requires extra file sharing mechanisms for data exchange between **OSs**	Separate partitions. Requires extra file sharing mechanisms for data exchange between **OSs**
Setup and Maintenance	Easy setup and management; Simple installation; Updates via *Windows Update*	Requires installation and configuration of additional virtualization software on host **OS**; periodic updates for both host and **VM**	Requires separate installation and management of the **OSs**; Updates are handled independently
Accessibility	Quick startup time; Guest **OS** typically starts within seconds	Longer startup time.	Requires rebooting the entire system to switch between **OSs**

Table 3.4: Characteristics Comparison Between **WSL**, **VMs** and *Dual-booting*.

The choice between **WSL**, **VM** or *dual-booting* offers different trade-offs in terms of performance, resource efficiency, integration, file access, setup and maintenance and also accessibility. Still, when comparing **WSL** to **VM**, the first one seems to be the better choice: **WSL** lightweight virtualization offers better performance when compared to running a full **VM**; **WSL** consumes fewer resources, with a more efficient use of **CPU**, memory, and storage; **WSL** allows an easier access to the system files; additionally, it also offers an easier setup and quick access/boot. On the other hand, **VM** offers a more complete isolation or the ability to simulate specific hardware configurations, although these aspects are not important for this specific work.

The choice between **WSL** and *dual-booting* is naturally also dependent on the specific needs and

preferences for the framework that is to be established. For instance, if a system requires isolation or maximum compatibility, using a separated *Linux* installation is the better choice. On the other hand, if convenience and integration/access to files are more important, the **WSL** choice provides a more simplified experience.

In this thesis work case, the ease of access to files is a key aspect, as the datasets are generated from within the simulation framework. Being more familiar with *Microsoft Excel*, and given the fact the *Microsoft Office* suite has some limitations in *Linux*, the data analysis are to be achieved in the *Windows* **OS** (both before and after the **ML** procedures). Naturally, this task chaining (models training/testing and data analysis) is also facilitated if the **ML** modules are installed in the *Windows* **OS**.

Taking this into consideration, the **WSL** solution was chosen, essentially for its convenience. This way, it is pretty straightforward to run the simulations within the simulation framework, access the generated datasets from *Windows* and then run the **ML** procedures and the data analysis as well (without the need to keep rebooting the entire system to switch between the **OSs**).

Figure 3.7 finally presents the complete development environment.

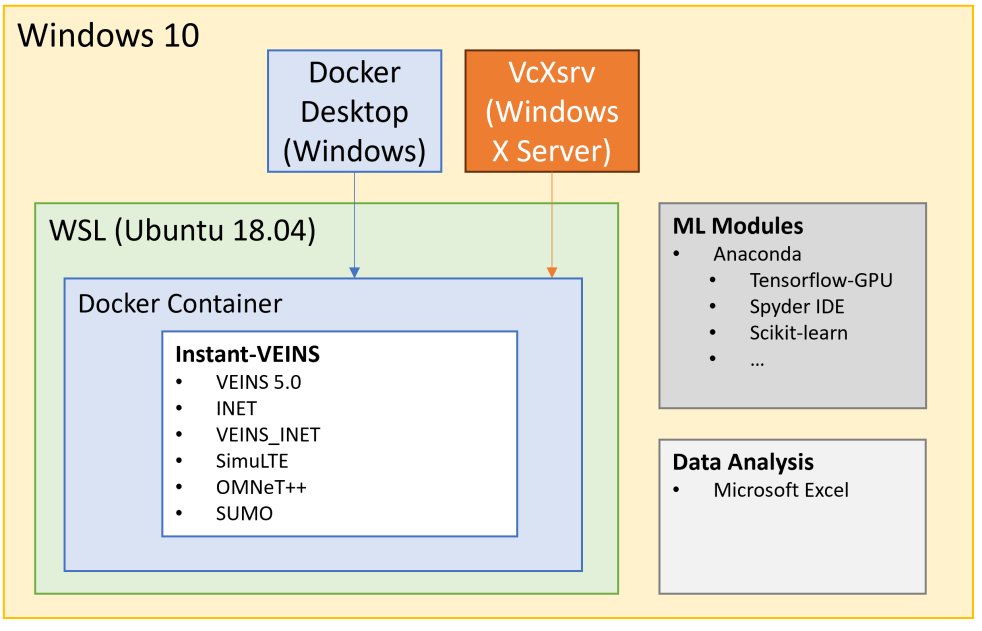


Figure 3.7: Development Environment Setup

In the final setup, the *Docker container* was shipped inside an *Ubuntu* that is running on **WSL**. This container contains all the simulation tools that are necessary to implement the proposed scenario and synthesize the datasets that are to be fed into the **ML** models. The **ML** modules were then installed directly in *Windows*, as well as *Microsoft Excel* for the data analysis. Additionally, this setup required the installation of the *VcXsrv X Server* [99] in *Windows* to provide a **GUI**. This was particularly important to acess the **GUI** from **OMNeT++** and to visually check the simulation scenario in **SUMO**.

Naturally, the final configuration turned out to be more complex than intended at first, when the *laptop* was being used. Still, it is important to highlight that the work that followed showed that the performance overhead that comes from using **WSL** is not noticeable when running the simulations.

3.4 Summary

In order to be able to establish and analyze realistic and valuable ITS simulation scenarios, there is a need to access to large datasets, involving both vehicles and vulnerable users. Currently, no dataset has been identified that specifically includes **VRUs** collision data along with **ITS** standard message exchanges from existing works or projects. Therefore, it is necessary to utilize simulation tools to synthesize such a dataset.

This chapter focused on **ITS** simulation tools, encompassing traffic simulation, network simulation, and coupled simulation environments. From the identified tools, a subset of coupled frameworks was identified for further analysis, based on their potential to model integrated traffic and network dynamics. The selected tools (**VEINS**, *Artery* and *Eclipse MOSAIC*) are then discussed, highlighting their functionalities and characteristics.

Preliminary tests were conducted on the *Artery* framework to assess its suitability for the intended research objectives. Some insights were gained into the strengths and limitations of the simulator, aiding in the selection of an appropriate tool for subsequent research tasks.

Considering that the tools have similar characteristics, the **VEINS** framework was selected, taking into account its good characteristics and previous knowledge and experience.

Furthermore, a development environment, suitable for programmed experimentation and also for processing the outputs of different ITS simulations, involving vehicles and vulnerable users, was also defined.

Chapter 4

System's Architecture and Simulation Scenario

The main goal of this research work is to evaluate the feasibility of using **V2X** data as input for **ML** models to predict collisions for **VRUs**. As previously discussed, predicting the behavior of **VRUs** using more traditional methods is a difficult task to perform. However, as road agents can be equipped with communication capabilities, they are able to exchange a lot of ego and environmental information that can be leveraged by **ML** techniques for the implementation of more advanced safety systems. Hence, **ML** models that are fed with vehicular data have the potential to predict **VRUs** movement and even compute probabilities of occurrence of collisions involving them. The implementation of said prediction system requires heavy computation resources and large amounts of storage for training and testing the model. Furthermore, the deployment and usage of the model/application also requires very low latencies, to allow the users enough time to safely actuate on collision predictions. Being a safety application, is is critical that the prediction is achieved within a reasonable time - including the exchange of information and also real-time analysis and treatment of such data.

A very suitable solution is resorting to the *Fog Computing* paradigm. The usage of *Fog Computing* introduces great benefits in terms of low-latency and mobility, given that it performs tasks of computation, communication and storage near the edge users of the network [100]. By using a distributed network of devices, this paradigm of computing (in comparison to more traditional *cloud* architectures) brings applications and services from the *cloud* to the *edge* of the network, greatly reducing the transfer times and meeting the demands of real-time applications (such as the short term prediction of collisions). Gomes et al. [101] presented an interesting and extensive survey on time-sensitive applications in fog computing environments, classifying the surveyed articles into five categories: *Fog Computing Concept, Faster Response, Low Latency, Data Streaming Application,* and *Time, Delay or Latency Constraint*.

An example of applying the *Fog Computing* concept to **ITS** can be found in [102], where Liu et al. propose a hierarchical system architecture, using both software **Software-Defined Networking (SDN)** and *Fog Computing* in **Internet of Vehicles (IoV)** paradigms. The architecture consists of four layers:

application layer, control layer, virtualization layer, and data layer. The system was tested by implementing two real-world environment prototype services:

See Through This service shares a real-time view of a front vehicle to its following vehicles. The vehicle that intends to share its view registers at the **SDN** controller using **LTE**. Then, based on the vehicle topology (and registered services), the **SDN** controller notifies available services to particular vehicles, via control messages. Vehicles are then able to request the services from the **SDN** controller using **LTE**. Once the service starts, the video can be streamed from the provider to the requesting vehicles using **DSRC** at the fog layer.

Collision Warning This service triggers warning messages if a potential collision between two vehicles is detected. The **SDN** controller communicates with vehicles via **LTE**. To support a large-scale and real-time service, the computation and communication workload is offloaded to the fog server. The vehicle constantly sends up-to-date information using **DSRC** to the fog server (10Hz), which then processes the data and estimates if there is a risk of collision - in positive cases, the warning message is triggered and sent to the vehicles - which are then displayed on a **Human-Machine Interface (HMI)** (along with sound and vibration).

Another example of the application of *Fog Computing* in **ITS** is introduced by Alemneh et al. [103], which present an infrastructure-less architecture (fog-based) named PV-Alert (Pedestrian-Vehicle Alert). In this architecture, the fog nodes process delay sensitive data (that are obtained from smartphones) and generates alerts for pedestrians and drivers when an imminent collision is detected - the collected data is also sent to the cloud for further analysis. The proposed solution was evaluated using the ns-3 and SUMO simulation tools. The architecture was compared to other (smartphone) **VRU**-related safety architectures, and the results showed that it was able to scale well and be reliable, while also providing low latencies.

This paradigm of computing brings applications and services from the *cloud* to the *edge* of the network, greatly reducing the transfer times and meeting the demands of real-time applications. In the proposed use-case context, this means that the **V2X** communications data (including vehicle's speed, acceleration, position, etc.) can be processed closer to the network edge, which allows the prediction algorithms to quickly analyze the data and make decisions. Also, by adopting a multi-layered architecture, it is possible to use an intermediate layer (the *Fog Layer*) to connect the vehicles (network's edge) and the centralized *cloud* server. Hence, on top of the the low-latency data processing, it is also possible to reduce the burden on the network infrastructure (as only aggregated relevant data is transmitted to the *cloud*). Moreover, the ability to distribute the computation in the *Fog Layer* also allows a more scalable solution for

the collision prediction across several road points, without the need to rely on a unique centralized *cloud* server.

In summary, by using a multi-layered architecture inspired on *Fog Computing*, it is possible to deploy **ML** prediction models that can be executed in real-time, providing timely warnings to drivers when a possible collision is predicted.

Furthermore, and because no datasets were found, it becomes necessary to generate and synthesize new datasets using simulation. For that reason, there is also a need to establish a proper use case scenario.

This chapter first presents the defined system's architecture, based on a *Fog Computing* approach. Then, this chapter details the defined use case, which is based on **ETSI** standards. Additionally, the implementation of the use case on the simulators is also discussed.

4.1 System's Architecture

The system's architecture is based on a multilayer hierarchical approach. The architecture design consists on three different levels, where the road entities play different roles and also have different functionalities, according to their needs and capabilities (in terms computation power, storage capacity, delay, etc.) and needs. The main purpose of the proposed design is to place the heaviest computational and less time-critical functions at the upper layer, and the lighter operations at the lower layer (while the middle layer is a middle ground). By allocating the different functionalities at each layer according to the road entities characteristics, rapid responses can be provided to the lower layers.

Figure 4.1 shows the hierarchical architecture of the proposed system, based on a *Fog Computing* approach. As illustrated on the figure, the architecture is composed of three hierarchical layers, which are described next.

Edge Layer This layer is the closest to the end users (drivers/vehicles), which are typically widely distributed in geographical terms. Most of the regular vehicles that travel on the road are equipped with a large number of sensors. The information that is collected by such sensors can be shared with other entities on the road using **OBUs** with communication capabilities. This information may be useful to other users on this layer and particularly for **RSUs** on the *Fog Layer*.

Fog Layer This layer is situated on the *edge* of the core network. The nodes on this layer (*Fog Nodes*) are also widely distributed - for instance, they can be located in every intersection on the road. They are responsible for interconnecting the *cloud* and the *end users*, which aim to obtain services

from these nodes. Additionally, they have ample capabilities to do heavier computation and to transmit/receive data (and also to process and store it). The implementation of low-latency and real-time applications can be achieved on this layer. Thus, this is the ideal place to deploy the **ML** models for prediction of collisions - the application can receive the data that is disseminated from the surrounding vehicles, treat it (aggregate it) and use it for the prediction of collisions. The *Fog Nodes* also connect with the top layer (*Cloud Layer*) in order to obtain more powerful computing and storage capabilities.

Cloud Layer The *Cloud Layer* consists of servers and storage devices that possess great performance capabilities (powerful computation and ability to store huge amounts of data) and can provide several services. In this case (a service for the prediction of collisions related to **VRUs**), this layer makes use of its ability to store the data that is sent from the *Fog Nodes* on the underlying layer and use it to train (and eventually retrain) the **ML** models.

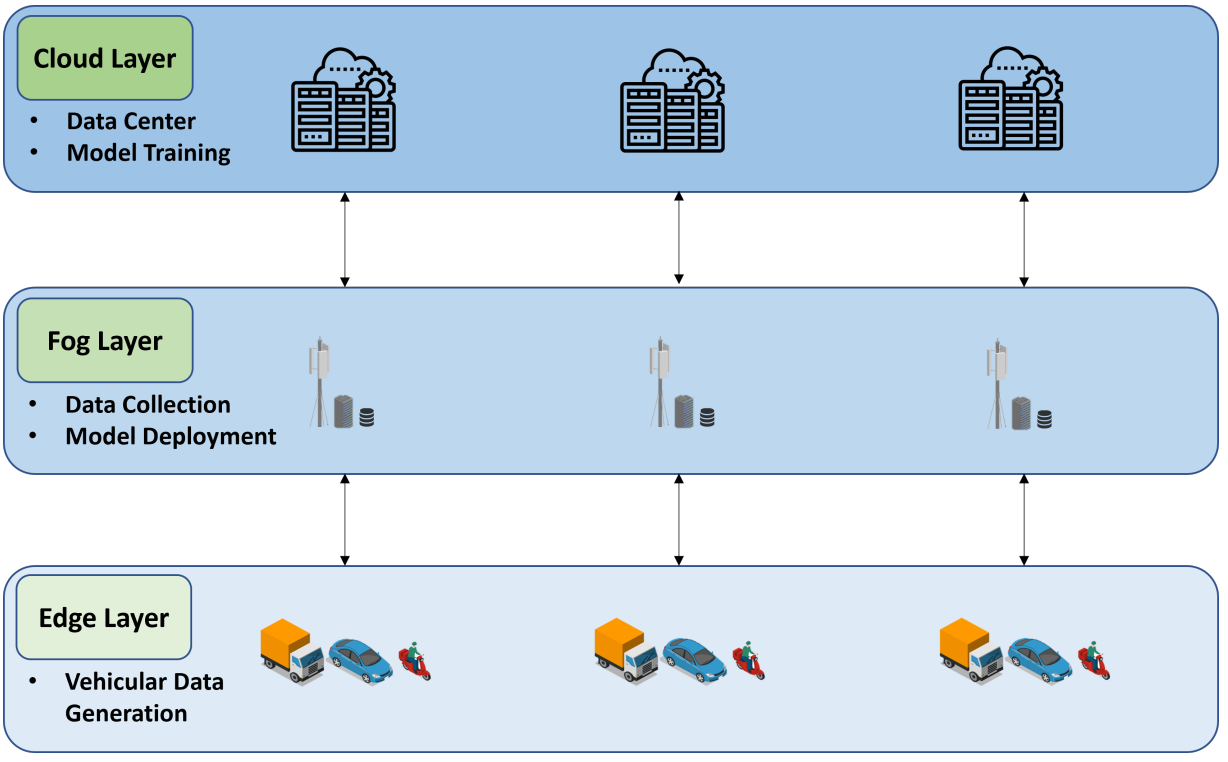


Figure 4.1: System's Hierarchical Architecture (Based on *Fog Computing*)

An example of the deployment of the complete system is illustrated in Figure 4.2.

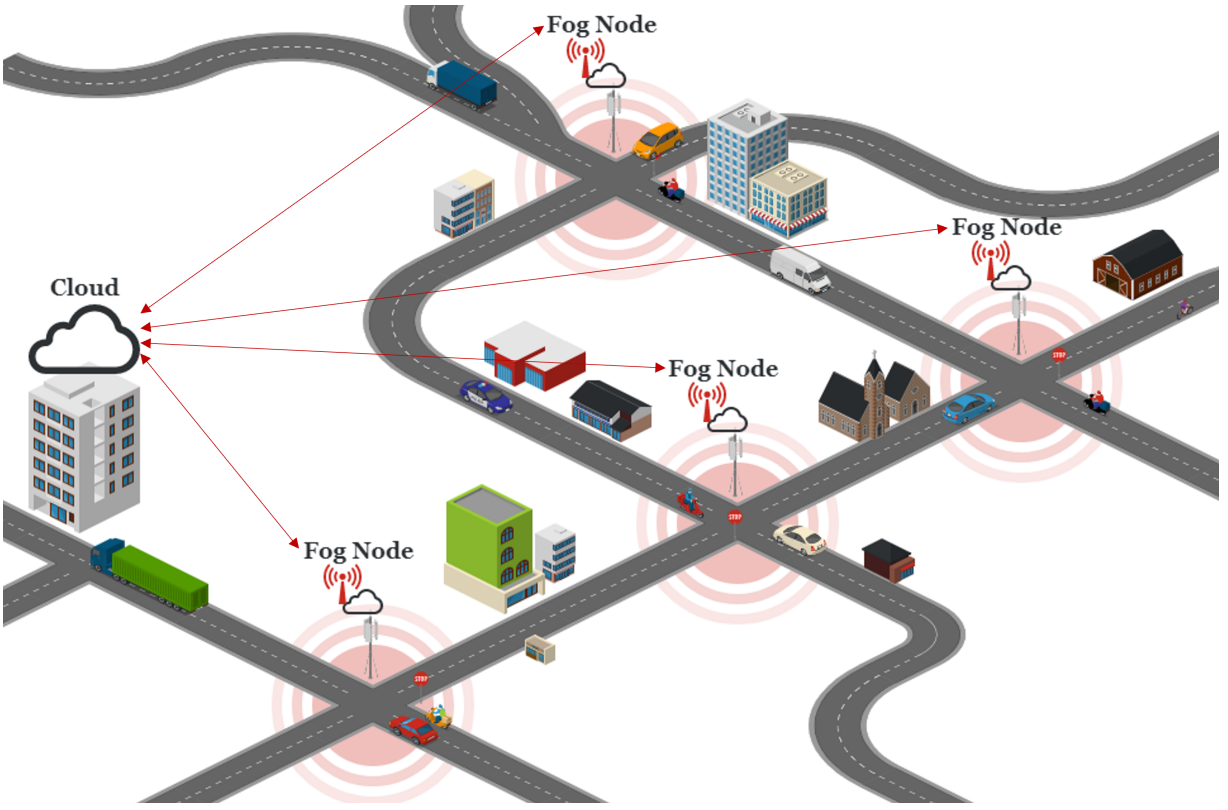


Figure 4.2: Fog Architecture - The Prediction of Collisions Related to **VRUs** Use Case

.

In this architecture, each end device (vehicle) connects with a *Fog Nodes* - that is located on a road intersection - using wireless access technologies (e.g. IEEE 802.11p). The *Fog Nodes* can be connected to the *Cloud* by using the **IP** core network.

The application of *predicting collisions related to* **VRUs** *using* **ML** can be divided into two stages - the offline stage (which consists on the model's training and testing) and the online stage (when the model is deployed). These stages are further explained next:

Offline On a first offline stage, it is important to collect sufficient data to parametrize, train/test and deploy the **ML** models, which will later be used for the prediction of the collisions. The end users (vehicles) broadcast data with fast rates (e.g., using **BSMs** every 100 *ms*) that can be collected by the *Fog Node* that is placed on an intersection (acting as a **RSU**). This data can then be treated and aggregated to be sent to the *Cloud*, which possesses better capabilities to store the huge amount of data and use it to train the models - on this case, in a supervised manner. Thus, the *Fog Node* should also complement the collected data with information related to collisions history, which is useful for the training process. When the process of training and testing the model is finished, the resulting model and weights can be sent back to the corresponding *Fog Node*, in order to be

deployed. This process should be repeated for each *Fog Node*.

Online When the first stage is accomplished, a model can finally be deployed on a *Fog Node* in order to start the prediction of the collisions. The *Fog Node* collects the broadcasted messages, treats them and uses that compilation of data as input for the prediction. The aggregated data resulting from this process is transmitted to the *cloud* for storage and subsequent utilization in retraining the model, in order to allow a continuous tuning of the model. The weights resulting from the retraining process are sent back to the *Fog Node* in order to update the model. When a collision is predicted by a *Fog Node*, a message can be disseminated to the relevant end users, in order to allow them to make an appropriate decision (e.g., perform an emergency brake).

4.2 Use Case Description

Although the first set of tests on *Artery* (discussed before on section 3.2) was important to get acquainted with the simulation tools and how their configuration/parameterization works, it soon became clear that the first developed scenario was problematic - too many variables on both traffic and communications, lack of scenario control, difficulties in understanding/locating the incidents, simulation performance issues, and so on. This led to the decision of establishing a smaller scenario to better control the environment, which results in lighter simulations which are faster to design, develop, test and debug.

In order to establish the scenario and its requirements, several use cases from **ETSI** standards were analyzed, such as *Collision Risk Warning from RSU* from [104], *Turning collision risk warning* from [105] and, with particular emphasis, the **Scooter/Bicyclist Safety with Turning Vehicle** from [106].

This use case, illustrated in Figure 4.3, is described as a critical (and typical) traffic situation where a vehicles makes a turn on an intersection and oversees a bicycle/scooter, which makes a collision between the agents possible. On this use case, the scooter/bicycle is not equipped with any kind of device, the vehicle possesses communication capabilities and an **HMI** and there is an **RSU** present on the intersection that is equipped with both a communication device and some kind of sensor (e.g. a camera). This sensor is used to detect both the **VRUs** and the vehicles, and the information is used to predict their path and compute/detect possible collisions. If an imminent collision is detected, the **RSU** broadcasts warning messages to the vehicles in the area. The vehicle, upon receiving the collision avoidance message, takes appropriate actions to avoid or mitigate the collision. The standard also presents alternative collision situations that are shown in Figure 4.4.

As is stated on the standard, the path prediction/collision detection plays a very important role in

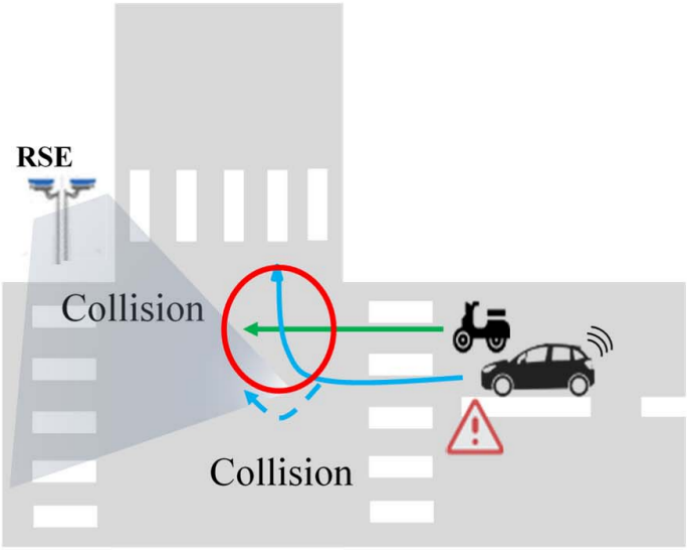


Figure 4.3: *Scooter/Bicyclist Safety with Turning Vehicle Use Case* [106]

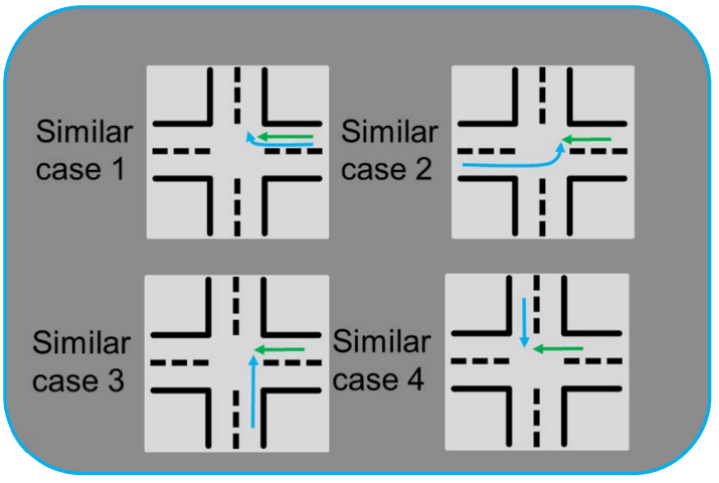


Figure 4.4: *Scooter/Bicyclist Safety with Turning Vehicle Use Case* - Possible collision situations [106]

this use case. On this example, the information that is used to feed the detection mechanisms is collected via sensors (e.g., Camera, RADAR, LIDAR).

However, this type of solutions may eventually perform poorly in situations where line of sight is non-existent or limited (e.g., the **VRU** is in a blind spot, behind a parked vehicle). This situation is aggravated due to their smaller size and high mobility, which make them harder to be detected. Hence, the implementation of such systems resorting to wireless communications between **VRUs** and regular vehicles/infrastructure may have a great impact on the general safety of those road agents.

For that reason, it is intended to study the feasibility of resorting to vehicular communications to feed such detection mechanisms on the proposed scenario. Naturally, this implementation is only possible if these users possess communication capabilities that allow communication between themselves and other road agents. Although entities such as bicycles may possess communication capabilities (for instance,

using the driver's smartphones), it is unlikely that they may communicate directly with other vehicles on the road - which are typically equipped with IEEE 802.11p transmission capabilities. On the other hand, motorcycles/scooters are fairly easy to equip with **OBUs** that possess communication technologies similar to regular vehicles.

Hence, from the **VRUs** group, motorcycles make the better subject to study the potential use of automatic solutions for collision prediction, which may *passively* improve their safety. Even the detection of collisions may, for instance, trigger emergency services that may reduce the severity of injuries after the collisions or even save lives. Even if motorcycles and regular passenger vehicles are not equipped with sophisticated safety systems that enable them to detect that they are involved in collisions, it is possible to resort to **RSUs** (infrastructure units on road environments) to implement such advanced safety systems (as the standard's use case suggests).

Taking into consideration the previous discussion, the proposed use case is described below:

Description

A passenger vehicle makes a turn on an intersection and oversees an approaching motorcycle, that intends to go straight on the road, which results in a possible collision.

Actors

In this scenario there are the following actors:

- Passenger Vehicle - Equipped with an **OBU** (with IEEE 802.11p).
- Motorcycle - Equipped with an **OBU** (with IEEE 802.11p).
- **RSU** - Equipped with IEEE 802.11p and automatic **ML** mechanisms for the prediction of collisions.

Pre-conditions

Passenger vehicles, motorcycles and one **RSU** which are able to receive and broadcast standard vehicular messages. **RSU** possesses mechanisms for the automatic prediction of collisions between passenger vehicles and motorcycles.

Triggers

- Motorcycle and passenger vehicle are close to the intersection.

- Motorcycle goes straight through the intersection.
- Vehicles makes a turn on the intersection, crossing roads with the motorcycle's route.

Normal flow

- Passenger vehicles and motorcycles broadcast ego information using standard vehicular messages.
- The **RSU** receives the standard vehicular messages and collects the data from those agents.
- The **RSU** performs collision prediction using automatic **ML** mechanisms.
- The **RSU** broadcasts collision warning messages to vehicles in the area.
- The vehicles receive the collision avoidance messages.

Post-conditions

Vehicles are alerted of potential collisions and take appropriate actions to avoid or minimize its effects.

4.3 Simulation Scenario Setup

Taking into consideration the scenario described on the previous section, a new traffic scenario was designed from scratch using the *netedit* tool provided in the **SUMO** package. This tool is a GUI application that allows the creation of new networks from scratch (and also their editing) and traffic demand (routes and trips for the different kinds of vehicles). This tool allows users to easily design road networks and to obtain the *.xml* files which are used as input on the **SUMO** simulator.

The design of the scenario is illustrated in Figure 4.5.

The **SUMO** tool aims to be collision-free when using the default mobility model (*Krauss*). For that reason, some configuration is needed in order to create collisions on the simulations. Hence, there was a need to follow the guide available in **SUMO**'s documentation [107] in order to simulate collisions (namely the *Collisions at Intersections* section). This guide suggests the configuration of junction model parameters on the *passenger* vehicles in order to implement collisions at intersections. In this case, the following parameters were tweaked:

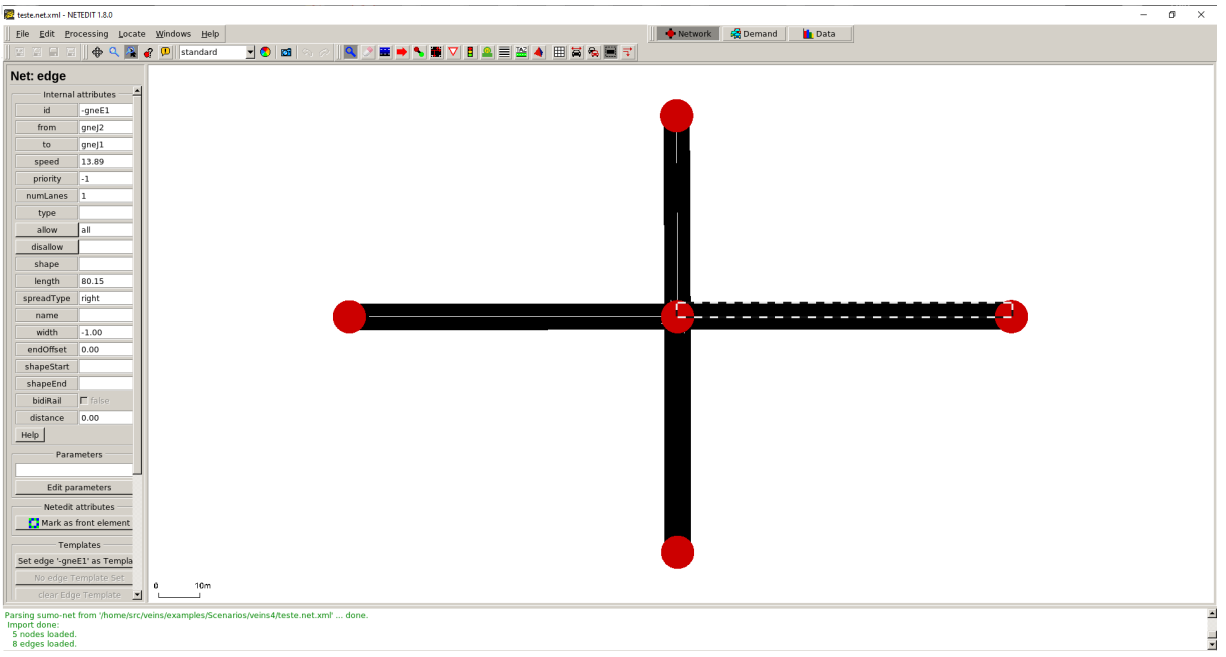


Figure 4.5: *netedit* Tool - Building the Intersection Scenario

jmIgnoreFoeProb / jmIgnoreFoeSpeed These parameters are used to ignore agents that are approaching the junction below the given speed with the given probability. The chosen values were *mIgnoreFoeProb = 1* and *jmIgnoreFoeSpeed = 100*. This results in ignoring all the agents approaching with 100% probability.

jmTimegapMinor This parameter sets the safety time gap when passing an intersection without priority. In this case, the chosen value was *jmTimegapMinor = 0*, the minimum possible value (in other words, no safety time gap).

Hence, the usage of unsafe values for these parameters results in passenger vehicles ignoring foes arriving at the intersection and even the safety time gaps when they have no priority. In this particular case, there was also a need to reduced the *emergencyDeceleration* default values, since vehicles were breaking in these emergency situations but were still not hitting using only said parameters. Lowering this value to *4.5* finally resulted in collisions.

In this case, the *motorcycle* vehicles on the simulation use the default values, as there was no need to change them. In that sense, the *passenger* vehicles are the ones causing the collisions.

Those parameters were defined on the *scenario*.rou.xml* configuration files of **SUMO**, which are used to parametrize *Vehicles, Vehicle Types,* and *Routes*.

In addition to the discussed configuration, there was also a need to set the *–collision.check-junctions* option to true, so that **SUMO** registers collisions on road junctions. Also, regarding collisions, **SUMO** tracks the distances (gaps) between vehicles and, when the gap is below a given threshold, it registers

a collision. In this case, by setting the option *–collision.mingap-factor = 0*, only physical collisions are considered.

Furthermore, and in order to simulate the effects of the collision itself, the vehicles involved on the collision halt on the lane for a fixed time. Since the remaining vehicles proceed normally according to their movement model, this results in traffic jams while the vehicles are stopped, as illustrated in Figure 4.6 below.

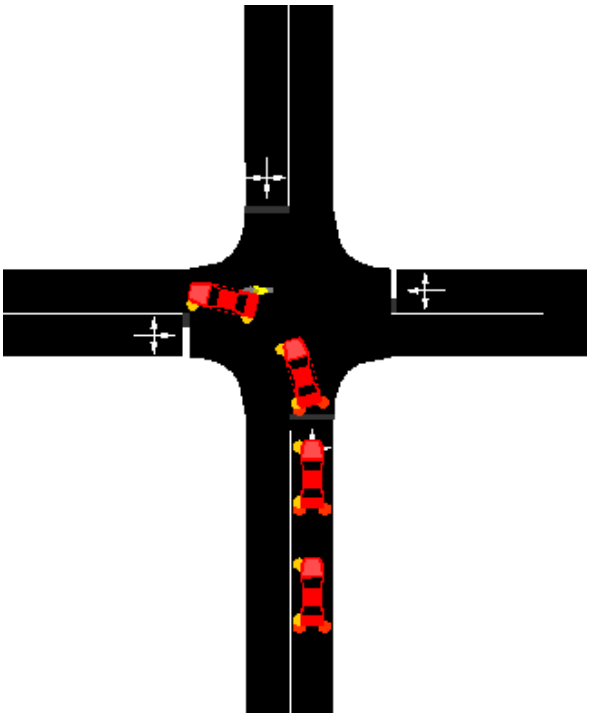


Figure 4.6: Traffic Jams Caused by Collisions on SUMO

In this case, each simulation run lasts for 24 hours of simulation time (86400 seconds), and the collision event is configured for lasting 500s (using the *collision.stoptime* parameter). When the *stop time* is over, the vehicles are removed from the simulation (as if they were moved to the side of the road), using the *collision.action = remove* parameter. These parameters were defined on the **.sumo.cfg* configuration files of **SUMO**.

Three different mobility instances were implemented on the intersection scenario, in order to simulate three different collision situations. These situations were based on the cases two, three and four from the previously discussed use case from [106] (shown in Figure 4.4).

The first use case that is depicted on the figure, where the passenger vehicles and the motorcycles are traveling on adjacent lanes, was not possible to implement on **SUMO**, since these kind of collisions (from adjacent lanes) are very hard to simulate.

The first implemented scenario case was the second collision case (from now on designated as

Scenario A), illustrated in Figure 4.7.

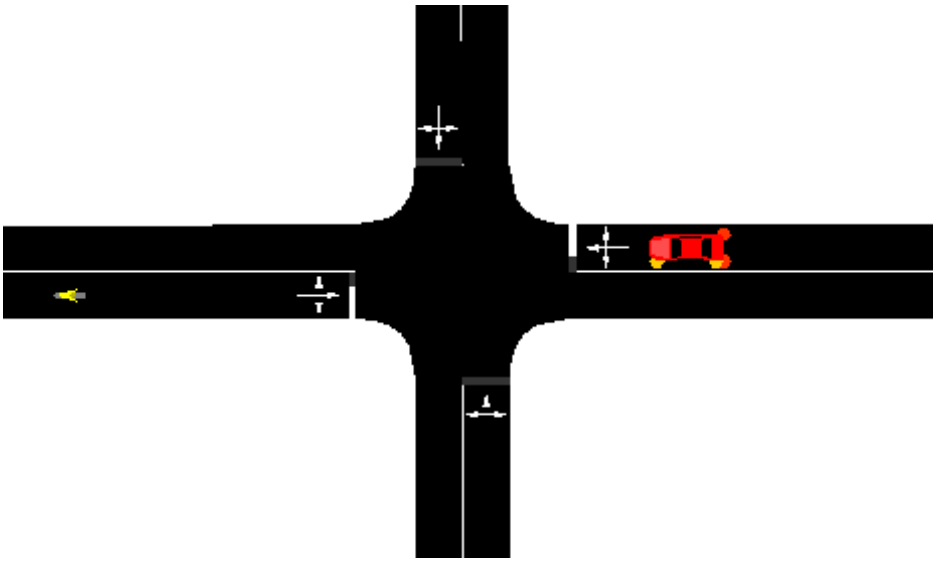


Figure 4.7: **SUMO** Scenario A (Cropped) - Collision Case 2

The scenario is not exactly as illustrated on the **ETSI**'s standard - for some reason, which could not be identified, the **VEINS** framework would sometimes crash (and close in a forced way) when some passenger vehicles would arrive at the end of the road's edge on the top side. The output logs did not allow to understand the exact cause of the issue. For that reason, the mobility pattern had to be rotated - avoiding the issue by simply not using the top edge of the road on the scenario. Still, the traffic behaves in a similar way - the passenger vehicles makes a left turn on the intersection while the motorcycle comes from the opposite side and follows straight.

The third collision case (from now on designated as Scenario B) is shown in Figure 4.8. On this case, the passenger vehicle makes a left turn on the intersection, while the motorcycle comes from the right and follows straight.

Finally, in the last implemented case (collision case 4 - Scenario C), the passenger vehicle makes a right turn, while the motorcycle comes from the left and follows straight. This fourth collision case is illustrated in Figure 4.9.

Initially, the parameters were set so that after two vehicles collided, they would stop on the road for a pre-defined time and later return to their normal behavior and proceed on the road on the next edge of the route. However, in some occasions, when the passenger vehicle that was involved in the collisions tried to move after the stop time, the next passenger vehicle on the traffic queue would collide again from the rear, causing another collision. This kept happening for the upcoming passenger vehicles on the traffic queue as well, causing a weird (and undesired) rear collision chain. The reason behind this behavior could not be found.

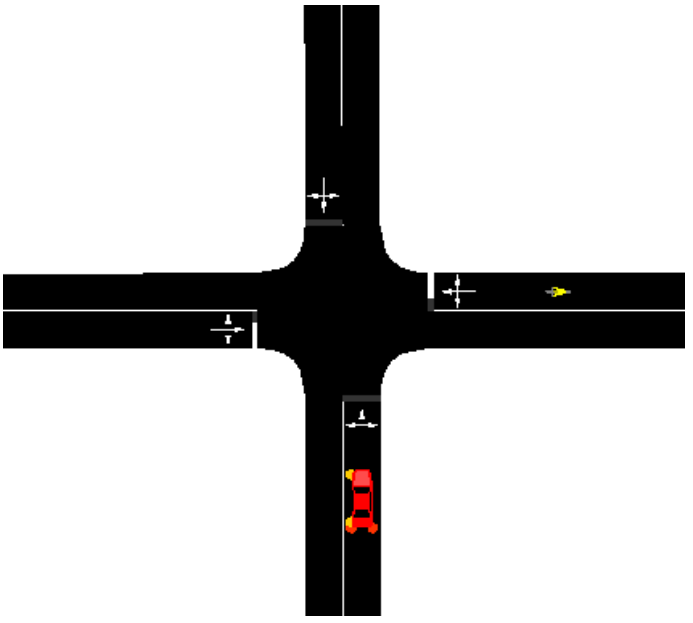


Figure 4.8: **SUMO** Scenario B (Cropped) - Collision Case 3

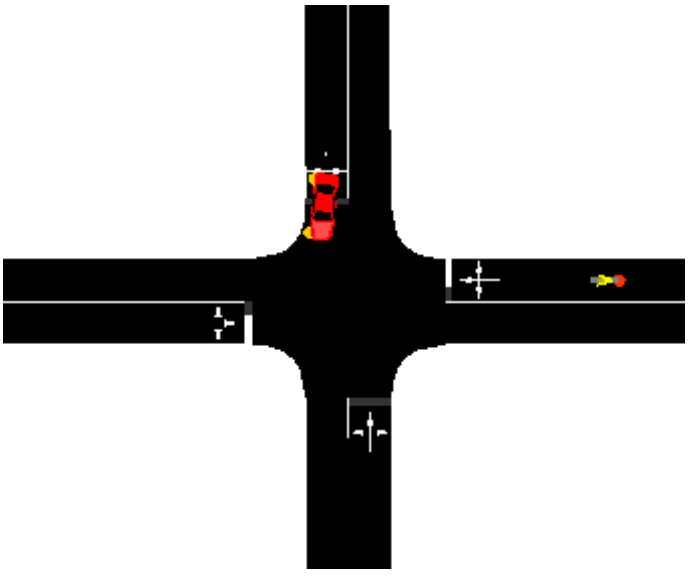


Figure 4.9: **SUMO** Scenario C (Cropped) - Collision Case 4

To address this issue, the vehicles involved in the collision are removed from the simulation, as if they were moved to the side of the road in a real collision situation. The remaining vehicles then proceed normally on their predefined routes. This strategy worked well for both Scenario A and Scenario B were all chain collisions were avoided. However, on Scenario C, there were still some chain collisions happening, which unfortunately could not be overcame (although they were rare). This unrealistic behavior of vehicles hitting each other immediately after starting moving (in a weird chain-collision loop) meant that the data that was gathered on the Scenario C simulations was not usable for the collision prediction system (which will be further discussed ahead).

4.3.1 Communications Setup

In terms of communications, all nodes on the simulation (both passenger and motorcycle vehicles and also **RSU** that is placed on the intersection) are equipped with the **DSRC**/IEEE 802.11p stack to communicate, using a special kind of beacons called **BSMs**. The **BSMs** beacons [9] are typically used to exchange safety data regarding the vehicles state. They are broadcasted frequently to surrounding vehicles with data content as required by safety and other applications.

On the simulation, the nodes are exchanging these **BSM**-like beacons with a 10Hz rate (typical when congestion control algorithms do not require a reduced rate) and the messages contents are filled in **VEINS** with information that is extracted from the traffic simulator **SUMO** via the **TraCI API**.

The core data frame of **BSMs** [9] (often referred to as *BSM Part One*), contains the critical core data elements necessary in every issued message, and are described on Table 4.1.

Unfortunately, it is not possible to obtain all the required data that a real **BSM** beacon usually possesses when using the simulators (which is naturally a limitation when comparing to a real scenario data collection). Still, even when using **SUMO**, most of the "core" data elements are currently being exchanged.

The simulation scenarios beacons are being filled with the following information:

- Station ID
- Longitude
- Latitude
- Elevation/Altitude
- Heading
- Speed
- Acceleration
- Vehicle Length
- Vehicle Width
- Vehicle Type (from Part II of **BSM**)

Type	Description
MsgCount	Provides a sequence number within a stream of messages from the same sender.
TemporaryID	Four octet random device identifier. For OBU devices, this value changes periodically for anonymity reasons.
DSecond	Time information.
Latitude	Geographic latitude of an object.
Longitude	Geographic longitude of an object.
Elevation	Geographic position above or below the reference ellipsoid.
PositionAccuracy	Parameters of quality used to model the accuracy of the positional determination with respect to each given axis.
TransmissionState	Provides the current state of the vehicle transmission.
Speed	Represents the vehicle speed.
Heading	Provides the current heading of the sending device.
SteeringWheelAngle	Represents the current steering angle of the steering wheel.
AccelerationSet4Way	Set of acceleration values in three orthogonal directions of the vehicle and with yaw rotation rates, expressed as a structure.
BrakeSystemStatus	Information about the current brake and system control activity of the vehicle. .
VehicleSize	Represents the vehicle length and vehicle width.

Table 4.1: **BSM** Core Data - Part One

Vehicle Type was also added to the scenarios beacons, although it does not make part of the core elements of **BSMs**, since it was easy to obtain that information resorting to the **API** available on **VEINS**. Furthermore, this particular information was useful for the initial tests regarding **ML** (discussed next on subsection 5.1.1).

The main default (out of the box) communication parameters for the network interface cards were used on the simulator. These parameters are presented on Table 4.2.

The simulation includes an *analog model* and a designated *decider*. The *analog model* specifies how the signal attenuation is calculated (compute the decrease in the power of a radio signal as it propagates away from the transmitter). In this case, the *SimplePathLossModel* is used. The *decider* decides if an incoming packet is to be received or not (based for instance on the **Signal to Noise Ratio (SNR)** threshold and on a defined bit error rate) - the simulation used the *Decider80211p* decider.

Network Interface Card Parameters	
TxPower	20 mW
BitRate	6 Mbps
minPowerLevel	-110 dBm
noiseFloor	-98 dBm
Decider	
Decider80211p	centerFrequency = 5.89 GHz
Analogue Model	
SimplePathLossModel	α = 2.0

Table 4.2: **OMNeT++** Network Interface Card - **IEEE** 802.11p Specific Parameters

4.3.2 Collecting Data for the Datasets

In order to compile the vehicle's **V2X** data (to be later processed and analyzed), it was necessary to simulate the scenarios that were previously introduced in section 4.2, collecting all the data exchanged. These *datasets* consist of the collection of beacons (**BSM**) that both passenger and motorcycle vehicles are exchanged using **DSRC**.

The collected and compiled data is to be used to feed the ML scripts aimed at the collisions detection and prediction. Each simulation *run* is executed for a total of 24h of *simulation time* and uses a different simulation *seed*. Using a different seed causes randomness when inserting vehicles on the simulation, which leads to different mobility behaviors for each *run* (and consequently different simulation results).

In order to build the *datasets*, the messages can be collected in two different ways:

1. Each individual vehicle writes to a **Comma Separated Value (CSV)** file (named after the vehicles ID on the simulation) all the generated messages (each line as a record). A simple bash script was built to compile all the generate files into a single **CSV** when the simulation is finished (hence building the *dataset* that is loaded later into the **ML** script).

2. The **RSU** that is installed in the intersection collects and saves all the received messages in that area to a single **CSV** file, line by line (each record is a message). In these particular scenarios, all nodes are at all times within communication range and able to exchange messages.

At this point, only the second collection method has been selected and used; in fact, apart from being simpler to implement, it is also the most reasonable approach for the purpose and also the one that operationally fits into the overall architecture design. In fact, the **V2X** messages which are relevant for the datasets constructs are generated by the vehicles around the geographic area where the **RSU** is located

and also where collisions are probable to take place.

On top of the aforementioned beacon fields, there is also an extra *inCollision* field (further discussed ahead). The final format of the stored information is:

Station ID Identification of the vehicle that sent the message.

Longitude Longitude of the sender vehicle in degrees.

Latitude Latitude of the sender vehicle in degrees.

Elevation Elevation of the vehicle in meters.

Heading Heading of the sender vehicle in degrees.

Speed Speed of the sender vehicle in m/s.

Acceleration Acceleration of the sender vehicle in m/s^2.

Vehicle Length Length of the sender vehicle in meters.

Vehicle Width Width of the sender vehicle in meters.

Sending Time Timestamp at which the message was generated in nanoseconds.

Vehicle Type Identification of the type of vehicle that sent the message.

In Collision Boolean stating if the sender vehicle is involved in a collision or not when the message was generated.

The final datasets compilation is achieved by performing different simulation runs for both Scenario A and Scenario B. Each simulation run uses a different simulation seed, which results in different mobility patterns in each run. This leads to non-deterministic and different number of collisions between vehicles and **VRUs**, occurring at different instants of time and with different traffic characteristics.

Firstly, for the collision detection system, ten different runs were used for each scenario. These ten resulting datasets were afterwards aggregated into three final datasets: one for training (80% of total data), one for validation (10%) and finally one for testing (10%) the collision detection **ML** approach. The datasets were concatenated in a sequential manner: from seed 0 to seed 7 for the training dataset; seed 8 for validation; seed 9 for testing.

Later, a total of fifty runs were used to compiled the three final datasets which have been used to evaluate the **ML** collision prediction system: 80% for training (from seed 0 to seed 39), 10% for validation

(from seed 40 to seed 44) and 10% for testing (from seed 45 to seed 49). The need to increase the number of simulation runs to compile the datasets was related to the fact that the amount of data was insufficient for the prediction models to converge when trying to predict the collisions (this issue is discussed further ahead). These final synthesized datasets (the basis for the collision prediction system) were submitted to a public on-line platform and can be accessed in https://zenodo.org/record/7376770 [108].

4.4 Summary

In this chapter, the design of a *Fog computing*-inspired system architecture was discussed. The chapter starts by explaining the fundamental principles of *Fog Computing* and how they apply to the system architecture, which aims to enhance efficiency and responsiveness - by decentralizing computing resources closer to the data sources (the vehicles), it's possible to deploy real-time collision prediction models to provide timely warnings to drivers.

A specific use case (derived from relevant **ETSI** standards) was then described in detail. This use case served as a practical scenario to demonstrate the system's functionalities.

This use case was then implemented using the **VEINS** simulator, combining **SUMO** for traffic modeling and **OMNeT++** for network simulation. The setup and configuration of these tools were detailed to provide insights into the simulation process.

Finally, the chapter delved into the process of data collection from the simulations to synthesize the datasets, discussing which concrete data is treated and how it can be collected. By collecting the messages data (**BSM** beacons) that is generated during simulations, meaningful datasets were synthesized to enable subsequent analysis and validation of the proposed system architecture.

Chapter 5

Machine Learning Techniques for Vulnerable Road Users Safety

ML may play a significant role in enhancing **VRUs** safety, which are typically more exposed to incidents when compared to other road users traveling on regular vehicles. This work considers that, by leveraging **V2X** data using **ML** algorithms and techniques, it may be possible to develop safety systems for their protection - for instance, a system for prediction and warning of potential collisions. Such systems may be able to learn from patterns and behaviors observed in historical data and use real-time inputs to accurately assess collision risks. Such predictions, if achieved in a timely manner, may also allow the sending of warnings to drivers - which ultimately may enable them to take preemptive safety actions to avoid the collision.

One of the first steps towards the analysis of the quality of the synthesized datasets and to explore some ML predicting techniques was the development of a simple system that aimed to classify the originator of the **V2X** messages. In this case, try to predict if the message was originated by a passenger vehicle or a motorcycle. This preliminary work, that is discussed on section 5.1, was important for the first interactions with the **ML** framework tools. Furthermore, this initial work was also important to reach the conclusion that traditional **ML** models are not suitable for the proposed final system. Hence, section 5.2 discusses two identified types of models that are more suitable to the use case in question: **LSTMs** and **MLPs**. The section presents the main characteristics of both techniques.

5.1 Machine Learning: Exploratory Analysis and Data Preprocessing

In order to get acquaintance with the **ML** tools, a simple script was developed, which aimed at classifying the *vehicles type* based on the messages content from a single run dataset (from *seed 0* of

Scenario A). In other worlds, the models classifies if a given message was originated by a *passenger* vehicle or a *motorcycle*.

This script (and also the more advanced systems that are discussed later on) were developed using the *Spyder* **Integrated Development Environment (IDE)** [109], which allows the usage of *Python* in an interactive environment, providing advanced editing, analysis, debugging, and profiling functionalities. The **IDE** environment is illustrated on Figure 5.1. The *Anaconda* tool [110] (Anaconda Navigator) was used to manage the *Python* distributions and modules.

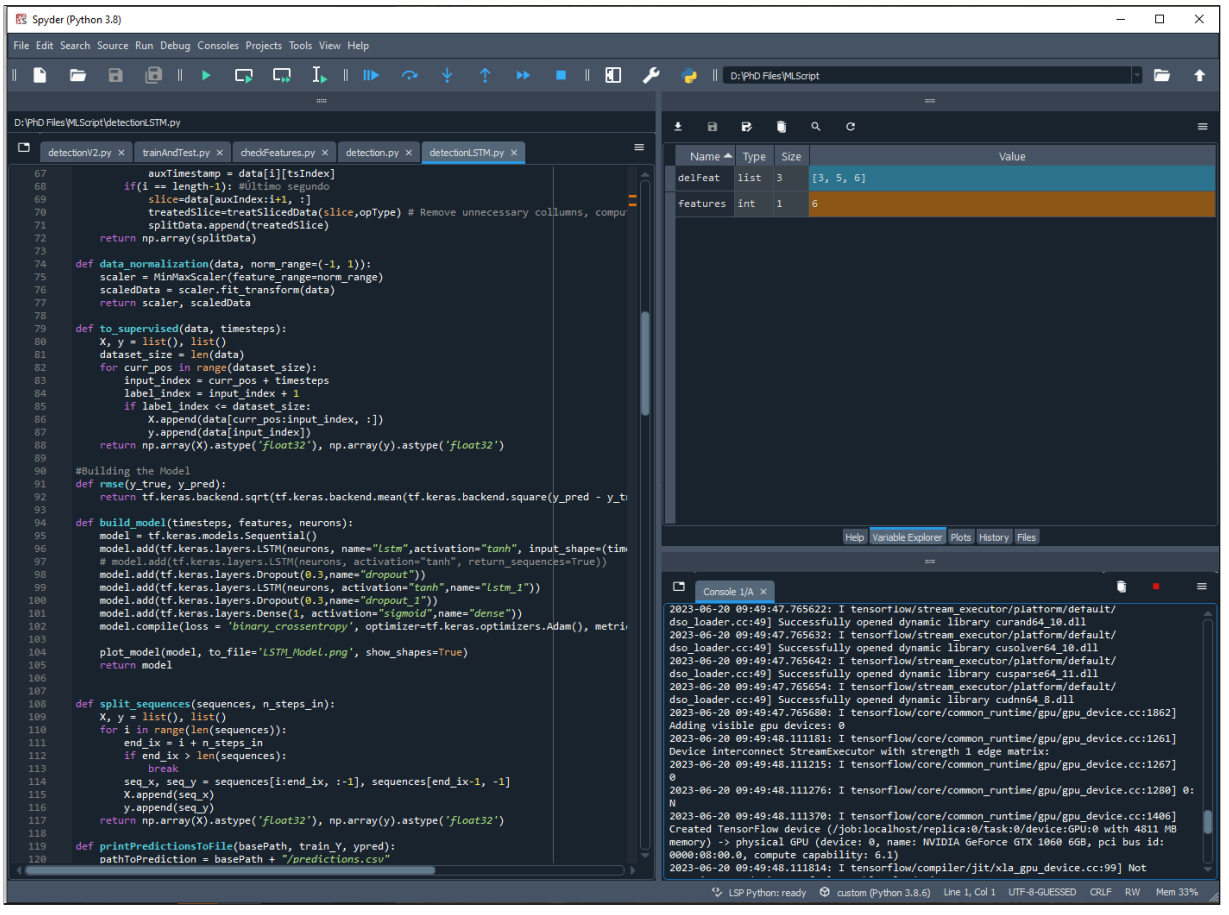


Figure 5.1: Spyder **IDE**

This first basic predictive model script was built resorting to the *Scikit-learn* [111] library. This library is available for the *Python* programming language [112] that provides a lot of efficient **ML** tools and statistical modeling.

Additionally, the Tensorflow-GPU library [113] was also installed. This library provides **ML** and Artificial Intelligence functionalities, namely the possibility for training and inference of *deep neural networks*. The *tensorflow-gpu* version allows the usage of a **GPU** instead of a **CPU**, as the process is more efficient. In this case, the following **GPU** was used: *MSI GeForce GTX 1060 Gaming X (6 GB DDR5)*.

The process of developing the first basic script is described on the next subsection.

5.1.1 Datasets Preprocessing and Automatic Classification of Vehicle's Type

In a first step, the *dataset* was loaded using the *read_csv()* function available on the *Pandas* library [114] (which provides tools for data analysis and manipulation).

Since the *dataset* is loaded without any treatment or pre-processing (the messages are simply received and written to a file on the simulation), the column titles were also defined, in order to clearly identify them, allowing an easier manipulation and analysis.

In order to prepare the *dataset* to be used classify the type of the vehicle, the first ten columns (*features*) are considered to be *data (x)*, while the *'SenderVehicleType'* feature is defined as being the *target (y)*.

Then, the *'NodeID'* and *'SenderVehicleType'* features must be encoded (in this case using the *LabelEncoder()* function). Although categorical features make the data more understandable (human-readable), **ML** algorithms typically don't handle well categorical data; in order to train and predict, this data should be transformed into numerical data.

The label encoder function encodes labels with values between "0" and "*numberOfClasses*-1". In this case, two different classes (*passenger - 0*, *motorcycle - 1*), as illustrated on the example on Figure 5.2.

...	SendingTime	SenderVehicleType
...	15879678	passenger
...	15879779	motorcycle
...	158796989	passenger

LabelEncoder

...	SendingTime	SenderVehicleType
...	15879678	0
...	15879779	1
...	158796989	0

Figure 5.2: Categorical Encoding Using Label Encoding

At this point, being the first basic **ML** script, only a few common **ML** prediction models were chosen for initial testing: **LR**, **KNN**, **Gaussian Naive Bayes (GNB)**, **SVM** and **ANN**.

The *K-Fold* method was used for the training and testing of the algorithms. This method divides the *dataset* into *k subsets* (in this case *k=10*) and each individual *subset* is then used as the *test set*, while the remaining *k-1 subsets* are used for training. The model's performance score consists on these results average.

This method is more robust than simply splitting the *dataset* into a *train* and *test* set (e.g. 80% - 20%). Cross-validation is usually a preferred method for that kind of problems because it gives a better indication of model performance on unseen data. On other hand, cross-validation requires more computation and

time resources.

Thus, using the data collected from the messages on the simulation, this initial script used the aforementioned **ML** algorithms to classify the type of vehicle that sent the message. In other words, the algorithms classified if the messages were originated from passenger vehicles or by motorcycles.

Naturally, and given the simplicity of the problem, all the algorithms had 100% *accuracy*. *Accuracy* is one of the most common metrics for evaluating classification models, and is defined as seen in Equation 5.1.:

$$Accuracy = \frac{Number\,of\,Correct\,Predictions}{Total\,Number\,of\,Predictions} \tag{5.1}$$

5.1.2 Datasets Preprocessing and Automatic Classification of Collisions

Even though the first developed script was a good transition from the simulation to the **ML** environment, the script still required a lot of work to achieve the global purpose of predicting collisions.

For that reason, and in order to introduce the notion of collisions into the *dataset*, a feature named *inCollision* was added to the *dataset*. In order to achieve this, a new *boolean* field was added to the beacon messages on the simulation, that states if the vehicle was involved (or not involved) in a collisions when that particular beacon was sent.

During every single simulation step, each individual node computes if he is involved in a collision at that instant, resorting to functionalities available on **TraCI**. In positive cases (i.e. if the vehicle is involved in a collision), the messages that these nodes generate are marked with a positive value.

Instead of marking each individual message during the simulation, another way of achieving this would be to feed the collisions information to the *dataset* during the *pre-processing* phase of the **ML** script. However, this would require to process the output logs of the **SUMO** simulator to identify which vehicles were involved in collisions (and at which instants) and later sweep the entire *dataset* in order to integrate the collision data. For its simplicity, the first solution was chosen - even though the beacons now possess an extra field that is not part of the standard message.

At this point, the script is updated to include the *inCollision* feature, which is now defined as being the *Target (Y)*, while *SenderVehicleType* is now added to *x*. The same algorithms were used to classify if the message belonged to a collision or not (without changes to the rest of the script), all achieving *accuracy* values above 80%.

At a first glance, the high value of the metric may sound promising. However, in fact, and as expected at this point, the solution of simply adding the new *inCollision* feature is a very poor one. Since

there are few collisions happening during the simulation (they are rare events), only a few messages are effectively marked as true on the *inCollision* feature column (in an universe of several hundreds of thousand of messages). For this reason, the *dataset* is considered to be **imbalanced** - too many messages marked as *false* when in comparison to the ones marked as *true*. This also explains why the algorithms have good levels of *accuracy* at this point - even if the models label every single record as *false* (messages from vehicles not involved in a collision), the *accuracy* is very high, because only a very little subset of records is being marked inappropriately.

Regarding the collision classification itself (the problem that is to be solved), all models performed poorly and were not able to classify properly. For that reason, it has become evident that the traditional models performance is unsatisfactory, and a different approach is necessary to address the problem effectively.

5.2 Time Series Forecasting

One of the key aspects of the proposed use case is that there are clearly very strong temporal concerns, which make the learning more complicated and difficult to handle by the traditional **ML** models, due to the sequence dependence among the input variables. Unlike most traditional **ML** models, that do not consider the time aspect at all (or operate on *datasets* that have a single slice of time), the core idea of *time series forecasting* modeling is to look at data from a time's perspective, defining patterns and predicting (in short or long-term) how target variables will change in the future. In other words, forecasting is the process of using machine learning models that were fit on historical data to predict future observations.

Based on the observed **V2X** messages, it is intended to automatically detect (and eventually predict) the occurrence of collisions, defining automatic **ML** mechanisms that send alerts to the driver in such cases (and thus try to avoid them). Hence, it is necessary to carry out a temporal analysis of the collected **V2X** messages data.

The state of the art works on time series forecasting systems refer architectures such as **Elman Recurrent Neural Network (ENN)**, **Gated Recurrent Units (GRU) Deep Recurrent Neural Network (DRNN)** or **LSTM** [115–117].

One particularly suitable deep learning model for the proposed use case is the **LSTM**. This deep learning model allows the capture of existing patterns in the data, as well as their long-term dependencies - they are very useful for its ability to hold information for long periods of time (information that is learned early on can still be impactful later on the model's decision).

LSTMs are a type of **Recurrent Neural Networks (RNNs)**, which in turn are a type of **ANNs**. This terms are briefly explained next.

ANN are computational systems that were modeled (in a simple way) on the central nervous system of human beings, using connections in order to solve problems - these networks are defined by a structure that interconnects computation units named *neurons* that possess learning abilities. These kind of *neural networks* consist in stacks of one or more *hidden layers* (which are used for learning) and a *dense* layer which is used to generate an output.

However, the usage of traditional **ANN** in this kind of problems has a major drawback because they do not possess *memory* - without the notion of *memory* present, it becomes harder to work on sequential data and on time-series data.

RNN are a subtype of **ANN** that aim to overcome this problem - unlike traditional **NNs**, **RNNs** address the persistence issue. In these systems, the evolution of the state depends both on the current and on the previous input, connecting outgoing nodes to the next incoming nodes. Such characteristic permits the process of context-dependent information and also the learning of long-term dependencies.

A **RNN** works as shown on Figure 5.3 below, where a *chunk* of the **NN** (A) looks at an input x_t and outputs the value h_t and then passes that information to the next step. This way, a **RNN** can handle data in a sequential manner, accepting both current input data and information from previous inputs.

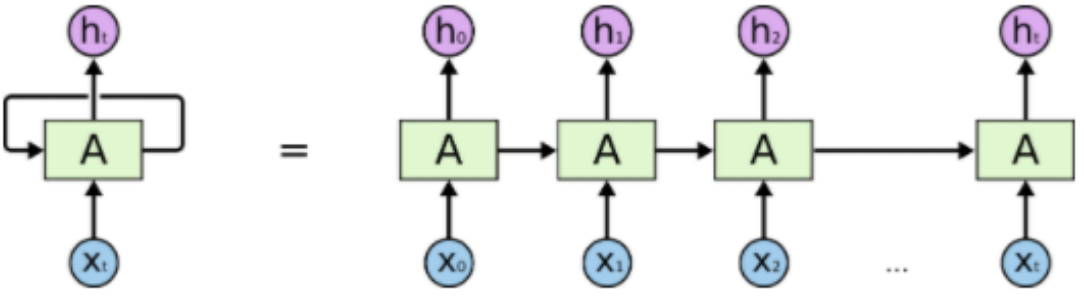


Figure 5.3: Unrolled Reccurent Neural Network [118]

In summary, **RNNs** may be interesting for their ability to connect recent data when performing a given task [118] - e.g. trying to complete the phrase: *"The sun is in the ____"*. The most evident answer to the problem is the word *sky* - as in *"The sun is in the sky"*. **RNNs** tend to perform well when the interval is short (between relevant data and the data that must be predicted). However, **RNNs** start to struggle when more context is needed to solve the problem. For example, when trying to complete the text: *"I was born in Portugal. I've been living there for 30 years. Therefore, i speak fluent _____"*. In this case, using only the most recent data is not enough to solve the problem. Data indicates the next word is probably a language but more context is needed from previous information to conclude that the answer should be

Portuguese. On some other more complex examples, the gap to the relevant information may become very large and, unfortunately, the **RNNs** performance will become deteriorated as the gap grows, since they are unable to connect the relevant data.

LSTMs [119] are a specialized type of **RNN** architecture, capable of better learning long-term dependencies - the main advantage of **LSTMs**, when comparing to traditional **RNNs**, is that they retain information for longer periods of time, which allows the early learned important information to also be impactful on the decision of the model, even if it is at the end of the sequence. Similarly to the previous discussion, and as shown on the top-side of Figure 5.4, A represents a complete **RNN** cell - which takes the current input (x_t) of the sequence and outputs the current hidden state (h_t), passing it to the next **RNN** cell. Each cell has a single layer that acts on its current state and input.

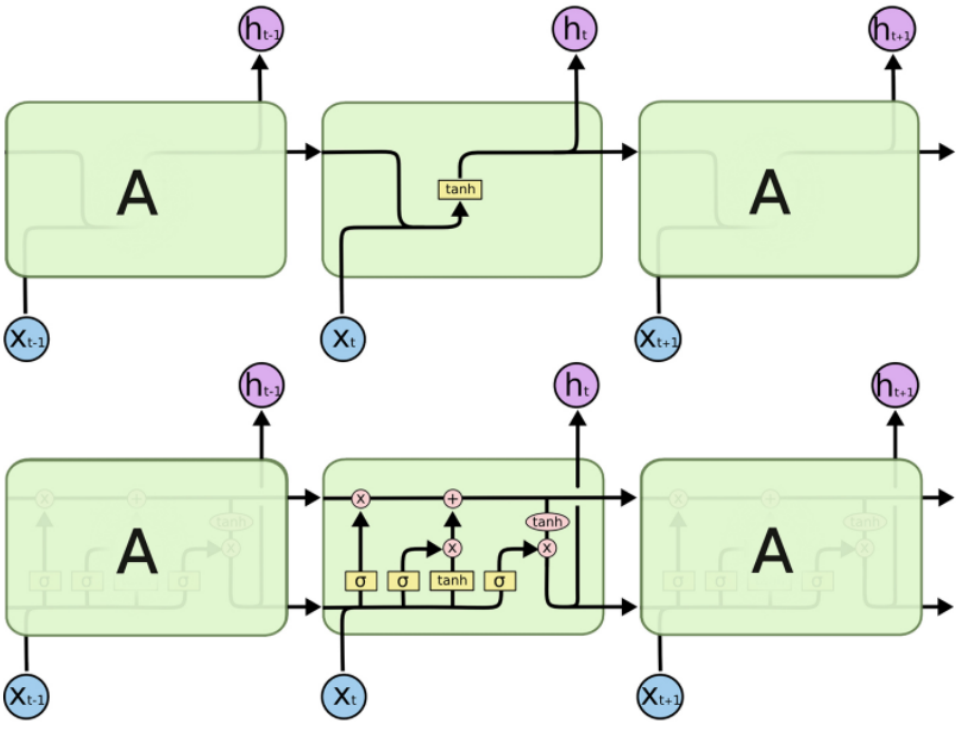


Figure 5.4: Unrolled RNN Cell (Top) vs LSTM Cell (Bottom) [118]

On the other hand, **LSTM** cells are more complex than traditional **RNN** cells, containing three internal layers acting on the state and input (shown on bottom-side of Figure 5.4). These internal gates (which are composed out of a sigmoid neural net layer and a pointwise multiplication operation) are the key to **LSTM** cells - they are weighted functions that govern the information flow (information state):

Forget Gate Decides what information to discard from the internal state (on other words, which information is maintained from the previous state).

Input (Update) Gate Decides which values from the input to add to the internal state (updates the state

based on the current input).

Output Gate Decides what to output based on the input and internal state (which information gets passed to the next state).

The control of the cell state is of utmost importance for **LSTMs** to properly work. Information is removed or added to the cell state using the aforementioned gates: the "forget" and "input" gates are used to update the internal state of the neuron, and the "output" gate is a final limiter on what the cell actually outputs.

Hence, **LSTMs** are very well suited for the task of predicting collisions, for its ability to capture long-term dependencies and handle sequential data. In the context of road safety for vulnerable road users, where context and temporal patterns are crucial, **LSTMs** are able to analyze and predict potential collisions patterns in real-time input data, and assist the road users in the decision making process if they are warned in a timely manner.

Furthermore, and as it is important to have a comparison point, another subtype of **ANN** was also identified for the implementation of the system: **MLPs**. **MLPs** are a classical type of neural networks [120] and they typically consists of one or more layers of interconnected artificial neurons, known as *perceptrons*. **MLP** consist of three main types of layers:

Input Layer This layer receives as input the initial data and passes it on to the next hidden layers. Here, each neuron in represents a feature of the input data.

Hidden Layer A hidden layer consists of multiple neurons which are connected to the previous and subsequent layers. Here, the input data is transform to extract higher-level features, providing levels of abstraction.

Output Layer This layer produces the final output (predictions) of the network, based on the input data and learned patterns.

MLPs use a series of equations with inputs, outputs and weights, and transform inputs into singular outputs between 0 and 1. That generated output serves as input to another layer, and the process continues until a singular output is reached. In other words, it is a feed-forward **NN** (the information moves only in one direction - from input nodes, through hidden nodes into output nodes). In that sense, information flows without any cycles or feedback connections.

MLPs can be used for time series forecasting and to predict future values based on previously observed data. Although they are not particularly designed to deal with long-term dependencies (unlike

LSTMs), they were still selected since they are very suitable for tabular datasets, as they have the ability to model complex non-linear relationships between the input features and the target variable. Also, **MLPs** can be particularly suitable for binary classification prediction problems - on this case, *collision* or *not in collision*. The output layer of the **MLP** can be designed to have two neurons that represent the probabilities of belonging to each class (1 or 0; *collision* or *not in collision*). Using an appropriate activation function (e.g the sigmoid function), the values that are outputted can be transformed into probabilities. Finally, the model can predict by assigning the fed input to the class with the highest probability outputted.

Considering their characteristics, both these **ML** techniques can contribute to the development of an automatic and intelligent system that can predict potential collisions involving **VRUs**, by leveraging the power of **V2X** data analysis and pattern recognition.

5.3 Summary

This chapter explores the application of **ML** methods to enhance safety measures within **ITS**. In a first instance, this chapter focuses on the development of a simple classification system aimed at predicting the originator of the messages (differentiating between passenger vehicles and motorcycles). This foundational work was an essential first step towards establishing interactions with the **ML** tools and assessing the quality of the synthesized datasets.

This exploratory investigation also revealed that traditional **ML** models are not suitable for the intended goal - the tested models performed poorly and were not able to classify the collisions properly. Thus, different types of models were identified that exhibit greater suitability for the specific use case: in this case, **LSTMs** and **MLPs**. Furthermore, the key characteristics and functionalities of **LSTMs** and **MLPs** techniques have been presented and analyzed.

The exploration of **ML** techniques that is achieved in this chapter lays the groundwork for subsequent model development, targeted at enhancing the safety and efficiency of **ITS**, particularly in safeguarding **VRUs**.

Chapter 6

Collision Detection System

A collision prediction system for **VRUs** is a critical safety feature that aims to reduce collisions involving pedestrians, cyclists, and other vulnerable users such as motorcycles. The main goal of such a system is to provide timely warnings to the interested actors when a potential collision is predicted, so that its effects can be avoided (or at least minimized).

Before achieving the prediction of potential collisions, an automatic collision *detection* system was first developed and evaluated, in order to have a starting point towards the more complex issue of prediction. This *detection* system aims to detect if a collision has occurred between passenger vehicles and motorcycles on an intersection. Despite not being able to prevent the collision, the development of such a system still allows the employment of various safety measures that try to minimize their severity and to mitigate further risks: for instance, by warning surrounding entities that a collisions has occurred; perform calls to alert emergency services and provide them with the collision location and details; control traffic and road closure to divert traffic away from the area; etc.

Hence, this chapter discusses the development of the first safety system: a collision detection system. It focuses on detailing the main characteristics of the system and also the results obtained from the evaluation. To achieve such a system that relies on leveraging **ML** mechanisms, two types of **NNs** were explored: **LSTMs** and **MLPs**.

Both types of models possess an input layer which expects (at most) 8 features - *Number of Vehicles, X Position, Y Position, Speed, Heading, Acceleration, Length and Width* - and an output layer with a *sigmoid* activation function, which outputs a value in the range 0 to 1. Several variations regarding the number of hidden layers (and dropout layers) and the set of input features were tested. The best performing models are presented in the results section 6.2.

The following sections describe how the data was gathered and processed, and also how the model was trained and tested.

6.1 System's Development - Considerations and Discussion

As stated before in section 4.3.2, the datasets that are built to feed the **ML** models consist of the collected messages that both *passenger* and *motorcycle* vehicles are exchanging through the use of communications. In order to compile the datasets, the **RSU** that is installed in the intersection collects all broadcasted messages and saves all the received messages in that area to a **CSV** file, line by line (each line corresponds to a collected message).

As each simulation run uses a different simulation *seed*, they produce different datasets with different results (in terms of mobility, as vehicles are inserted with randomness).
In this case, only Scenario A was considered for training and testing of the collision detection models. A total of 10 simulation runs were used to build the datasets:

- **Training dataset** - Aggregation of datasets from *seed* 0 to *seed* 7 (80%).
- **Validation dataset** - Dataset from *seed* 8 (10%).
- **Test dataset** - Dataset from *seed* 9 (10%), containing a total of 14 collisions.

The **ML** model must be constantly updated with all the changes in the whole environment so that it can take them all into account when performing the classification.

A possible solution for the later prediction of collisions could consist on training a model to predict the path of the nodes individually, using the predicted positions to compute *Cartesian* distances between the nodes and the corresponding probabilities of collisions (based on that predicted distance). However, such an approach requires the treatment of data in an individual manner (keeping information for each individual node) and the computation of distances to all nodes that are traveling in the vicinity. Naturally, as the number of vehicles may increase, the management of the system becomes more complicated, which may cause some performance issues.

As an alternative, in order to avoid the problem of having a large collection of *singular* vehicular data, another possible solution is to aggregate data in a temporal fashion - turning individual records into environmental information. Not having to treat data in an individual manner allows for a more straightforward and simpler implementation.

Hence, the datasets were split in fixed time intervals (1s, 0.5s and 0.1s were tested) and several methods were tested for aggregation (*minimum, maximum, sum and average*). Figure 6.1 below illustrates an example where messages are aggregated using 1 second for aggregation time and the *sum* function.

Node ID	Pos X	Pos Y	Pos Z	Speed	Heading	Acceleration	Length	Width	Timestamp	VehicleType	Accident
motorcycle.0	181.60	23.40	0.00	10.10	269.98	6.00	2.20	0.90	0.1848	motorcycle	0
passenger.0	181.54	20.03	0.00	14.00	359.98	5.74	5.00	1.80	0.2185	passenger	0
motorcycle.0	181.43	23.42	0.00	10.15	269.98	5.99	2.20	0.90	0.2849	motorcycle	0
...											
passenger.1	181.60	20.05	0.00	13.95	359.98	5.85	5.00	1.80	1000.1547	passenger	0
motorcycle.1	181.54	23.45	0.00	12.50	269.98	5.74	2.20	0.90	1000.2480	motorcycle	1
passenger.2	181.43	20.07	0.00	13.70	269.98	5.99	5.00	1.80	1000.2545	passenger	1

Condensing all messages within 1 second using the "sum" operation

Nº of Vehicles	Pos X	Pos Y	Speed	Heading	Acceleration	Length	Width	Accident
2	544.57	66.85	34.25	899.95	17.73	9.40	3.60	0
...								
3	544.57	63.57	40.15	899.95	17.58	12.20	4.50	1

Figure 6.1: Aggregating Several Messages into a Single Record - Example with 1s *sum* Aggregation

Since the system is now using condensed environmental data, some features are now removed from each record, since they no longer make sense as aggregated information, namely the *Station ID, Vehicle Type* and *Timestamp*. Additionally, the *Position Z (elevation)* feature is also removed. The simulation scenario does not consider this information and all values are equal to zero, which makes the feature irrelevant for the model. However, a new *Vehicle Count* feature is added to each record, in order to complement the aggregation data - which states how many different vehicles sent messages during that period of time.

The main challenge in the defined use case (detecting collisions related to **VRUs** on a crossing) is that the dataset can be considered as *imbalanced*. There are only a few collisions in each simulation, hence only a few records on the dataset set to be *inCollision = true*. In a way, these records represent anomalies in the complete data (may be considered outliers).

In order to address the issue of having unequal representation of classes in the dataset *target* value there are some approaches that can be followed, such as:

- **Data Resampling** - If a class is significantly underrepresented, one can generate more instances of the minority class (*oversampling*) or remove some instances from the majority class (*undersampling*) to achieve a more balanced dataset.

- **Weighted Loss Function** - Tune the *loss* function during training to give more importance to the underrepresented class, helping the model to achieve better predictions.

- **Ensemble Methods** - Combine multiple prediction models to improve the performance. Their prediction aggregation may help to reduce the impact of data imbalance.

- **Adjust Decision Threshold** - In binary classification problems, the classification *threshold* is set by default to 0.5 (i.e., <0.5 equals to a negative classification) . Adjusting this decision threshold may help to predict the classes more accurately.

- **Feature Engineering** - Transforming the input features, or create new ones, may help the models in capturing patterns in the imbalanced data.

In this case, different class weights were estimated for the models training - the models *loss* function is assigned higher value to the positive instances, which are rarer. In other words, this intends to tell the models to pay more attention to the positive cases (collisions) during classification. The models were compiled using *binary crossentropy* for the *loss* function (typical loss function for binary classification problems, measuring the dissimilarity between the predicted and the true labels) and the *Adam* optimizer (responsible for updating the model's weights during training).

The pseudo-code for the data loading and treatment is illustrated in Code Block 6.1.

```
#Load and Treat Datasets
#Compile VEINS datasets into three datasets: Train - 80%, Validation
↪  - 10%, Test - 10%
trainDataset, validationDataset, testDataset = Load_csv_datasets()
For each dataset:
        Split into Aggregation Time second slices
        For each slice:
                Remove undesired features
                Aggregate slice into single record using Aggregation
                ↪  Type
                Insert VehicleCount feature
        Normalize data (in a [-1, 1] interval)
```

Code Block 6.1: Collision Detection System - Data Processing

After loading and treating the data, the training and testing models are slightly different between the **MLPs** and the **LSTMs**. Both of them resorted to the usage of two *callbacks* during the model fitting process:

ModelCheckpoint Used to save the weights of the model into a checkpoint file. This is useful so the weights can be loaded later to continue the training from the saved state.

EarlyStopping Useful for stopping the training when a monitored metric is no longer improving after a certain number of training epochs.

The pseudo-code for the **MLPs** training and testing is illustrated in Code Block 6.2.

```
# MLP Training Procedure
# Build the MLP model, inputing the (sub)set of features to be used
 ↪  and the number of neurons on the layers
mlpModel = build_MLP_model(features, neurons)

# Compute class weights
class_weights = compute_class_weights(targetData)

#Training (fitting) the model
mlpModel.fit(trainData, validationData, epochs,
 ↪  callbacks=[modelCheckpoint, earlyStopping],
 ↪  class_weight=class_weights)

#MLP Testing Procedure
# Obtain test results metrics
results = mlpModel.evaluate(test_X, test_Y)

#Obtain the predicted values and write to a .csv file for later
 ↪  analysis on Excel
predicted_Y = mlpModel.predict(test_X)
printPredictionsToFile(test_Y, predicted_Y)
```

Code Block 6.2: Collision Detection System - Training and Testing the **MLPs**

The **LSTMs** training and testing process is similar but slightly different. This is related to the fact that **LSTMs** require 3D inputs (batch size; timesteps; features), which requires some extra steps to be able to train. The pseudo-code for the **LSTMs** training and testing is illustrated next in Code Block 6.3.

```
#LSTM Training procedure
# Build the LSTM model, inputing the number of timesteps, the
↪  (sub)set of features to be used and the number of neurons on the
↪  layers
build_LSTM_model(timesteps, set_of_Features, neurons)

#Split train and validation data into sequences as LSTM expects 3D
↪  inputs [batch, timesteps, feature]
trainX, trainY = split_sequences(trainData, timesteps)
validationX, validationY = split_sequences(validationData,
↪  timesteps)

# Compute class weights
class_weights = compute_class_weights(targetData)

#Training (fitting) the model
lstmModel.fit(trainData, validationData, epochs, batchSize,
↪  shuffle=False, callbacks=[modelCheckpoint, earlyStopping],
↪  class_weight=class_weights)

#LSTM Testing Procedure
test_X, test_Y = split_sequences(testData, timesteps)
# Obtain test results metrics
results = lstmModel.evaluate(test_X, test_Y)
#Obtain the predicted values and write to a .csv file for later
↪  analysis on Excel
predicted_Y = lstmModel.predict(test_X)
printPredictionsToFile(test_Y, predicted_Y)
```

Code Block 6.3: Collision Detection System - Training and Testing the **LSTMs**

A different model was trained and tested for each combination of parameters (presented ahead), saving the results onto **CSV** files, using the format *[Real Value, Model's Prediction Value]* to allow a later more in-depth analysis.

The analysis of the obtained results is discussed next.

6.2 Evaluation of the Collision Detection System

This section focuses on the evaluation of the collision detection system performance. On this particular use case, unlike some other systems evaluation, simply analyzing the accuracy of the models to determine its performance is misleading. Although the high value of the metric may sound promising, the model's performance in terms of detecting the collisions may actually be poor. Since there are only a few collisions happening during the simulation time (they are rare events), only a few records on the dataset are effectively marked as *true* on the *inCollision* target column. For this reason, the dataset is considered to be **imbalanced** - too many messages marked as *false* when in comparison to the ones marked as *true*. This also explains why the models have good levels of *accuracy* in every combination of parameters. Even if, hypothetically, a model classifies every single record as *false* (vehicles are not involved in an collision), the *accuracy* metric has still a very high value, because only a very little subset of records are being marked inappropriately. For this fact, the following metrics were also considered:

- **Precision** - measure of how many of the positive predictions are actually correct (the accuracy of positive predictions).
- **Recall** - also sometimes described as *sensitivity* or *true positive rate*, it measures how many of the actual positive instances were correctly classified by the model. In other words, it measures the ability to find all positive instances.
- **f-Score** - combines the *precision* and *recall* metrics in a single score value (which may be useful in imbalanced datasets analysis).
- **Specificity** - sometimes described as *true negative rate*, it measures the model's ability to identify negative instances correctly (proportion of *actual* negative instances that were correctly predicted by the model).

Still, all those metrics had values very close to 1 (maximum value) when the models were being evaluated. Hence, they do not allow, by their own, to make a proper decision on which subset of parameters performs better. As an example, Table 6.1 shows the average results of the best performing parameters (for each aggregation time) when analyzing the **MLPs** results.

As the table shows, the results are close to perfection, but are still misleading - a more in-depth analysis of the results is necessary, with particular focus on the **False Positives (FPs)** and **False Negatives (FNs)** cases.

Aggregation Time	Neurons	Accuracy	Precision	Recall	F-Score	Specificity
1	32	1.000	0.999	0.998	0.998	1.000
0.5	64	1.000	0.999	0.999	0.999	1.000
0.1	32	1.000	0.998	0.999	0.998	1.000

Table 6.1: Example Subset of Results (**MLP** Analysis)

In particular, and taking into consideration the goal of the system (detecting collisions), lowering the number of **FNs** is of the utmost importance - it is crucial that the model is able to detect all collisions and, furthermore, detect it as soon as possible. Naturally, a high number of **FPs** may also be a problem - it is not intended for the model to classify normal traffic situation as collisions.

Hence, in summary, in order to conclude which parameters perform better, it is important to analyze how fast the collision is detected and the total number of **FP** cases.

Several parameters were experimented when training and testing the models, in order to find which one performed better:

- **Aggregation Time** - 1s, 0.5s and 0.1s.
- **Aggregation Type** - max, min, sum and average.
- Number of **neurons** on the models layers - 32, 64 and 128.
- Different (sub)sets of **input features**.
- **Timesteps** - 5, 10, 15 and 20, in the **LSTMs** case.

Additionally, several model variations in terms of number of *hidden* layers (layers between the *input* and *output* layers) and *dropout* layers were evaluated (*dropout* layers are used to prevent *overfitting* in a **NN**). Every model possessed an output layer with a *sigmoid* activation function - producing a probability output value (value between 0 and 1).
To simplify the analysis, the results are presented regarding the best performance models for each aggregation time. All the following discussed results consider the *sum* aggregation type - which performed better in all cases.

6.2.1 Analysis Using Aggregation Intervals of 1.0 Seconds

Starting with the analysis of the *1s* results, a **LSTM** model with two *hidden* layers and two *dropout* layers performed best - which had 64 neurons on the *hidden* layers, 5 features in the *input* layer (*Number of vehicles, Position X, Position Y, Heading, Vehicle Width*) and 20 *timesteps*, as shown on Figure 6.2.

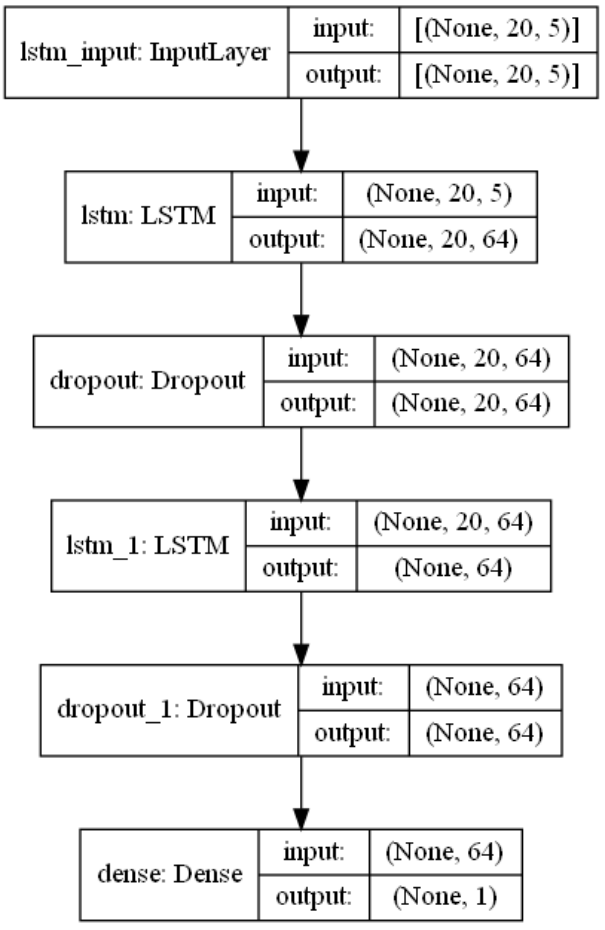


Figure 6.2: **LSTM** Model Summary - Aggregation Interval of 1.0s

In this case, and considering a *threshold* of 0.9 on the output (which means that a value greater than 0.9 is considered a positive classification), 2 **FPs** and 28 **FNs** were found (out of 167435 entries on the test dataset).

Most of the **FNs** classifications happen right at the beginning of the collision, which means that the model is not able to detect it immediately.

As exemplified on Figure 6.3, taken from the analysis of the first collision happening on the test dataset, the model outputs an higher value on the first occurrence (compared to the when the collision is not yet happening) but still not high enough to be above the threshold of what is considered to be positive (in this case $>$0.9). In this case, the collision is only detected on the second positive instance (in other words, after 2s).

Naturally, lowering this threshold (e.g. *threshold = 0.17*) would allow to correctly classify the first collision record and detect the collision sooner but, unfortunately, it also results in a very high number of **FPs**, which makes the solution unfeasible.

Real Value	Predicted Value
0	0.017
0	0.011
1	0.173
1	0.996
1	0.999

Figure 6.3: False Negative Example - First Collision on the Test Dataset

In summary, Table 6.2 presents the results obtained for the 1s aggregation time (how long it took to detect the collision).

Detection Time (s)	**Number of Detected Collisions**
2	2
3	10
4	2

Table 6.2: Results for an Aggregation Interval of 1.0s

Considering a total of 14 collisions that happen on the test data, two collisions were detected in 2 seconds, ten in 3s and two in 4s, which results in an average detection of 3s.

At this point, and although the model is performing good in absolute values (very low number of **FPs** and **FNs**), the collision detection results are below the expectations for an automatic system.

6.2.2 Analysis Using Aggregation Intervals of 0.5 Seconds

Looking at the results when aggregating the values in a smaller time window (now using 0.5s), the best performing model was an **MLP** with two *hidden* layers (with 64 neurons each) and all 8 features on the input layer (*Number of vehicles, Position X, Position Y, Speed, Heading, Acceleration, Vehicle Length, Vehicle Width*), as shown on Figure 6.4.

Using this model, and considering a *threshold* of 0.95, a total of 29 **FPs** and 12 **FNs** were achieved. The first noticeable difference from the previous case is the increase of **FPs** and **FNs** classifications. This behavior was expected: using a smaller time window for the aggregation time results in more condensed records on the dataset (using half the aggregation time roughly doubles the number of existent records). Although 29 **FPs** may be considered good results in terms of the model's overall performance, it still may raise issues in terms of the goal of the use case: the number of **FPs** cases is more than double the number of collisions happening on the training dataset (fourteen collisions), which leads to the need of establishing a strategy to mitigate this issue (which was not a problem so far on the 1s case).

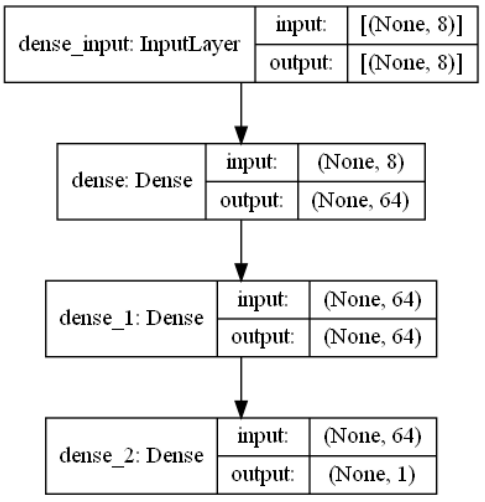


Figure 6.4: **MLP** Model Summary - Aggregation Interval of 0.5s

The list of **FPs** from the test dataset is shown in Figure 6.5.

Real Value	Predicted Value	Record Index	
0	0.96038	46123	
0	0.99084	104656	
0	0.99230	104658	
0	0.99908	123676	Immediatly after Acident
0	0.99760	123677	
0	0.98224	123678	
0	0.98345	139291	
0	0.97748	145654	
0	1.00000	149144	
0	1.00000	149145	
0	1.00000	149146	Immediatly after Acident
0	0.99999	149147	
0	0.99977	149148	
0	0.98209	149149	
0	0.99161	149150	
0	0.96081	185581	
0	0.96008	191944	
0	0.96156	204029	
0	0.97178	218979	
0	0.97291	245862	
0	0.95392	253536	
0	0.95089	274851	
0	0.98226	307559	
0	0.95696	318679	
0	0.95686	319829	
0	0.99997	330023	Immediatly after Acident
0	0.99957	330024	
0	0.99634	330025	
0	0.98126	330795	

Figure 6.5: Collision Detection System - List of **FPs** on the Test Dataset when using 0.5s for Aggregation Time

From the analysis of the list of **FPs**, two main conclusions were drawn. First, thirteen of the **FPs** happen immediately after the collision is over, and in a consecutive way. This is most likely related to the behavior of the vehicles in the mobility simulation - when the collision is over, the configuration of the vehicles on the road is still somewhat similar for some time, which may cause the model to continue to classify the entries as positive for some time in the end. To overcome this problem, it is proposed that these **FPs** are ignored if they happen immediately after a collision has already occurred - they can safely be disregarded.

The second conclusion is that the remaining **FPs** happen in isolated cases (there are not two consecutive **FPs** classifications). So, to avoid **FPs** classifications, it is established that a collision is detected only if the model classifies two consecutive record as positives. On the other hand, this also means that even if the model classifies the first entry of a collision correctly, we will only consider the collision as detected on the second entry. In other words, if the model correctly classifies the first two *0.5s* records, the collision itself will only be effectively detected in *1s*.

Thus, there is a certain trade-off when implementing this strategy to overcome the problem of the **FPs** cases - one minimizes/mitigate the **FPs** problem, but also delays the actual detection on positive cases.

This collisions detection logic, illustrated on Figure 6.6, removes all **FPs** cases.

t	Predicted Value	Real Value	
1s	0	0	
2s	1	0	Not considered
3s	0	0	
4s	1	0	False Positive
5s	1	0	
6s	0	0	
7s	1	1	True Positive
8s	1	1	
...			
308s	1	1	
309s	1	0	False Positive (ignored)
310s	1	0	
...			

Figure 6.6: Collisions Detection Logic

In this case, regarding the **FNs**, they also tend to happen right in the beginning of the collision, similarly to the 1s case. Table 6.3 presents the results obtained for the 0.5s aggregation time - how long it took to detect the collision. In summary, a total of twelve collisions were detected in 1.5s and two in 1s

Detection Time (s)	Number of Detected Collisions
1	2
1.5	12

Table 6.3: Results for an Aggregation Interval of 0.5s

(averaging 1.43s). Hence, in terms of the time needed to detect collisions, this solution performs better than the previous one, despite the model's initial worse performance in terms of increasing absolute values for **FPs** and **FNs** cases.

6.2.3 Analysis Using Aggregation Intervals of 0.1 Seconds

Finally, the *0.1s* results are consistent with the previous cases: lowering the aggregation time results in higher **FPs** and **FNs** numbers (141 and 83, respectively). Similarly, **FNs** also follow the same pattern as the other options and most of the **FPs** happen immediately and consecutively after a collision, while the remaining ones happen as isolated cases, which allows to apply the same strategy, thus mitigating the problem. Table 6.4 presents the results obtained for the 0.1s aggregation time.

Detection Time (s)	Number of Collisions
0.3	2
0.4	1
0.5	1
0.6	4
0.7	3
0.8	1
0.9	1
1.0	1

Table 6.4: Results for an Aggregation Interval of 0.1s

This way, two collisions were detected in 0.3s, one was detected in 0.4s, one in 0.5s, four in 0.6s, three in 0.7s, one in 0.8s, one in 0.9s and one in 1s (averaging 0.62s).

These results were obtained using exactly the same model and parameters as in the 0.5s use case.

6.3 Summary

This chapter described the results obtained when evaluating a system aimed at improving road safety, by collecting and treating **ITS** data, using **ML** techniques in order to detect collisions related to **VRUs** (motorcycles).

The tested models tend to perform relatively well but they present some limitations, specially regarding the high number of **FPs** - which were mitigated through the use of a specific collision detection logic after the model finishes its classification. On the other hand, the use of such a detection logic of the collision also slightly delays the detection itself, so there is a certain trade-of when applying such solution - between the performance of the model *per si* and the performance of the collision detection.

Additionally, and although the performance of the models *per si* gets worse when lowering the aggregation time (resulting in an higher absolute number of **FPs** and **FNs**), its capacity to detect collisions also becomes more efficient in terms of timeliness.

With the best parameter configuration - a **MLP** model with two dense layers (64 neurons) and all features on the input layer - the system was able to detect every collision in 1s or less (taking 0.62s on average), when using the 0.1s aggregation time.

This fast detection opens the possibility to trigger passive safety measures, such as notify surrounding vehicles of the collision, call emergency services or divert traffic to other roads.

The next chapter of this thesis presents a more complex system for the prediction of collisions, instead of just detecting them.

Chapter 7

Collision Prediction System

The previous chapter focused on the development and evaluation of an automatic collision detection system. This chapter presents a more complex safety system for **VRUs** - a collision prediction system. Such a system goes beyond the simpler detection of collisions and aims to predict them in advance. It utilizes more complex and advanced models to analyze the collected data and estimate the likelihood of a potential collision scenarios before they occur. Naturally, by providing a predictive capability beyond the simpler detection, and if achieved in a timely manner, it allows for preemptive driving decisions to enhance road safety, particularly for **VRUs**. This chapter describes the main enhancements that were made on top of the previously developed system, describing different possible strategies for the forecasting and several other important considerations regarding the system optimizations, including a discussion on another main issue in the defined use case - the *dataset* is *imbalanced*.

The previous system aimed to detect if a collision is occurring between passenger vehicles and motorcycles at an intersection. Until this point, at every instance *t*, the models would take *N* previously observed values (timesteps) and try to classify if that instant is part of a collision or not, as it is illustrated in Figure 7.1.

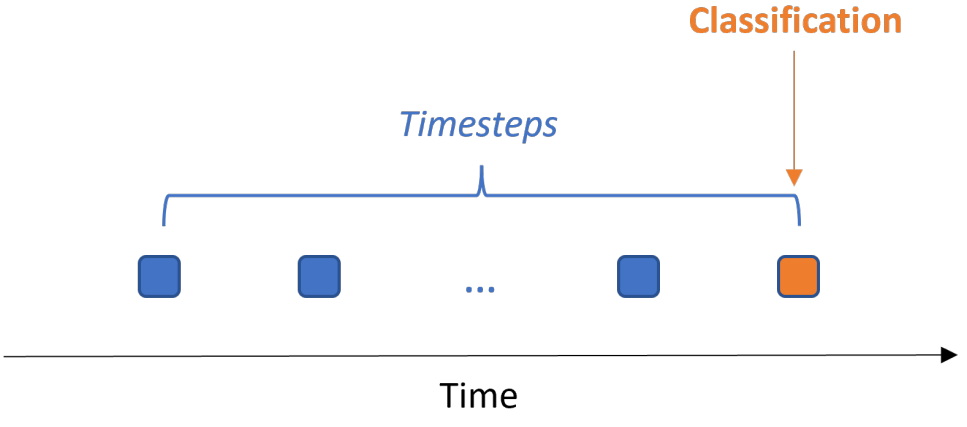


Figure 7.1: Collision Detection Classification

Now it is intended to go one step further on the complexity of the solution, and try to *predict* the collision in the future, instead of just detecting it. Generally speaking, when talking about *forecasting*, the most common use case is the *one-step ahead forecasting*, where a model tries to predict one value into the future, taking N *timesteps* of previously observed values - e.g. using the weather data of the last 7 days to predict the weather for tomorrow. The generic logic of *one step-ahead prediction* is illustrated in Figure 7.2.

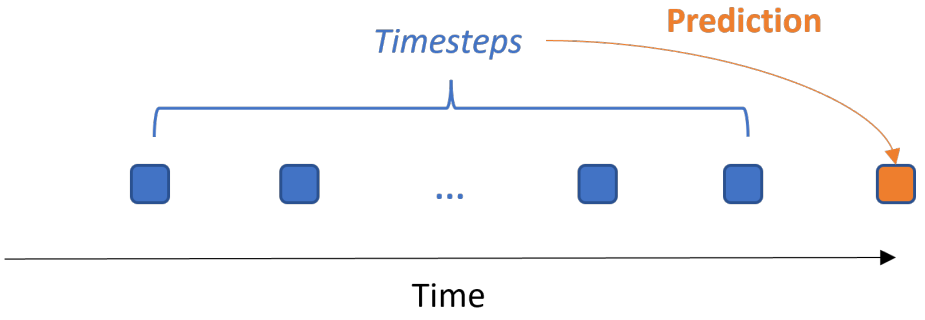


Figure 7.2: One-step Ahead Forecasting

On the other hand, there is the case of *multi-step forecasting*, where N *timesteps* are used to predict M **Multi-steps (MS)** in the future, as exemplified in Figure 7.3. An example of such solution is using the weather data of the last 12 months to predict the weather for the next two months.

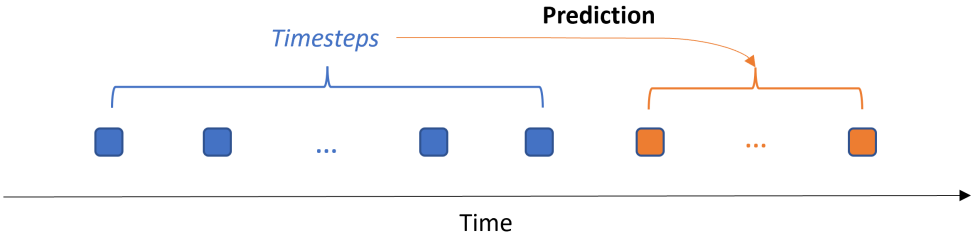


Figure 7.3: Multi-step Forecasting

Taking into consideration the proposed use case of predicting collisions, where the data aggregation is achieved in a 1s time window (further discussed ahead), the *one-step ahead* scheme may not be sufficient, since it only allows for the forecasting of one second into the future (as one step equals to one second).

Predicting collisions just one second before they happen may not allow for a proper preemptive action (e.g. drivers won't have enough time to manually react to the warning) and even the usage of automatic safety mechanisms (e.g. automatic breaking) may fall short, although they could minimize the impacts of the collisions. Still, naturally, resorting to automatic safety mechanisms may also be problematic

when considering the existence of possible *False Positive* cases during the forecasting - it is not intended for the system to automatically act on such cases (breaking in situations where it shouldn't).

For that reason, it is pre-established as a requirement that the forecast of the collisions should be achieved using the *multi-step forecasting*, and it should be within a reasonable time frame that allows for preemptive defensive actions by the drivers (further discussed ahead).

7.1 Multi-Step Forecasting Strategies

When talking about *multi-step forecasting*, there are different strategies that could be followed to make the forecasts [121], which are discussed next.

The first possible strategy is the *Direct Multi-step Forecasting*. This direct method requires the use of a separate model for each forecast **MS**. This means that, if one want to forecast 2 seconds into the future using the last 5 observed seconds, two different and separated models should be developed - one for each step in the future. However, using a solution where a unique model is needed for each step in the future may soon become very heavy in terms of computation requirements (and also on the maintenance of the solution itself), which can be a major drawback. This method is illustrated on Figure 7.4.

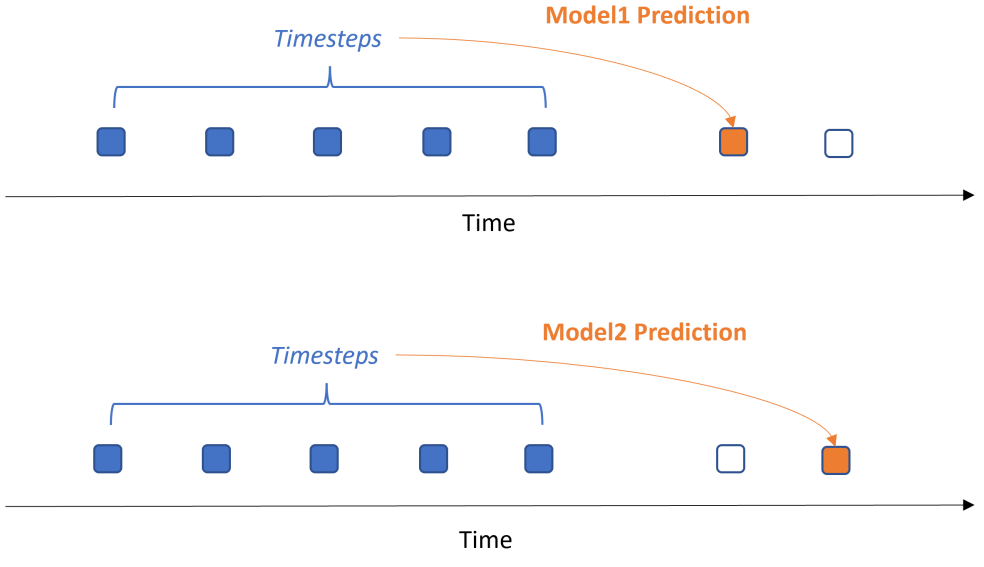


Figure 7.4: Multi-step Forecasting - Direct

A second possible strategy - *Recursive Multi-step Forecasting* - consists in the usage of a *one step ahead* model where the prediction of the model is recursively used as input for the forecast of the next steps. This means that, using the same analogy of predicting the next two seconds, the model would first predict the value for the 1st second and then use that prediction as an observation value (timestep) for the next prediction. This strategy is illustrated on Figure 7.5.

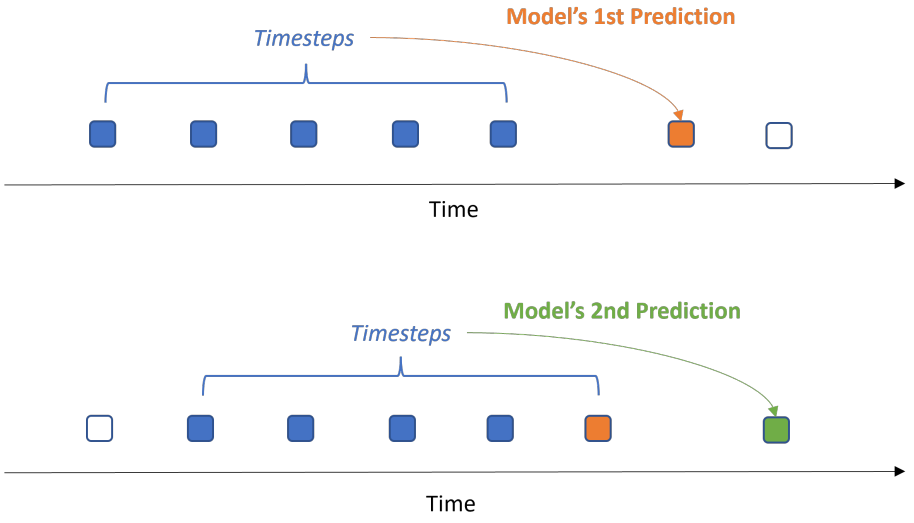


Figure 7.5: Multi-step Forecasting - Recursive

The main downside of this strategy is that it relies on the usage of predictions instead of real observations, which can cause the accumulation of prediction errors and ultimately lead to degradation of the performance of the model's forecast (specially as the number of **MS** grows).

A third possible solution is the usage of a strategy that combines the first two - *Direct-Recursive Hybrid Forecasting*. Using such mechanism, it is possible to developed separated models for each step to be predicted but use the prior model's predictions as input for the next ones. This behavior is illustrated on Figure 7.6.

Naturally, and although this solution may try to overcome the limitations of the *Recursive Multi-step* forecast, it still may possess the same limitations of the *Direct Multi-step* - heavy in terms of computation and maintenance.

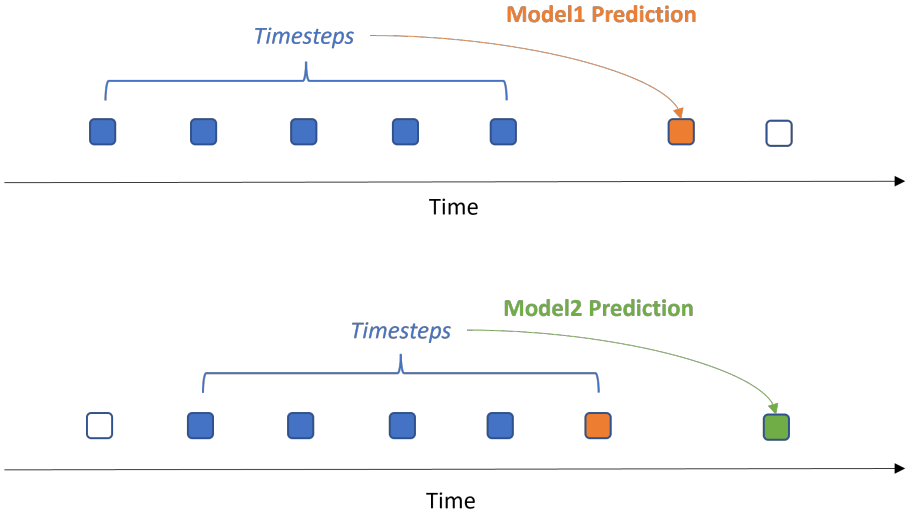


Figure 7.6: Multi-step Forecasting - Direct-Recursive Hybrid

Finally, a more straightforward solution is the simple usage of a single model to predict the entire forecast (all time steps in the future) - called *Multiple Output Forecasting*, as illustrated on Figure 7.7. These kind of models may eventually become more complex as they learn the dependencies between the inputs and the multiple outputs, which can cause them to be slower during training and be prone to *overfitting* (they typically require more data for proper training).

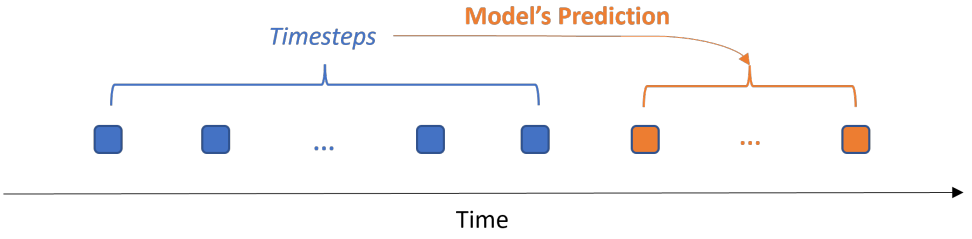


Figure 7.7: Multi-step Forecasting - Multiple Output

The main advantages and disadvantages of the four described strategies are presented on Table 7.1.

Strategy	Advantages	Disadvantages
Direct	Flexible: Each individual model can be adapted to the data of a specific timestep. Allows individual Tuning: Models parameters can be tuned individually, potentially leading to better results.	Complex: Developing and managing several models is challenging. High computational cost: Multiple models requires significant computational resources. Lack of global knowledge: Using separate models for each step may not capture the overall system dynamics as effectively.
Recursive	Simple to implement	Accumulated Errors: Errors can propagate with each successive step (potentially amplifying each time). Slower inference: Having a sequential process for prediction is slower (compared to direct strategies).
Direct--Recursive Hybrid	Combines the advantages of both direct and recursive strategies. Better error handling than direct multistep	More complex implementation
Multiple--Output	Simple to implement Less error accumulation than the other strategies	Models are more complex, as they forecast several steps Potential high computation cost: forecasting multiple steps simultaneously can be computationally demanding.

Table 7.1: Main Advantages and Disadvantages of the Multi-step Forecasting Strategies

Taking into consideration the pros and cons of said solutions, it was established that the *Multiple Output Forecasting* would be used to predict the collisions. This strategy requires only the usage of a single model and relies only on real observed values to predict future ones (avoiding error propagation), despite the potential higher computational costs.

7.2 System's Development - Considerations and Discussion

The process of developing the model's for the prediction was reasonably similar to the detection process and the same two types of **NNs** were tested: **MLPs** and **LSTMs**. The same data was used to feed the models and the same variations of parameters were trained and tested (set of input features, number of layers, number of neurons, and so on).

However, regarding the prediction of the collision, only the 1.0s *aggregation time* was considered. Lowering the value of the time window makes it harder to perform the forecast, as more **MS** are required (e.g. using the 0.5s aggregation requires the double of the **MS** to forecast the same time window, when compared to the 1.0s). On the other hand, enlarging the time window (e.g. aggregating for 2.0s, 3.0s and so on) dilutes the information and makes it less precise, which could lead to worse performances. For that reason, that option was not explored.

The process of developing the best performing models was arduous and immensely time consuming. It occupied most of the time dedicated to the thesis and it took several months of work to be able to achieve the first promising results (which were still very far away from the expected). Optimizing the prediction models was hard to learn and required great effort, especially as there wasn't any kind of previous experience on the (complex) subject.

At first, this process of trying to establish a stable and good performing model was achieved considering only Scenario A. The first attempts to predict the collisions were mostly unproductive and unsuccessful. This was mostly related to the fact that the datasets were highly *imbalanced*. Despite the models having good training results in terms of the traditional metrics such as *Accuracy, Precision, Recall*, and even *F-Score* measures (all values very close to 100%), the collisions could simply not be predicted. Figure 7.8 presents a fictitious (simple) example to better understand why the predictions were not working as intended. In this simplified example, the model tries to predict 1 second into the future, outputting a value between 0 and 1 using a sigmoid output dense layer (one assumes a positive classification if the value is equal or greater to 0.5). In this example, the model is able to predict accurately 90% of the time. However, the model does not predict correctly the first instance of the collision (t=5, identified in red, is

the most crucial point of prediction). Despite accurately predicting t=6, the collision had already occurred at that stage. Hence, the collision was not in fact predicted in useful time.

t	Real Value	Predicted Value	
0	0	0.12	
1	0	0.15	
2	0	0.21	
3	0	0.16	
4	0	0.19	
5	1	0.21	Collision
6	1	0.56	
7	1	0.75	
8	1	0.99	
9	1	1.00	

Figure 7.8: Failing Collision Predictions - Simple Example

The results were similar to the example even when using larger values for **MS** (predict more seconds into the future). Lowering the threshold value for the output (e.g., equal or greater than 0.2 as in the example) was a possible solution to overcome the problem. However, and as expected, using lower threshold values resulted in very large numbers of **FPs** (considering the example, using a threshold of 0.2 results in a **FP** in t=2). Hence, it was not a viable solution.

The first attempt at improving the model's performance was to test different feature selection methods. The feature selection process is useful to reduce the number of input variables when developing a predictive model. By using less variables, the computation cost is also reduced and, in some cases, the performance of the models may actually be improved. The following methods were explored:

Variance Threshold This method removes features with low-variance (variance is a measure of how much the values of a feature vary in the dataset. Features with low variance have similar values for most of the dataset, while features with high variance have a wider range of values. For instance, features that have constant values in all samples (zero variance) don't contribute much to the learning process, and should be removed. On the other hand, features with higher variance should be kept, as they show more significant variations in the data. Figure 7.9 presents the selected features (green cells) when using a threshold $t = 0.1$ (a typical value).

Variance Threshold	nVehicles	PosX	PosY	Speed	Heading	Acceleration	Length	Width
threshold = 0.1								

Figure 7.9: Feature Selection Results - Variance Threshold

Univariate feature selection This technique works by selecting the best features based on univariate statistical tests. In this case, two were tested: *selectkBest* (removes all but the *k* highest scoring features) and *selectPercentile* (removes all but a highest scoring percentage of features). Figure 7.10 presents the selected features (green cells) for those methods. Both selection methods achieve the same results, consistent with *Variance Threshold* when considering six features.

Univariate Feature Selection		nVehicles	PosX	PosY	Speed	Heading	Acceleration	Length	Width
SelectKBest	k=3								
	k=4								
	k=5								
	k=6								
SelectPercentile	p=30								
	p=50								
	p=60								
	p=80								

Figure 7.10: Feature Selection Results - Univariate Feature Selection

Recursive feature elimination **RFE** is a wrapper-type feature selection method. A different **ML** algorithm is wrapped by **RFE** and used to help select features. This method fits a model and removes the weakest feature(s) - it ranks features by importance and discards the least important feature, and re-fits the model. This process is repeated until achieving the specified number of desired features. Naturally, this approach is more heavy in computational terms when in comparison to the first ones. Figure 7.11 presents the selected features (green cells) when using the **DT** and **LR** estimators.

Recursive Feature Selection		nVehicles	PosX	PosY	Speed	Heading	Acceleration	Length	Width
Logistic Regression	k=3								
	k=4								
	k=5								
	k=6								
Decision Tree Classifier	k=3								
	k=4								
	k=5								
	k=6								

Figure 7.11: Feature Selection Results - Recursive Feature Selection

Sequential Feature Selection **Sequential Feature Selection (SFS)** is a greedy method for selecting features. It can work in two manners: Forward-**SFS** starts with zero features and adds a new best feature iteratively (based on the score obtained by the estimator), until the desired number of features is reached; Backward-**SFS** works similarly but instead of starting with zero features, it

starts with all features and greedily remove features from the set. Figure 7.12 presents the selected features (green cells) when using the same estimators as **RFE** (**DT** and **LR**).

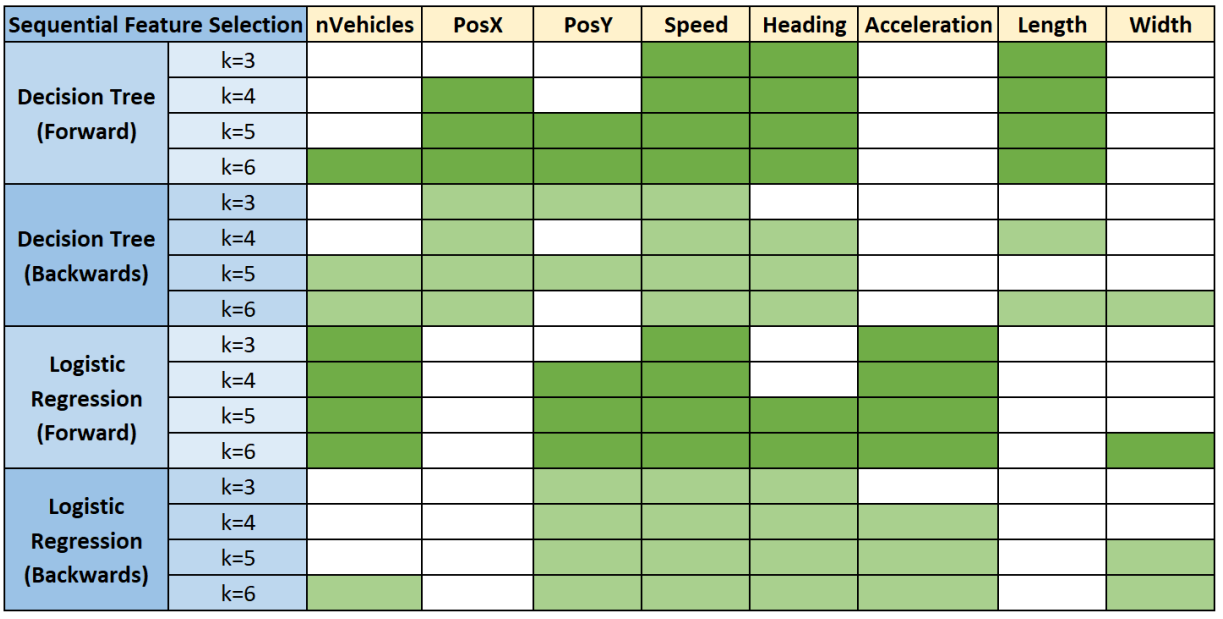


Figure 7.12: Feature Selection Results - Sequential Feature Selection

Table 7.2 presents the summary results of the achieved study. The table presents the number of total selections and the percentage of selections for each feature. This approach assigns more weight to features which are first selected for lower *k* values (meaning they are more relevant). Considering the example of **LR** (Backwards) from **SFS**, *PosY*, *Speed* and *Heading* will have the higher value (4), followed by *Acceleration* (3), then *Width* (2) and finally *nVehicles* (1). In this case, *PosX* and *Length* have a value of 0, as they were not selected. Based on that logic, the table finally presents the ranking results of the features, being *Speed* the most relevant and *Acceleration* the least relevant.

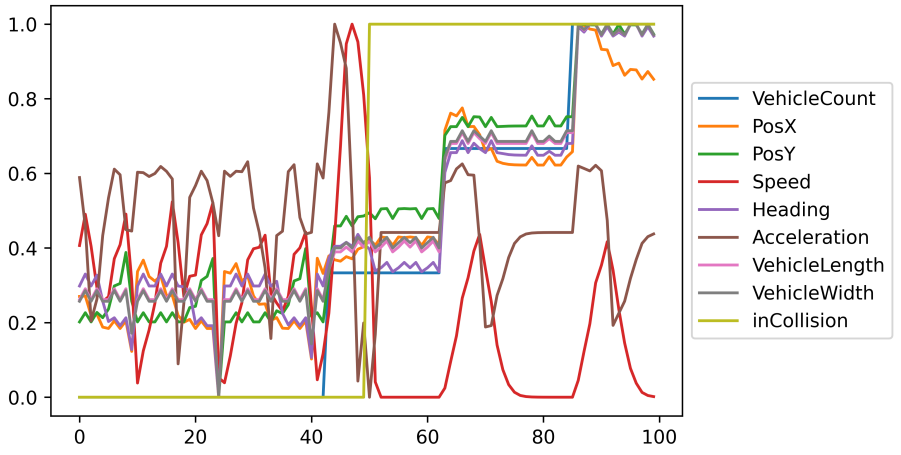
Features	nVehicles	PosX	PosY	Speed	Heading	Acceleration	Length	Width
Total selections	26	23	23	32	28	13	18	23
Percentage	14%	12%	12%	17%	15%	7%	10%	12%
Ranking	3	4	4	1	2	6	5	4

Table 7.2: Feature Selection - Summary Results

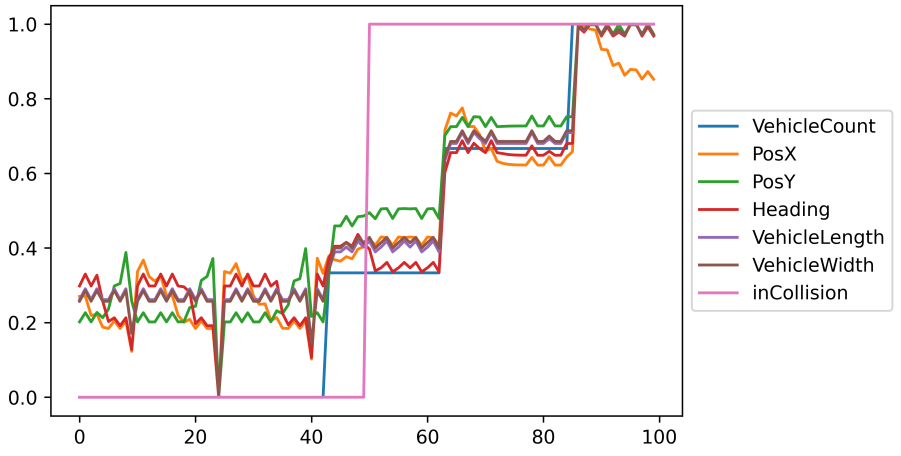
Additionally, a subset of features was also selected from an analysis of the features values on collision points. Figure 7.13 presents the features values from 50 seconds before and after a collision (that happens on *t*=50, represented by *inCollision*), taken from the 1st collision on the train dataset of Scenario A.

Subfigure 7.13a presents all the features, while subfigure 7.13b presents only the selected features which values that tend to grow and approach 1 when a collision occurs. In this case, the *Speed* and

Acceleration features were removed from this particular set, as they followed a different pattern from the other features.



(a) All Features



(b) Selected Features

Figure 7.13: Feature Selection: Collision Point Analysis (Scenario A 1st Collision on Train Dataset)

Hence, considering the previous discussion, the following subsets of features were selected for training and testing the models:

- **[speed, Heading, nVehicles]**
- **[speed, Heading, nVehicles, PosX, PosY]**
- **[speed, Heading, nVehicles, PosX, PosY, Width]**
- **[speed, Heading, nVehicles, PosX, PosY, Width, Length]**

- **[speed, Heading, nVehicles, PosX, PosY, Width, Length, Acceleration]** - All Features
- **[nVehicles, Heading, PosX, PosY, Width, Length]** - Taken from manual observation of the collision

Testing several conjugations of parameters took a lot of time to perform, and the gain was at first, minimal. Despite the efforts, the best performing models would still only be able to predict one third (roughly) of the collisions accurately, while still having a high number of **FPs**. These results were consistent (both in **LSTMs** and **MLPs**) despite several attempts to improve the training of the model: using different data preprocessing techniques (*standardization* vs *normalization*); experimenting with hyper-parameter optimization techniques (*Grid Search* and *Random Search*); and other tweaks (e.g., using *sum* vs *average* when aggregating the data).

The fact that the models were unable to produce acceptable results raised questions regarding the datasets that were being used. In particular, whether the available amount of data was sufficient for the model's to converge and learn. Despite the fact that the datasets contained hundred's of thousands of entries, the collisions were still rare occasions - a few tens of collisions on the training data. So, to try to understand if this could be the cause for the model's poor performance, more simulation runs were performed to collect more data and build new datasets - first 20 runs were trained and tested, then 30, 50 and finally 100.

In fact, increasing the amount of data greatly improved the performance of the forecasts, which led to the conclusion that there was an insufficient amount of data for the model's to converge properly - although this was previously not an issue, when performing the collision detection. The models were achieving higher numbers of accurate positive collision predictions when increasing the data collection to 20, 30 and 50 runs - the accuracy was getting higher as the number of total simulation runs to compile the datasets was raising. However, no meaningful differences were noticed when raising from 50 to 100 simulation runs in terms of the prediction capacity ability, but the 100 runs made the learning computation much heavier. For that reason, the results presented ahead rely on the usage of the 50 datasets to compile a very large dataset for training (80%), for validation (10%) and testing (10%).

However, increasing the amount of data naturally makes the learning process much slower and heavier in terms of computation (it even caused issues in terms of memory when using the larger values of *timesteps* and **MS**). In order to overcome that limitation, the training data was truncated on those large periods of time where there are no collisions. Different time windows (1000s, 1500s and 2000s) were tested to check which one performed better - e.g. keep 1000s before and after every collision, while removing the remaining information. The validation and test data remained unmodified. This solution is

illustrated on Figure 7.14.

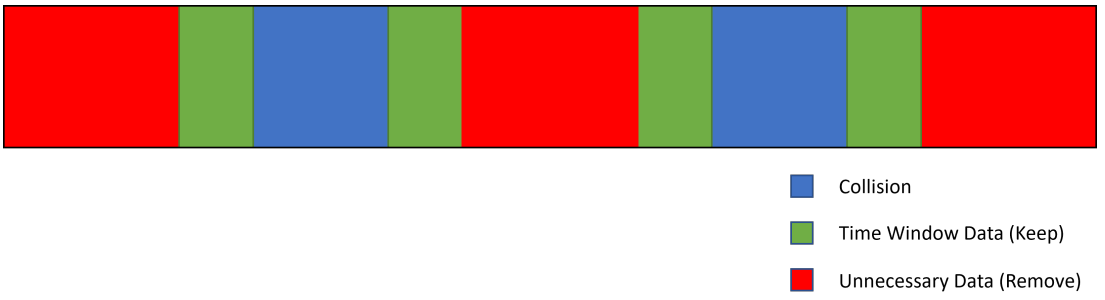


Figure 7.14: Truncating Unnecessary Data - Time Window Logic

Such a solution also helps to deal with the data imbalance. By removing those large amounts of less useful information, one is also reducing the amount of negative samples - in other words, *undersampling* the majority class, as discussed before on section 6.1. Naturally, by making the dataset more balanced, it also helps the model to converge better and to perform more accurate predictions.

All things considered, the established architecture (the one which had the highest number of predicted collisions so far) consisted on three **LSTM** layers and two *dropout* layers (used to prevent *overfitting*). It used all features on the *input layer* and the *sum* aggregation method for the input data. Additionally, it resorted to the *hyperbolic tangent (tanh)* activation function in all **LSTM** layers (the activation function helps determine the output of that layer - how the weighted sum of the input is transformed into an output from a node). The output layer used a *sigmoid* activation function, taking the input from the previous layer and producing a probability (value between 0 and 1) to produce the output. The comparison between the models performance (in this stage of establishing an architecture) considered a *threshold*=0.5 on this output layer.

The model architecture was compiled using *binary crossentropy* for the *loss* function and the *Adam* optimizer.

The final established architecture of the model is presented on Figure 7.15.

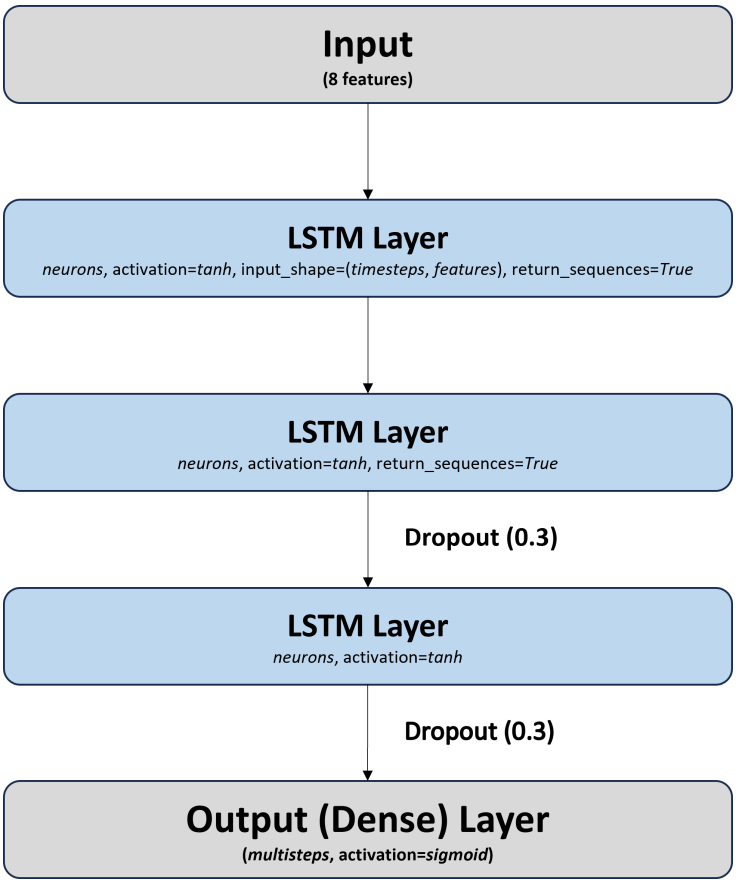


Figure 7.15: The Established **ML** Model Architecture for the Collision Prediction System

This architecture was established to be used for the collisions prediction system evaluation that is presented on section 7.3 (in both scenarios). The next section results explored the usage of the following parameters:

- Batch size - 64, 128, 256
- Neurons - 16, 32, 64
- Timesteps - 10, 15, 20
- **MS** - 1, 2, 3, 4, 5
- Time Window - 1000s, 1500s, 2000s

7.3 Evaluation of the Collisions Prediction System

The results that are discussed next were achieved in an iterative fashion. This means that first, different sets of parameters for *one-step ahead forecast* (1s in the future) were trained and tested. Then, the best performing runs were used to perform the *multi-step forecast*. Naturally, this procedure may not end up presenting the optimal results - some other parameters that were not tested could eventually perform better. However, performing all runs on all is very consuming, both in terms of time and computation - which makes it impractical.

As stated on the before on section 4.3, only two scenarios were explored due to some limitations on the *SUMO* simulator behavior regarding collisions. Results are then presented in an individual manner.

7.3.1 Results for Scenario A

This subsection presents the best achieved results for Scenario A, where the **Correct Decision Percentage (CDP)** is higher than 33% (this metric is further discussed ahead). The remaining runs, who achieved lower **CDP**, are disregarded from this point on.

The first part of the results, presented on Table 7.3, show some partial results of the *Model.evaluate()* function that is available on the *Tensorflow* built-in **API** (which returns both the loss and metrics values of the model after testing it, using the test dataset).

Run	Time Window	Batch Size	Neurons	Timesteps	Precision	Recall	F-Score
3	2000	256	16	10	0.9981	0.9990	0.9985
6	2000	128	32	10	0.9979	0.9993	0.9986
8	1500	256	32	15	0.9986	0.9990	0.9988
11	1500	128	32	20	0.9986	0.9825	0.9905
15	2000	128	64	15	0.9977	0.9983	0.9980
17	1500	256	64	20	0.9979	0.9992	0.9986
21	2000	256	64	10	0.9981	0.9991	0.9986

Table 7.3: Scenario A: *One-step Ahead Forecasting* Results - Part I

Similarly to what was discussed before, typical metrics such as *Precision, Recall* or *F-Score* (exemplified on the table) do not allow by themselves to make decisions on which model performs best - all values are very close to 1 (100%). For that reason, a more in-depth analysis was also needed.

Considering the use case of predicting collisions, the most important metric is, in a first instance, the **Collision Prediction Percentage (CPP)** - how many collisions from the test dataset were in fact

predicted. The second most important metric is the number of **FPs** - situations where the model wrongly forecasts a collision.

In order to make a proper decision related to these two metrics, a new metric that combines them was defined:

$$\textit{Correct Decision Percentage} = \frac{\textit{Predicted Collisions}}{\textit{False Positives} + \textit{Total Collisions}}$$

The **CDP** metric gives a value related to the number of correct decisions in the *critical points of decision*: the collisions and the **FPs** instances as well. In other words, it states how many positive predictions were actually correct (in percentage).

As a simple example, imagining a situation where a model predicts 5 collisions accurately (out of 10 existent collisions) and 15 **FP**, it means the system had a **CDP** of:

$$\textit{Correct Decision Percentage} = \frac{5}{15 + 10} = 0.2 = 20\%$$

In other words, only 20% of the decisions taken at the critical points were correct.

For each run, different *thresholds* were tested in order to achieve the best **CDP** value. This analysis was achieved using *Microsoft Excel*, using a spreadsheet that computes the number of predicted collisions (percentage as well) and the list of **FPs** for a given threshold - which ultimately allows to compute the **CDP**.

The summary of the analysis of that metric is presented on Table 7.4 and on the bar charts on Figure 7.16.

Run	Threshold	Predicted Collisions	Collisions Prediction Percentage	False Positives	Correct Decision Percentage
3	0.50	38	67%	39	40%
6	0.70	41	72%	27	49%
8	0.69	42	74%	20	55%
11	0.50	40	70%	30	46%
15	0.70	41	72%	36	44%
17	0.60	44	77%	36	47%
21	0.80	37	65%	22	47%

Table 7.4: Scenario A: *One-step Ahead Forecasting* Analysis - Part II

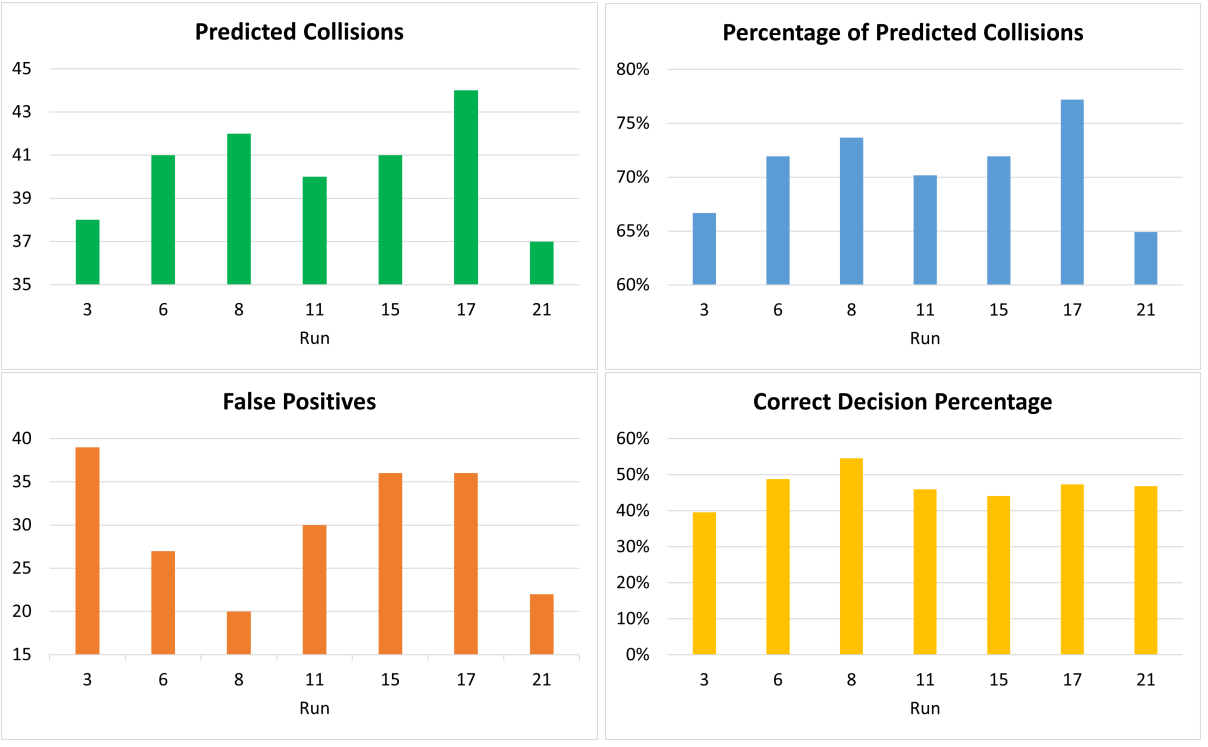


Figure 7.16: Scenario A: *One-step Ahead Forecasting* Analysis (Bar Charts)

When looking at the results, the best performing run regarding the **CDP** metric is number 8, with 55% of correct decisions. Although, for instance, *Run 17* predicted correctly more collisions (44 instead of 42 on *Run 8*), it has an higher number of **FPs** (36 vs 20), which leads to globally worse performance. This is the reason why the comparison is mostly based on **CDP**, which relates both metrics, and not simply on the number of total predicted collisions.

All of the aforementioned runs were selected to train and test new models for the *multi-step forecasting*, using the same parameters from Table 7.3. The obtained results are presented on Table 7.5.

Globally, the obtained results were positive. If one takes *Run 8* as an example (which was the best performing run on the one-step ahead forecast), the **CPP** was high for every **MS** possible - the lowest score was obtained at 2 **MS**, but still all of them managed to predict at least 50 collisions (in 57 possible).

However, there are still present a relatively high number of **FPs**, specially as the number of **MS** increases. Taking the 2 **MS** run as example, we have a total of 42 **FPs** and 50 predicted collisions (out of 57). This means that the number of **FPs** is somewhat close to the number of actual correct predictions.

The two and three **MS** runs were able to keep the **CDP** above the 50% - in other words, at least one in every two predictions actually predicts in collision. However, as the number of **MS** increases, this metric's values decrease - at 5 **MS**, the number of correct decisions is roughly one in every four (24%), a much lower value.

Run	Multi--steps	False Positives	Predicted Collisions	Collisions Predictions Percentage	Correct Decision Percentage	Average Prediction Time (s)
3	2	51	51	89%	47%	1.76
	3	54	47	82%	42%	2.49
	4	44	49	86%	49%	2.29
	5	69	52	91%	41%	3.33
6	2	43	52	91%	52%	1.81
	3	34	47	82%	52%	2.70
	4	55	50	88%	45%	2.48
	5	81	55	96%	40%	4.51
8	2	42	50	88%	51%	2.06
	3	40	52	91%	54%	2.67
	4	59	53	93%	46%	2.74
	5	170	54	95%	24%	4.56
11	2	45	43	75%	42%	1.98
	3	45	52	91%	51%	2.90
	4	39	49	86%	51%	2.76
	5	78	55	96%	41%	4.53
15	2	39	45	79%	47%	1.82
	3	39	49	86%	51%	2.76
	4	49	52	91%	49%	2.88
	5	84	53	93%	38%	4.26
17	2	49	47	82%	44%	2.00
	3	36	49	86%	53%	2.63
	4	64	54	95%	45%	2.72
	5	82	55	96%	40%	4.56
21	2	38	47	82%	49%	2.00
	3	41	52	91%	53%	2.65
	4	49	50	88%	47%	3.02
	5	88	54	95%	37%	4.50

Table 7.5: Scenario A: **MS** *Forecasting* Results

In this case, the sweet spot seems to be at 3 **MS**, where the model is able to keep the **CDP** above 50% and achieved an **Average Prediction Time (APT)** of 2.67s.

Here, the model's **APT** is slightly lower than 3 seconds because the model doesn't always necessarily predict the collision exactly 3 seconds before it happens - the model tries to forecast three values at every instance (every second) but the correct prediction may be achieved earlier or later.

Table 7.6 illustrates an example (taken from real results) for a prediction that happens exactly three seconds before the collision happens - at index $t = 13330$, being $t = 13333$ the moment the collision occurs.

Index (t)	Real value			Predicted Value		
	t+1	t+2	t+3	t+1	t+2	t+3
13328	0	0	0	0.0000	0.0000	0.0000
13329	0	0	0	0.0000	0.0000	0.0000
13330	0	0	1	0.0000	0.0058	0.7695
13331	0	1	1	0.0006	0.9155	0.9556
13332	1	1	1	0.8933	0.9599	0.9687
13333	1	1	1	0.8867	0.9473	0.9335
13334	1	1	1	1.0000	0.9999	0.9999

Table 7.6: Scenario A: Run 8, 3 **MS** - 3 Seconds Prediction

Table 7.7, on the other hand, presents an example (also from real results) of a later prediction, achieved 2 seconds before the collision.

Index (t)	Real value			Predicted Value		
	t+1	t+2	t+3	t+1	t+2	t+3
2496	0	0	0	0.0000	0.0000	0.0000
2497	0	0	0	0.0000	0.0000	0.0000
2498	0	0	1	0.0000	0.0009	0.3633
2499	0	1	1	0.0001	0.7597	0.8680
2500	1	1	1	0.8649	0.9342	0.9487
2501	1	1	1	0.9562	0.9686	0.9692
2502	1	1	1	1.0000	0.9998	0.9998

Table 7.7: Scenario A: Run 8, 3 **MS** - 2 Seconds Prediction

As it is possible to see, at index $t = 2498$ (3 seconds before the collision), the model was not yet able to predict correctly the collision. Although the predicted value for $t + 3$ is already higher when in comparison to the previous rows (0.3633), it still does not reach the threshold value for positive cases (on this particular case, *threshold* = 0.5).

Similarly to the problem that existed in detection of collisions, lowering the threshold value would result in earlier predictions, but it would also raise the number of **FPs** cases (which are meant to be avoided). From that perspective, it exists a certain trade-off - one sacrifices a better **APT** to avoid further **FPs** cases.

Hence, and since the model only correctly predicts at index $t = 2499$, the collision is predicted only 2 seconds before it actually happens.

Finally, on Table 7.8, there is an example of earlier prediction (4 seconds).

Index (t)	Real value			Predicted Value		
	t+1	t+2	t+3	t+1	t+2	t+3
19615	0	0	0	0.0000	0.0000	0.0000
19616	0	0	0	0.0000	0.0042	0.6829
19617	0	0	1	0.0008	0.9464	0.9713
19618	0	1	1	0.9761	0.9889	0.9920
19619	1	1	1	1.0000	0.9995	0.9996
19620	1	1	1	1.0000	0.9999	0.9999
19621	1	1	1	1.0000	1.0000	0.9999

Table 7.8: Scenario A: Run 8, 3 **MS** - 4 Seconds Prediction

In this example (also from real results), the prediction time is higher than the 3 seconds established for the prediction. As it is possible to see on index $t = 19616$, the model *wrongly* classifies the $t + 3$ value - positive when the real value is negative. This is not considered towards the **FPs** case count, since the collision happens immediately after.

Finally, the prediction time results for Run 8 (3 **MS**) are presented on Table 7.9.

Prediction Time (s)	Number of Collisions
4	4
3	29
2	17
1	2
Not detected	5

Table 7.9: Scenario A: Run 8, 3 **MS** - Prediction Time Results

In summary, the Run 8 (3 **MS**) model was able to correctly predict 91% of the collisions (52 out of 57), with a **CDP** of 54% when considering the **FPs**. The collisions prediction was achieved on a 2.67s average.

Although this discussion regarding Run 8 is helpful to understand how the results were computed, it is more interesting to debate the results organized by **MS**.

Table 7.10 presents the results for two **MS** on Scenario A.

Run	False Positives	Predicted Collisions	Collisions Prediction Percentage	Correct Decision Percentage	Average Prediction Time (s)
3	51	51	89%	47%	1.76
6	43	52	91%	52%	1.81
8	31	45	79%	51%	1.84
11	45	43	75%	42%	1.98
15	39	45	79%	47%	1.82
17	49	47	82%	44%	2.00
21	38	47	82%	49%	2.00

Table 7.10: Scenario A: Two **MS** Results

Here, only two models had a **CDP** over 50% - *Run 6* and *Run 8*. When comparing these two, it is possible to conclude that Run 6 outperforms Run 8 - although it has more **FPs** (43 vs 31), it performed much better by predicting more collisions (91% vs 79%), which eventually resulted in a marginally higher **CDP** value.

Regarding the **APT**, they performed similarly (1.81s vs 1.84s). Thus, one can conclude that *Run 6* is the best performing one when considering 2 **MS** forecasting.

Table 7.11 presents the results for three **MS** on Scenario A.

Run	False Positives	Predicted Collisions	Collisions Prediction Percentage	Correct Decision Percentage	Average Prediction Time (s)
3	54	47	82%	42%	2.49
6	34	47	82%	52%	2.70
8	40	52	91%	54%	2.67
11	45	52	91%	51%	2.90
15	39	49	86%	51%	2.76
17	36	49	86%	53%	2.63
21	41	52	91%	53%	2.65

Table 7.11: Scenario A: Three **MS** Results

Here, the results are similar between the runs - with exception to *Run 3*, which did not reach a **CDP** value of at least 50%. In terms of the collision prediction performance, the best runs (*Run 8, 11* and *21*) performed equally well - with 91%. In that sense, they only differ in terms of **FP** and the **APT**: *Run 11* had the best **APT**, while *Run 8* and *21* were very similar (2.67s vs 2.65s); However, in terms of **FPs**, *Run 8* was the best - 40 vs 45 and 41.

Thus, considering all points, *Run 8* was considered the best performing overall.

The results for four **MS** on Scenario A are illustrated on Table 7.12.

Run	False Positives	Predicted Collisions	Collisions Prediction Percentage	Correct Decision Percentage	Average Prediction Time (s)
3	44	49	86%	49%	2.29
6	55	50	88%	45%	2.48
8	59	53	93%	46%	2.74
11	39	49	86%	51%	2.76
15	49	52	91%	49%	2.88
17	64	54	95%	45%	2.72
21	49	50	88%	47%	3.02

Table 7.12: Scenario A: Four **MS** Results

When using 4 **MS** for the forecasting, only *Run 11* was able to achieve a **CDP** of at least 50%. Although some other runs had better collision prediction capabilities (e.g. Run 17 with 95%), they also had a larger number of **FPs**, which lowered the **CDP** value. Thus, and since *Run 11* was the only run who could at least make a correct decision in every two, it was considered the best performing run overall.

Finally, table 7.13 shows the results for five **MS** on Scenario A.

Run	False Positives	Predicted Collisions	Collisions Prediction Percentage	Correct Decision Percentage	Average Prediction Time (s)
3	69	52	91%	41%	3.33
6	81	55	96%	40%	4.51
8	170	54	95%	24%	4.56
11	78	55	96%	41%	4.53
15	84	53	93%	38%	4.26
17	82	55	96%	40%	4.56
21	88	54	95%	37%	4.50

Table 7.13: Scenario A: Five **MS** Results

On these forecasts, none of the model's was able to achieve a **CDP** over 50% - although they all had excellent results in terms of collisions prediction percentage (all above 90%), they all also had high **FPs** classifications. *Run 3* possessed the least **FPs** but, in the other hand, it also had the worst **APT**. All things considered, *Run 11* was the best performing model, with the higher values for **CPP** and **CDP**.

Table 7.14 summarizes the results for the best performing runs by multi-step. *Runs 6* and *Run 8* performed better for lower **MS** while *Run 11* performed better for higher **MS**. Furthermore, and in terms of **CPP**, all the results can be considered good since most of the collisions were correctly predicted.

MS	Run	False Positives	Predicted Collisions	Collisions Prediction Percentage	Correct Decision Percentage	Average Prediction Time (s)
2	6	43	52	91%	52%	1.81
3	8	40	52	91%	54%	2.67
4	11	39	49	86%	51%	2.76
5	11	78	55	96%	41%	4.53

Table 7.14: Scenario A: **MS** Summary Results

One of the main difference in the results is related to the number of **FPs** classifications: they get higher when using the highest value of **MS** (the further we try to predict into the future, the higher the number of **FPs**). When comparing the three **MS** and four **MS** forecasts, there is little difference on the **APT** (which is somewhat transversal to all runs). From that point of view, and as the **CPP** is similar, using an additional multi-step results in an heavier computing with no performance payoff.

However, things get different when using five **MS** instead of just four. In this case, the **APT** get larger, although they come at the cost of more **FPs** and a worse **CDP**. There is a trade-off notion present here: we have worse performance overall but, from the road users perspective, there is more time to make decisions and act upon the predictions.

Thus, in summary, the results can be then evaluated from two points of view: if one considers that the **CPP** and **CDP** are the most important metrics, using three **MS** is the best solution; on the other hand, if one considers that the **CPP** and **APT** are the most important, it is better to resort to five **MS** forecasting.

7.3.2 Results for Scenario B

This subsection presents the best results for scenario B. Similarly to Scenario A, one only considers the runs where the **CDP** is higher than 33% (the remaining runs are disregarded). The first part of the results show part of the output results of the *Model.evaluate()* function, presented on Table 7.15.

Run	Time Window	Batch Size	Neurons	Timesteps	Precision	Recall	F-Score
5	1500	128	32	10	0.9981	0.9993	0.9987
9	2000	128	32	15	0.9982	0.9994	0.9988
12	2000	128	32	20	0.9978	0.9991	0.9984
14	1500	128	64	15	0.9981	0.9992	0.9987
17	1500	256	64	20	0.9984	0.9990	0.9987
18	2000	256	64	20	0.9985	0.9990	0.9988
20	1500	256	64	10	0.9978	0.9992	0.9985

Table 7.15: Scenario B: *One-step Ahead Forecasting* Results - Part I

Once again, these typical metrics (*Precision, Recall* or *F-Score*) do not allow by themselves to make decisions on the model's performance - all values are very close to 1 (100%), so a a more in-depth analysis is also needed, considering the **CDP**. The summary of the analysis of the correct decisions is presented on Table 7.16.

Run	Threshold	Predicted Collisions	Collisions Prediction Percentage	False Positives	Correct Decision Percentage
5	0.50	41	72%	52	38%
9	0.60	41	72%	34	45%
12	0.60	43	75%	42	43%
14	0.65	35	61%	27	42%
17	0.75	36	63%	18	48%
18	0.74	37	65%	15	51%
20	0.50	44	77%	54	40%

Table 7.16: Scenario B: *One-step Ahead Forecasting* Analysis - Part II

When looking at the results, and at this point of using only *one-step ahead forecast*, the best performing run is number 18, with 51% **CDP**. As was done before in Scenario A, these best performing run parameters were selected to train and test new models for the *multi-step forecasting*. The results are presented on Table 7.17.

Run	Multi-steps	False Positives	Predicted Collisions	Collisions Predictions Percentage	Correct Decision Percentage	Average Prediction Time (s)
5	2	73	42	74%	32%	2.00
	3	51	50	88%	46%	2.88
	4	57	51	89%	45%	3.00
	5	96	54	95%	35%	4.59
9	2	31	41	72%	47%	2.00
	3	58	41	72%	36%	3.00
	4	36	35	61%	38%	2.60
	5	86	50	88%	35%	4.04
12	2	49	36	63%	34%	1.97
	3	57	42	74%	37%	2.57
	4	68	48	84%	38%	2.83
	5	89	54	95%	37%	4.59
14	2	53	43	75%	39%	2.02
	3	57	48	84%	42%	2.77
	4	47	51	89%	49%	2.90
	5	68	54	95%	43%	4.44
17	2	61	50	88%	42%	1.90
	3	41	50	88%	51%	3.00
	4	55	52	91%	46%	2.90
	5	73	54	95%	42%	4.48
18	2	43	46	81%	46%	1.98
	3	42	54	95%	55%	2.87
	4	33	51	89%	57%	2.94
	5	80	53	93%	39%	4.45
20	2	53	47	82%	43%	1.98
	3	75	53	93%	40%	2.75
	4	58	52	91%	45%	2.94
	5	105	54	95%	33%	4.54

Table 7.17: Scenario B: **MS** *Forecasting* Results

The first difference that can be found, when comparing to the results of Scenario A, is that the models ability to take correct decisions is globally lower (only a few of the runs were able to keep a **CDP** above the 50% threshold). Nonetheless, the models were still able to have high values regarding the number of predict collisions and similar results regarding the **APT**.

The remaining results will be presented following the same structure as on the previous section for Scenario A, organized by **MS**.

Table 7.18 presents the results for two **MS** on Scenario B.

Run	False Positives	Predicted Collisions	Collisions Prediction Percentage	Correct Decision Percentage	Average Prediction Time (s)
5	73	42	74%	32%	2.00
9	31	41	72%	47%	2.00
12	49	36	63%	34%	1.97
14	53	43	75%	39%	2.02
17	61	50	88%	42%	1.90
18	43	46	81%	46%	1.98
20	53	47	82%	43%	1.98

Table 7.18: Scenario B: Two **MS** Results

Here, none of the runs was able to achieve a **CDP** value above 50% (the higher values were achieved by *Run 9* and *Run 18*, with 47% and 46%, respectively). Although the first one achieved an higher value for **CDP**, the **CPP** is much lower (72% vs 81%). Given that both had similar results for the **APT**, *Run 18* can be considered the best performing one for two **MS**.

Table 7.19 presents the results for three **MS**.

Run	False Positives	Predicted Collisions	Collisions Prediction Percentage	Correct Decision Percentage	Average Prediction Time (s)
5	51	50	88%	46%	2.88
9	58	41	72%	36%	3.00
12	57	42	74%	37%	2.57
14	57	48	84%	42%	2.77
17	41	50	88%	51%	3.00
18	42	54	95%	55%	2.87
20	75	53	93%	40%	2.75

Table 7.19: Scenario B: Three **MS** Results

In comparison to the two **MS** results, the **CDP** values got globally higher. Here, two runs were able to make at least a correct decision for every two decisions: *Run 17* and *Run 18*. The results regarding **FPs** and **APT** were very similar - 41 vs 42 and 3s vs 2.87s. The main difference relies on the **CPP** where *Run 18* outperformed *Run 17* - 95% vs 88%. For that reason, *Run 18* was considered the best run overall.

The results for four **MS** are illustrated on Table 7.20.

When using four **MS** for forecasting (similarly to what happened on Scenario A), only *Run 18* was able to achieve a **CDP** over 50% (57% on this case). Although *Run 17* and *Run 20* had one more collision detected, the number of **FPs** was much higher, which made them worst solutions. Hence, *Run 18* was the best performing run.

Run	False Positives	Predicted Collisions	Collisions Prediction Percentage	Correct Decision Percentage	Average Prediction Time (s)
5	57	51	89%	45%	3.00
9	36	35	61%	38%	2.60
12	68	48	84%	38%	2.83
14	47	51	89%	49%	2.90
17	55	52	91%	46%	2.90
18	33	51	89%	57%	2.94
20	58	52	91%	45%	2.94

Table 7.20: Scenario B: Four **MS** Results

Finally, Table 7.21 presents the best results for five **MS** on Scenario B.

Run	False Positives	Predicted Collisions	Collisions Prediction Percentage	Correct Decision Percentage	Average Prediction Time (s)
5	96	54	95%	35%	4.59
9	86	50	88%	35%	4.04
12	89	54	95%	37%	4.59
14	68	54	95%	43%	4.44
17	73	54	95%	42%	4.48
18	80	53	93%	39%	4.45
20	105	54	95%	33%	4.54

Table 7.21: Scenario B: Five **MS** Results

When using the five **MS** forecasting, none of the models was able to achieve a **CDP** above 50%, although they all had very good results for **CPP** (at least 50 collisions out of 57 predicted). The **CDP** is lower because all models presented a high number of **FPs** classifications. *Run 14* can be considered the best, since it had the fewest **FP** classifications and an **APT** of 4.44s - most runs also performed around 4.5s, with exception to *Run 9*.

Table 7.22 summarizes the results for the best performing runs by multi-step.

MS	Run	False Positives	Predicted Collisions	Collisions Prediction Percentage	Correct Decision Percentage	Average Prediction Time (s)
2	18	43	46	81%	46%	1.98
3	18	42	54	95%	55%	2.87
4	18	33	51	89%	57%	2.94
5	14	68	54	95%	43%	4.44

Table 7.22: Scenario B: **MS** Summary Results

On Scenario B, *Run 18* performed better for two, three and four **MS**, while *Run 14* outperformed the other ones for five **MS**. In terms of collision prediction, only the two **MS** was unable to reach at least 50 collisions (still it managed to predict 81% correctly). The best results were achieved using three and five **MS** - 95% of predicted collisions.

Regarding the **FPs**, the values were similar using two and three **MS**. For four **MS**, there are only 33 **FPs** (which explains the higher value for the **CDP** of 57%). Five **MS** had the most **FPs**, similarly to what happed in Scenario A, which caused a lower **CDP** value.

Again, when comparing the **APTs** for three and four **MS**, it is noted that there is little difference. In that sense, the heavier computation for using additional steps also has no overall performance payoff. On the other hand, the **APT** when using five **MS** is much higher (4.44s), giving road users more time to make decisions and to actuate.

As already stated before on Scenario A, the results can be then evaluated from two points of view: if one considers that the **CPP** and **CDP** are the most important metrics, using three **MS** is the best solution; on the other hand, if one considers that the **CPP** and **APT** are the most important, it is better to resort to five **MS** forecasting.

7.3.3 Summary Results and Discussion

This section presented the results for the Collision Prediction System evaluation, organized by scenarios. The models ability to predict collisions involving **VRUs** was analyzed, focusing on the metrics: *Number of False Positives, Predicted Collisions, Collisions Prediction Percentage, Correct Decision Percentage and Average Prediction Time*.

The results from both scenarios are summarized on Table 7.23.

Globally, the proposed system achieved very good results in terms of collision prediction, as most of the collisions were predicted correctly in every **MS**. The worst performance was achieved in 2 **MS** for Scenario B, were 81% of the collisions were still predicted.

Despite the good performance for the prediction, the main issue of the system comes from the high number of **FPs**, which results in generally low values in terms of **CDP** - roughly, one in every two critical decisions made by the the system are accurate.

When comparing **MS** two and three, one can conclude that it is actually better to resort to the latter - it achieved better **CPP** and also had an higher **CDP**. Additionally, it also had better values for **APT**.

On the other hand, when comparing **MS** three and four, one can conclude that the results are very similar: similar values for **CDP** and **APT**, although the **CPP** are worse in the four **MS** case. From that

MS	Scenario	Run	False Positives	Predicted Collisions	Collisions Prediction Percentage	Correct Decision Percentage	Average Prediction Time (s)
2	A	6	43	52	91%	52%	1.81
	B	18	43	46	81%	46%	1.98
3	A	8	40	52	91%	54%	2.67
	B	18	42	54	95%	55%	2.87
4	A	11	39	49	86%	51%	2.76
	B	18	33	51	89%	57%	2.94
5	A	11	78	55	96%	41%	4.53
	B	14	68	54	95%	43%	4.44

Table 7.23: Collision Prediction Summary Results

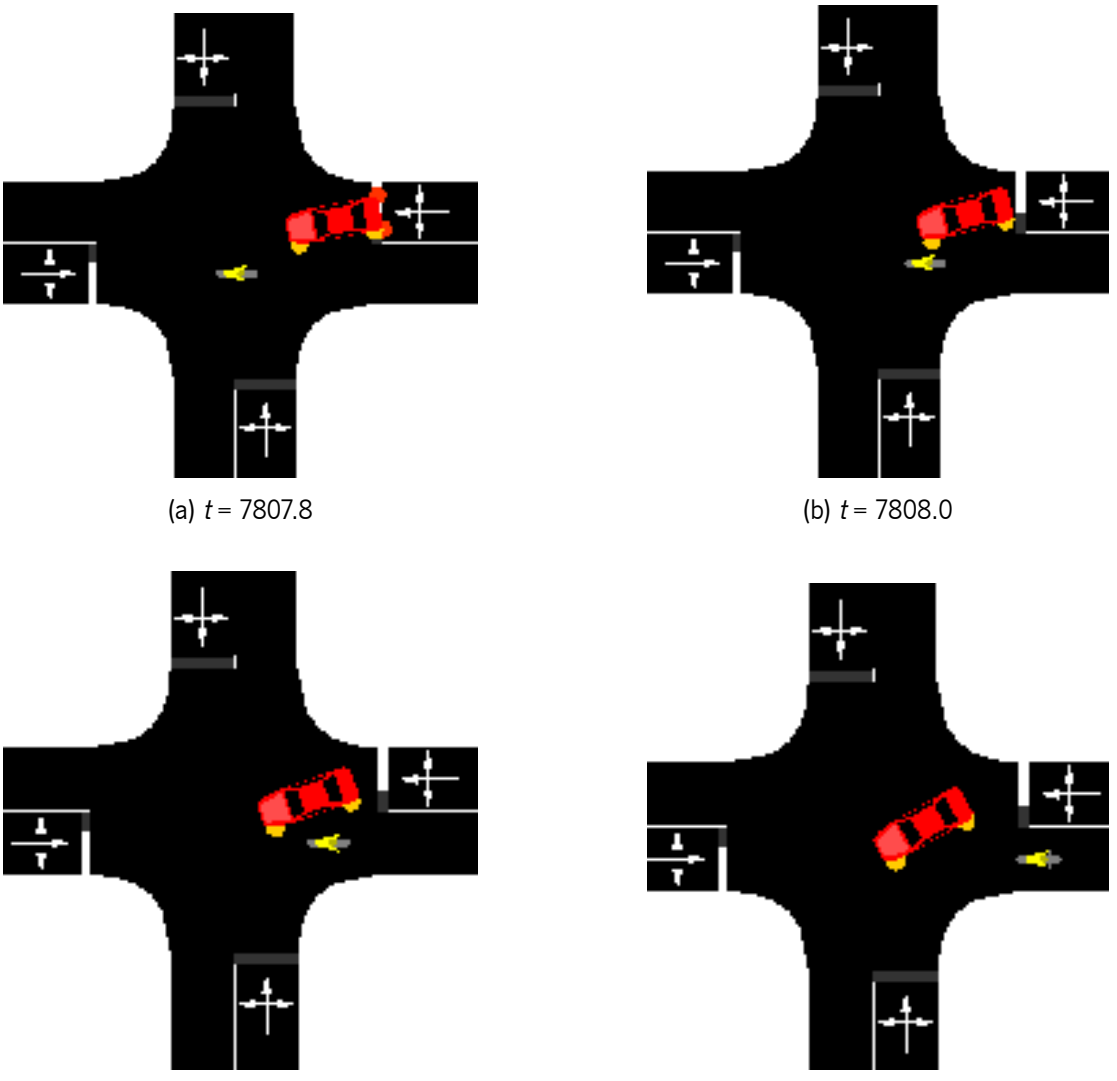
point of view, no performance gain is noted when using four **MS** instead of three.

Finally, the results from five **MS**: it achieved very good **CPP** values but has problems regarding the **FPs**, which cause low values on the **CDP** - below one in every two decisions is correct. On the other hand, the **APT** were very high (around 4.5s).

Hence, results can be looked from two perspectives: if one considers that the prediction of the collisions and the **CDP** are a priority, using three **MS** is the best solution; On the other hand, if higher **APT** are preferred, then using the five **MS** is the way to good, despite having a lower **CDP**. This way, an high number of notifications received by the driver's are *false alarms* but, on the other hand, the system may be able to avoid at least **95%** of the collisions on the proposed scenarios if the drivers are able to safely actuate on those (more or less) 4.5 seconds, so a trade-off notion is also noted in here.

Naturally, **FPs** can be in fact prejudicial, as having many **FPs** may cause the drivers annoyance. This may lead them to not react as expected, ignore the warnings or even disable the safety system if the driver does not trust its performance. In this case, an explanation to the high number of **FPs** may be related to how the scenario is implemented on the simulator. Given that **SUMO** does not have collisions enabled on the simulation by default, and they had to be deliberately caused using parameters, there are a lot of *near-collision* incidents happening during the simulations runtime. The mobility pattern and configuration of the vehicles on the scenarios is very similar to collisions, although the road users don't actually end up colliding. Figure 7.17 illustrates an example of a *near-collision* collision taken from a simulation of Scenario A (*seed* 0).

From the *Collisions Prediction System* point of view, the situation on the scenario is very similar to the simulated collisions, which may result in positive predictions. Naturally, from a statistical perspective,



(a) $t = 7807.8$ (b) $t = 7808.0$

(c) $t = 7808.2$ (d) $t = 7808.5$

Figure 7.17: *Near-collision* Example - Scenario A, *Seed* 0

these cases are counted as **FPs**. Still, in a real-world environment, these situations should also be avoided, as they can be considered dangerous.

For that reason, another way of reading the **FP** cases is that they are not necessarily all bad results. The notification of imminent danger may also help to avoid those *near-collision* incidents, since the drivers have time to adjust their behavior on the traffic and be more careful.

In summary, the proposed solution allows the usage of safety measures (such as notifying the drivers of imminent danger and performing an emergency break) which could greatly improve the **VRUs** safety on the roads.

Nonetheless, the **FPs** issue does not allow to implement *automatic* mechanisms regarding the passenger vehicles, as it is not desirable to actuate on such cases. Using such a system, the most practical way of actuating is *passively*, by simply notifying the drivers of the vehicles, leaving up to him to

perform defensive actions to prevent the collision.

From that point of view, it is better to resort to the five **MS** solution, since it has higher **APTs** (when comparing to three **MS**) which results in more time for the driver to take notice of the environment and actuate if necessary. However, one must be conscious that the **CDP** is slightly lower.

7.4 Evaluation in Terms of Collision Avoidance

The previous section focused on evaluating the collision prediction system using metrics such as **CPP**, **FP**, **FN**, **CDP** and also the **APT**. Besides analyzing the system ability to predict collisions, it is important to evaluate if the prediction gives enough time for the road entities to actuate in order to avoid the collision, or at least minimize its effects.

In order to further evaluate the performance of the system, the model needs to be deployed on-line within the **VEINS** simulation framework, where the use case scenarios were implemented. To achieve this, the *Python/C API* was used to interconnect the application that is running on **VEINS** and the *Python* model script. This library allows to call the *Python* functions from within the simulation environment, in order to process the data that the **RSU** is collecting (aggregate and scale) and also to perform the predictions. The simulation framework architecture is illustrated on Figure 7.18.

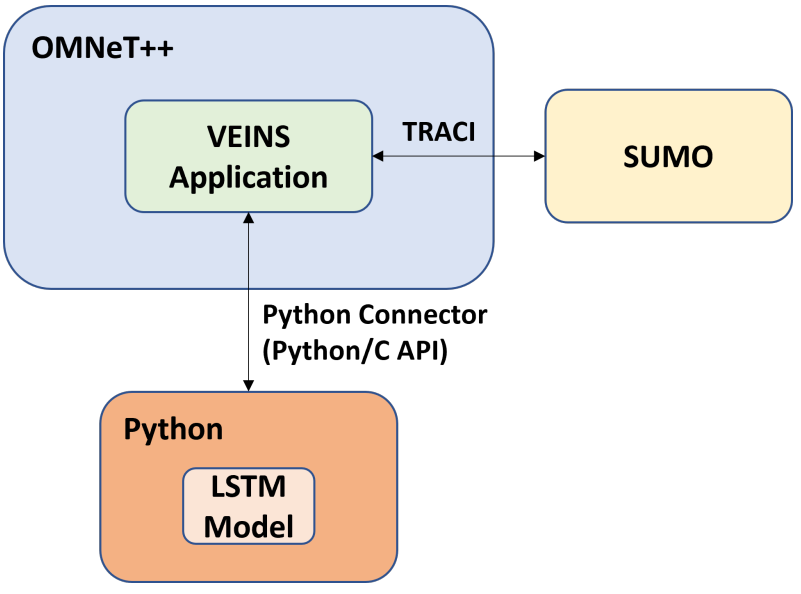


Figure 7.18: Simulation Framework Architecture

In summary, to implement the final collision prediction system, the following steps were taken:

1. Collect and preprocess the V2X communications data on the simulation framework.
2. Generate fifty datasets (one for each simulation run).
3. Divided the fifty datasets into training, validation and test datasets.
4. Train **LSTM** models using the training and validation data.
5. Evaluate the models performance on the test data (off-line).
6. Use the models to predict collisions in simulation runtime (on-line), on the same simulation runs that were used to compile the test datasets.
7. Measure the system overall performance for collision avoidance.

Figure 7.19 illustrates the chronological events and the timings associated with the collisions prediction system.

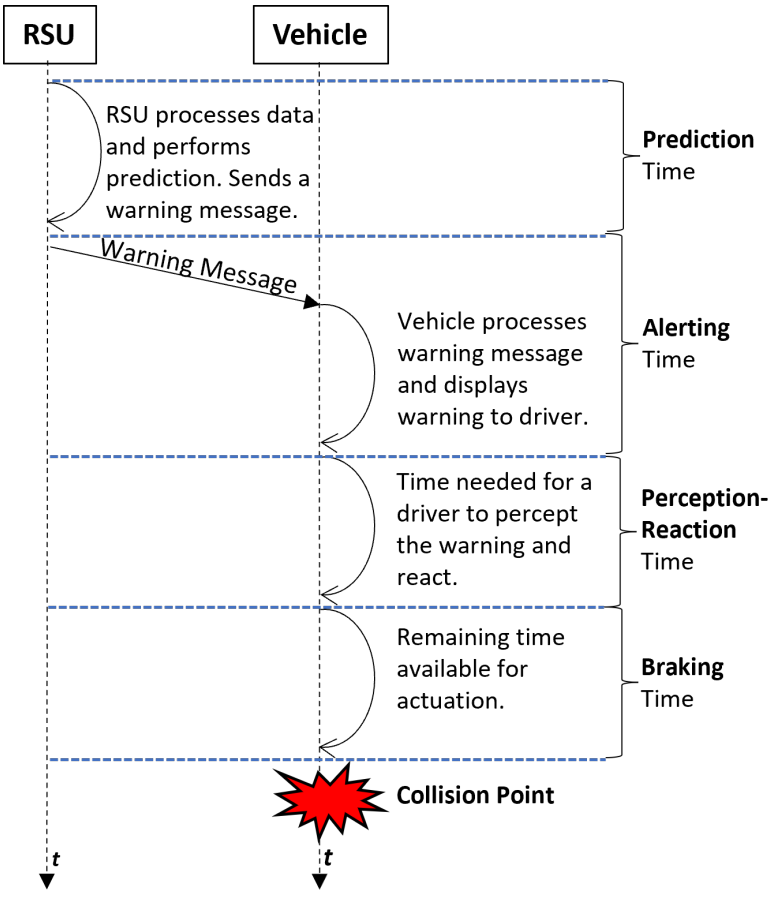


Figure 7.19: The Chronological Events of a Predicted Collision

The timing terms are further discussed next:

Prediction Time Time needed by the **RSU** to collect the environmental data, treat it and feed it as input to the **LSTM** model. The model then outputs the probability for a collision to happen. The time needed for the prediction process is dependent on the complexity of the model: the more complex the model (e.g. higher number of parameters and layers), the longer it takes to perform the prediction. On the other hand, more complex models also provide better predictions, which improve safety overall. Still, considering that it is possible to use high-performance computing resources (such as potent **GPUs**) on the **RSU**, the time needed for this prediction is significantly reduced - these times typically range from microseconds to a few milliseconds.

In positive prediction cases, the **RSU** generates a warning message that is sent to the vicinity. Here, the total time is impacted by the complexity of the message (in this case it is a very simple and short message) and the dissemination protocol that is used (**DSRC**/**WAVE** protocol).

In summary, the *Prediction Time* includes all of the aforementioned processes and, in this case, it is mostly dependent on the capabilities of the system in terms of computational power.

Alerting Time Time needed for the message to be propagated, plus the time needed by the vehicle to process the received warning message and display it on an **HMI**. Similarly to the *Prediction Time*, the processing time is mostly dependent on the computation resources that are available. On the other hand, the propagation time is dependent on the communication characteristics (distance between nodes and wave propagation speed).

Perception-Reaction Time This period corresponds to the sum of two different times. The first one is often called the *perception time* - time needed by the driver to percept the **HMI** warning. The second one is the *reaction time* - time needed by drivers to react to the perception of danger (e.g. how long it takes for the driver to actually start braking).

Both *perception time* and *reaction time* can be influenced by several factors - e.g. the warning design, the driver's experience/skill level, the driver's physical/mental state and age, among others. Naturally, the effectiveness of the system is largely dependent on the *perception-reaction time* and it is important to understand how much time a driver needs to react to warning messages. This work does not focus on the study of these two aspects, as it is not possible to analyze it using the selected simulation framework. Instead, some reference values are used, based on related state of the art works. Some of the most relevant recent works can be found on [122–125].

Braking Time Remaining time that is available for the drivers to perform the braking action in order

to avoid the collision. The *braking time* can be computed for each collision situation within the simulation. When a vehicle receives a warning message, it is possible to log the values related to current speed and current position. This allows (as individual simulations are deterministic) to compute the time (and distance) to the collision point and how long it takes to perform an emergency brake. Naturally, the time that a vehicle needs to perform an emergency brake is also related to their (braking) deceleration values. In real situations, this value is dependent on several factors such as the brake state or road conditions (e.g. dry or wet asphalt, snow or ice-covered roads). Furthermore, in motorcycles, if both wheels or only one wheel is braked, the deceleration value is also affected.

The **SUMO** tool uses values that are based on real world values, although they assume optimal conditions by default (which is also the case on the proposed scenario, which considers a dry asphalt road). Thus, the default values of the simulator were used to compute the braking times that each vehicle needs to perform an emergency break - $9m/s^2$ for passenger vehicles and $10m/s^2$ for motorcycles.

The following subsections focus on the achieved results related to these times. The results were gathered from the same simulation runs that were first used to compile the test datasets for both scenarios. The simulation now included the prediction models running *on-line*, which means that the simulation is now working on a hybrid mode (the simulation is discrete, but the *Python* calls are performed in *real-time*). This way, the **RSU** is able to use the *Python's* functions to process the collected beacons data and to perform the predictions during the simulation runtime. This also allows for the simulation application that is running on the **RSU** to gather all important results for the evaluation of the system. The models that were deployed in the simulation correspond to the *5* **MS** models presented before on Table 7.23.

7.4.1 Results for Prediction Times

As stated before, the time that a model needs to achieve predictions is highly dependent on the hardware computation resources. In this case, a desktop was used with the following characteristics: *AMD Ryzen 5 5600x CPU, 32 GB DDR4 1200 MHz RAM and a NVIDIA GeForce GTX 1060 6GB GPU.*

On average, the prediction, which is performed every second, took 110 μs (real-time). Hence, the prediction time can be considered insignificant.

7.4.2 Results for Alerting Times

In this use case, the proposed scenario is set in a rural environment with a minimum number of vehicles traveling close to the **RSU**. This means that there are no cases of networking congestion. Also, given that the beacons are very small packets (42 bytes), the communications are achieved with very short latencies - the transmission and propagation delays of the packets throughout the simulations are very small.

Table 7.24 presents the messages delays statistics regarding the average delay and standard deviation.

	Scenario A		**Scenario B**	
	Average Delay	**Standard Deviation**	**Average Delay**	**Standard Deviation**
Warning Messages	163.22	0.03	163.23	0.03
Regular Beacons	112.66	10.78	112.50	10.90

Table 7.24: Messages Delays (μs)

As the table shows, the average latency of the messages is really low. Hence, and similarly to the *prediction time*, the *alerting time* is also negligible from a global point of view.

7.4.3 Results for Perception-Reaction and Braking Times

As stated before, using the selected simulation framework, it is not possible to study the *perception-reaction times* of the drivers. Instead, some reference values are used, based on related works (1s, 1.5s, 2.0s and 2.5s).

For that reason, logic is somewhat inverted here. First, one calculates the *braking time* that a vehicle needs to perform an emergency brake. Then, depending on the time that a driver needs to *percept and react*, one computes if it is possible to avoid the collision.

As the simulation is deterministic, it is possible to compute the exact **Time to Collision (TTC)**, which is the amount of time between when the warning arrives and the point of collision. Once the **TTC** is known, it can be compared with the sum of the *Perception-Reaction* and *Braking* times, in order to assess the possibility of avoiding the collision.

The amount of time a vehicle needs to perform an emergency brake and stop is related to two

values: its current speed and its deceleration capabilities. As an example, if a motorcycle is traveling at $20\ m/s$ and has a deceleration capability of $10\ m/s^2$, then it needs $2s$ to brake and avoid a collision.

Table 7.25 illustrates example calculations achieved for the first two collisions that happen on Seed A from Scenario A.

Vehicle Type	Time to Collision (s)	Current Speed (m/s)	Braking Time (s)	Perception--Reaction Time (s)	Total Time (s)	Time Difference (s)
passenger	1.74	8.21	0.91	1.5	2.41	-0.67
motorcycle	1.74	16.39	1.64	1.5	3.14	-1.40
passenger	4.60	10.17	1.13	1.5	2.63	1.97
motorcycle	4.60	11.26	1.13	1.5	2.63	1.97

Table 7.25: Time Measurements - 1st and 2nd Collisions from Seed A, Scenario A (1.5s Reaction Time).

In the first collision case, the vehicles have a **TTC** of 1.74s - in other words, a maximum of 1.74s to *perceive- react* and *brake* to prevent the collision. In this case, as the **TTC** is very short, when subtracting the *braking* and *perception-reaction* times (*total time*) it results in a negative *time difference* for both vehicles. Thus, one can't assume that the collision can be avoided by either one.

In the second example, the **TTC** is much higher (4.60s). In this situation, the final *time difference* is positive - the total amount of time is enough to *perceive-react* and *brake* in order to prevent the collision. Hence, one can consider that the collision can be prevented by both vehicles.

Next subsection presents and discusses the summary results for both scenarios, obtained using similar calculations.

7.4.4 Summary Results and Discussion

As discussed previously, the *prediction* and *alerting times* are in the order of μ-seconds and thus are negligible from a global standpoint. Hence, the performance of the system is more related to the amount of time that drivers have available to *percept-react* and *brake*.

Table 7.26 presents the summary results for each scenario, regarding the preventable collisions when drivers receive a warning.

The performance of the system was similar in both scenarios. In the best-case (1.0s *reaction time*), 94% of the collisions can be prevented in Scenario A and 96% in Scenario B, if any of the drivers performs an emergency brake. Regarding the worst-case (2.5s *reaction time*), still 75% and 69% of the collisions can be avoided.

Reaction Time (s)	Scenario A	Scenario B
1.0	94 %	96 %
1.5	81 %	76 %
2.0	74 %	70 %
2.5	74 %	69 %

Table 7.26: Preventable Collisions - Summary Results

As discussed before on the examples from Table 7.25, one considers that the collision is avoidable if any of the drivers has enough time to perform an emergency braking. Still, however, it may be possible to avoid the collision if both vehicles actuate simultaneously (by braking and/or steering). Naturally, these situations are very hard to account for (specially due to the limitations of the simulation tools). Hence, and although they are not considered towards the preventable collisions count, one can't say for certain that these collisions are absolutely unavoidable. For instance, in real scenarios, both vehicles may steer in opposite directions and still avoid the collision as they are more aware of the situation (even if the times didn't seem to be enough at first).

Results showed that the system was able to predict most collisions in a timely manner: by performing emergency brakes on either involved vehicle, the system is able to prevent at least 74% of the collisions of Scenario A and 69% of Scenario B on the worst-case *perception-reaction* times (2.5s); in the best-cases (1.0s *reaction time*), the system is able to prevent 94% of the collisions of Scenario A and 96% of Scenario B.

As a drawback, using this approach, it is very hard to account for situations when both vehicles can actuate simultaneously (particularly by steering). In that sense, this research work can be extended in the future by performing more complex and specific calculations related to this cases (and analyze if these situations can also be prevented).

7.5 Summary

This chapter discussed a Collision Prediction System, tailored to anticipate collisions involving **VRUs** at intersections. The development of the system was meticulously considered and discussed, emphasizing possible multi-step forecasting strategies and the application of data processing techniques, with particular focus on feature selection and data imbalance issues.

A comprehensive evaluation of the prediction system was then conducted to assess its performance and effectiveness. Several key metrics were analyzed to validate the system's performance: *Number of*

False Positives, Predicted Collisions, Collisions Prediction Percentage, Correct Decision Percentage and *Average Prediction Time*. Globally, the system performed well, as most of the collisions were predicted correctly. The false positives issue was also investigated, and it was found to be related to *near-collision* incidents.

Furthermore, the timeliness of the predictions was also evaluated, to determine if measures could be taken with sufficient lead time to prevent the collisions. This assessment showed that the system's predictions allowed for timely actuation of safety measures by the drivers (by performing emergency braking), thereby enhancing overall intersection safety for **VRUs**.

Chapter 8

Conclusions and Future Work

This final chapter presents a critical analysis of the obtained results and the conclusions taken from the overall research work. Additionally, some drawbacks and limitations presented by this research's work are also discussed. Finally, some future steps are proposed, indicating subsequent research work that may complement the work presented in this thesis.

8.1 Conclusions

This thesis investigates the feasibility of leveraging the data that is exchanged by road agents by means of **V2X** communications, in order to improve **VRUs** safety. This document discusses the proposal, implementation, test and evaluation of a system's architecture that aims to improve **VRUs** safety, by predicting potential collisions at an intersection.

The first part of this research work focused on presenting and analyzing different **ITS** contexts and corresponding scientific background. Some of the most important **ITS** communications standards were identified, for both American and European standards; the main vehicular communication modes and technologies are also explored and discussed.

Given the fact that **V2X** communications generate huge amounts of useful road data, **ML** techniques have the potential to leverage it and solve many problems in **ITS**. **ML** systems can build knowledge extracting patterns and time dependency relations from historical **V2X** data collections, combining it with real-time information in order to implement and establish useful **ITS** systems.

In that sense, it is pertinent to evaluate the usefulness of **ML** in this context. Hence, existing **ML** solutions for **ITS** were surveyed - both for generic **ITS** solutions, but also related to the detection/prediction of incidents (and in particular those related to **VRUs**). A very large part of the surveyed solutions are related to the managing and optimizing, although some also focus on improving communication issues. Regarding the use of **ML** in the prediction of incidents, works tend to focus on data that is collected via in-vehicle

sensors, such as cameras or similar in-vehicle devices (even for **VRUs** incidents prediction). Furthermore, **VRUs** related works tend to focus on *pedestrians* and *bicycles* and not so much on other vulnerable users such as *reduced mobility persons* or *motorcycles*.

During the related work survey, no datasets related to **VRUs** collisions that contains **ITS** standard messages exchanges were found - most of the identified works don't have the datasets publicly available, or even present details about them. For that reason, there was a need to construct new datasets in order to gather **V2X** data, to classify this data and make it publicly available, so that **ML** models can be extensively trained and tested. Two possible solutions were identified to generate the datasets: using real world testing or simulation.

Performing **ITS** field tests on real prototypes is a very expensive task, both in terms of money and time. Alternatively, simulators are a very popular choice for evaluating **ITS** solutions within the research community, since they bring many advantages in terms of cost, repeatability, scale and practical requirements. Hence, several simulation tools and frameworks were identified and compared - both for communications and mobility simulation. Most of the surveyed tools were not open source or in active development, which made them unsuitable solutions. Additionally, coupled frameworks are more interesting solutions than using individual tools. From the available coupled tools, *Artery*, **VEINS** and *Eclipse Mosaic* are the most interesting solutions. Finally, considering the *Artery's* lack of documentation and that both **VEINS** and *Eclipse Mosaic* have similar characteristics and features, the **VEINS** tool was selected to be used (ultimately, it was a matter of personal preference).

As stated above, the main goal of this thesis is to evaluate the feasibility of using **V2X** data as input for **ML** models to predict collisions involving **VRUs**. Naturally, such a prediction system requires large amounts of storage and computation resources for training and testing **ML** models. Also, considering that it is a safety system, the usage of the model for predictions also requires very low latencies, so that users have enough time to safely actuate when a warning message is disseminated.

This thesis proposes a system's architecture, based on Fog Computing, that organizes the different road entities and functionalities of the system in hierarchical layers according to their characteristics (mainly in terms of computation requirements, storage, timeliness and utility). The proposed architecture is divided in three levels: *Cloud Layer*, *Fog Layer* and *Edge Layer*. In this design, the heaviest computational and less time-critical functions are placed at the upper layer (*Cloud Layer*), and the lighter operations at the lower *Edge Layer* (while the middle *Fog Layer* is a middle ground). At the *Edge Layer*, the end users (vehicles/drivers) collect ego information from sensors and share it with other users using vehicular communications. The *Fog Layer* is the ideal place to deploy the **ML** models for prediction of collisions - they

receive the data that is disseminated from the surrounding road users, treat and use it for the prediction. Also, it connects with the top layer (*Cloud Layer*) to obtain more powerful computing and storage capabilities. This top layer, being the most powerful in terms of capabilities, is the most adequate place to train (and retrain) the **ML** models.

In order to synthesize the datasets that are to be fed as input for the **ML** models, a simulation scenario was designed and implemented on the **VEINS** framework. The established scenario was based on use cases from **ETSI** standards and consists on a critical traffic situation where a passenger vehicle makes a turn on an intersection and oversees an approaching motorcycle that intends to go straight, resulting in a possible collision. In terms of communications, both passenger vehicles and motorcycles possess an **OBU** equipped with *IEEE 802.11p* and are able to exchange periodical beacons (**BSMs**) with ego-vehicle information. These beacons are filled with the following information: *Station ID, Longitude, Latitude, Elevation/Altitude, Heading, Speed, Acceleration, Vehicle Length, Vehicle Width and Vehicle Type*. Additionally, a **RSU** is placed next to the intersection, which also possesses *IEEE 802.11p* (enabling it to also received the beacons) and is able to run a **ML** mechanism to treat the **V2X** data for the prediction of collisions. If the **RSU** predicts a potential collision, it is able to trigger and broadcast collision warning messages to vehicles in the area. Furthermore, the **RSU** is responsible for compiling the datasets - in this case, they consist of the collection of beacons (**BSM**) that both passenger and motorcycle vehicles are exchanging during the simulation run. Each simulation run is executed for 24h of simulation time with a different simulation seed. This assures that each run results in different mobility behaviors and, consequently different simulation results and datasets. By the end, a total of fifty simulation runs were used to compiled the three final datasets which have been used to train (80% of the data), validate (10%) and test (10%) the **ML** collision prediction system. These final synthesized datasets were made publicly available and can be accessed in https://zenodo.org/record/7376770 [108].

This first developed model's goal was to classify the vehicle's type based on the messages content - in other words, if the message was originated by a passenger vehicle or a motorcycle. Several common **ML** prediction models were chosen for initial testing: **LR**, **KNN**, **GNB**, **SVM** and **ANN**. Naturally, being such a simple problem, all models achieved an *accuracy* of 100%.

In a first step towards the collision prediction, a new feature was added at this point - *inCollision*. This feature stated if a record was related to a collision (or not). Then, those aforementioned models were used to classify if the message belonged to a collision and they all achieving accuracy values above 80%. The high accuracy values sound promising but, in fact, this simple solution is a very poor one. This is related to the fact that the dataset is highly **imbalanced**. Only a few collisions happen during the simulation

runs, for they are rare events. This results in only a few messages being marked as *true*. Naturally, this explains the high *accuracy* values on the models results: even if every single record is labeled as *false*, the final metric value is very high, since only a few records are being wrongly labeled. Each of the tested models performed very poorly and were not able to classify the collisions properly. Naturally, given that the traditional models performance was poor, the problem needed to be approach in a different way.

Considering the use case of predicting collisions at intersections, a key aspect is that there are very strong temporal issues and that there is a need to carry out a continuous temporal analysis of the collected **V2X** messages data. A suitable model for solving such type of problems is **LSTM**, for its ability to hold information for long periods of time (early learned information can still be useful later on the model's decision). Besides **LSTMs**, **MLPs** were also identified as being a suitable solution, since they work well on tabular datasets and in binary classification prediction problems. Both types of **NN** models were used for the collision detection and prediction systems.

A collision detection system was first developed, in order to have a starting point towards the more complex issue of prediction. In summary, the detection system aims to detect if a collision has occurred between *passenger* vehicles and *motorcycles* on an intersection. Most of the tested models performed relatively well, although they presented some limitations regarding the high number of **FPs**. Even so, this problem was mitigated through the use of a specific *collision detection logic* after the classification itself. As a drawback, using such a logic slightly delays the collision detection itself, so there is a certain trade-of notion present. The best performing model consisted on a **MLP** model with two *hidden* layers (64 *neurons*) which used all features on the *input layer*. This model was able to detect every single collision in 1s or less, taking 0.62s on average.

After developing, testing and evaluating the collision detection system, a similar process was achieved for a more complex system for their prediction. Firstly, different possible strategies for *multi-step forecasting* have been identified and analyzed. When weighting the pros and cons of the proposed solutions, the *Multiple Output Forecasting* strategy was selected to predict the collisions, as it requires a single prediction model and relies only on real observed values to predict future ones (despite requiring heavier computation). The scripts for collision detection were then reworked and adapted to the established forecasting strategy.

In order to tune the prediction model to be able to predict the collisions in a satisfactory way, several optimization techniques have been performed: several model's variations (in terms of number of layers, *batch size*, number of *neurons*, among others); testing different *feature selection* methods; exploring alternative data *preprocessing* methods (*standardization* vs *normalization*); exploring automatic

hyper-parameterization optimization techniques (e.g. *grid search*); tweaking some data preprocessing techniques (e.g. using *sum* vs *average* values when performing the data aggregation).

This process of exploring these different methods and techniques in order to improve the model's performance was arduous and took several months of work, without producing adequate results. Despite the great effort, the tested models (both **MLPs** and **LSTMs** variants) were only able to predict roughly one third of the collisions accurately in the best cases. Moreover, they still had problems regarding having a high number of **FPs** classifications. As the models were not able to properly converge, unlike in the collision detection system case, it raised questions regarding the datasets quality. Despite the initial set of data (taken from ten simulation runs) having hundreds of thousands of messages records, collisions are rare occasions on the simulation runs. In that sense, despite having large amounts of raw data, the available amount of information on collisions could not be sufficient for the model's to converge and learn. As the amount of collisions data was scarce, more simulation runs were executed and new and bigger datasets were synthesized, in order to overcome the initial poor performance - first using twenty runs, then thirty, fifty and finally one hundred (always using 80% for training, 10% for validation and 10% for testing).

In fact, using larger datasets greatly improved the performance of the model's forecasts, which proved that there wasn't sufficient collision data for the model's to properly converge. The results from the evaluation achieved next relied on the usage of data from fifty simulation runs, as no improvement was noted when using one hundred (and the computation cost was much higher). Naturally, using greater amounts of data turned the learning process much slower (heavier in terms of computation), and even caused issues in terms of memory. To solve that issue, the training data was truncated on those large periods of time where there are no relevant information on collisions - which also helped in dealing with the data imbalance issue.

The final results from the evaluation were achieved in an iterative fashion - performing runs on all possible combinations would be highly consuming, both in terms of time and computation, making it impractical.

Naturally, such an approach may end up not presenting the optimal results, as some other parameters that were not tested could eventually perform better.

The *Collision Prediction System* models were analyzed, with particular focus on the following metrics: *Number of False Positives, Predicted Collisions, Collisions Prediction Percentage, Correct Decision Percentage* and *Average Prediction Time*.

The best performing models were grouped by multi-steps (how many seconds in the future one is forecasting) and, globally, they all achieved good results. Even the worst model (the model for 2**MS** on

Scenario B) predicted 81% of the collisions on the test dataset. On the other hand, the main issue of the prediction systems were associated with high number of **FPs** - which was transversal to all models.

Still, when considering the **MS** alternatives, the results can be looked from two perspectives: if one considers that the prediction of the collisions and the **CDP** are a priority, using three **MS** is the best solution; On the other hand, if higher **APT** are preferred, then using the five **MS** is better (despite lower **CDP** values due to higher **FPs**).

Having too many **FPs** can be severely prejudicial, as it may annoy the drivers - leading them to not react, ignore the warnings or even disable the safety system. Being a crucial point, this issue was investigated on the mobility simulations. In fact, this **FPs** issue was related to *near-collision* situations that happen frequently on the simulation. As the mobility pattern of the vehicles is very similar to collisions, this causes the system to raise *false alarms* (although the vehicles don't actually collide). Despite being **FPs** from a statistical point of view, they are traffic situations that one wishes to avoid nonetheless. Hence, this **FPs** shouldn't be seen as necessarily bad results - a warning notification may also help to avoid those near-collisions, improving traffic in a broader manner.

Naturally, the system is not suited (but was never aimed) for automatic actuation mechanisms, such as *automatic emergency braking*. The most practical way of actuating is in a passive way: it is better to warn the drivers of imminent danger, leaving up to them to access the situation and perform defensive actions to prevent the collisions. In real-world situations, the warnings may consist on: visual alerts (visual warnings displayed on an **HMI**); auditory warnings (in the form of beeps or spoken messages); haptic feedback (tactile sensations, such as vibrations on the steering wheel); augmented reality (highlighting potential collision areas using virtual objects); among others.

Finally, considering that discussion, it makes more sense to resort to the five **MS** solution, since it has higher **APTs**. In other words, it provides more time for the drivers to acknowledge the warning and actuate if necessary. The best performing model for Scenario A was able to predict 96% of the collisions (with an **APT** of 4.53), while the best model for Scenario B predicted 95% of the collisions (with an **APT** of 4.44s).

More than analyzing the system's ability to predict collisions, it is also crucial to evaluate if the **APT** gives enough time for the vehicles to actuate in order to avoid the collisions. Hence, these two final models were evaluated regarding the timeliness of the collision: in terms of *prediction time* (time for the model's prediction), *alerting time* (time needed to send a warning message to the drivers), *perception-reaction time* (time that the drivers need to percept the warning and to react - e.g. start braking) and *braking time* (remaining time available for actuation).

In particular, one wishes to verify if the drivers have enough time to avoid the collision after receiving the warning message (which is sent after the model computes a positive prediction). As it is not possible to study the *perception-reaction* times in the selected simulation framework (**VEINS**), some reference values were identified from related works - 1s, 1.5s, 2.0s and 2.5s.

To compute if the drivers are able to avoid the collision, one first computes the exact **TTC** (amount of time between when the warning arrives and the collision time). Then, using logged values at the moment of the reception of the warning, one calculates the *braking time* that a vehicle needs to perform an emergency brake. Then, depending on the time that a driver needs to percept and react, one finally checks if the collision can be avoided or not.

On the performed tests, the *prediction* and *alerting times* are in the order of μ-seconds and thus are negligible from a global standpoint. Hence, the system's performance is more related to the *perception-reaction time* and *braking time*.

The performance of the system was similar in both scenarios: in scenario A, 74% of the collisions can be avoided in the worst *perception-reaction* case (2.5s) and 94% on the best case (1.0s); in scenario B, 69% of the collisions can be avoided in the worst case and 96% on the best case.

This study considers that the collision is avoidable if at least one of the drivers (*passenger vehicle* or *motorcycle*) has enough time to perform an emergency brake. It does not consider the possibility of both vehicles actuating at the same time (either by braking and or steering), since these situations are hard to account for.

Hence, one can't say for certain that these collisions are absolutely unavoidable - still, they are not considered towards the preventable collisions count. Naturally, in real situations, both vehicle's may steer in completely opposite directions and still avoid the collision, even if the computed timings didn't seem to be enough at first.

In conclusion, this PhD thesis has successfully defined a system architecture and a **ML** system to predict collisions for **VRUs**, which is an important step forward in increasing safety measures for **ITS**. As a consequence of extensive data collection and analysis, this research has shown the potential of **ML** techniques for an accurate forecast of potential collisions involving **VRUs** (in this case, motorcycles on an intersection). By leveraging this predictive capabilities, one is able to take proactive measures to prevent collisions (or at least minimize its effects) and protect lives, thus improving traffic safety in general. The most relevant publications that present results directly related to this thesis work can be found in [12–15].

8.2 Limitations and Future Work

This thesis presents a **ML**-based safety systems for **VRUs**, which relied on the usage of datasets that were synthesized resorting to a simulation framework (**VEINS**). Naturally, together with the fact that the datasets are also self-produced, this could introduce involuntary *bias* in the results. Ideally, the datasets should have used data collected from real world environments. Or, as an alternative, these systems could have relied on *third-party* datasets. Unfortunately, no datasets were found when surveying the state of the art works. Still, as a positive aspect, the synthesized datasets that result from this thesis work are available publicly - which allows third parties to evaluate and scrutinize the results.

As for the datasets synthesis process, it relied on two different mobility patterns on a similar scenario on **SUMO**. Although the scenario is based on **ETSI**'s technical reports on typical use cases relevant to traffic safety that involve **VRUs**, the resulting system still has some limitations. In particular, considering that the models were trained using simple mobility patterns, it may limit the prediction capabilities of the system in more complex traffic environments. In that sense, it is unclear how the models would behave in more complex situations, in terms of traffic density and/or road configuration. Nevertheless, the implemented system can still be easily extended to consider new traffic situations using the same methodology on new scenarios (with different geographical areas or mobility patterns).

At this point, when a potential collision is predicted, the warning messages that are triggered by the system are being simply broadcast, which means that all the vehicles within range are able to receive and process the warning. On more advanced systems, it is also important to determine which vehicles should in fact be warned or, in a different approach, which vehicles should take an action upon receiving a warning message - the system should aim to alert vehicles that are in immediate danger or have a high probability of colliding. Hence, on future works, the system should take into consideration factors such as: its vicinity to the intersection (consider only vehicles that are close enough to the intersection); direction of travel (vehicles that are approaching the intersection, and not vehicles that are moving away); traffic light current status (e.g. prioritize vehicles who are approaching a green or yellow light); lane configuration (e.g. not sending warnings to vehicles that are circulating on a lane that will not cross the intersection); etc. Such a targeted approach ensures that the warning messages are either only sent to the relevant parties or only reacted upon by specific vehicles, avoiding unnecessary distractions for other users, further optimizing the overall road safety.

Finally, the proposed system is built upon a hierarchical architecture that is inspired on *Fog Computing*. However, this thesis does not explore or analyze the architecture from a global standpoint. As future

work, one considers that it could be interesting to study the architecture as a whole (as in a city-wide view) - in terms of resources allocation, communications efficiency, data aggregation, scalability, adaptability and any other kind of integration with other smart city systems and applications. Also, specific security measures should be taken into the system, in order to avoid security attacks and preserve data privacy.

Bibliography

[1] European Commision. ITS & Vulnerable Road Users. URL https://ec.europa.eu/transport/themes/its/road/action_plan/its_and_vulnerable_road_users_en. [Online; Last accesed on: 15-04-2021].

[2] Yahoo. Global Automotive V2X (V2V, V2I, V2P, V2G, V2C, and V2D) Markets, 2022-2028 with Qualcomm, Autotalks, Continental, Cohda Wireless, and Robert Bosch Dominating. https://uk.finance.yahoo.com/news/global-automotive-v2x-v2v-v2i-123300325.html, September 2022. [Online; Last accessed on: 20-09-2022].

[3] Research and Markets. Global Automotive V2X Market by Connectivity (DSRC, and Cellular), Communication (V2V, V2I, V2P, V2G, V2C, and V2D), Vehicle Type (Passenger Cars, and Commercial Vehicles), Propulsion (ICE and EV), Unit, Offering, Technology and Region - Forecast to 2028. https://www.researchandmarkets.com/reports/4991293/global-automotive-v2x-market-by-connectivity, August 2022. [Online; Last accessed on: 20-09-2022].

[4] PR Newswire. By 2025, More than 10 Million Vehicles Will Be Capable of Short-Range V2X Communication. https://www.prnewswire.com/news-releases/by-2025-more-than-10-million-vehicles-will-be-capable-of-short-range-v2x-communication-301623836.html, September 2022. [Online; Last accessed on: 20-09-2022].

[5] ABI Research. Vehicle-to-Everything (V2X). https://www.abiresearch.com/market-research/product/7779722-vehicle-to-everything-v2x/, August 2022. [Online; Last accessed on: 20-09-2022].

[6] Cohda Wireless. Cohda Wireless is Proud to Supply V2X Technology for the Recently Launched Cadillac CTS. https://www.cohdawireless.com/cohda-wireless-is-proud-to-supply-v2x-technology-for-the-recently-launched-cadillac-cts/. [Online; Last accessed on: 20-09-2022].

[7] Yasser L Morgan. Notes on DSRC & WAVE standards suite: ITS architecture, design, and characteristics. *IEEE Communications Surveys & Tutorials*, 12(4):504–518, 2010.

[8] David Eckhoff, Nikoletta Sofra, and Reinhard German. A performance study of cooperative awareness in ETSI ITS G5 and IEEE WAVE. In *2013 10th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, pages 196–200. IEEE, 2013.

[9] SAE. SAE J2735 - V2X Communications Message Set Dictionary, 2020-07.

[10] ETSI. ETSI EN 302 637-2 V1.4.1 Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service, 2019-09.

[11] ETSI. ETSI EN 302 637-3 V1.3.1 Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 3: Specifications of Decentralized Environmental Notification Basic Service, 2019-04.

[12] Bruno Ribeiro, Maria João Nicolau, and Alexandre Santos. Leveraging vehicular communications in automatic VRUs accidents detection. In *2022 Thirteenth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 326–331. IEEE, 2022.

[13] Bruno Ribeiro, Maria João Nicolau, and Alexandre Santos. Machine Learning for VRUs accidents prediction using V2X data. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, pages 1789–1791, 2023.

[14] Bruno Ribeiro, Maria João Nicolau, and Alexandre Santos. Using Machine Learning on V2X Communications Data for VRU Collision Prediction. *Sensors*, 23(3):1260, 2023.

[15] Bruno Ribeiro, Maria João Nicolau, and Alexandre Santos. Evaluation of a Collision Prediction System for VRUs Using V2X and Machine Learning: Intersection Collision Avoidance for Motorcycles. In *28th IEEE Symposium on Computers and Communications (ISCC) 2023)*. IEEE, 2023.

[16] Fábio Gonçalves, Bruno Ribeiro, João Santos, Oscar Gama, Francisco Castro, João Fernandes, António Costa, Bruno Dias, Maria João Nicolau, Joaquim Macedo, et al. Enhancing VRUs Safety with V2P communications: an experiment with hidden pedestrians on a crosswalk. In *2022 14th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, pages 96–103. IEEE, 2022.

[17] Oscar Gama, Antonio Costa, Maria Joao Nicolau, Alexandre Santos, Joaquim Macedo, Bruno Dias, Fabio Goncalves, and Bruno Ribeiro. Design and Evaluation of an Adaptive Virtual Traffic Light System for VANETs. In *2022 14th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, pages 177–184. IEEE, 2022.

[18] Fábio Gonçalves, Bruno Ribeiro, Oscar Gama, Maria Joao Nicolau, Bruno Dias, António Costa, Alexandre Santos, and Joaquim Macedo. Agnostic Middleware for VANETs: Specification Implementation and Testing. In *WINSYS- 19th International Conference on Wireless Networks and Mobile Systems, 2022*, 2022.

[19] Fábio Gonçalves, Bruno Ribeiro, Oscar Gama, João Santos, António Costa, Bruno Dias, Maria João Nicolau, Joaquim Macedo, and Alexandre Santos. Synthesizing datasets with security threats for Vehicular Ad-hoc Networks. In *GLOBECOM 2020-2020 IEEE Global Communications Conference*, pages 1–6. IEEE, 2020.

[20] Oscar Gama, Alexandre Santos, Antonio Costa, Maria João Nicolau, Bruno Dias, Joaquim Macedo, Bruno Ribeiro, Fabio Goncalves, and Joao Simoes. Evaluation of push and pull communication models on a VANET with virtual traffic lights. *Information*, 11(11):510, 2020.

[21] Oscar Gama, Alexandre Santos, Antonio Costa, Bruno Dias, Joaquim Macedo, Maria Joao Nicolau, Bruno Ribeiro, and Fabio Goncalves. Evaluation of broadcast storm mitigation techniques on vehicular networks enabled by WAVE or NDN. In *Intelligent Transport Systems. From Research and Development to the Market Uptake: Third EAI International Conference, INTSYS 2019, Braga, Portugal, December 4–6, 2019*, pages 219–236. Springer, 2020.

[22] Fábio Gonçalves, Bruno Ribeiro, Oscar Gama, Alexandre Santos, António Costa, Bruno Dias, Joaquim Macedo, and Maria Joao Nicolau. A systematic review on intelligent intrusion detection systems for VANETs. In *2019 11th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, pages 1–10. IEEE, 2019.

[23] The European Parliament, Council of European Union. DIRECTIVE 2010/40/EU. `https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32010L0040`, July 2010. [Online; Last accesed on: 20-12-2022].

[24] ETSI. ETSI ETSI EN 302 665 V1.1.1 Intelligent Transport Systems (ITS); Communications Architecture, 2010-09.

[25] ETSI. ETSI TR 102 638 V1.1.1 Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Definitions, 2009-06.

[26] ETSI. ETSI EN 302 895 V1.1.1 Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Local Dynamic Map (LDM) , 2014-09.

[27] ETSI. ETSI EN 302 636-1: Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 1: Requirements , 2014-04.

[28] ETSI. ETSI TS 102 636-5-1 V2.2.1 Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 5: Transport Protocols; Sub-part 1: Basic Transport Protocol, 2019-05.

[29] ETSI. Final Draft ETSI EN 302 663 V1.3.1 - Intelligent Transport Systems (ITS); Access layer specification for Intelligent Transport Systems operating in the 5 GHz frequency band, 2019-10.

[30] IEEE. IEEE Guide for Wireless Access in Vehicular Environments (WAVE) Architecture. *IEEE Std 1609.0-2019 (Revision of IEEE Std 1609.0-2013)*, pages 1–106, 2019.

[31] IEEE. Trial-Use Standard for Wireless Access in Vehicular Environments (WAVE) – Resource Manager. *IEEE Std 1609.1-2006*, pages 1–71, 2006.

[32] IEEE. IEEE Standard for Wireless Access in Vehicular Environments (WAVE) – Over-the-Air Electronic Payment Data Exchange Protocol for Intelligent Transportation Systems (ITS). *IEEE Std 1609.11-2010*, pages 1–62, 2011.

[33] IEEE. IEEE Standard for Wireless Access in Vehicular Environments (WAVE) – Networking Services. *IEEE Std 1609.3-2020 (Revision of IEEE Std 1609.3-2016)*, pages 1–210, 2021.

[34] IEEE. IEEE Standard for Wireless Access in Vehicular Environments (WAVE) – Identifiers. *IEEE Std 1609.12-2019 (Revision of IEEE Std 1609.12-2016)*, pages 1–17, 2019.

[35] IEEE. IEEE Standard for Wireless Access in Vehicular Environments (WAVE) – Multi-Channel Operation. *IEEE Std 1609.4-2016 (Revision of IEEE Std 1609.4-2010)*, pages 1–94, 2016.

[36] IEEE. IEEE Standard for Information Technology – Local and metropolitan area networks – Specific requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 6: Wireless Access in Vehicular Environments. *IEEE Std 802.11p-2010*

(Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, IEEE Std 802.11n-2009, and IEEE Std 802.11w-2009), pages 1–51, 2010.

[37] Bluetooth. Bluetooth Core Specification v5.4. `https://www.bluetooth.com/specifications/specs/core-specification-5-4/`, 2023. [Online; Last accessed on: 23-07-2023].

[38] Raphael Frank, Walter Bronzi, German Castignani, and Thomas Engel. Bluetooth Low Energy: An alternative technology for VANET applications. In *2014 11th annual conference on wireless on-demand network systems and services (WONS)*, pages 104–107. IEEE, 2014.

[39] IEEE. IEEE Standard for Local and metropolitan area networks–Part 15.7: Short-Range Optical Wireless Communications, IEEE Std IEEE 802.15.7-2018 . 2018.

[40] Khaled Shaaban, Md Hosne Mobarok Shamim, and Khadija Abdur-Rouf. Visible light communication for intelligent transportation systems: A review of the latest technologies. *Journal of traffic and transportation engineering (English edition)*, 8(4):483–492, 2021.

[41] Daniel Jiang and Luca Delgrossi. IEEE 802.11p: Towards an International Standard for Wireless Access in Vehicular Environments. In *VTC Spring 2008-IEEE Vehicular Technology Conference*, pages 2036–2040. IEEE, 2008.

[42] 3GPP. Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Overall description; Stage 2 (v14.3.0, Release 14). Tech. Rep. 36.300, 2017-06.

[43] 3GPP. "Study on enhancement of 3GPP support for 5G V2X services" (v15.0.0, Release 15). Tech. Rep. 22.886, 2017-03.

[44] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning, 2nd Edition*. MIT press, 2018.

[45] Kajaree Das and Rabi Narayan Behera. A survey on machine learning: concept, algorithms and applications. *International Journal of Innovative Research in Computer and Communication Engineering*, 5(2):1301–1309, 2017.

[46] Mohammad Saeid Mahdavinejad, Mohammadreza Rezvan, Mohammadamin Barekatain, Peyman Adibi, Payam Barnaghi, and Amit P Sheth. Machine learning for internet of things data analysis: A survey. *Digital Communications and Networks*, 4(3):161–175, 2018.

[47] Saint John Walker. Big data: A revolution that will transform how we live, work, and think. *International Journal of Advertising*, 33(1):181–183, 2014.

[48] P. Bedi and V. Jindal. Use of big data technology in vehicular ad-hoc networks. In *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1677–1683, Sep. 2014.

[49] Juan Contreras-Castillo et al. Solving vehicular ad hoc network challenges with big data solutions. *IET Networks*, 5(4):81–84, 2016.

[50] F. Terroso-Saenz, M. Valdes-Vela, C. Sotomayor-Martinez, R. Toledo-Moreo, and A. F. Gomez-Skarmeta. A Cooperative Approach to Traffic Congestion Detection With Complex Event Processing and VANET. *IEEE Transactions on Intelligent Transportation Systems*, 13(2):914–929, June 2012. ISSN 1524-9050.

[51] Wei Kuang Lai, Mei-Tso Lin, and Yu-Hsuan Yang. A Machine Learning System for Routing Decision-making in Urban Vehicular Ad Hoc Networks. *Int. J. Distrib. Sen. Netw.*, 2015:6:6–6:6, January 2015. ISSN 1550-1329.

[52] Jyoti Grover, Nitesh Kumar Prajapati, Vijay Laxmi, and Manoj Singh Gaur. Machine learning approach for multiple misbehavior detection in VANET. In *International Conference on Advances in Computing and Communications*, pages 644–653. Springer, 2011.

[53] Gitanjali Bhutani. Application of machine-learning based prediction techniques in wireless networks. *International Journal of Communications, Network and System Sciences*, 7(05):131, 2014.

[54] Pierre-Luc Gregoire, Charles Desjardins, Julien Laumonier, and Brahim Chaib-draa. Urban traffic control based on learning agents. In *2007 IEEE Intelligent Transportation Systems Conference*, pages 916–921. IEEE, 2007.

[55] Hamzah Al Najada and Imad Mahgoub. Big vehicular traffic data mining: Towards accident and congestion prevention. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2016 International*, pages 256–261. IEEE, 2016.

[56] Jair Ferreira Júnior, Eduardo Carvalho, Bruno V Ferreira, Cleidson de Souza, Yoshihiko Suhara, Alex Pentland, and Gustavo Pessin. Driver behavior profiling: An investigation with different smartphone sensors and machine learning. *PLoS one*, 12(4):e0174959, 2017.

[57] Nejdet Dogru and Abdulhamit Subasi. Traffic accident detection using random forest classifier. In *2018 15th Learning and Technology Conference (L&T)*, pages 40–45. IEEE, 2018.

[58] Murat Ozbayoglu, Gokhan Kucukayan, and Erdogan Dogdu. A real-time autonomous highway accident detection model based on big data processing and computational intelligence. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 1807–1813. IEEE, 2016.

[59] Zhenhua Zhang, Qing He, Jing Gao, and Ming Ni. A deep learning approach for detecting traffic accidents from social media data. *Transportation research part C: emerging technologies*, 86: 580–596, 2018.

[60] Mario Munoz-Organero, Ramona Ruiz-Blaquez, and Luis Sánchez-Fernández. Automatic detection of traffic lights, street crossings and urban roundabouts combining outlier detection and deep learning classification techniques based on gps traces while driving. *Computers, Environment and Urban Systems*, 68:1–8, 2018.

[61] Imran A Zualkernan, Fadi Aloul, Fayiz Basheer, Gurdit Khera, and Shruthi Srinivasan. Intelligent accident detection classification using mobile phones. In *2018 International Conference on Information Networking (ICOIN)*, pages 504–509. IEEE, 2018.

[62] Bo Xu, Tiffany Barkley, Andrew Lewis, Jane MacFarlane, Davide Pietrobon, and Matei Stroila. Real-time detection and classification of traffic jams from probe data. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 79. ACM, 2016.

[63] Bin Pan and Hao Wu. Urban traffic incident detection with mobile sensors based on svm. In *2017 XXXIInd General Assembly and Scientific Symposium of the International Union of Radio Science (URSI GASS)*, pages 1–4. IEEE, 2017.

[64] Yu Chen, Yuanlong Yu, and Ting Li. A vision based traffic accident detection method using extreme learning machine. In *2016 International Conference on Advanced Robotics and Mechatronics (ICARM)*, pages 567–572. IEEE, 2016.

[65] Xiaohui Huang, Pan He, Anand Rangarajan, and Sanjay Ranka. Intelligent intersection: Two-stream convolutional networks for real-time near accident detection in traffic video. *arXiv preprint arXiv:1901.01138*, 2019.

[66] Yiru Zhao, Bing Deng, Chen Shen, Yao Liu, Hongtao Lu, and Xian-Sheng Hua. Spatio-temporal autoencoder for video anomaly detection. In *Proceedings of the 25th ACM international conference on Multimedia*, pages 1933–1941. ACM, 2017.

[67] VC Maha Vishnu, M Rajalakshmi, and R Nedunchezhian. Intelligent traffic video surveillance and accident detection system with dynamic traffic signal control. *Cluster Computing*, pages 1–13, 2017.

[68] Michael Goldhammer, Sebastian Köhler, Stefan Zernetsch, Konrad Doll, Bernhard Sick, and Klaus Dietmayer. Intentions of Vulnerable Road Users – Detection and Forecasting by Means of Machine Learning. *IEEE transactions on intelligent transportation systems*, 21(7):3035–3045, 2019.

[69] Raúl Parada, Anton Aguilar, Jesus Alonso-Zarate, and Francisco Vázquez-Gallego. Machine Learning-based Trajectory Prediction for VRU Collision Avoidance in V2X Environments. In *2021 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2021.

[70] Md Mostafizur Rahman Komol, Md Mahmudul Hasan, Mohammed Elhenawy, Shamsunnahar Yasmin, Mahmoud Masoud, and Andry Rakotonirainy. Crash severity analysis of Vulnerable Road Users using Machine Learning. *PLoS one*, 16(8):e0255828, 2021.

[71] Mariana Vilaça, Eloísa Macedo, and Margarida C Coelho. A Rare Event Modelling Approach to Assess Injury Severity Risk of Vulnerable Road Users. *Safety*, 5(2):29, 2019.

[72] Muhammad Ijaz, Muhammad Zahid, Arshad Jamal, et al. A comparative study of machine learning classifiers for injury severity prediction of crashes involving three-wheeled motorized rickshaw. *Accident Analysis & Prevention*, 154:106094, 2021.

[73] Benjamin Völz, Holger Mielenz, Gabriel Agamennoni, and Roland Siegwart. Feature relevance estimation for learning pedestrian behavior at crosswalks. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 854–860. IEEE, 2015.

[74] Andreas Jahn, Klaus David, and Sebastian Engel. 5G/LTE based protection of Vulnerable Road Users: Detection of crossing a curb. In *2015 IEEE 82nd Vehicular Technology Conference (VTC2015-Fall)*, pages 1–5. IEEE, 2015.

[75] Maarten Bieshaar, Günther Reitberger, Stefan Zernetsch, Bernhard Sick, Erich Fuchs, and Konrad Doll. Detecting Intentions of Vulnerable Road Users Based on Collective Intelligence. *arXiv preprint arXiv:1809.03916*, 2018.

[76] SUMO - Simulation of Urban MObility. https://eclipse.dev/sumo/, . [Online; Last accessed on: 01-04-2023].

[77] Michael Behrisch, Laura Bieker, Jakob Erdmann, and Daniel Krajzewicz. SUMO – Simulation of Urban Mobility: an Overview. In *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*. ThinkMind, 2011.

[78] OSM - OpenStreetMap. https://www.openstreetmap.org. [Online; Last accessed on: 01-04-2023].

[79] Feliz Kristianto Karnadi, Zhi Hai Mo, and Kun-chan Lan. Rapid Generation of Realistic Mobility Models for VANET. In *2007 IEEE Wireless Communications and Networking Conference*, pages 2506–2511. IEEE, 2007.

[80] PTV VISSIM - Multimodal Traffic Simulation Software. https://www.myptv.com/en/mobility-software/ptv-vissim. [Online; Last accessed on: 26-07-2023].

[81] VanetMobiSim. http://vanet.eurecom.fr/. [Online; Last accessed on: 01-07-2020].

[82] TSIS-CORSIM: Traffic Software Integrated System - Corridor Simulation. http://mctrans.ce.ufl.edu/mct/index.php/tsis-corsim/. [Online; Last accessed on: 01-07-2020].

[83] ns-3: Network Simulator 3. https://www.nsnam.org/. [Online; Last accessed on: 15-04-2023].

[84] ns-2: Network Simulator 2. http://www.isi.edu/nsnam/ns/. [Online; Last accessed on: 15-04-2023].

[85] George F Riley. The Georgia Tech Network Simulator. In *Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*, pages 5–12, 2003.

[86] OMNeT++ - Objective Modular Network Testbed in C++. https://omnetpp.org/. [Online; Last accessed on: 15-04-2023].

[87] MiXiM - Mixed Simulator. http://mixim.sourceforge.net/. [Online; Last accessed on: 15-04-2023].

[88] INET Framework. https://inet.omnetpp.org/. [Online; Last accessed on: 15-04-2023].

[89] JiST/SWANS - Java in Simulation Time / Scalable Wireless Ad hoc Network Simulator. http://jist.ece.cornell.edu/. [Online; Last accessed on: 15-04-2023].

[90] Antonio Virdis, Giovanni Stea, and Giovanni Nardini. SimuLTE-A modular system-level simulator for LTE/LTE-A networks based on OMNeT++. In *2014 4th International Conference On Simulation And Modeling Methodologies, Technologies And Applications (SIMULTECH)*, pages 59–70. IEEE, 2014.

[91] Francisco J et al. Ros. A survey on modeling and simulation of vehicular networks: Communications, mobility, and tools. *Computer Communications*, 43:1–15, 2014.

[92] Daimler Center for Automotive IT Innovations (DCAITI). Eclipse Mosaic - Smarter Mobility with Simulation. https://www.dcaiti.tu-berlin.de/research/simulation/mosaic/, . [Online; Last accessed on: 15-04-2023].

[93] Daimler Center for Automotive IT Innovations (DCAITI). MOSAIC as Co-Simulation Framework. https://www.eclipse.org/mosaic/about/, . [Online; Last accessed on: 15-04-2023].

[94] VEINS. VEINS - Vehicles in Network Simulation. veins.car2x.org, . [Online; Last accessed on: 15-04-2023].

[95] VEINS. VEINS Architecture. https://veins.car2x.org/documentation/, . [Online; Last accessed on: 15-04-2023].

[96] Raphael Riebl, Hendrik-Jörn Günther, Christian Facchi, and Lars Wolf. Artery: Extending Veins for VANET applications. In *2015 International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, pages 450–456. IEEE, 2015.

[97] Raphael Riebl, Christina Obermaier, and Hendrik-Jörn Günther. *Recent Advances in Network Simulation: The OMNeT++ Environment and its Ecosystem*, chapter Artery: Large Scale Simulation Environment for ITS Applications, pages 365–406. Springer International Publishing, 2019.

[98] Dirk Merkel. Docker: lightweight Linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[99] VcXsrv. VcXsrv Windows X Server. URL https://sourceforge.net/projects/vcxsrv/. [Online; Last accessed on: 25-09-2022].

[100] Pengfei Hu, Sahraoui Dhelim, Huansheng Ning, and Tie Qiu. Survey on fog computing: architecture, key technologies, applications and open issues. *Journal of network and computer applications*, 98: 27–42, 2017.

[101] Eliza Gomes, Felipe Costa, Carlos De Rolt, Patricia Plentz, and Mario Dantas. A survey from real-time to near real-time applications in fog computing environments. In *Telecom*, volume 2, pages 489–517. MDPI, 2021.

[102] Kai Liu, Xincao Xu, Mengliang Chen, Bingyi Liu, Libing Wu, and Victor C. S. Lee. A hierarchical architecture for the future internet of vehicles. *IEEE Communications Magazine*, 57(7):41–47, 2019. doi: 10.1109/MCOM.2019.1800772.

[103] Esubalew Alemneh, Sidi-Mohammed Senouci, and Philippe Brunet. PV-Alert: A fog-based architecture for safeguarding Vulnerable Road Users. In *2017 Global Information Infrastructure and Networking Symposium (GIIS)*, pages 9–15. IEEE, 2017.

[104] ETSI. ETSI TR 102 638 V1.1.1 Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Definitions, 2009.

[105] ETSI. Intelligent Transport Systems (ITS); V2X Applications; Part 2: Intersection Collision Risk Warning (ICRW) application requirements specification, 2018.

[106] ETSI. Intelligent Transport System (ITS); Vulnerable Road Users (VRU) awareness; Part 1: Use Cases definition; Release 2 , 2019.

[107] SUMO - Deliberately causing collisions. `https://sumo.dlr.de/docs/Simulation/Safety.html#deliberately_causing_collisions`, . [Online; Last accessed on: 23-04-2022].

[108] Bruno Ribeiro, Maria João Nicolau, and Alexandre Santos. V2X Datasets - Accidents between Passenger Vehicles and Motorcycles, November 2022. URL `https://doi.org/10.5281/zenodo.7376770`. [Online; Last accessed on: 30-11-2022].

[109] Spyder IDE. `https://www.spyder-ide.org/`. [Online; Last accessed on: 20-04-2023].

[110] Anaconda. `https://www.anaconda.com/`. [Online; Last accessed on: 20-04-2023].

[111] Scikit-learn - Machine Learning in Python. `https://scikit-learn.org/`. [Online; Last accessed on: 20-03-2022].

[112] Python - Programming Language. https://www.python.org/. [Online; Last accessed on: 20-03-2021].

[113] TensorFlow. https://www.tensorflow.org/. [Online; Last accessed on: 20-03-2023].

[114] pandas - Python Data Analysis Library. https://pandas.pydata.org/. [Online; Last accessed on: 28-09-2022].

[115] Peter T Yamak, Li Yujian, and Pius K Gadosey. A Comparison between ARIMA, LSTM, and GRU for Time Series Forecasting. In *Proceedings of the 2019 2nd international conference on algorithms, computing and artificial intelligence*, pages 49–55, 2019.

[116] José F Torres, Dalil Hadjout, Abderrazak Sebaa, Francisco Martínez-Álvarez, and Alicia Troncoso. Deep learning for time series forecasting: a survey. *Big Data*, 9(1):3–21, 2021.

[117] Pedro Lara-Benítez, Manuel Carranza-García, and José C Riquelme. An experimental review on deep learning architectures for time series forecasting. *International journal of neural systems*, 31 (03):2130001, 2021.

[118] Christopher Olah. Understanding LSTM Networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/, August 2015. [Online; Last accessed on: 30-04-2021].

[119] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-term Memory. *Neural computation*, 9(8): 1735–1780, 1997.

[120] Marius-Constantin Popescu, Valentina E Balas, Liliana Perescu-Popescu, and Nikos Mastorakis. Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, 8(7): 579–588, 2009.

[121] Jason Brownlee. 4 Strategies for Multi-Step Time Series Forecasting. https://machinelearningmastery.com/multi-step-time-series-forecasting/, March 2017. [Online; Last accessed on: 21-06-2022].

[122] Myeongkyu Lee, Sehan Kim, Daehyun Jung, Hyojun Lee, Jihun Choi, Hyunseo Han, and Ji Hyun Yang. Simulator-based study of the response time and defensive behavior of drivers in unexpected dangers at an intersection. In *Adjunct Proceedings of the 14th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, pages 10–14, 2022.

[123] Xiaomeng Li, Andry Rakotonirainy, and Xuedong Yan. How do drivers avoid collisions? a driving simulator-based study. *Journal of safety research*, 70:89–96, 2019.

[124] Hyunseo Han, Songhui Kim, Jihun Choi, Hasun Park, Ji Hyun Yang, and Jonghyuk Kim. Driver's avoidance characteristics to hazardous situations: A driving simulator study. *Transportation research part F: traffic psychology and behaviour*, 81:522–539, 2021.

[125] Remo MA van der Heiden, Christian P Janssen, Stella F Donker, and Chantal L Merkx. Visual in-car warnings: How fast do drivers respond? *Transportation research part F: traffic psychology and behaviour*, 65:748–759, 2019.