UNIVERSIDAD DE INVESTIGACIÓN DE TECNOLOGÍA EXPERIMENTAL YACHAY

Escuela de Ciencias Matemáticas y Computacionales

TÍTULO: Calibration of mobility and traffic simulation models through machine learning.

Trabajo de integración curricular presentado como requisito para la obtención del título de Ingeniero en tecnologías de la información

Autor:

Franklin Steven De la Cruz Paucar

Tutor:

Tito Rolando Armas Andrade, PhD.

Co-tutor:

Manuel Eugenio Morocho Cayamcela, PhD.

Urcuquí, noviembre 2023

Autoría

Yo, **FRANKLIN STEVEN DE LA CRUZ PAUCAR**, con cédula de identidad 1727256057, declaro que las ideas, juicios, valoraciones, interpretaciones, consultas bibliográficas, definiciones y conceptualizaciones expuestas en el presente trabajo; así cómo, los procedimientos y herramientas utilizadas en la investigación, son de absoluta responsabilidad de el/la autor/a del trabajo de integración curricular. Así mismo, me acojo a los reglamentos internos de la Universidad de Investigación de Tecnología Experimental Yachay.

Urcuquí, noviembre 2023.

Franklin Steven De la Cruz Paucar
CI: 1727256057

Autorización de publicación

Yo, **FRANKLIN STEVEN DE LA CRUZ PAUCAR**, con cédula de identidad 1727256057, cedo a la Universidad de Investigación de Tecnología Experimental Yachay, los derechos de publicación de la presente obra, sin que deba haber un reconocimiento económico por este concepto. Declaro además que el texto del presente trabajo de titulación no podrá ser cedido a ninguna empresa editorial para su publicación u otros fines, sin contar previamente con la autorización escrita de la Universidad.

Asimismo, autorizo a la Universidad que realice la digitalización y publicación de este trabajo de integración curricular en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación

Urcuquí, noviembre 2023.

Franklin Steven De la Cruz Paucar
CI: 1727256057

Dedication

I dedicate this work to the scientific community, whose unwavering dedication to the pursuit of knowledge is a boundless source of inspiration. I hope that my efforts contribute in some way to the progress of information technology.

To my beloved family, thank you for your unwavering support throughout this journey. Without your love, encouragement, and sacrifice, I would not have achieved this success.

To Yachay Tech, my beloved university: Thank you for the education and opportunities that have enabled my growth and learning. You are a beacon of knowledge and academic excellence.

This work is the result of the dedication and effort of many people, and I dedicate it with gratitude and appreciation to all those who have been part of my academic and personal career.

Franklin Steven De la Cruz Paucar

Acknowledgment

I want to express my sincere gratitude to those who played a fundamental role in the success of my thesis. Their support, guidance, and encouragement were invaluable throughout the process.

I would like to thank my exceptional thesis mentor, PhD. Rolando Armas, for your professional assistance and unwavering devotion. His dedication and knowledge were essential at every level of this thesis' development.

I also like to thank my thesis co-tutor, PhD. Eugenio Morocho, whose thoughts and experience served as a source of inspiration and collaboration even when the outcomes did not match expectations. His insight and recommendations were essential.

I offer a special appreciation to my mother, Hirma Paucar, for her moral support and unconditional affection. Her presence and support made the whole thing pleasant.

I want to thank my entire family for always believing in me and supporting me.

I am thankful to my buddy Saul Figueroa, who has been a constant companion throughout my career, for his collaboration and unwavering support.

And to all my friends and my girlfriend, I thank you for your words of encouragement, the time you spent listening to my development, and the helpful advice you gave me.

Franklin Steven De la Cruz Paucar

Resumen

La movilidad urbana es uno de los principales elementos del transporte inteligente, debido a su importancia y avance tecnológico, los ingenieros de tráfico aprovechan los modelos computacionales de movilidad y simulación de tráfico. Sin embargo, para obtener un escenario similar a la vida real es necesario cambiar parámetros en el simulador. Este proceso es iterativo y requiere mucho tiempo. Tradicionalmente se realiza de forma manual, es decir, el ingeniero de tráfico va cambiando los parámetros del simulador hasta obtener un escenario similar al observado. La importancia de las simulaciones radica en que ayudan a mejorar el flujo de tráfico y predecir la congestión si algo cambia en la red.

Simulation of Urban Mobility (SUMO) es una herramienta muy popular en el mundo de la simulación de tráfico. Este software es un paquete de tráfico microscópico y de código abierto que simula el comportamiento de la red urbana. Sin embargo, estas simulaciones son computacionalmente costosas debido a problemas en el tamaño del escenario y la cantidad de vehículos. La cantidad de automóviles en un sistema puede llevar mucho tiempo de procesamiento y calibración, por lo tanto esta investigación propone una metodología para calibrar automáticamente simulaciones de tráfico mediante el conteo de automóviles en tiempo real para obtener datos precisos de entrada del simulador. Primero, se realizan muchas simulaciones para crear un extenso conjunto de datos de ejemplos utilizando diferentes volúmenes de vehículos en los carriles de entrada y probabilidades en los carriles de intersección. Luego, los datos se intercambian de entrada/salida a salida/entrada para entrenar los modelos. Se aplican diferentes técnicas de aprendizaje automático, como Redes Neuronales Artificiales, Bosque Aleatorio (RF) y k-Vecino más cercano (kNN) que son capaces de estimar resultados de los parámetros de entrada para la simulación. Se presenta otra opción de calibración que combina modelos de aprendizaje automático y un algoritmo genético si el método propuesto no funciona bien. Se seleccionó la ciudad de Ibarra como principal área para la calibración y dos escenarios alternativos con alta prevalencia en áreas urbanas así como el hecho de que sus estructuras de red difieren entre sí.

Los resultados han demostrado que las redes neuronales tienen un mejor rendimiento en el primer escenario para predecir valores de entrada al simulador. En el segundo escenario, las redes neuronales también tuvieron un mejor rendimiento, sin embargo, los resultados no fueron tan precisos. Es por ello que se realizó la alternativa que combina modelos de aprendizaje automático con un algoritmo genético. kNN logró un mejor rendimiento al predecir las salidas del simulador sin su ejecución. Una vez desarrollado un modelo con alta precisión, se implementó un algoritmo genético para obtener los valores de entrada de la simulación teniendo el conteo de automóviles en las intersecciones.

Palabras Clave:

Redes Neuronales Artificiales, Bosque aleatorio, k- Vecino más Cercano, Aprendizaje Automático, Algoritmo Genético

Abstract

Urban mobility is one of the main elements of intelligent transportation. In this context, computational mobility and traffic simulation models are harnessed by traffic engineers. Nevertheless, to obtain a scenario similar to real life needs to change parameters in the simulator, this process is iterative and time-consuming. It is traditionally done manually. These simulations help to enhance the traffic flow and predict congestion if something in the network changes.

Simulation of Urban Mobility (SUMO) is very popular in the world of traffic simulation. This software is a package of microscopic traffic and open source that simulates urban network behavior. However, these simulations are computationally expensive because of problems in the size of the scenario and the number of vehicles. The number of cars in a system can take a long processing time. This research proposes a methodology to automatically calibrate traffic simulations by counting cars in real time to obtain precise data of input of the simulator. First, many simulations are done to create an extensive dataset of examples by using different volumes of vehicles in entry lanes and probabilities in intersection lanes. Then data is interchanged from input/output to output/input to train the models. It is applied different machine learning techniques, such as Artificial Neural Networks, Random Forest (RF), and k-Nearest Neighbors (kNN) that are capable of estimating simulation results. It is presented with another option for calibration that combines machine learning models and a genetic algorithm if the proposed method does not work well. Ibarra city was selected as the main for calibration and two alternative scenarios with high prevalence in urban areas as well as the fact that their network structures differ from one another.

Results have shown Neural Networks have better performance in the first scenario to predict input values to the simulator. In the second scenario, Neural Networks also had better performance, however, the results were not so accurate. That is why the alternative that combines machine learning models with a genetic algorithm was performed. kNN achieved better performance in predicting the outputs from the simulator without its execution. Once a model with high precision was developed, a genetic algorithm was implemented to obtain the input values of the simulation having the counting of cars in intersections.

Keywords:
Artificial Neural Networks, k-Nearest Neighbors, Random Forest, Machine learning, Genetic algorithm

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Nowadays, traffic congestion has become a recurrent challenge in cities, which affects in a negative way the mobility. To tackle this problem some techniques and tools have been developed, one of them is the simulation of vehicular traffic in urban environments [7]. As a consequence, these tools can be used to simulate future conditions in existing transportation systems. The simulation offers an effective platform to analyze and understand traffic behavior in different conditions. It can be used to evaluate the effectiveness of different transportation plans and policies, allowing traffic engineers to make decisions about traffic management, such as the addition of new roads and intersections or the implementation of new rules and regulations related to driver behavior [7]. As a result, traffic simulation has the potential to save governments time and money by simulating different scenarios and evaluating the impact of different strategies and conditions, cities can reduce the cost of implementing new transportation projects and make more effective use of limited resources [8].

Traffic simulation has been used for many years to analyze the impact of changes in transportation infrastructure. For example, it can provide the consequences of the effects of changes to the design of streets and highways and the routing of traffic through the system [8]. Different types of traffic simulations have been created to accomplish this task. In this project, a microscopic traffic simulator called SUMO [9] allows modeling traffic flow in a network, having in count several factors, such as volume of vehicles, traffic density, and different network topologies. This software was designed in 2001 by the German Center for Aerospace (DLR, a German abbreviation). Sumo is an open-source traffic simulator that includes a number of apps for planning and creating various simulations in realistic situations. [10]. This tool works with networks in a very flexible way. They can be made from scratch or modified from an existing map using Netedit. For its working a route file needs to be generated, which contains some characteristics such as type of vehicle and routes for all vehicles in the simulation. To obtain accurate results is essential to have trusty and updated data that could describe the conditions in the real world. To obtain optimal conditions it is necessary to modify the parameters in the simulator in an iterative way until some of them match with the behavior observed, this is known as calibration[11].

Calibration is one important area in computer science that helps to ensure the accuracy and reliability of calculations. To understand this concept, it is necessary to know about the process of calibration and the different types of calibration that exist. It is also important to understand the role that calibration plays in everyday computing and how it can be applied in other fields of study. The process of calibration is about taking the measurements that are essential to a task and then adjusting the computer or device in a way such that those measurements are accurate and reliable [12]. There are different methods that can be used to achieve the proper level of calibration. In mobility and traffic simulation models is an important step in ensuring the accuracy and reliability of these models[13].

In this work, we present a machine-learning approach to calibrate mobility and traffic simulation models.. We demonstrate that our approach, which uses datasets of simulated traffic and mobility information, can improve the simulation models' accuracy and predictive power.

1.2 Problem statement

In modern cities, traffic congestion is a critical problem that affects urban mobility, quality of air, and efficiency of transport systems. Successfully planning strategies to manage and optimize vehicle flow is essential to reduce these problems. In this context, traffic vehicle simulation has emerged as a valuable tool to understand and predict traffic behavior in different scenarios [14].

Nevertheless, the precision of simulations in some models, such as SUMO depends on the quality of used data as input. The results of simulations can not be enough close to reality if the estimation of the volume of vehicles in entry lanes and exit route probabilities are not well defined. One approach to this problem is to use vehicle count in the real world as a base-calibrated scenario for the simulator. The idea is to take the simulation performed by the counting data as the initial reference and adjust gradually the simulator parameters of the simulator to have a scenario very close to the real world. This approach not only can enhance the precision of simulations but also, can deliver a solid base to make decisions related to traffic management.

1.3 Objectives

1.3.1 General Objective

This research proposes a methodology to train machine learning models to automatically calibrate mobility and traffic simulation models.

1.3.2 Specific Objectives

- Automate the process for the creation of a dataset and execute a large number of simulations based on the volume of vehicles and route probabilities.

- Identify the most common methods used to calibrate traffic simulators and the research gap.
- Develop Artificial Neural Networks, Random Forest, and k-Nearest Neighbors that reuse the information from the history of simulations to predict inputs (vehicle volume and route decision) for the calibration in the simulator and validate the proposed methodology.
- Develop a genetic algorithm that, in union with machine learning models, could predict the volume of vehicles and route probabilities for different complex scenarios.

Chapter 2

Theoretical Framework

The principles required to comprehend this project are covered in this chapter, along with an introduction, a thorough description of the fundamentals of traffic simulation, the SUMO simulator interface, and components that generate scenarios for traffic flow simulation with user-defined variables. It also explains the machine-learning models, Artificial Neural Networks, Random Forest regression, k-Nearest Neighbors (kNN) regression, their structure, and how they work. Finally, it is explained what a genetic algorithm is and how it and machine-learning models worked together to develop this work.

2.1 Traffic Simulation

Traffic simulations make easier the evaluation of vial structure changes, the same as the impact of new politics before being implemented in daily life [15]. Nowadays, mobility is required for all human activities, whether they be social or commercial. The distance and amount of time needed for travel between two locations have increased along with the expansion of urban and rural areas. This time extension is caused by an increase in the number of vehicles on the road worldwide each year [16].

All calculated parameters for all vehicles in the simulation, including locations, speeds, and accelerations, are delivered as outputs by the simulator and are all recognizable and traceable. But it is deemed unreasonable for the simulation to be entirely deterministic [17], and as a result, the simulation injects some unpredictability into the computations. The simulator enables the development of measurement tools for trip times, automobile queue counts, and vehicle counts, among other outcomes. Practically speaking, these aggregate data are regarded as the outputs of the simulator and are technically processed as statistics.

One possible interpretation of the traffic simulator is that of a mathematical function that takes all settings and parameter inputs, including the length of the simulation and outputs the states of the road network at each simulation step, along with the output metrics that were set by the user and refer to the entirety of the simulation, or to intervals within the duration of the simulation (for example, measure vehicle counts on a road only within a specific time interval).

2.2 SUMO

Simulation of Urban Mobility (SUMO) is a very powerful vehicle simulation package that over the years has been adding a wide variety of tools that allow for increasingly realistic simulation scenarios and conditions [18] [19] [20]. Starting in 2001, it was created by staff members at the German Aerospace Center. The Graphic User Interface (GUI) of SUMO is shown in Figure 2.1.

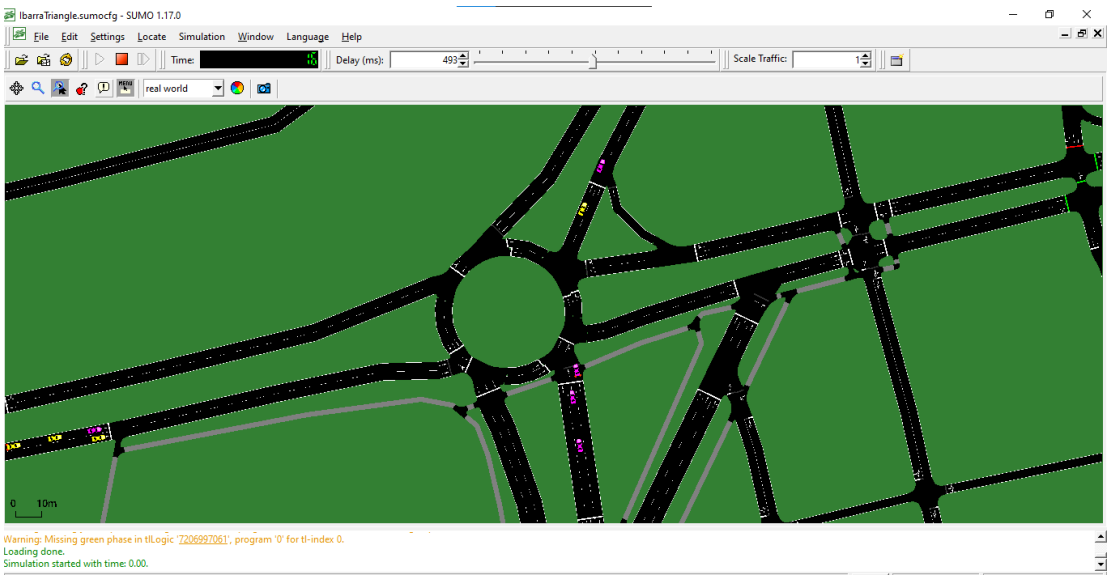


Figure 2.1: SUMO Graphic User Interface. Retrieved from [3]

Depending on the level of detail of the simulators, four traffic flow models are distinguished: (a) Macroscopic: the traffic flow is the basic entity. (b) Microscopic: the movement of each vehicle is simulated. (c) Submicroscopic: Vehicles are considered microscopic and divided into substructures. (d) Mesoscopic: the boundary between the microscopic and macroscopic models. The models with a higher level of detail require longer calculation times, which restricts the size of the network to be simulated [4].

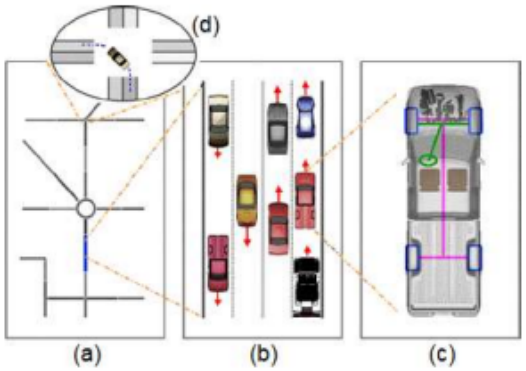


Figure 2.2: Traffic flow models: (a) Macroscopic,(b) Microscopic, (c) Submicroscopic, (d) Mesoscopic. Retrieved from [4]

2.2.1 Network Building

A directed graph is used in SUMO's network representation. The nodes use a coordinate from the map to represent a point in the network. In the map, a unidirectional street is represented by the union of two nodes, which is described by the edges. For vehicles to understand the direction of the roadway, edges must have a starting and ending node. Each edge may have several parallel lanes. Each lane has fixed settings for its width, speed, and other characteristics that control how a moving vehicle behaves in that lane. Figure 2.3 shows a network representation in SUMO.

Each network representation is contained in a file (*.net.xml). This file includes all the previously described components as well as additional ones that permit accurate network interpretation. These files can be created by users using programs like NETEDIT and NETCONVERT. A command-line utility called NETCONVERT enables the importation of networks from several sources, including OpenStreetMap (OSM) and other simulators, including MATSim [21]. An editor called NETEDIT is used to create, review, and update network files as well as to manually manipulate converted networks [22].

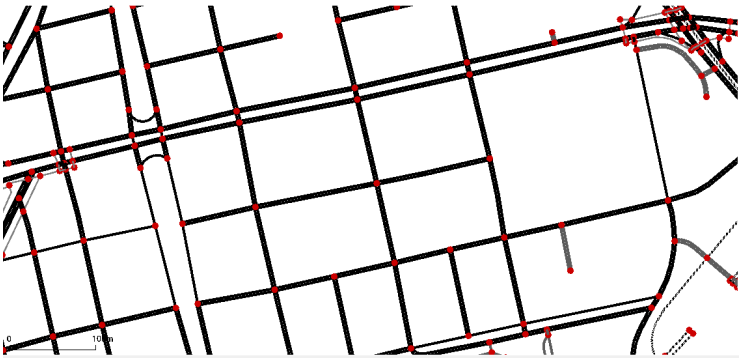


Figure 2.3: Network representation in SUMO

The key components of network files, which are encoded as XML files and contain precise information about how a graph is represented, are the cartographic projection, edges, junctions, and connections.

Coordinates

Cartesian projection is used in network files to represent the spatial distribution. The bottom node is at y=0 while the leftmost node is at x=0. Such projections are represented by the label location, and Table 2.1 lists their properties.

Table 2.1: Parameters in a network file to be coordinated

Attribute	**Description**
netOffsset	The distance needed to move the network to (0, 0).
convBoundary	The present network's perimeter.
origBoundary	The original network's original boundaries before projection.
projParameter	Information about the projection of the network.

Edges and Lanes

An edge, as previously mentioned, is a union of two nodes (junctions), and each edge may have a number of lanes, which are parallel, unidirectional representations of streets. Table 2.2 displays the properties of an edge, while Table 2.3 depicts a lane.

Table 2.2: Parameters for edges in network file

Attribute	Description
id	The edge's ID.
from	Node ID where it starts.
to	Node ID where it finishes.
priority	The significance of the edge.
function	Edge purpose.

Table 2.3: Parameters for lanes in a network file

Attribute	Description
id	The lane's ID.
index	A running total that begins at 0.
speed	The lane's maximum speed is (m/s).
length	The lane's length is in meters.
shape	Depiction of contours.

Junctions

The intersections between lanes are represented by the junctions, which can be thought of as nodes in the directed graph. The traits of them are shown in Table 2.4. The junctions may have a variety of demands. The requests list the streams that are more important for each link's streams. The requests' characteristics are shown in Table 2.5.

Table 2.4: Settings for junctions and intersections in a network file

Name	Description
id	The intersection's ID.
x	The location of the x-axis intersection.
y	The location of the y-axis intersection.
incLanes	The group of lanes that come to an intersection's end.
intLanes	An inventory of the junction's lanes.
shape	Describes the junction's boundaries.

Table 2.5: Parameters for requests in network files

Attribute	Description
index	The connection's index.
response	Bitstring indicating if it forbids passing through the crossing without slowing down.
foes	Bitstring describing conflicts between alternative connections and the real connection.
cont	Whether a car can proceed past the first stop line and wait until no other cars have a higher priority.

Connections

The connections, or links, show the first route to take after passing the intersection and specify which lane the vehicle can use from an oncoming lane. The properties of connections are provided in Table 2.6.

Table 2.6: Parameters for connections in a network file

Attribute	Description
from	The incoming edge's ID.
to	The outgoing edge's id.
fromLane	The entering lane's ID.
toLane	The outgoing lane's ID.
via	The lane that is used first following the connection.
dir	Connection direction.
state	The connection's state.

2.2.2 Demand Modelling

After network generation, traffic demand must be put in place. It has access to vehicles and the details of each travel. There are various ways to represent traffic demand in SUMO, including trips, flows, or routes [18]. The representation in routes is revealed in this study. We may categorize several sorts of cars according to their various attributes, such as size, behavior, and others. The creation of each vehicle that will feature in the simulation is a crucial component of this file. Each of them will be represented as a set of edges that move around the simulation and have a kind, a departure time, and a predetermined route. Route files (*.rou.xml), which represent this traffic requirement.

We may make use of SUMO's randomTrips tool to create a route file. It is a Python script that makes use of a network file and creates a number of routes in a file based on input from the user. These routes frequently lack balance, so the characteristics of the cars will be random. Utilizing Origin and Destination (O-D) matrices is a distinct strategy [21].

The tools used by SUMO to create routes are as follows: (1) DUAROUTER, which lets users import demand definitions and create routes based on the shortest path algorithm. (2) JTRROUTER is used to statistically predict traffic utilizing turn rates at street intersections and flow specifications. (3) OD2TRIPS allows OD matrices (origin/destination matrix) to be transformed into driving trips. (4) DFROUTER can determine routes using data collected from detectors placed along the map's roads. (5) MAROUTER requires less computer resources because it constructs the routes as vehicular flows and computes the trips using macroscopic assignment.[18].

Typically, a route file includes comprehensive information about the traffic demand. The representation of a route file is shown in Figure 2.4.

```
<routes>
    <vType id="type1" accel="5.0" decel="7.5" sigma="0.5" length="5" maxSpeed="80"/>
    <route id="route1" color="1,1,0" edges="d1 d2 d3 d4"/>
    <vehicle id="0" type="type1" route="route1" depart="100" color="1,0,0"/>
</routes>
```

Figure 2.4: Route file representation

Vehicle type

The simulation's first step defines one or more vehicle kinds (vType). Each type specifies a number of variables that are used to define the shape and behavior of the vehicle during the simulation. In Table 2.7, some of these traits are displayed.

Table 2.7: Attributes behavior of vehicles in a route file

Attribute	**Description**
id	The type of vehicle's name.
accel	This car's ability to accelerate (m/s^2).
decel	The vehicle's ability to decelerate (m/s^2).
sigma	The flaw in the car-flowing model's driver.
length	Vehicle length in meters.
maxSpeed	The car's top permitted speed.

Route

The route specifies the boundaries and other features that will make up a vehicle's journey within the simulation. To prevent simulation mistakes, the set of straight edges must be connected. The features of the routes are shown in Table 2.8.

Table 2.8: A route file's depiction of a route

Attribute	Description
id	The route's name.
edges	List of the edge identifiers that the car will go along.
color	Route color

Vehicle

The simulated object that will go across the full simulation in the chosen network is defined by the vehicle. The key traits of the vehicles are displayed in Table 2.9.

Table 2.9: Depiction of a vehicle in a route file

Attribute	Description
id	The vehicle's name.
type	User-specific type id for this vehicle.
route	Id of the path that the vehicle will take, with explicitly stated edges.
color	The vehicle's color
depart	The moment when the car first joins the network.

JTRROUTER

This tool generates vehicle routes based on flow demand and turn percentages at street intersections [23]. Figure 2.5 shows the flowchart of this tool, where in addition to the road network and traffic demand as vehicle flows, there is a description of the turning ratios at each intersection.

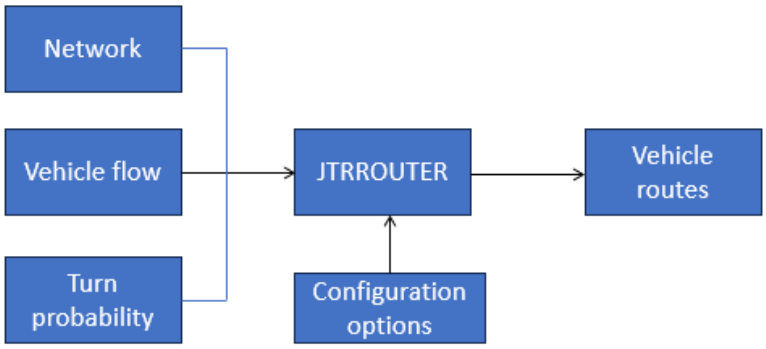


Figure 2.5: JTRROUTER flowchart

2.2.3 TraCI

The Traffic Control Interface (TraCI) [5] gives users access to a running SUMO traffic flow simulation, allowing them to retrieve simulation agent variables and control their behavior. TraCI uses a TCP/IP-based framework to provide users control over SUMO. A client/server topology is supported by the design. As a result, SUMO functions as a server that initiates command-line actions using a TraCI script (often written in Python or C++) and manipulates simulation objects in real time. Up until it runs the simulation step function, TraCI supports many clients and executes the desired activities in the clients. The TraCI architecture is shown in Figure 2.6.

The client application delivers commands to SUMO using the protocol depicted in Figure 2.6 to control the simulation execution, a specific vehicle behavior, or to extract values from simulation objects. SUMO responds to each command with a status response and, depending on the request, with one or more additional values.

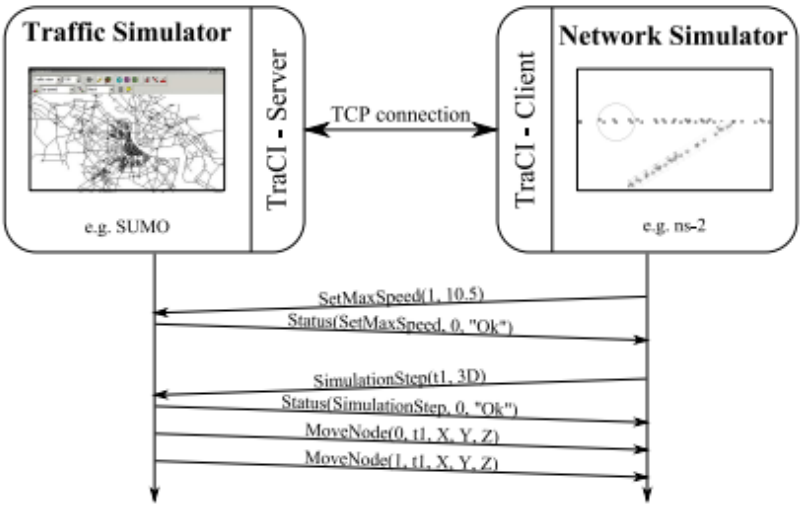


Figure 2.6: TraCI architecture. Retrieved from [5]

Python is the preferred language for creating scripts that use TraCI since it has thorough documentation, supports all TraCI commands, and the community regularly tests this library. TraCI libraries are supported by other languages including C++ and Matlab, but their support isn't entirely comprehensive. We need to import TraCI into the script in order to connect SUMO and TraCI using Python. The Python load path must contain the SUMO HOME/tools directory. Figure 2.7 illustrates how to do this.

```
if 'SUMO_HOME' in os.environ:
    tools= os.path.join(os.environ['SUMO_HOME'],'tools')
    sys.path.append(tools)
else:
    sys.exit("please declare environment variable 'SUMO_HOME'")
```

Figure 2.7: Import TraCI in a Python script.

We begin our simulation and establish a connection to it using the constructed script after loading TraCI into the Python load path. Figure 2.8 shows a simple Python script that runs 1000 simulation steps using TraCI.

```
import traci

sumoBinary= "/path/to/sumo"
sumoCmd= [sumoBinary, "-c", "yourConfiguration.sumocfg"]

import traci
traci.star(sumoCmd)
step=0
while step<1000:
    traci.simulationStep()

    step+=1
traci.close()
```

Figure 2.8: Basic TraCI script.

Several commands can be issued after connecting to the simulation, which will then carry out a simulation step till needed. The simulation must be finished by closing it.

2.3 Machine learning

The field of artificial intelligence known as Machine Learning (ML) focuses on creating algorithms and models that can recognize patterns and relationships in data [24]. Its use in the calibration of simulators, like SUMO, has emerged as a viable strategy to raise the accuracy of simulations of vehicle traffic [25]. In this context, models may understand underlying patterns and correlations in data to generate predictions and judgments, which is the foundation of ML. In order to learn from examples and generalize from them, machine learning algorithms are trained using a set of input data and matching outputs. The calibration of the simulator can make use of several ML algorithms. Relationships between simulator settings and count data can be found using regression models, such as polynomial and linear regression.

2.4 Neural Networks

Famous machine learning techniques that mimic the biological organisms' learning mechanisms include artificial neural networks [26]. Cells that function as neurons are incorporated into the human nervous system. Dendrites and axons connect the neurons, and the areas where dendrites and axons meet are known as synapses [27]. In this study, the term "neural networks" will be used to describe artificial neural networks rather than biological ones. Weights, which have the same purpose as the potency of synaptic connections in biological organisms, are used to connect computing units. Each bit of data that a neuron receives is scaled with a weight, which has an impact on the function determined at that unit [28].

2.4.1 Neural networks architecture

Single-layer and multi-layer neural networks will be introduced. A set of inputs is quickly mapped to an output in the single-layer network using a generalized linear function variation. The perceptron is another name for this basic neural network implementation [29]. In multi-layer neural networks, the neurons are organized in layers with the input and output layers being divided by a set of hidden layers. A feed-forward network is another name for the neural network's layer-wise architecture [30].

2.4.2 Single-layer neural networks

The simplest kind of neural network is a single layer, sometimes referred to as a perceptron. They don't have any hidden layers in between; they only have an input layer and an output layer. All of the input layer's neurons are coupled to every neuron in the output layer. These linear models are typically applied to binary classification issues [29]. Perceptrons were the starting point for the creation of deeper and more complicated neural networks, despite their limited capacity to capture complex interactions. A layer's perceptron can be thought of as a linear function that receives inputs, combines them with weights and biases, and outputs the result. Figure 2.9 shows the perceptron model.

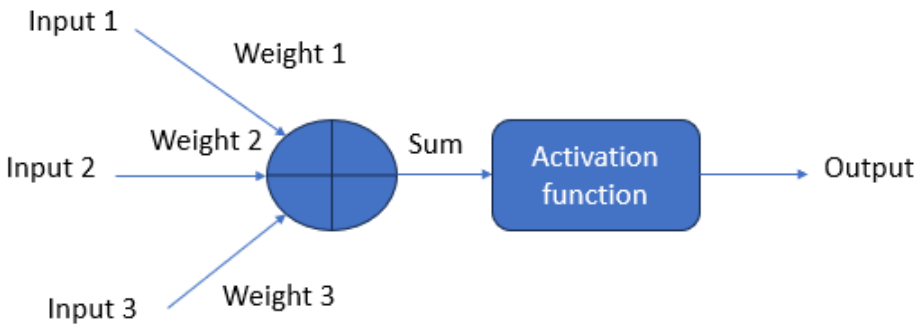


Figure 2.9: The perceptron model.

2.4.3 Multi-layer neural networks

Multi-layer neural networks are more advanced and flexible models that include multiple layers, including an input layer, one or more hidden layers, and an output layer. Hidden layers allow the network to learn more complex, non-linear representations of the data. Each neuron in one layer is connected to all the neurons in the layer before and after it, forming a pattern of interconnected connections.[30]. It is illustrated in Figure 2.10. A detailed explanation of its structure and operation follows:

Input layer

The input characteristics of the problem are sent to this layer. In this layer, each neuron stands for a distinct trait. Neurons in the hidden layers receive and analyze the inputs

before being sent [31].

Hidden layers

These intermediary layers are in charge of teaching the data more complicated and abstract representations. Every neuron in a hidden layer takes input from neurons in the layer above, processes that input using weighted operations, and then passes the outcome through a nonlinear activation function. [31].

Output layer

The model's final outputs are produced by this layer. The hidden layer outputs are fed onto the neurons in this layer, which then use weighted operations to produce the final predictions or classifications [31].

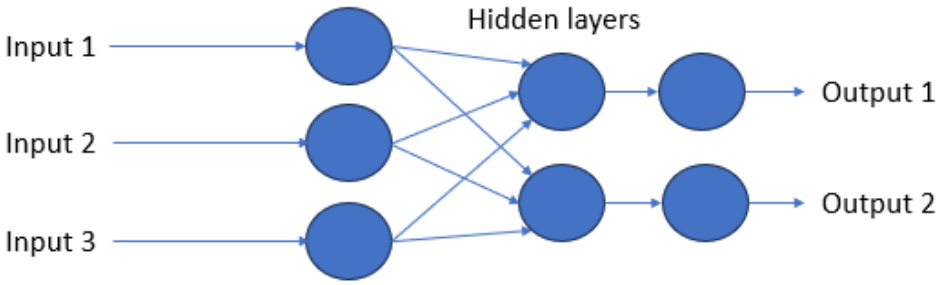


Figure 2.10: The multi-layer perceptron model.

2.4.4 Optimizers

The goal of optimization approaches is to avoid settling for local minimum solutions. Gradient descent (GD) optimization is a prominent approach in deep learning models. However, numerous GD approaches for deep networks are utilized to discover the optimal set of parameters that minimize the loss function [32]. The following are some of the most often-used optimizers:

Stochastic Gradient Descent (SGD)

SGD is an iterative optimization approach that tries to minimize a neural network's loss function (also known as the cost or objective function) during training by modifying the model's parameters (weights and biases). It is dubbed "stochastic" because it acts in each iteration on a randomly selected portion of the training data (a mini-batch) rather than the complete dataset [33].

Adaptive Gradient Descent (AdaGrad)

It is designed to enable quicker convergence by decreasing the learning rate from big gradient parameters and raising the learning rate from small gradient parameters. In other words, this approach determines the learning rates based on the gradient scenario [34].

Adaptive Delta (AdaDelta)

It, like AdaGrad, adjusts the learning rate for each parameter. The primary difference is that this approach is more stable since it employs the root mean square of the gradients. It minimizes memory needs by using multiple fixed-size windows rather than collecting the gradient [35].

Adaptive Moment Estimation (Adam)

It combines RMSProp and Momentum. The learning rates are determined by the first and second moments of the gradients. One of the primary benefits of this optimizer is that it decreases the computational cost and the execution memory [36].

Maximum Adaptive Moment Estimation (AdaMax)

In contrast to the Adam optimizer, AdaMax employs an infinity norm (L-infinity) rather than the L2 norm. Furthermore, this optimizer employs the entire value of the second momentum from the Adam approach, resulting in a more robust and stable solution [37].

Nesterov-accelerated Adaptive Moment Estimation (Nadam)

Nadam combines the benefits of both Nesterov Accelerated Gradient (rapid convergence) and Adaptive Moment Estimation (adaptivity to varied learning rates), making it an effective optimizer for deep neural network training. In many cases, it converges quicker and more consistently than classic optimizers like SGD and even Adam [38].

2.5 Random Forest

One of the most common machine learning regression models is the random forest regression. It is a tree-based ensemble learning algorithm that predicts numeric values [39]. Certainly, a random forest algorithm can be used for classification and regression. Classification problems are related to discrete labels or specific categories, while regression predicts continuous values [40]. Then, random forest regressions can be considered as an extension of the main random forest algorithm. To understand all the mechanisms involved in random forest procedure, essential concepts, and sub-processes need to be known.

Ensemble Learning

Ensemble learning is a technique that combines multiple machine learning algorithms to obtain a single prediction from many base model predictions. The idea of applying this

method is to avoid choosing a poor model for solving a particular problem [41]. Instead, by combining results, a more accurate solution can be obtained. when referring to random forest approach, parallel ensemble learning is needed.

Bootstrapping Bootstrapping is a probabilistic technique whose objective is to create different subsets by extracting random samples from an original dataset [42]. The key strategy of bootstrapping is that this extraction process is sequential. In other words, the first subset is obtained by randomly selecting N observations from the original dataset, then each observation is returned to the original dataset before extracting the next N sample for the second subset [43]. This process is repeated for all the subsets to be created. Therefore, one observation is not only contained in one subset, it has the chance to be in different subsets. This mechanism makes the bootstrapping technique a robust one since it does not depend on applying any distribution to the original data.

Bootstrap Aggregation Bootstrap Aggregation or Bagging is the basis of the random forest algorithm. It consists of creating a training dataset from the original dataset through the sampling with replacement technique (Bootstrapping) [44]. Indeed, bagging is used to generate multiple versions of a predictor. Here is where aggregation comes up to get this aggregated predictor based on those versions. In the case of regression problems, a numerical output is expected. Hence, this new predictor is calculated by computing the average among all the versions [45]. Figure 2.11 shows a representation of the bagging process.

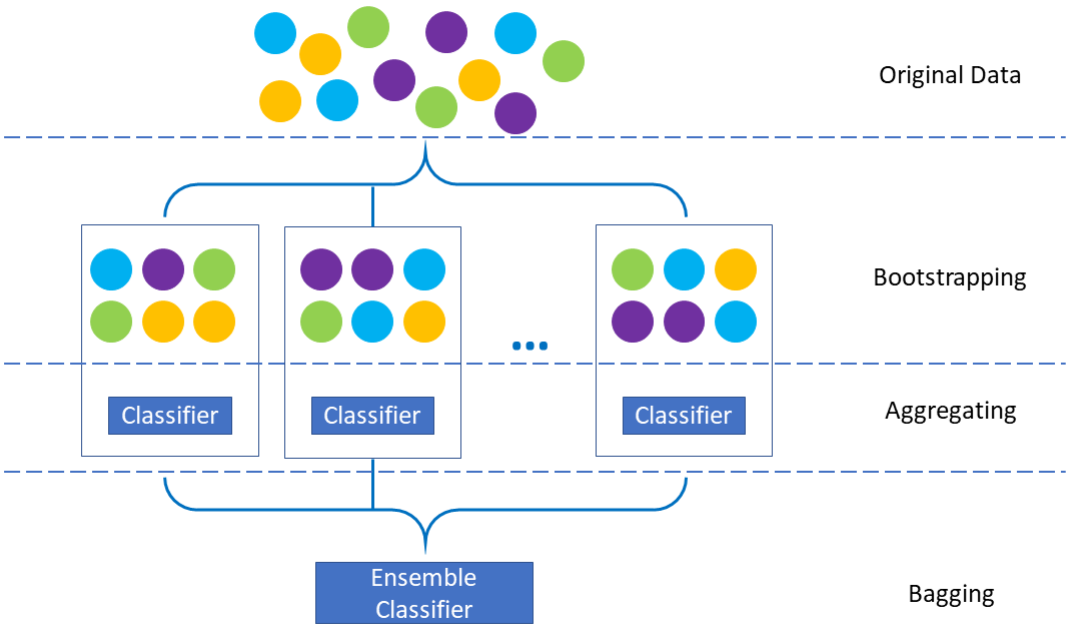


Figure 2.11: Bootstrap aggregation process represented as decision trees

Random Forest Algorithm for Regression

Due to the decision tree structures, and considering that random forest is based on a bagging process, it is suitable to perform parallel training such as required by a bootstrapping aggregation method. Then, it is possible to implement this technique by following the next procedure:

1. Set k, the number of decision trees to be used (bootstrap set), set N, the number of samples to randomly extract observations from the original dataset D, and set $F = \sqrt{Number\ of\ features\ from\ D}$.
2. Create k subsets from D.
3. Split each decision tree by considering the best feature F that minimizes the loss.
4. For regression, computes the means by taking each prediction resulting from each tree.
5. Evaluates the performance by using an appropriate error metric depending on the problem studied. One of the commonly used error metrics for regression is the squared error loss function $L(Y, f(X)) = (Y - f(X))^2$ [39].

2.6 *k*-Nearest Neighbors (*k*NN)

It is an algorithm that saves all past (available) examples and utilizes them to anticipate values based on a similarity measure. It predicts the values for test data/new data points using 'feature similarity'. The new point's value is determined by how closely it resembles other training data samples [46]. This explanation is better represented in Figure 2.12.

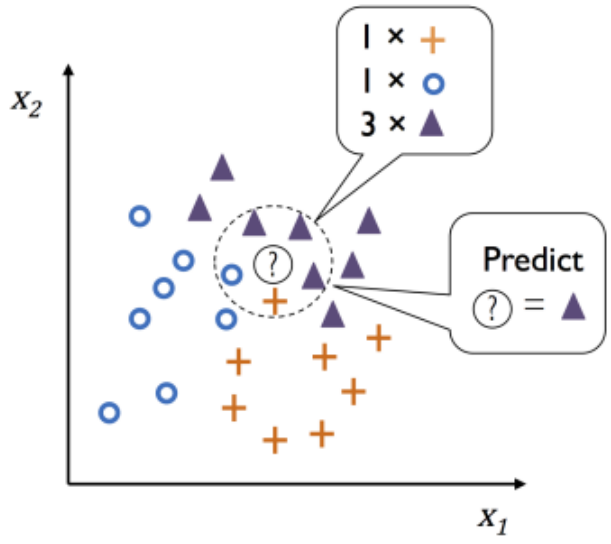


Figure 2.12: Example of kNN for a 3-class problem with k=5. Retrieved from [6]

There are two techniques for kNN regression. The first step is to compute the average of the target of the k-Nearest Neighbors. The second method involves calculating an inverse distance weighted average of the k-Nearest Neighbors. The same distance functions presented in Table 2.10 that are used in kNN classification are used in kNN regression: Euclidean, Manhattan, and Minkowski[47]. Where p and q: They are vectors in an n-dimensional space, r: It is a parameter that determines the distance between the vectors "p" and "q", n: It is the dimension of the space in which the vectors "p" and "q" are found.

Table 2.10: Distance Functions and Formulas. Retrieved from [1]

Name	**Formula**
Euclidean	$d_{\text{Euclidean}}(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$
Manhattan	$d_{\text{Manhattan}}(\mathbf{p}, \mathbf{q}) = \sum_{i=1}^{n} \lvert p_i - q_i \rvert$
Minkowski	$d_{\text{Minkowski}}(\mathbf{p}, \mathbf{q}, r) = \left(\sum_{i=1}^{n} \lvert p_i - q_i \rvert^r \right)^{\frac{1}{r}}$

Stepy-by-step

- **Definition of *k*:** Begin by choosing a value for K, which represents the number of nearest neighbors to consider when making a prediction. This K value is a hyperparameter that needs to be tuned based on the problem and the data.
- **Distance measurement:** To determine the K closest points to a query point (the point you want to predict), a distance metric, commonly Euclidean distance, is used. The square root of the sum of the squared differences between two locations in an n-dimensional space is used to determine the Euclidean distance between them.
- **Neighbor selection:** The distance metric is used to identify the k nearest points to the query point. "Nearest neighbors" refers to these points.
- ***k*NN Regression:** In the regression context, the prediction is made by taking a weighted average of the k nearest neighbors' target values (labels). The closer a neighbor is, the more influence it has on the prediction. Instead of class labels, the target values are the values you want to forecast [6].

2.7 Genetic Algorithm

Genetic Algorithms are a sort of optimization algorithm that is influenced by biological evolution and Darwin's theory of natural selection. They were created to handle search and optimization issues in a variety of applications. These algorithms use principles like population, selection, crossing (crossover), mutation, and adaptation to simulate the process of natural evolution [48].

2.7.1 Genetic Algorithm Development

The classical genetic algorithm is essentially based on starting with a collection of unoptimized random candidate solutions that reflect a solution to the optimization issue that we wish to solve. When these solutions are run through the evolutionary algorithm, potential solutions with improved features start to emerge [49]. When a solution is examined using the fitness function, it is considered a possible candidate. The answer or group of solutions might be maximal or minimal. The representation of the answers is critical since it influences the type of genetic operators utilized. In general, as defined by [50], the solutions' representation might be scalar or bit strings. The genotype or chromosome refers to the solution's coding, which is subject to evolution [49].

Some special features or characteristics endure throughout time under Natural Selection. Genetic algorithms use this characteristic by drawing parallels with natural evolution. GA is inspired by the natural selection mechanism, in which stronger individuals are more likely to win in a competitive context [51].

The procedure starts with a group of individuals known as a Population. Each individual is a solution to the situation at hand. An individual is defined by a set of factors (variables) called as Genes. A Chromosome (solution) is formed by stringing together genes. A genetic algorithm represents an individual's set of genes as a string in terms of an alphabet. Typically, binary values (strings of 1s and 0s) are utilized. We call this encoding the genes on a chromosome. It is better represented in Figure 2.13.

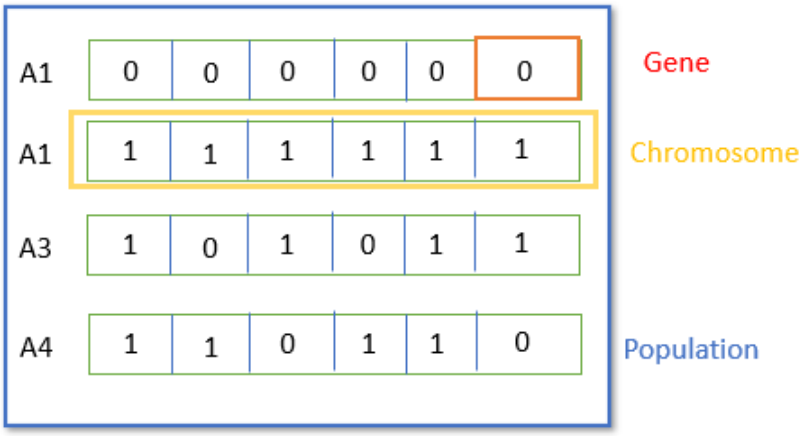


Figure 2.13: Popultion, Chromosomes, and Genes

Step-by-step

- **Initialization:** The first step is to create a group of possible solutions called an initial population. These solutions are encoded as chromosomes, which consist of genes [52].
- **Fitness evaluation:**When evaluating each solution in the population, its fitness is measured to determine its effectiveness in achieving the problem's goal. The level of fitness may vary depending on the problem, with some aiming to maximize it, while others aim to minimize it [52].
- **Selection:** Solutions in the population are selected for breeding based on their fitness. The fittest solutions have a higher probability of being selected, but some less fit solutions may also be selected to maintain genetic diversity [52].
- **Crossover:** The selected solutions are combined to create new solutions called descendants. This is done by exchanging parts of the chromosomes of the parent solutions. Crossing simulates genetic recombination and may lead to promising solutions [52].
- **Mutation:** Random modifications are sometimes made to descendant solutions to introduce variability and prevent the algorithm from becoming trapped in local optima [52].
- **Replacement:** Descendant solutions are incorporated into the population, often replacing less fit solutions. This ensures that the population remains constant in size [52].
- **Stopping Criterion:** The selection, crossing, and mutation process is repeated for a set number of generations or until a predefined stopping criterion is met, such as reaching a satisfactory solution [52].
- **Results:** Once the genetic algorithm has converged or has reached the stopping criterion, the best solution found is reported as the result of the algorithm [52].

2.8 Calibration

Calibration has been regarded as one of the most difficult phases in the construction of traffic simulation scenarios due to its repeated and time-consuming nature Chu et al. [53]. Furthermore, automation and the understanding of calibration as an optimization issue have resulted in favorable outcomes in terms of enhancing calibration task accuracy, dependability, and repeatability. While genetic algorithms have been identified as the most commonly used optimization methods for calibration, one drawback is that a large number of simulations are performed throughout the iterations for the heuristic to achieve a small set of highly specific solutions to the single scenario under calibration.

The goal of the research is to develop automatic calibration models that can be used for different simulations of the same map but under different traffic conditions (for example,

the same avenue or intersection but with varying traffic demands, vehicle compositions, and routing decisions).

We will use error functions to evaluate the calibration and convergence of machine learning models. Mean Squared Error (MSE) was selected to be used in the experiments, where the predictions of a model (represented by $\hat{y}_i$) are measured with the actual observed values (represented by y_i), n: It is the total number of observations.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.1}$$

The user can utilize the variables that were modeled with acceptable error while disregarding the machine learning models' suggestions for the variables that were modeled badly.

The testing dataset is used to assess performance for each variable. The machine learning models process the test inputs, and the outputs are compared to the intended outputs from the dataset. Furthermore, the Pearson correlation between the network's predictions and the intended test outputs was chosen as the metric to quantify performance in a normalized manner. The correlation serves as a score of the calibration's quality for each variable and is determined using Equation 2.2, where x and y are the expected and desired output values [54].

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{2.2}$$

Once machine learning models are trained and tested, the calibration will be measured with the results of Pearson correlation for each variable. It means that the manual calibration process done by a traffic engineer will be reduced for those variables with a correlation close to 1. This explanation is expanded in Chapter 4 referred to the Methodology.

Chapter 3

State of the Art

Microsimulation has developed as a critical tool in traffic planning and management, providing dynamic and detailed insights into traffic system behavior. The careful calibration of multiple factors is critical for the correct reproduction of real-world settings within microsimulators. The complexity of these simulation systems, combined with the enormous parameter space, has made optimal calibration difficult. The following review presents an in-depth examination of approaches and methodologies for identifying essential factors for microsimulator calibration, with the goal of improving the accuracy and efficiency of traffic modeling, and machine learning techniques for traffic problems.

3.1 Selection of parameters for calibration

The effective calibration of microsimulators is crucial for producing reliable simulation outcomes. Various parameters such as driver behavior, vehicle interactions, and traffic flow dynamics [55] are the more commonly used to achieve a good calibration. However, the sheer number of parameters complicates calibration, often requiring substantial time and computational resources. The identification of critical parameters, that have a significant impact on simulation results, is essential to streamline the calibration process and enhance the reliability of simulation outcomes. Documentation from traffic engineers about the process of traffic microsimulation calibration Llanque [56] and Miller [57] show that parameters, such as the simulation configuration and the infrastructure construction are considered constant or fixed. In addition, other authors make reference to the importance of driving behavior parameters for calibration [58][59], especially in smaller road networks.

On the other hand, Fellendorf and Peter Vortisch [60] introduce the importance of combining driving behavior, vehicle volume, and route decisions to have a good microscopic traffic representation. Chu et al. [53] added that calibration models based only on driving behavior do not work for larger or more complex networks, and to solve this use an Origin-Destination (O-D) matrix as a reference. Then, the importance of determining the traffic volume demand in networks is highlighted by Tettamanti et al [61] that uses a genetic algorithm to determine the traffic volume on a group of roundabouts.

Conducting sensitivity analysis is a typical method for discovering key parameters. This strategy comprises systematically changing individual parameters and observing the

changes in simulation outputs. Critical parameters are those that have a significant influence on model outputs and require careful calibration [62]. One-at-a-Time (OAT) and Global Sensitivity Analysis (GSA) methodologies provide insights into parameter interactions and their effects on simulation outcomes [63]. According to all authors, the literature research revealed that there is no one set of parameters to be calibrated. Mkadziel [64], Ciuffo, and Azevedo [65] explored parameter selection and the parametric sensitivity of the simulations, respectively. The conclusion is that the subset of parameters (as well as the other subset that can be left on the program defaults) varies by circumstance. In addition, Punzo et al. [66] support the notion that a subset of parameters may be sufficient to deem traffic microsimulation calibration satisfactory.

3.2 Concerning the calibration procedure

Microsimulator calibration can be thought of as an optimization problem. Because microsimulators have a plethora of characteristics that influence the behavior of simulated traffic, determining the optimal combination of these parameters is analogous to locating a sweet spot in a multidimensional space. The objective is to minimize the disparity between the simulated and observed or desired data [67].

This optimization procedure entails searching for a solution that meets particular performance requirements, such as average speed accuracy, traffic density, and journey times, among others . However, due to various problems, such as the intricacy of the interactions between the parameters and the results, the existence of multiple acceptable solutions, and the possibility that some parameters interact with each other in a non-linear manner, this search is not straightforward [68]. In practice, calibration as an optimization issue entails iteratively altering parameter values and evaluating how they affect simulation outcomes. This iteration process is repeated until a convergence is reached that yields simulated results that satisfactorily fit the observed data or the desired goals.

The assessment of the squared error between the simulation results and the observed data is a frequently used and theoretically validated technique for assessing the accuracy of the calibration [69]. The squared difference between the simulated and genuine values for several observations is measured by the mean square error (MSE). By minimizing the MSE, one aims to discover the ideal parameter values that generate the best overall agreement between the simulated and observed results [17].

Chu et al. [53], Balakrishna et al. [12] and, Jha et al. [70] are examples where the iterations of calibration were not automated and, where traffic specialists manually calibrate microsimulators by subjectively modifying the parameters based on their expertise and knowledge. Due to the complexity and multidimensionality of the parameter spaces, this can origin in unsatisfactory results and error proneness. This methodology is known as manual iteration based on preliminary results.

Some research used semi-automated heuristic methods, in which genetic algorithms or search methods were used to generate initial parameter combinations. For instance, Henclewood et al. [71] offer a calibration approach based on simulation automation. The procedure runs a large number of simulations with random parameters and calculates squared errors; 1000 simulations were run to extract 93 and 34 calibrated models for noon and evening periods in their testing location, respectively, demonstrating the calibration

differences between different time periods at the same location. In addition, Aghabayk et al. [72] and Bethonico [73] automated calibration by applying Genetic Algorithms to tackle an optimization problem. The Genetic Algorithm is a search heuristic that iterates through generations of calibration sets; inspired by evolutionary theory, the new generations are created from combinations of the most successful calibration sets from the previous generation, with a random element added [74]. Success is determined by maximizing a fitness function (or, alternately, minimizing a cost function), and the method produces successful calibration parameters after a huge arbitrary number of rounds. However, there is no mathematical certainty that the method will eventually produce the best calibration set.

Shafiei and Saberi [75] proposed a methodology for calibrating traffic flow fundamental diagrams in the dynamic traffic assignment (DTA) model that involves using a machine learning-based technique. This technique utilizes 1-year worth of traffic data from 239 loop detectors across the network to calibrate the traffic flow fundamental diagrams. The calibrated DTA model demonstrated reasonably high accuracy in simulating link volumes and path travel times, considering the size of the network and its congestion level.

3.3 On the use of machine learning to solve traffic problems

Machine learning is a rapidly expanding field that has the potential to transform the way we address traffic challenges. Machine learning algorithms can be used to forecast traffic patterns or optimize traffic signals. Both regression and classification difficulties have been identified in the traffic engineering literature, and machine learning has been offered as a solution [76]. The use of predictive analytics to forecast traffic patterns is one of the most promising applications of machine learning to traffic concerns. Predictive analytics forecasts future traffic conditions using historical data. This data can be used to optimize traffic signals, schedule road building projects, and provide drivers with real-time traffic updates [77].

The Artificial Neural Network (ANN), a family of computing models inspired by biological neurons and utilized for machine learning of regression and classification problems [26], is one machine learning technique of relevance in this research. In the car-following model of microsimulations, a recurrent ANN model was presented by Zhou and Li [78] to forecast traffic oscillations. The recurrent ANN is a sort of network that is trained from time-series sample data to predict future data. This model is used to replace car-following models, which are typically sets of equations used to calculate the interaction of cars in the same road lane. The recurrent ANN outperformed the classical models in forecasting the trajectory of subsequent vehicles; the better the recurrent ANN outperformed the classical models, the farther the succeeding vehicles were from the reference vehicle. Tang et al. [79] used a type of network called fuzzy ANN to estimate travel times in the real world from road loop detectors, thus the ANN implicitly computes a traffic simulation and delivers the travel time outputs, if we consider loop detection counts and speeds as the inputs of this hypothetical simulation. Chen et al. [80] employed a combination of Genetic Algorithms and ANNs to construct a system to forecast rear-end crashes as another example of ANN use outside of traffic calibration. Instead of training a single ANN to anticipate a collision

based on car-following patterns fed from the Vehicle-to-Infrastructure (V2I), Vehicle-to-Vehicle (V2V), and Global Positioning System (GPS) infrastructures; their approach is to train many ANNs and combine them with genetic algorithms.

On the other hand, Shi et al. [81] found to have better prediction accuracy compared to other data-driven methods, such as Artificial Neural Networks (ANN), while requiring lower computational resources. They used Random Forest to construct a car-following model based on high-precision, high-refresh-rate, and large-scale vehicle trajectory data. The RF model accurately described the car-following behavior by exploring the internal connections of the data. The RF method's random selection of samples and attributes ensured high prediction accuracy and strong generalization ability, while its parallel training capability improved efficiency. Also, Saadi et al. [82] provide a method for predicting origin-destination (O-D) matrices in transportation that are based on random forests (RF). To boost accuracy, the RF technique employs disaggregated travel diary data and location information as predictors. The RF approach mitigates mistakes and minimizes variation by utilizing bagging and random subspace principles. The preliminary results show that the RF technique provides intriguing O-D traffic flow estimates.

In addition to solving traffic problems, Katrakazas et al. [83] used the k-Nearest Neighbors (kNN) algorithm to identify conflict-prone traffic conditions. The kNN is a nonparametric approach that is widely applied in classification and regression tasks provides case-based explanations for classification results and is easily transferrable without requiring prior knowledge of any datasets. The effectiveness of the kNN algorithm for predicting conflict-prone traffic conditions was demonstrated in previous studies, where it was shown to be more accurate and transferable compared to other classifiers such as Neural Networks. Moreover, Yu et al. [36] used kNN to identify the similarities between current and historical traffic states and integrate generations of the most similar k historical states as prediction results. The most similar k historical states are referred to as k nearest neighbors. kNN is implemented by defining the state series, selecting a distance metric, and using a prediction function. kNN regression has been used successfully in road safety analysis. Park et al. [84], for example, employed kNN regression to predict sight and safe stopping distances in low visibility conditions. Scott et al. [85] used kNN regression to examine the distribution of crashes in Miami-Dade County. Iranitalab and Khattak [86] investigated the effectiveness of various data mining techniques for assessing the severity of two-vehicle crashes and found that kNN classification performed best. kNN has been used successfully in road safety analysis and short-term traffic situation prediction. It has demonstrated encouraging results in predicting traffic situations during exceptional events where data may be scarce or unevenly distributed.

3.4 On SUMO calibration

Manjunatha et al. [87], Ge and Menendez [88], Punzo and Ciuffo [89] agree that calibration, the default values of each parameter are modified until the difference between the simulated and actual measurements is reduced. Prior to calibration, it is critical to identify the parameters that influence the model's output. Many parameters influence SUMO's driving behavior and traffic flow. Calibrating SUMO for all of these parameters may not be necessary because all of these aspects may not have a substantial impact on the driving

behavior of a specific model.

Bagheri et al. [90] present a model based on an artificial neural network (ANN) for modeling automobiles' gap acceptance behavior at unsignalized junctions. The simulation package Simulation of Urban Mobility (SUMO) is used to create the model, then compared to default and calibrated models. The simulation model is performed by using ground truth data to accurately replicate the driver's behavior, such as gap acceptance, lane changing, car following, and route choice. The results reveal that the ANN-based model outperforms the other models in terms of wait time and acceptable gap values.

Harth et al. [25] using real-world observations, calibrate a traffic flow simulation in SUMO. The suggested calibration technique includes replicating realistic traffic light programs as well as adjusting simulated traffic flows. The calibration considers the comparison of simulated measures to the equivalent real-world data, as well as the calibrated traffic signals' ability to handle the specified traffic demand.

Jayasinhe et al. [91], the calibration process was carried out utilizing the Simultaneous Perturbation Stochastic Approximation (SPSA) algorithm. SPSA is a stochastic optimization approach that updates model parameters using a gradient approximation. The approach is intended for use with noisy and computationally expensive objective functions, making it appropriate for traffic simulation models. To reduce the Mean Absolute Percentage Error (MAPE) between the simulated and observed speeds, the SPSA method was applied. Iterative calibration was used until the MAPE was decreased to an acceptable level. Scatter graphs of simulated traffic counts and speeds vs their observed equivalents were used to evaluate the calibrated model.

3.5 On measuring the accuracy of the model

Calibration entails adjusting the unknown parameters until a good match between the simulation results and the corresponding physical measurements for the response(s) of interest is established. Calibration is frequently done inefficiently by trial and error [92].

Pool et al. [93] use Pearson correlation to evaluate the performance of Runoff models. Where the Pearson correlation coefficient values for high discharge levels led to fewer constrained simulations at low flows. The specified sensitivity of the Pearson correlation coefficient and the usage of the FDC decreased the range in simulated hydrographs under high-flow circumstances.

Wang et al. [94] provide this research to calibrate traffic flow scenarios in China by using genetic algorithms. To evaluate the accuracy of the predictions it was assign a Pearson correlation score to each variable. Thus, these parameters were adjusted to provide a considerably greater correlation (Pearson correlation) between the frequency of simulated conflicts and genuine conflicts. As a result, the majority of the findings in this study were similar to earlier research such as Huang et al. [95]. Because both research focus on replicating urban traffic flow in China, there appears to be a possibility of parameter transferability for simulation. That is why Using the Pearson evaluation coefficient might help you compare the results to other works or studies that utilize this measure to evaluate models in similar scenarios. This is especially beneficial to compare models to existing research or check the obtained results against earlier studies. On the other hand, Hoot et al. [96] developed a computer simulation for the specific purpose of real-time forecasting

of stands for Emergency Department (ED) operating conditions and to validate the ability of the simulation to forecast several measures of ED crowding. They used the Pearson coefficient of correlation was used to measure the reliability of the simulation forecasts for each continuous outcome measure in comparison with the reference standard. The Pearson r measures the strength of linear association and, when squared, summarizes the fraction of explained variation in the outcome.

Chapter 4

Methodology

4.1 Phases of Problem-Solving

This study suggests a way for automatically calibrating reusable models of traffic microsimulations that are thought to be similar to one another. The methodology's objective is to complete all activities required to produce a reusable automatic calibration model from a setup for an originally uncalibrated traffic microsimulation. In the validation tests of this research, Artificial Neural Networks, Random Forest regression, k-Nearest Neighbors regression, and genetic algorithm were used to develop the calibration model, which was created using supervised learning, a branch of machine learning techniques.

As a result, the process is broken down into four steps: building a training dataset with enough calibration input-output examples, training machine learning models to do the task accurately, evaluating the "r" score (Equation: 2.2), and obtaining a machine learning model for calibration, or retraining the machine learning model to work union with genetic algorithm for calibration. The input/output order of the traffic simulator is reversed in this novel approach, where a large number of traffic simulation results are used to build the dataset. The simulation outputs are used as inputs for the machine learning models, and the simulation inputs are the desired outputs

Training dataset creation

Several simulations are performed to build the dataset. The input/output order of the traffic simulator is reversed in this novel approach, where a large number of traffic simulation results are used to build the dataset. The simulation outputs are used as inputs for the machine learning models, and the simulation inputs are the desired outputs as presented in Figure 4.1.

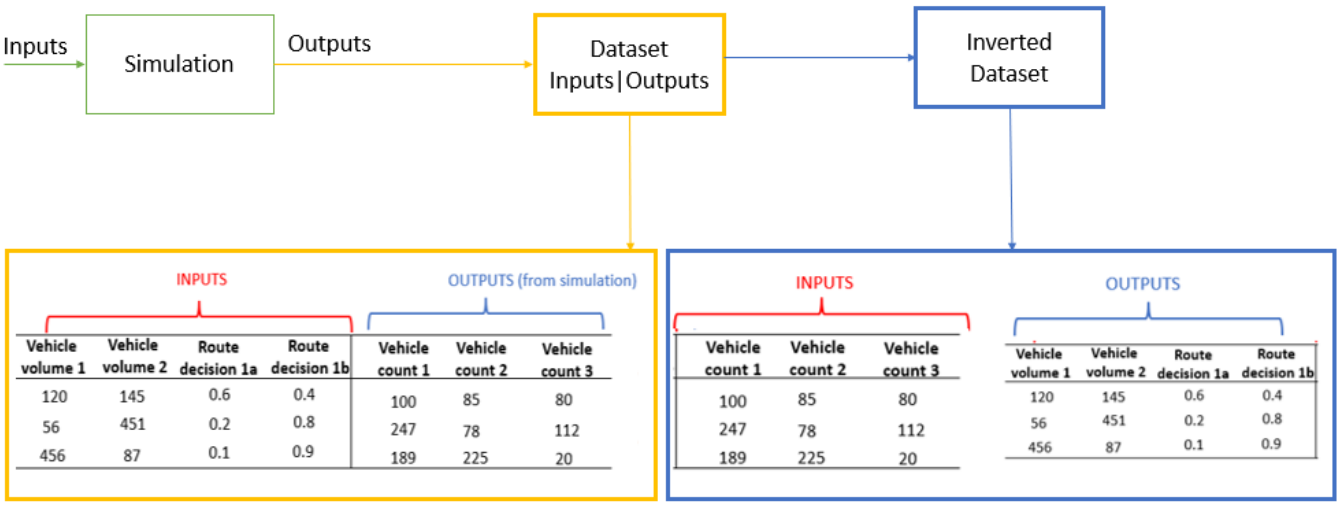


Figure 4.1: Training dataset creation

Training machine learning models

Artificial Neural Networks, Random Forest regression, and k-Nearest Neighbors regression are trained to evaluate their performance in the predictions of input parameters for the simulation. This evaluation is carried on by a correlation analysis average of each variable of the model. According to the Pearson correlation value, we must follow two different options.

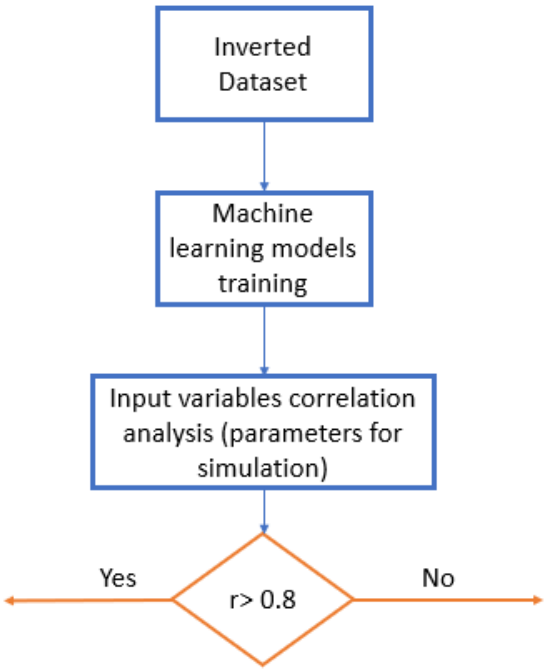


Figure 4.2: Machine learning training and evaluation

Option 1

The first path corresponds to the option where the Pearson correlation coefficient is greater than 0.8. In this case, the model or models that belong to this group can already be used to perform the calibration, now the model has the ability to take the number of vehicles as input and return the simulator parameters for that count. Therefore, these parameters can be simulated, thus obtaining a new vehicle count that will be contrasted with the one given in the machine-learning model. Figure 4.3 shows the flow diagram for this section.

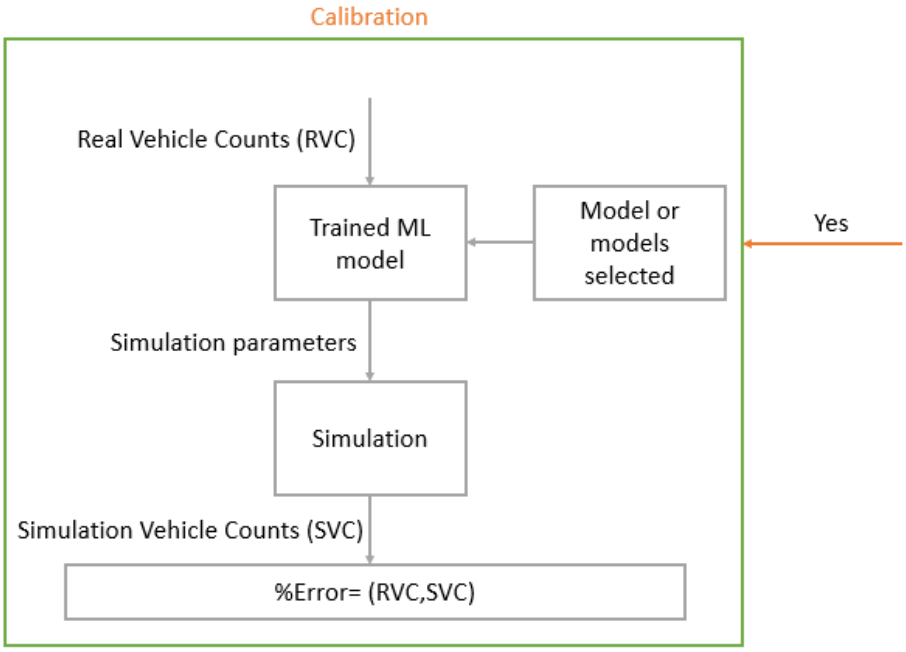


Figure 4.3: If $r > 0.8$

Option 2

The second path corresponds to the scenario where no model reached a Pearson correlation coefficient greater than 0.8. For this section, a retraining of the machine learning models is carried out, with the difference that the dataset will be used before being inverted, with the aim that the models are capable of predicting the count values of the simulator without its execution. This action is carried out in order to take advantage of the fact that the dataset was created. Generally, this measure is adopted for those more complex scenarios where the proposed model of inverting the dataset does not work as expected. This alternative combines the machine learning model with the best performance with a genetic algorithm whose objective is to iteratively find the best individual to obtain the real-world vehicle count.

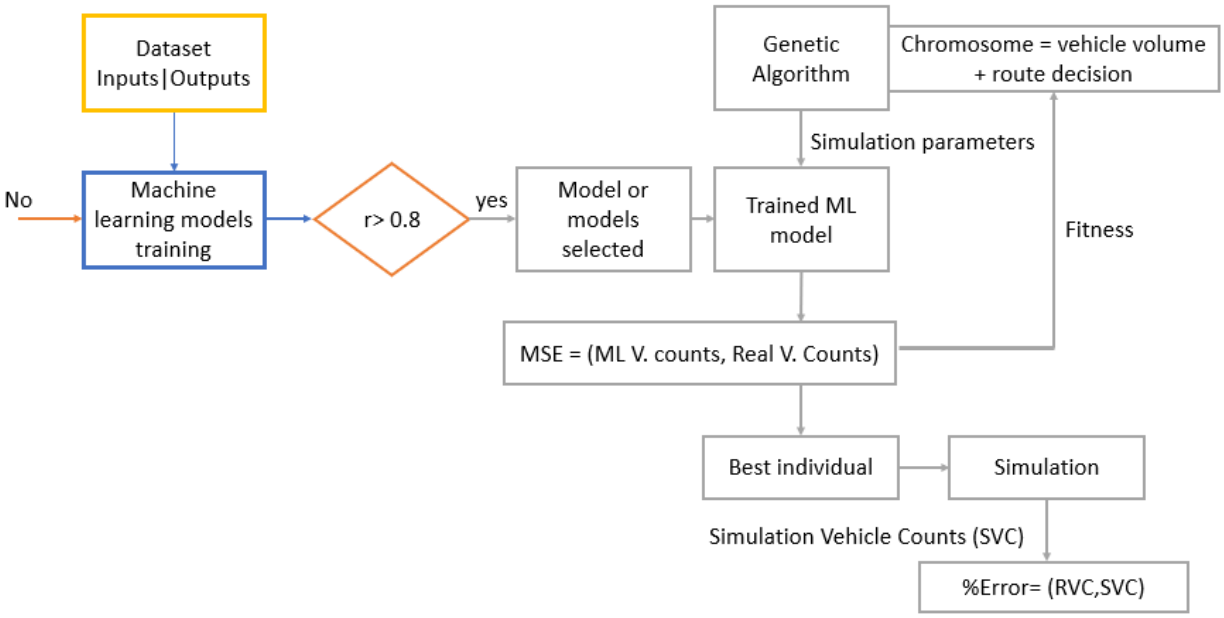


Figure 4.4: If $r < 0.8$

4.1.1 Description of the Problem

Traffic congestion is a major issue in modern cities, affecting urban mobility, air quality, and the efficiency of transportation networks. It is critical to successfully create ways to control and optimize vehicle flow in order to reduce these issues. In this respect, traffic vehicle simulation has developed as a useful tool for understanding and forecasting traffic behavior in various settings.

However, the precision of simulations in some models, such as SUMO, is dependent on the quality of the data provided as input. If the number of cars in entrance lanes and departure route probabilities are not adequately characterized, simulation results cannot be sufficiently near to reality. One solution to this challenge is to utilize real-world vehicle count as a base-calibrated scenario for the simulator. The goal is to start with the simulation done by counting data and progressively altering the simulator parameters to create a situation that is extremely near to reality.

4.2 Model Proposal

This work proposes machine learning models capable of calibrating a traffic scenario based on the count of vehicles at intersections in the SUMO simulator. For this, we will use a custom tool based on JTRROUTER made in Python with the Traci module for a connection with SUMO. In addition, a method is proposed to automate the simulations in order to obtain the greatest amount of variation in the input parameters and thus have a richer dataset. Two different traffic scenarios will be used for its development. Finally, by performing several simulations for each scenario and training the machine learning models with the simulator outputs as their inputs, and the simulator inputs as their outputs, the

calibration will be carried out.

4.2.1 Parameters selection

As a preliminary step, it is crucial to define what simulations are regarded as similar. The methodology that is proposed as the answer to the problem is used in the validation experiments, which are created using SUMO as the preferred traffic simulator. Remembering Chapter 2, SUMO (along with other simulators) conducts the simulation function by converting the set of all simulator inputs (such as driving behavior parameters and network inputs) into the set of all simulation outputs (such as simulation states at each step and output aggregate metrics).

The suggested categories for simulation inputs are parameters for driving behavior, network inputs, infrastructure development, and simulation configuration. In the context of this study, comparable traffic simulations—which, practically speaking, are simulations across the same road network map and with the same duration—share the same values and parameters for simulation setup and infrastructure building. Similar simulations also share a number of network inputs and parameters governing driving behavior. These variables' values can change, nevertheless. The only practical outputs needed for calibration are the aggregated data findings, which are shown in Figure 4.5 with the fixed and variable inputs for SUMO.

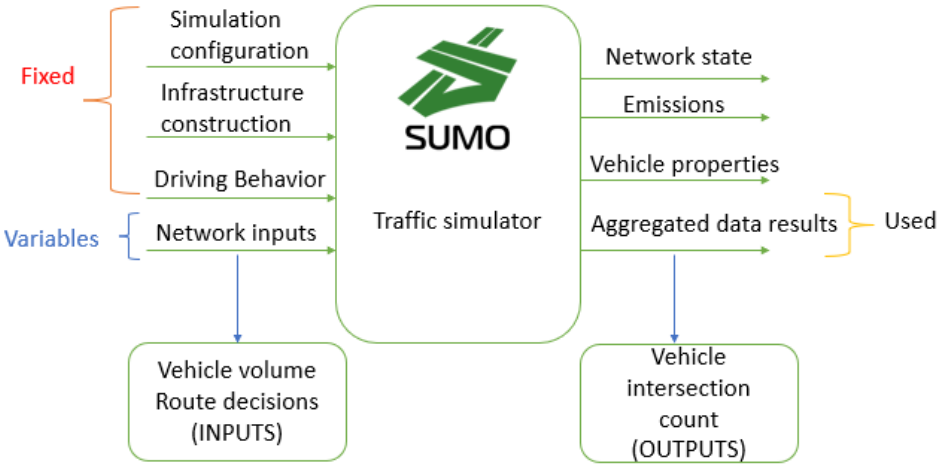


Figure 4.5: Simulations with fixed and variable inputs.

Simulation configuration corresponds to the time of simulation, and infrastructure construction to the network or scenario. Inputs from the network, such as vehicle volume and route decisions, are used for our purposes. Consequently, the simulator's default driving behavior is maintained. The outcomes to be used from aggregated data results correspond to the count of the number of vehicles that pass through intersections.

4.2.2 Customized JTRROUTER

Chapter 2 introduced JTROUTER, a traffic generation tool for SUMO. Its input parameters include the network, vehicle flow, and turn probability. The output is the routes for the vehicle flow, illustrated in Figure 2.5. This tool's biggest drawback is that it only supports straightforward pathways. For paths with just two edges to turn, that is. The tool won't operate if there are cutting edges before turning. Complex scenarios cannot be simulated as a result. It has been modified to be able to execute spin probabilities independent of the presence of intermediate edges, though, based on the design of this tool. Larger and more intricate circumstances can thus be recreated. The rationale behind this change is that the Python script already includes those middle edges.

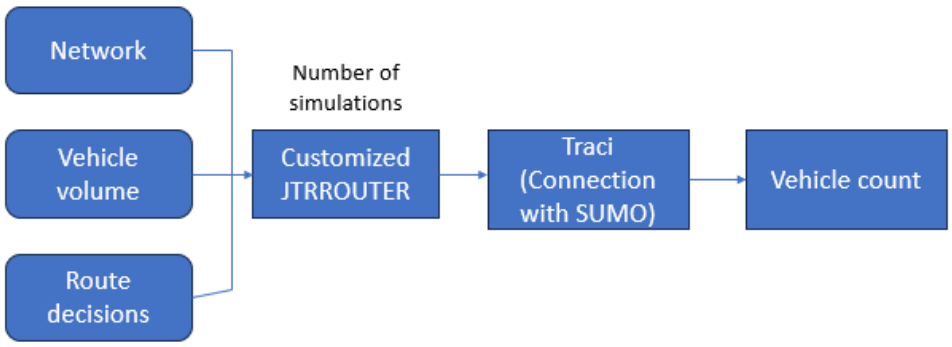


Figure 4.6: Customized JTRROUTER.

4.2.3 Selection of scenarios

For the calibration, two alternative scenarios were used. The city of Ibarra was employed in this instance. SUMO representation Figure 4.8.

Heleodoro Ayala Avenue and Mariano Acosta Avenue serve as the first calibration scenario Figure 4.9a and the roundabout at "La Madre" serving as the second calibration scenario 4.9b. These scenarios were chosen because of their high prevalence in urban areas as well as the fact that their network structures differ significantly from one another. Figure 4.7 shows the networks from Google Maps.

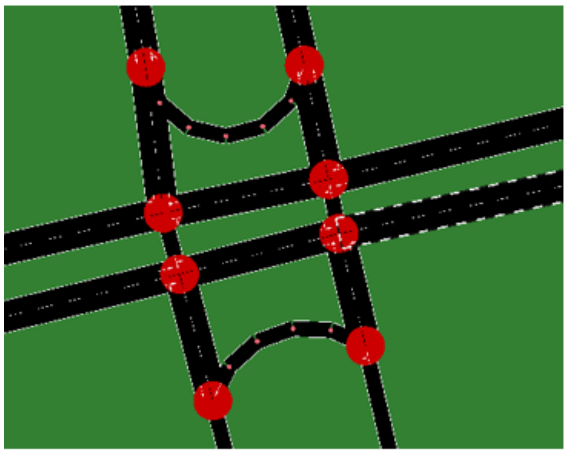
(a) Heleodoro Ayala Avenue and Mariano Acosta Avenue. Retrieved from Google Maps (2023)

(b) Roundabout at "La Madre". Retrieved from Google Maps (2023)

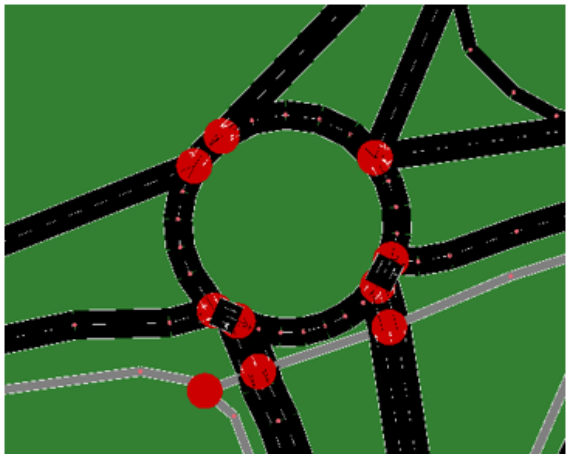
Figure 4.7: Google Maps networks representation.

Following the selection of the scenarios, the network was built using OpenStreetMap and NETCONVERT in order to later have a controllable representation in SUMO. Figure 4.9 shows the result of the scenarios in SUMO GUI.

Figure 4.8: Ibarra city SUMO representation.



(a) Heleodoro Ayala Avenue and Mariano Acosta Avenue.



(b) Roundabout at "La Madre".

Figure 4.9: SUMO networks representation.

4.2.4 Assignment of the volume of vehicles and route decisions for each scenario

For this section, the lanes that feed the network will be taken into account as the inputs for the vehicle volumes, and the probabilities that they can take one route or another based on how the network is designed. For our first scenario, there are four vehicle volumes and six route decisions. Figure 4.10 provides a more detailed illustration.

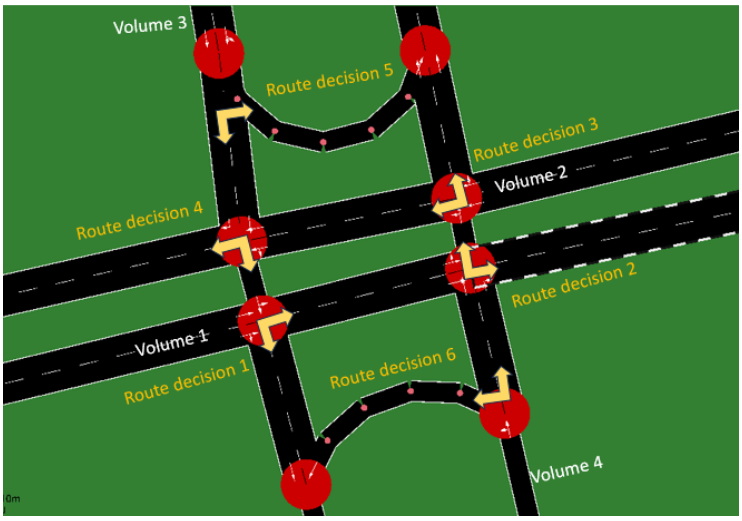


Figure 4.10: First scenario volumes and route decisions.

For the second case, the same technique is employed, there are four vehicle volumes and four route decisions. Figure 4.11 depicts a more accurate portrayal of the aforementioned.

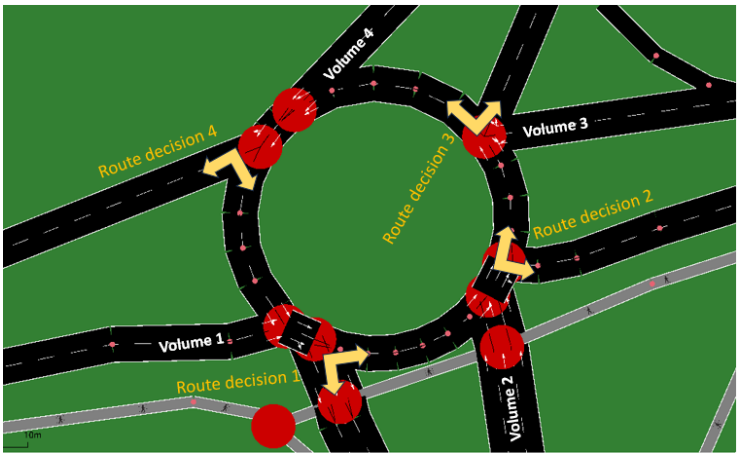


Figure 4.11: Second scenario volumes and route decisions.

4.2.5 Machine learning models

The division of the learning dataset into a training dataset and a testing dataset is one of the most basic methods for developing and testing machine learning models. The machine learning models are then trained using only the training samples, and after training is complete and no further model adjustments are needed, the testing samples are used to assess how well the machine learning models perform in a deployment scenario (for example, mean quadratic error in regression problems and accuracy in classification problems). In order to remove bias from the models, it is crucial that the training phase be conducted without knowledge of the instances that will be used for testing. In Figure 4.12, the conventional dataset split is displayed.

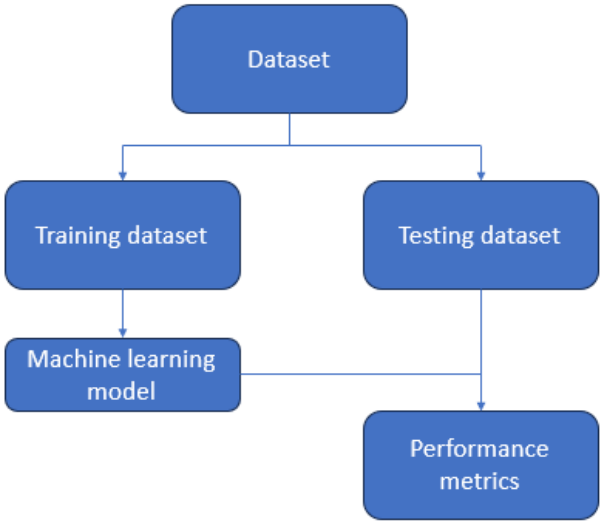


Figure 4.12: Training and testing dataset split.

Because an overfitted model performs better for the training data than a model with fewer epochs, but worse for a new batch of testing data, a portion of the training dataset

is isolated for blind validation at each training epoch, as shown in Figure 4.13.

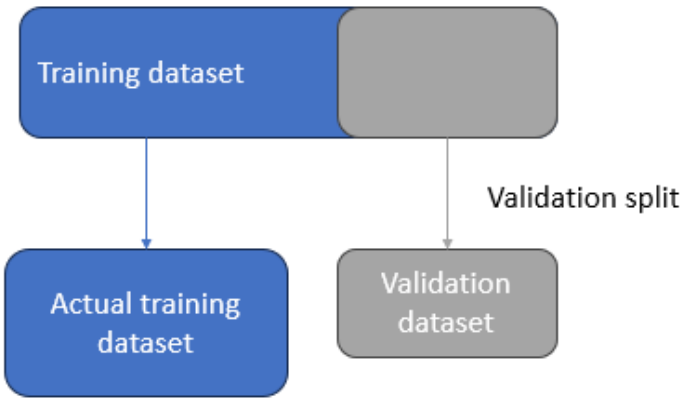


Figure 4.13: A portion of the training dataset is separated for validation.

The validation dataset is essential to automatically split the training data into a training set and a validation set during the training process. This helps monitor model performance on unseen data and stop training early if overfitting is detected.

To train the models it will be done normalization of the inputs (subtraction of the variable's mean and division by its standard deviation) is, therefore, a step that reduces the computational complexity of the iterations. While route decision variables are fractions between zero and one, outputs with broader ranges (such as vehicle volume inputs ranging from zero to thousands of vehicles/hour) may bias the training away from fitting the outputs with smaller ranges. As a result, normalizing of the output samples is also preferred to maintain the parity of importance of all the outputs in the calibration function modeling.

Artificial Neural Network

Since the input and output layers depend on the number of variables in the dataset, feedforward networks are used in this study, and the layer configuration refers to the hidden layers. As a consequence, the results notation indicates that, for instance, a 25-25-25 topology denotes a neural network with three layers that are each 25 neurons wide. To prevent the over-fitting problem an approach is to halt training after a predetermined number of training epochs, which are equal to all rounds of iterations over the training dataset. To prevent the Neural Network from being under or over-fit, the ideal number of training epochs must be determined. Underfitting occurs when there are insufficient training epochs for the network to reach a local minimum on the error function surface, resulting in a high training error. On the other hand, overfitting refers to the network memorizing the training samples rather than thoroughly learning the regression model from the training data. Because neural network experiments can be flexible, Keras environment was required for this purpose. The ability to automatically change the size of the Neural Network layers is one of the benefits of utilizing Keras in Python programs. A collection of networks with various depth and width configurations is trained and tested after the dataset is ready for training.

The technique of early-stopping the training is already in place with Keras, along with a number of widely used activation functions and optimizer algorithms. To ensure the consistency of the findings that support the suggested methodology, these settings are varied. The created neural networks were preserved with the default sigmoid function, one of the non-linear continuous activation functions that is suitable for regression, as advised in the Keras documentation, despite the large range of activation functions—eleven at the time of writing of this project.

The Keras library, on the other hand, contains the implementation of multiple optimizer algorithms for network training, and they had an effect on training times and the number of epochs before the early-stopping trigger. The six tested optimizers are mentioned below, from those recommended in the manual for usage in regression problems; all were left with their default internal settings.

- Stochastic Gradient Descent
- Adagrad
- Adadelta
- Adam
- Adamax
- Nadam

Finally, the structure of this neural network will be a 50-50 hidden layer, a sigmoid activation function, and 6 different optimizers for evaluating its performance in terms of training and number of epochs before early stop. As is represented in Figure 4.14.

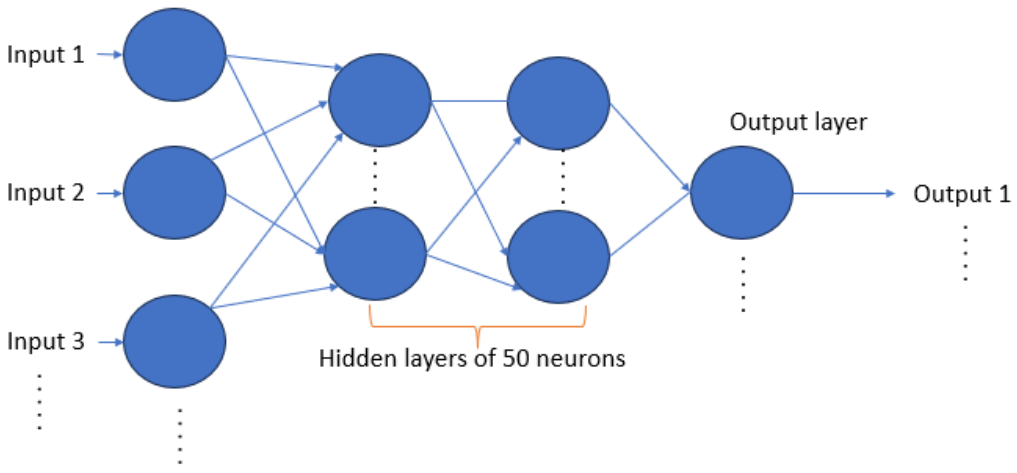


Figure 4.14: Neural Network topology.

Random Forest regression

Advanced methods to address the calibration of traffic microsimulators were investigated during this algorithm design phase. The usage of Random Forest Regression, a potent method discovered in assembly methods in the field of machine learning, was one of the chosen strategies. Building numerous distinct decision trees, and then merging their outputs to provide more precise and reliable predictions, is how Random Forest Regression functions. The diversity and resilience of the ensemble are enhanced by the fact that each tree is trained using a random subset of the training data. Additionally, to avoid overdependence on any particular attribute, splits are performed at the nodes of each tree using a random subset of attributes (input variables).

To build Random Forest regression Scikit-learn, also known as sklearn will be used. It is a Python machine-learning library that provides a wide range of tools and algorithms for developing machine-learning models and performing data mining and predictive analytics. In the scikit-learn implementation, the **"RandomForestRegressor"** model was instantiated with the following pair of parameters:

- **n_estimators:** 100 to generate an ensemble of 100 decision trees. In general, the accuracy and robustness of the model are improved with more trees.
- **random_state:** Set to 0 to ensure the results are reproducible across different runs.

k-Nearest Neighbors (*k*NN) Regression

k-Nearest Neighbors (kNN) Regression was introduced as an additional method to handle the calibration of traffic microsimulators during this stage of algorithm design. kNN is a machine learning method that bases predictions on the closeness of data points and employs a non-parametric approach. kNN runs in a straightforward but effective manner. Regression, as it relates to this discussion, employs kNN to locate the k training points that are the closest to a new location and use their average output values to generate a forecast at the new point. In other words, the prediction is based on the k nearest points' average outputs. **"KNeighborsRegressor"** model was instantiated with the following pair of parameters:

- **n_neighbors:** This parameter indicates the number of nearest neighbors to consider when making predictions. In this case, 5 near neighbors were chosen.

Genetic algorithm

A genetic algorithm mixed with a machine learning model will be employed as part of the alternative. The mix of vehicle volumes and route decisions is the composition of chromosomes. The genetic algorithm will iterate till it finds the best individual for the real vehicle count by using the parameters presented in Table 4.1

Table 4.1: Genetic algorithm parameters

Parameter	Value	Description
Population_size	100	The number of chromosomes in each generation of the population.
Mutation probability	0.2	The probability that a chromosome will mutate after reproduction.
Number of Generations	500	The number of complete iterations of the evolution cycle.

Highlights:

- **Fitness Evaluation Function (fitness(chromosome))**: This function measures how fit a chromosome is based on its performance. In this case, the inverse of the mean square error between the ML model predictions (Best ML model) and the desired outputs is used. A higher fitness indicates a more promising solution.
- **Selection of the Best Chromosomes:** Chromosomes with higher fitness are more likely to be selected for reproduction. This reflects the principle of "natural selection", whereby the fittest individuals have the best chance of passing their characteristics on to the next generation.
- **Reproduction and Mutation:** Reproduction is the process of merging the features of parent chromosomes to produce offspring. Mutation promotes variety in the population, allowing for the development of new features and preventing premature convergence to inferior solutions.
- **Evolution over the Generations:** Because of selection, reproduction, and mutation, chromosomes tend to get more fit as generations pass. This depicts how the algorithm strives to improve the population iteratively depending on fitness.
- **Search and Optimization:** The genetic algorithm seeks solutions that minimize the machine learning model's prediction error. The population evolves to get closer to solutions that better suit the facts and are more likely to produce accurate forecasts.

4.3 Analysis Method

The analytical technique was developed to thoroughly assess the effectiveness of the genetic algorithm and machine learning models. It explained how to compare various models and approaches and the precise criteria used to rate the accuracy of the forecasts. This made it possible to evaluate the approaches' accuracy and dependability on a quantitative and qualitative level.

To assess the adjustment and convergence of a machine learning model, a number of error functions are used. Literature contends that the usage of quadratic errors, mean quadratic errors, and root mean quadratic errors is appropriate for the specific application of traffic microsimulations due to the stochastic character of traffic. Given that it was one of the default options in Keras and Scikit-learn, the Mean Squared error from a list of suitable error functions is supplied in Table 4.2 and will be utilized in the experiments.

Table 4.2: Error Formulas. Retrieved from [2]

Name	**Formula**
Square Error	$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
Mean Squared Error	$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
Root Mean Squared Error	$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$
Root Mean Squared Normalized Error	$\frac{1}{n}\sqrt{\sum_{i=1}^{n}\left(\frac{y_i - \hat{y}_i}{y_i}\right)^2}$

Where the predictions of a model (represented by $\hat{y}_i$) are measured with the actual observed values (represented by y_i), and n: It is the total number of observations.

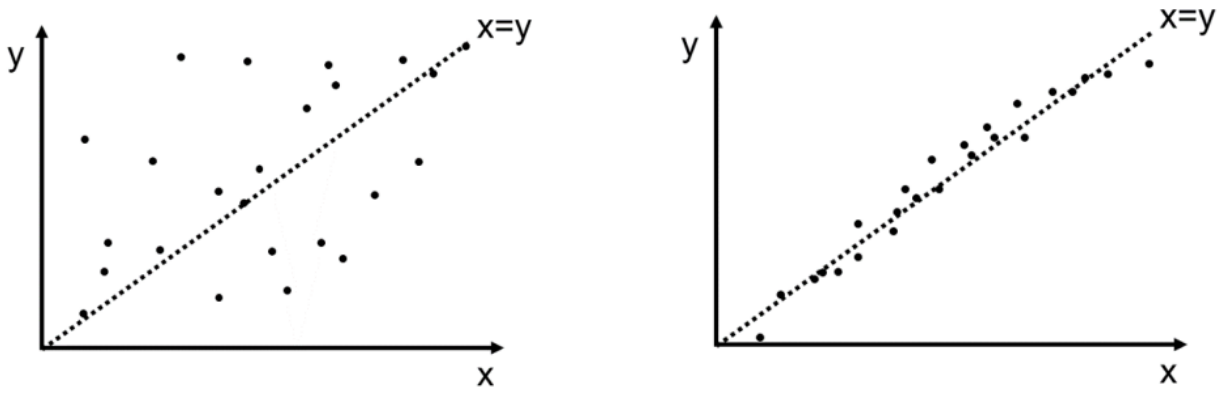
In addition, the process suggests that the user (for instance, a traffic engineer) has the knowledge to assess the trained model for each distinct calibration variable. Additionally, the user can validate the variables that were modeled with acceptable errors and use them, ignoring the machine learning models' recommendations for the variables that were poorly modeled.

The testing dataset is used to assess each variable's performance. Machine learning models process the test inputs, and the results are compared with the desired results from the dataset. Additionally, the Pearson correlation between the model predictions and the desired test outputs is the chosen metric to assess performance in a normalized manner. Equation 4.1 is used to determine the correlation, which serves as a score for the accuracy of the prediction for each variable. x and y represent the expected and intended output values, respectively.

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{4.1}$$

Variables with negative and tiny correlation values should be ignored by the user since they indicate that the model's predictions do not match the intended values with precision, and therefore the model's accuracy following deployment is suspect. On the other hand, variables having correlation values near to 1 may be calibrated with precision by the network upon deployment.

A plot of the x and y variables on the XY plane shows a poor connection when the points are randomly distributed, but a significant association when the points are near the x=y line, as shown in Figure 4.15a and 4.15b.



(a) (b)

Figure 4.15: Plot of x and y variables with weak (a) and strong (b) correlations.

Works made by Pool et al. [93], Wang et al. [94], and Huang et al. [95] use the Person correlation to correlate input variables and the prediction's models. They argue that when there exists a linear relationship between two quantitative variables this metric is relevant and provides valuable information about the quality of predictions based on the linear relationships between variables. In addition, using the Pearson correlation coefficient we can make the results comparable to these works that also use this metric to evaluate models in similar situations.

In the context of the problem presented in this work, we remark that the nature of the data waits to be predominantly linear, which means that as more vehicles are counted in the network, more volume of vehicles are introduced in the network, and the same principle for route decisions, while route probability for a specific route is greater, more number of vehicles that route will have. This is why the Pearson correlation is an adequate metric to evaluate the performance of the model.

4.4 Experimental Setup

The calibration function dataset is formatted as shown in Figure 4.16, which emphasizes how the input and output roles change depending on whether the calibration function or simulation is being used. The rows list the simulation runs, while the columns list the input and output variables. For example, vehicle volumes are non-negative integers that range from 0 vehicles/hour to the maximum theoretical capacity of the road, according to a uniform distribution, and are set to a variety of randomly generated values within a defined range that is appropriate to the application.

In the suggested methodology, after creating the infrastructure map and selecting the relevant variables, the user specifies the value ranges and quantity of simulation runs. The tables can be managed with Python libraries like NumPy or Pandas, which provide optimized functions for operations on huge tables and random-value generation, but this is not required to keep the scripts within the same programming environment. To quickly generate the variable values, spreadsheet applications such as Microsoft Excel or other open-source substitutes can be utilized.

Vehicle volume 1	Vehicle volume 2	Route decision 1a	Route decision 1b	Vehicle count 1	Vehicle count 2	Vehicle count 3
120	145	0.6	0.4	100	85	80
56	451	0.2	0.8	247	78	112
456	87	0.1	0.9	189	225	20

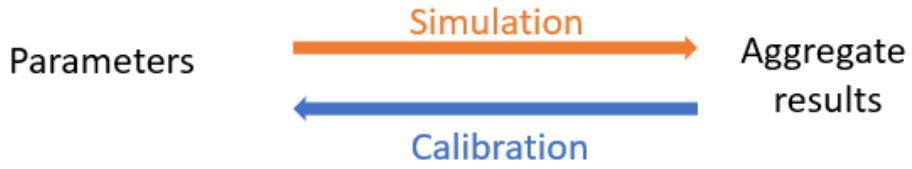


Figure 4.16: Dataset example.

It is important to recall that for the sake of simplicity, route choices are always modeled as pairs, and the sum of the routing possibilities (a) and (b) is always equal to 1. Since just the value of option (a) needs to be estimated for each route decision, As a consequence machine learning models need to estimate fewer parameters.

The traffic engineer can opt to disregard some of the recommendations from the calibration model since the software defines the performance metrics for each variable to be calibrated during implementation. The list of tasks below explains the recommended methodology presented in Section 4.1.

Start with: Simulation road network.

- **Dataset creation**
  - Produce random values for the calibration inputs (volume of vehicles and route decisions).
  - Run a number of simulations.
  - Exchanging simulation inputs and outputs will help you generate the calibration function dataset.
- **Machine learning models**
  - Separate datasets for training and testing.
  - Training machine learning models
  - Determine which inputs are appropriately estimated.
- **Analysis of performance's models**
  - If Person correlation is greater than 0.8; **End with:** machine learning model for automatic calibration.
  - Else incorporate genetic algorithm; **End with:** machine learning model combined with genetic algorithm for automatic calibration.

An important aspect to consider in research and testing is the number of simulations conducted to create the training dataset. It's important to find a balance between enough simulations to accurately represent the behavior of the simulator, while also ensuring that it's feasible to handle within a given time frame. These values serve as the minimum and maximum limits for the number of simulations required.

Chapter 5

Results and Discussion

In order to validate the strategy suggested for the calibration of mobility and traffic simulation models through machine learning, two significant experiments that were undertaken are presented in this dissertation. SUMO program was used to execute the traffic simulations, and the Python, Keras, and Scikit-learn environments were used to create the machine learning models. Due to their intricacy and regular traffic jams, the two road networks used in the tests were of interest. Heleodoro Ayala Avenue and Mariano Acosta Avenue Figure 4.10 and the roundabout at "La Madre" Figure 4.11.

The network utilized machine learning models to adjust the number of vehicles that enter the map's edges and the percentage of cars that take each direction at every fork in the road. All tests employed the same vehicle model to calculate volumes, with only car entities presented in the simulation. Motorcycles were assumed to be less than one car, while buses and trucks were considered to be more than one car.

5.1 First scenario results

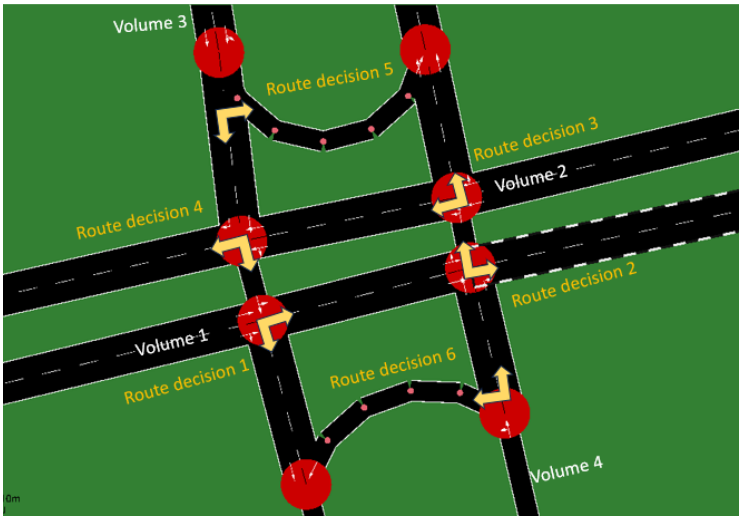


Figure 5.1: First scenario volumes and route decisions.

The desired inputs for calibration are the 4 vehicle volumes that enter the edges of the map, and the ratios of the 6 route decisions that are part of the road network. The output performance metrics from SUMO that are used by machine learning models to calibrate simulations on this map are 10 vehicle counts.

The setup given below was used to prepare the simulation runs.

- **Number of simulations:** 10.000. The number was chosen empirically. Training and testing are split 80/20, with 20% of the training dataset used for validation.
- **Inputs**
  - 4 volume of vehicles, following a uniform distribution between 0 and 1000 vehicles per hour.
  - 6 route decisions.
- **Simulation duration:** 3600 seconds, chosen empirically.
- **Outputs:** 12 vehicle counts at every fork in the road.

To train all the models the inputs/outputs will be inverted. It means that the outputs from the simulator will be used as inputs to the models, and the output from the simulator as inputs to the models to build the calibration function as it was presented in Chapter 4, Section 4.1

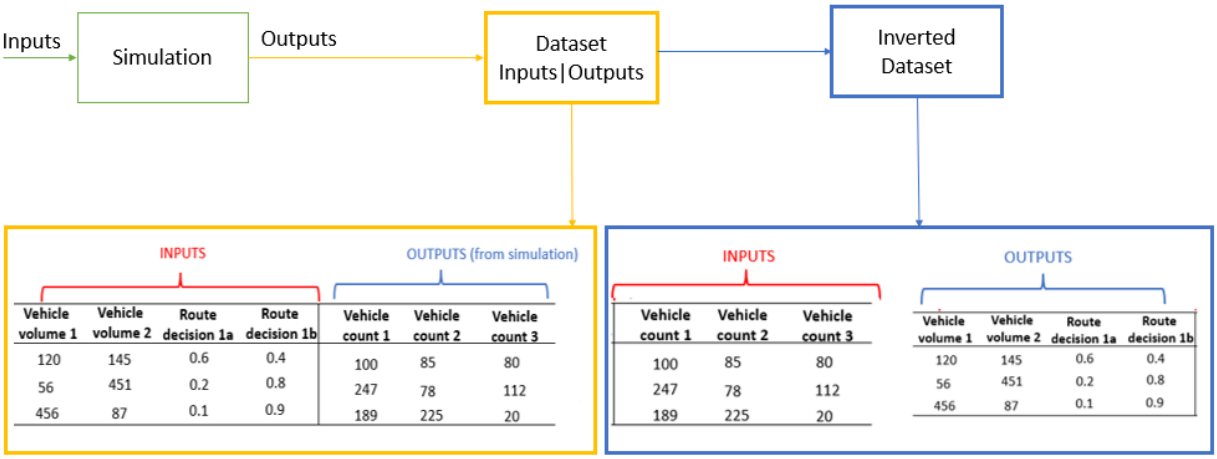


Figure 5.2: First scenario volumes and route decisions dataset example.

To train the models it will be done normalization of the inputs (subtraction of the variable's mean and division by its standard deviation) is, therefore, a step that reduces the computational complexity of the iterations. While route decision variables are fractions between zero and one, outputs with broader ranges (such as vehicle volume inputs ranging from zero to thousands of vehicles/hour) may bias the training away from fitting the outputs with smaller ranges. As a result, normalizing of the output samples is also preferred to maintain the parity of importance of all the outputs in the calibration function modeling.

The Pearson correlation between the model predictions and the desired test outputs is the chosen metric to assess performance in a normalized manner. It is used to determine the correlation.

5.1.1 Neural Network

Following the simulations and the development of the training dataset, a number of neural networks were trained and put to the test, with the performance metric being the correlation between the target values for calibrating the simulation and the estimations from the neural networks. The training/testing dataset split was empirically calculated as 80/20 for all trained networks across all experiments based on the custom in the examined literature. The validation split was also 80/20 inside the training dataset.

Configuration for early-stopping is:

- **monitor='val_loss':** This means that the callback is monitoring the loss metric in the validation set ('val_loss'). Training will stop if this metric does not improve.
- **min_delta=0:** The min_delta parameter specifies how much improvement is considered significant. A value of 0 means that no minimum improvement in the metric is required for the callback to trigger. In other words, training will stop if the loss metric on the validation set does not improve at all during the number of epochs specified in patience.
- **patience=2:** The patience parameter indicates how many epochs the callback should wait without improvement in the loss metric on the validation set before stopping training. In this case, if the loss metric does not improve for two consecutive epochs, training will stop.

To experiment with the six optimizer methods suggested for regression in Keras documentation, we fixed the structure of two hidden layers with 50 neurons each (noted as a 50-50 configuration). This configuration was necessary due to the large number of input and output variables. Figure 5.3 displays the number of epochs required for training the neural networks with each optimizer before implementing the early-stopping callback.

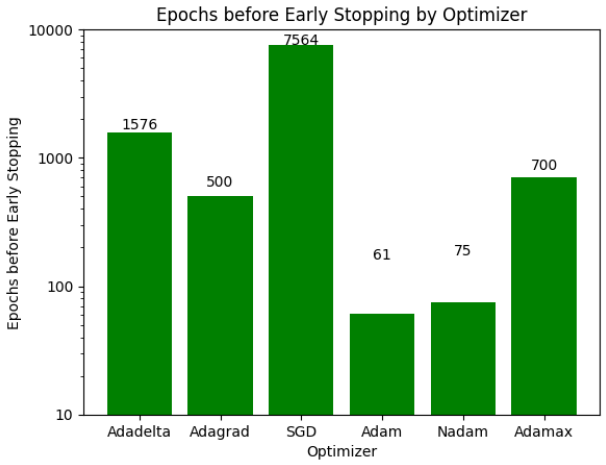


Figure 5.3: Epochs before early-stopping for each optimizer in the first scenario.

The Stochastic Gradient Descent (SGD) optimizer, which uses a fixed step size for navigation on the error function's surface, was discovered to require more epochs than other optimizers. In comparison, the other five optimizer options use adaptive size steps

and require roughly 2000 epochs before the models have overfitted. In contrast to the other four, Adam and Nadam required the fewest number of epochs before overfitting. Both "Nadam" and "Adam" are gradient descent algorithms with various tweaks that can make them more efficient in specific situations. One explanation could be that "Adam" and "Nadam" include a momentum term in their weight updates.

Figure 5.4 depicts a comparison of the average correlation metrics for the six tested optimizers. Because the fluctuation in performance is less than 1%, it has been concluded that the optimizers influence the training duration of the Neural Networks but not their performance in this experiment.

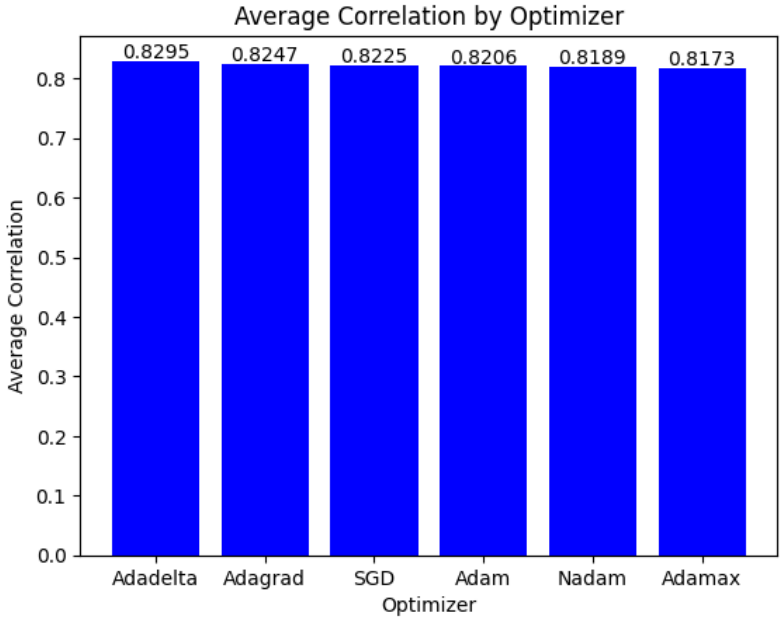


Figure 5.4: Correlation for each optimizer in the first scenario.

Figure 5.5 compares the calibration performance of each variable in SUMO by the Neural Network.

In the case of the volume variables (Vol_), we observe that the correlations are in a relatively high range, ranging between 0.905 and 0.8609. This indicates that there is a positive and moderately strong relationship between these volumes. That is, as one volume increases, it is likely that another will also increase by a certain proportion. This may indicate some consistency in the behavior of these variables.

On the other hand, the route decision variables (Rout_) show correlations in a slightly lower range, varying between 0.8174 and 0.7444. This suggests a positive relationship, although less strong than in the case of volumes. Paths may influence other variables less consistently, resulting in slightly weaker variance. Finally, all variables estimated by the Neural Network can be used by a traffic engineer to calibrate and build a scenario based on vehicle counting.

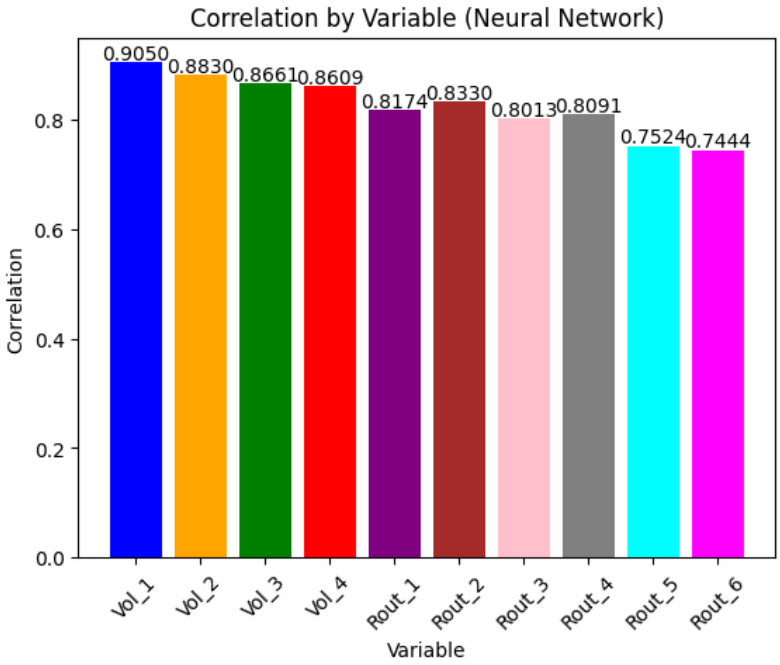


Figure 5.5: Correlation for each variable in the first scenario by Neural Network.

5.1.2 Random Forest regression

Figure 5.6 compares the calibration performance of each variable in SUMO by the Random Forest regression.

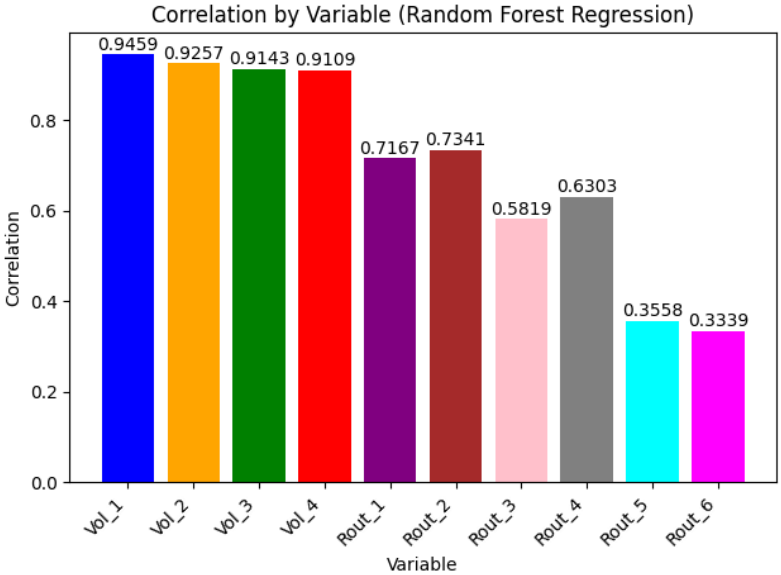


Figure 5.6: Correlation for each variable in the first scenario by Random Forest regression.

The volume variables (Vol_1, Vol_2, Vol_3 and Vol_4) show very high correlations with the simulation results, with values between 0.9109 and 0.9459. This indicates that the

volume of traffic in different sectors or points of the traffic network has been predicted with high precision by Random Forest regression.

The route decision variables (Rout_1 to Rout_6) also have positive correlations with the simulation results, although the correlations are generally lower than the volume variables. The correlations vary between 0.3339 and 0.7341. This suggests that the routes chosen by vehicles in different areas of the traffic network have not been ideally predicted. Therefore the traffic engineer has more freedom to choose the parameters from Rout_3 to Rout_6, but can take as reference the results from the Random Forest.

5.1.3 *k*NN regresssion

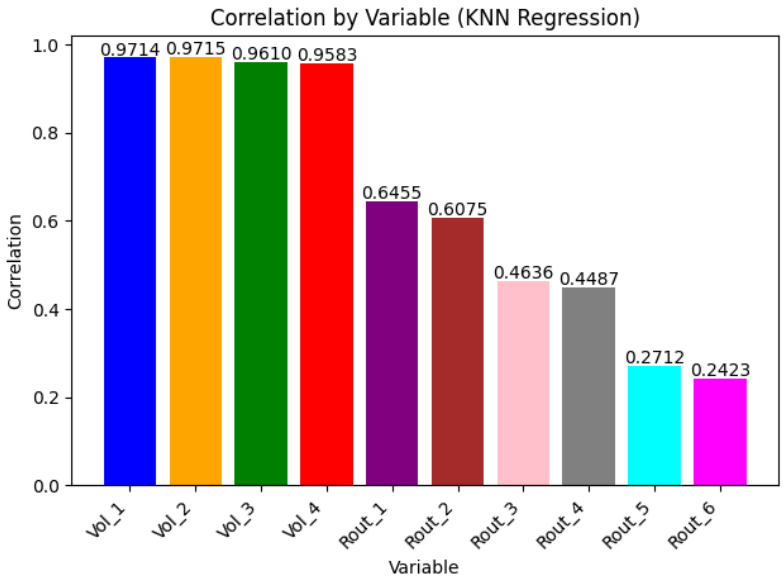


Figure 5.7: Correlation for each variable in the first scenario by kNN regression.

In this experiment represented in Figure 5.7, we looked at the relationship between the input variables and the outcomes predicted by the kNN regression model. The variables are divided into two categories: "Vol_" represents the volume of cars at the crossings, and "Rout_" represents the alternative routes in the scenario.

We see that all of the volume variables ("Vol_") show high and positive correlations with the results predicted by the kNN regression model. This implies that the model can accurately capture and anticipate vehicle volumes at crossings. The variables "Vol_1" and "Vol_2" in particular have the strongest correlations, exceeding the value of 0.97. This suggests that the actual volume values and the model predictions are quite similar.

The routing variables ("Rout_"), on the other hand, have smaller correlations than the volume variables. This could be because routes are more complex and diversified than simple volumes. The correlations between the routes range from 0.24 to 0.6455, indicating that the kNN regression model has a more difficult time properly forecasting the likelihood of taking one route or the other.

These findings illustrate the kNN regression model's ability to estimate vehicle quantities at crossings, with very strong correlations. However, the model has difficulty fore-

casting path probabilities, which results in lower correlations. These insights can help to better understand the model's strengths and limitations based on the different factors and drive potential changes in future model iterations.

5.1.4 Summary Results

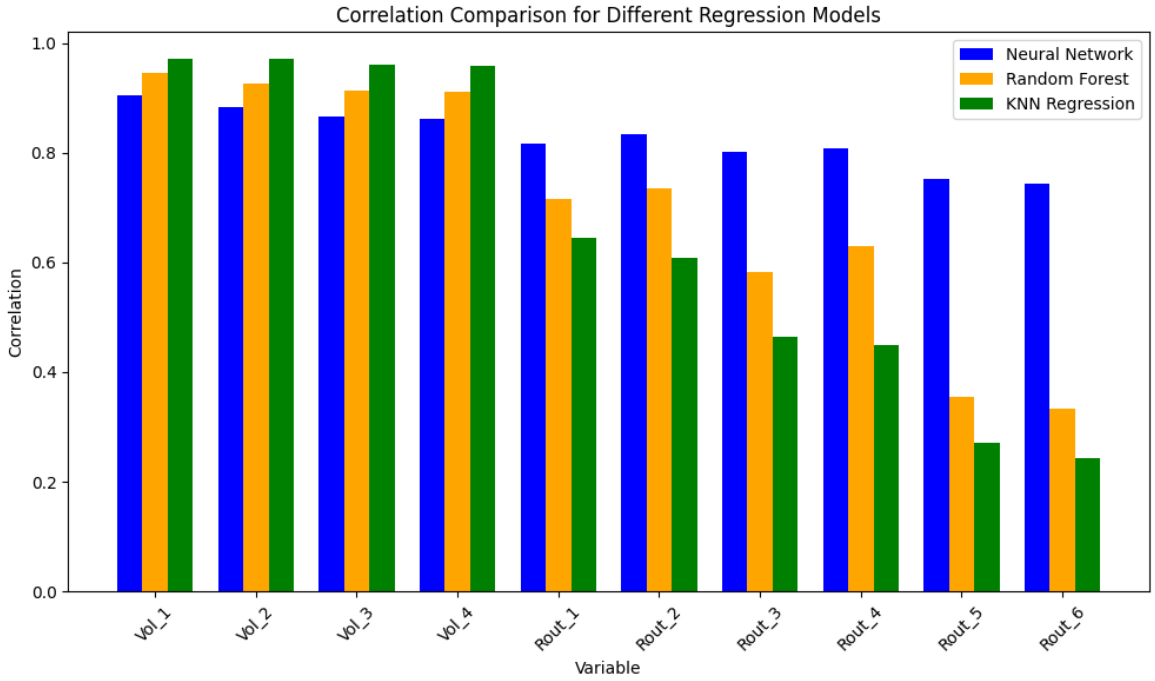


Figure 5.8: Correlation for each variable in the first scenario by each machine learning model for the first scenario.

According to Figure 5.8 for volume variables, all three models show high and similar relationships. This shows that the models, regardless of approach, accurately capture the volumes of cars at crossings.

For route decision variables the correlations between the models show a greater degree of variation. The strongest correlations are seen in the kNN Regression model, indicating that it may be particularly good at capturing links between vehicle paths and other factors in the scenario. The Random Forest model, on the other hand, performs well in some pathways, while the Neural Network has slightly lower correlations.

The results of analyzing the correlations between variables and predictions of three machine learning models (Neural Network, Random Forest, and kNN Regression) provide valuable information about each model's ability to capture patterns and relationships for the prediction of traffic simulation inputs. In general, the Neural Network stands out by having the highest correlations across the board. This constancy in correlations indicates that the Neural Network is capable of modeling complicated and non-linear interactions between input data and output predictions. Its greater performance may be due to your capacity to recognize abstract patterns and learn deeper representations of the material. Although the Random Forest and kNN Regression models exhibit substantial correlations

on numerous variables, such as volume variables, their performance on route decision variables is limited in comparison to the Neural Network. Random Forest, despite its ability to capture complicated associations by mixing several decision trees, may struggle to capture abstract patterns. The kNN Regression, on the other hand, being a model closer to the data and capable of recognizing local interactions, may be less effective in capturing non-linear correlations in bigger data sets.

Table 5.1: Summary results correlation and MSE of machine learning models

Machine learning model	**Pearson correlation (r)**	**MSE**
Neural Networks	0.8189	0.2948
Random Forest	0.7049	0.3459
k-Nearest Neighbors	0.6541	0.4209

Finally, according to results in Table 5.1, Neural Networks is the machine learning model that has a Pearson correlation greater than 0.8. As a consequence, following the proposed methodology, we must follow Option 1 presented in Figure 4.3 to perform the calibration.

To measure the capacity of the Neural Networks to find the optimal parameters input in order to obtain the desired output (vehicle count) we performed four experiments. The percentage error was calculated to evaluate the results as is shown in Table 5.2 where column 1 corresponds to the scenario control and column 2 the results obtained by using the best machine learning model. As a consequence, column 3 has the quality of the simulation using the percentage error.

Table 5.2: Neural Networks results part 1

Vehicle count (expected)	**Vehicle count (obtained)**	**Percentage error (%)**
Count_1 = 200	Count_1 = 170	15%
Count_2 = 120	Count_2 = 150	25%
Count_3 = 95	Count_3 = 125	31.58%
Count_4 = 340	Count_4 = 310	8.82%
Count_5 = 50	Count_5 = 45	10%
Count_6 = 75	Count_6 = 90	20%
Count_7 = 125	Count_7 = 105	16%
Count_8 = 200	Count_8 = 175	12.5%
Count_9 = 300	Count_9 = 330	10%
Count_10 = 150	Count_10 = 190	26.67%
Count_11 = 250	Count_11 = 225	10%
Count_12 = 100	Count_12 = 130	30%

Table 5.3: Neural Networks results part 2

Vehicle count (expected)	**Vehicle count (obtained)**	**percentage error (%)**
Count_1 = 250	Count_1 = 215	14%
Count_2 = 150	Count_2 = 175	16.67%
Count_3 = 110	Count_3 = 140	27.27%
Count_4 = 380	Count_4 = 350	7.89%
Count_5 = 55	Count_5 = 65	18.18%
Count_6 = 85	Count_6 = 105	23.53%
Count_7 = 130	Count_7 = 150	15.38%
Count_8 = 190	Count_8 = 170	10.53%
Count_9 = 240	Count_9 = 220	8.33%
Count_10 = 135	Count_10 = 160	18.52%
Count_11 = 180	Count_11 = 210	16.67%
Count_12 = 75	Count_12 = 100	25%
Count_1 = 300	Count_1 = 285	5%
Count_2 = 180	Count_2 = 198	10%
Count_3 = 80	Count_3 = 110	37.5%
Count_4 = 420	Count_4 = 400	4.76%
Count_5 = 60	Count_5 = 72	20%
Count_6 = 95	Count_6 = 85	10.53%
Count_7 = 140	Count_7 = 160	14.29%
Count_8 = 220	Count_8 = 240	9.09%
Count_9 = 260	Count_9 = 280	7.69%
Count_10 = 120	Count_10 = 130	8.33%
Count_11 = 190	Count_11 = 180	5.26%
Count_12 = 70	Count_12 = 90	28.57%

Table 5.4: Neural Networks results part 3

Vehicle count (expected)	**Vehicle count (obtained)**	**percentage error (%)**
Count_1 = 180	Count_1 = 210	16.67%
Count_2 = 110	Count_2 = 140	27.27%
Count_3 = 70	Count_3 = 85	21.43%
Count_4 = 320	Count_4 = 350	9.38%
Count_5 = 45	Count_5 = 55	22.22%
Count_6 = 65	Count_6 = 80	23.08%
Count_7 = 100	Count_7 = 120	20%
Count_8 = 140	Count_8 = 160	14.29%
Count_9 = 180	Count_9 = 200	11.11%
Count_10 = 95	Count_10 = 110	15.79%
Count_11 = 120	Count_11 = 140	16.67%
Count_12 = 50	Count_12 = 70	28.57%

As it is presented in Table 5.2, Table 5.3, and Table 5.4 Most of the findings have an absolute inaccuracy larger than 20%, which may be due to the fact that the chosen model (Neuronal Networks) in general has a correlation value of 0.8189, indicating that this model would aid in simulator calibration by 81%. This is why the findings remain within that margin of error. As a result, this model's calibration quality is as expected. As a result, the traffic engineer will have to tweak the correction settings, starting with a 20% inaccuracy, substantially simplifying his task and saving him time. Furthermore, this model may assist you in the same way for other traffic conditions on the same network. The traffic engineer might also utilize the Pearson correlation coefficients of each variable to determine the weakest ones and start the calibration procedure from there. In the situation of Neuronal Networks, where the compensation coefficients for Rout_5 and Rout_6 are smaller than 0.76.

5.2 Second scenario results

The desired inputs for calibration are the 4 vehicle volumes that enter the edges of the map, and the ratios of the 4 route decisions that are part of the road network. The output performance metrics from SUMO that are used by machine learning models to calibrate simulations on this map are 4 vehicle counts.

The setup given below was used to prepare the simulation runs.

- **Number of simulations:** 10.000. The number was chosen empirically. Training and testing are split 80/20, with 20% of the training dataset used for validation.
- **Inputs**

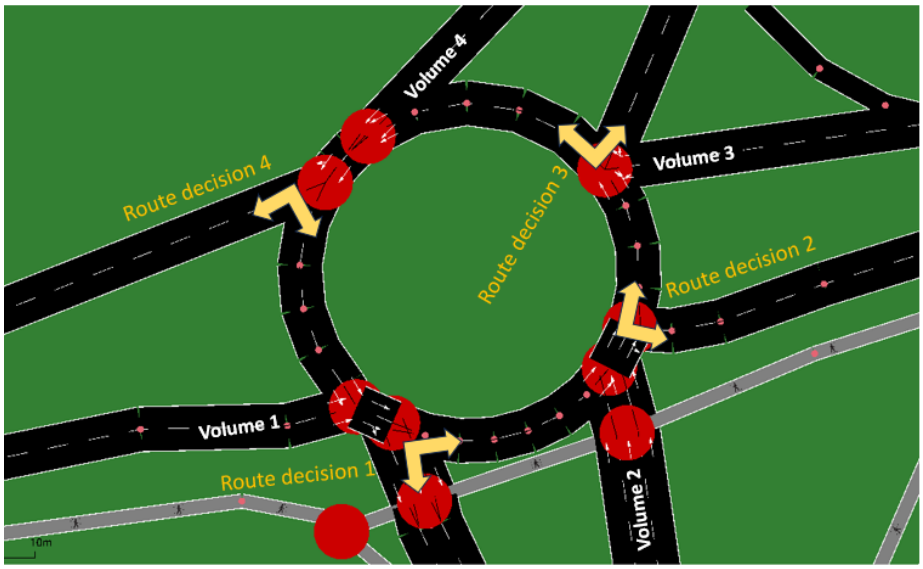


Figure 5.9: Second scenario volumes and route decisions.

  - 4 volume of vehicles, following a uniform distribution between 0 and 1000 vehicles per hour.
  - 4 route decisions.

- **Simulation duration:** 3600 seconds, chosen empirically.
- **Outputs:** 4 vehicle counts at every fork in the road.

To train all the models the inputs/outputs will be inverted. It means that the outputs from the simulator will be used as inputs to the models, and the output from the simulator as inputs to the models to build the calibration function as it was presented in Chapter 4, Section 4.1

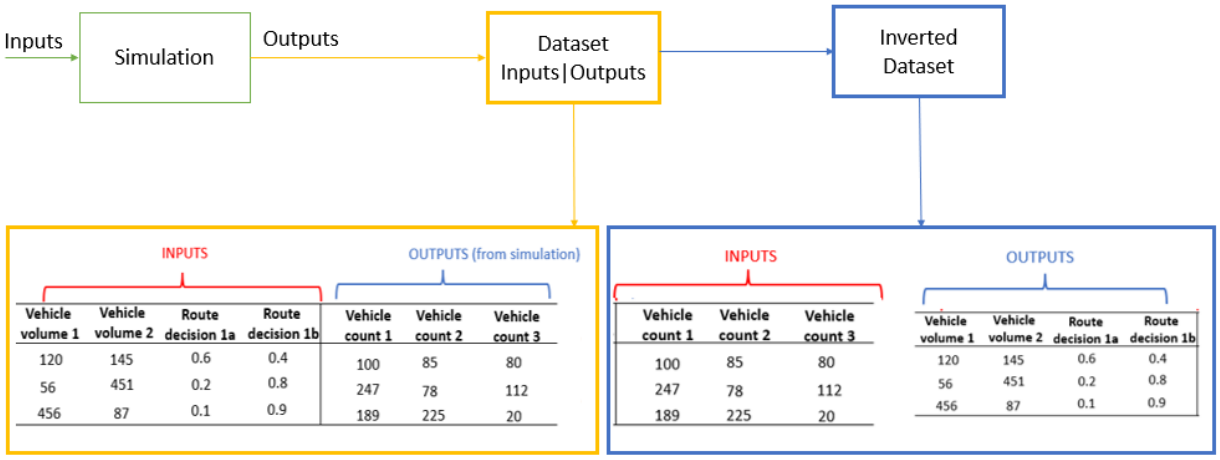


Figure 5.10: Second scenario volumes and route decisions dataset example.

To train the models it will be done normalization of the inputs (subtraction of the variable's mean and division by its standard deviation) is, therefore, a step that reduces

the computational complexity of the iterations. While route decision variables are fractions between zero and one, outputs with broader ranges (such as vehicle volume inputs ranging from zero to thousands of vehicles/hour) may bias the training away from fitting the outputs with smaller ranges. As a result, normalizing of the output samples is also preferred to maintain the parity of importance of all the outputs in the calibration function modeling.

The Pearson correlation between the model predictions and the desired test outputs is the chosen metric to assess performance in a normalized manner. It is used to determine the correlation.

5.2.1 Neural Network

Following the simulations and the development of the training dataset, a number of neural networks were trained and put to the test, with the performance metric being the correlation between the target values for calibrating the simulation and the estimations from the neural networks. The training/testing dataset split was empirically calculated as 80/20 for all trained networks across all experiments based on the custom in the examined literature. The validation split was also 80/20 inside the training dataset. First, was tested each optimizer to measure the number of epochs that each one needs before early stopping. The rules are displayed in the Figure 5.11

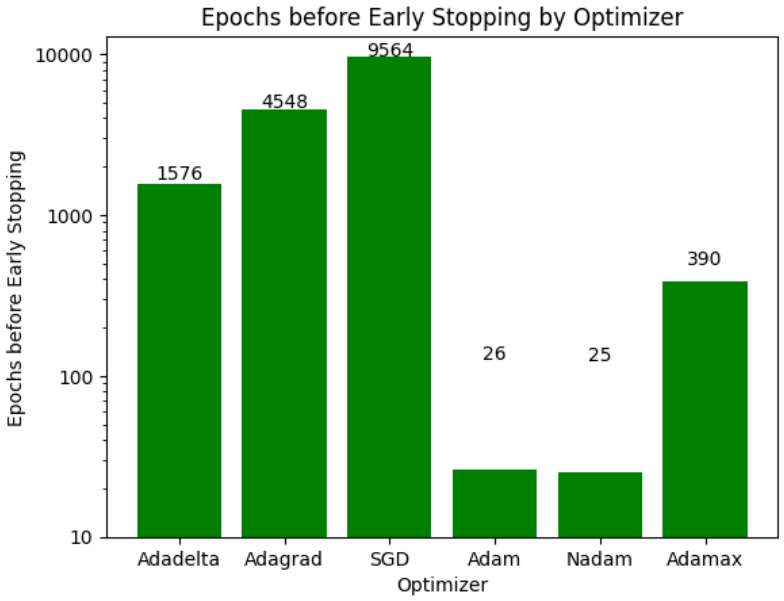


Figure 5.11: Epochs before early-stopping for each optimizer for the second scenario.

Looking at the findings, one can see that the number of epochs required before each optimizer hits the early stopping threshold varies significantly. The "Adam" and "Nadam" optimizers require the fewest epochs before stopping training, with 26 and 25 epochs, respectively. This could be attributable to the optimizers' ability to swiftly adapt learning rates and converge toward optimal solutions.

The "SGD" and "Adagrad" optimizers, on the other hand, require a greater number of epochs before stopping training, with 9564 and 4548 epochs, respectively. This could

imply that these optimizers are less efficient in terms of convergence speed and loss function adaption.

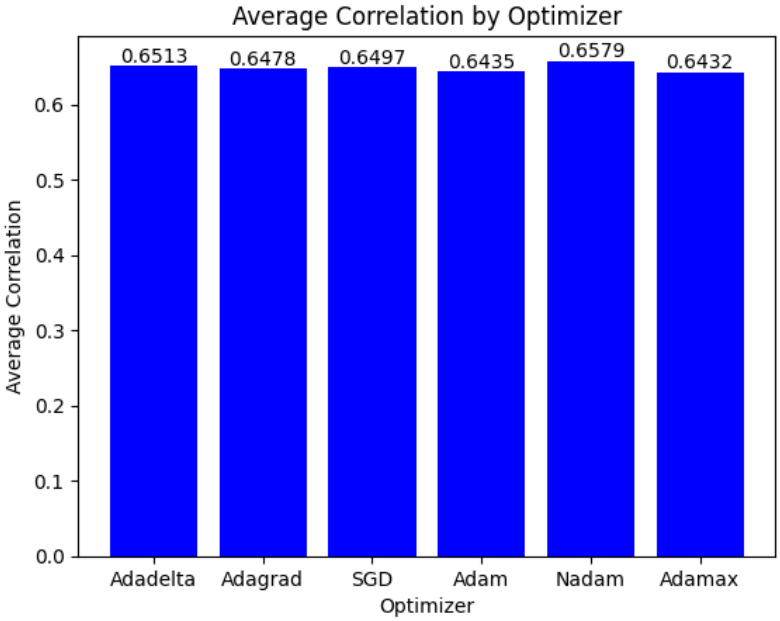


Figure 5.12: Average correlation by each optimizer for the second scenario

The number of epochs before stopping early training (early stopping) for different optimizers, combined with the average correlations obtained with each optimizer, provides a more complete picture of the performance of machine learning models under different conditions. Some interesting trends can be detected by comparing the number of epochs required for training with the average correlations obtained in Figure 5.12. Not only do the "Adam" and "Nadam" optimizers require a small number of epochs before terminating, but they also achieve remarkable average correlations of 0.6435 and 0.6579, respectively. This implies that these optimizers are not only efficient in terms of convergence but also in terms of correlation with the vehicle count to be predicted.

The "Adadelta", "Adagrad", "SGD", and "Adamax" optimizers, on the other hand, exhibit a similar trend in terms of the number of epochs required and average correlations. Although these optimizers may require more epochs to complete training, their average correlations are close to 0.6432 and 0.6513. This suggests that, despite taking longer to converge, they get fairly strong outcomes in terms of correlation to forecast the values related to vehicle count.

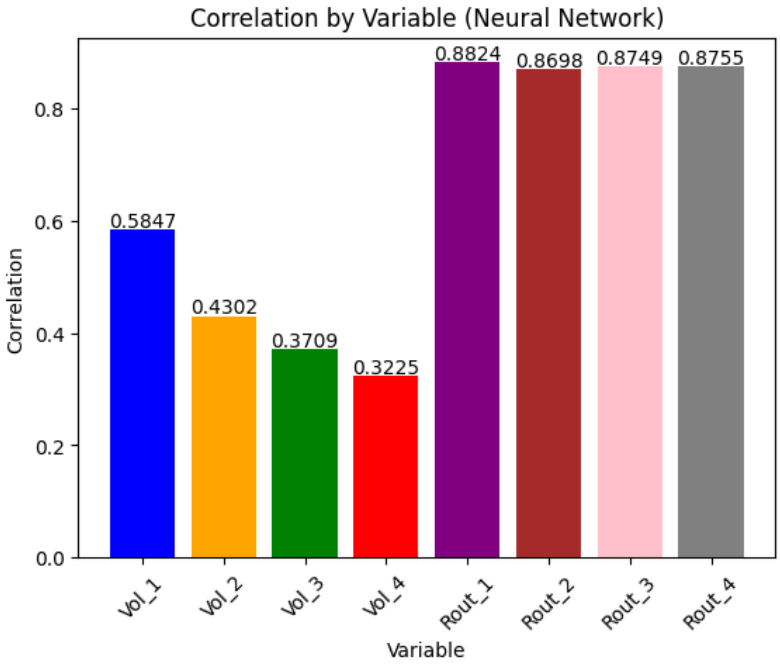


Figure 5.13: Correlation for each variable in the second scenario by Neural Network.

The volume variable correlations are weaker than in the first case, suggesting that the Neural Network may struggle to capture the complicated interactions between inputs and volume predictions in this more complex scenario. This could be because the inputs are more dimensional and there are more interactions between the characteristics. In contrast, the route decisions have relatively strong correlations, showing that the Neural Network is successful in capturing the links between input attributes and path probabilities. The high correlation indicates that the Neural Network is capable of modeling how traffic conditions at intersections influence vehicle routing decisions.

In general, while the correlations for volume factors are smaller in this second case, they are relatively strong for route variables. This implies that the Neural Network is critical in anticipating and modeling vehicle routing decisions, which is required for an accurate and realistic traffic simulator. Users can take as a reference for calibration the values given for volume vehicles.

5.2.2 Random Forest regression

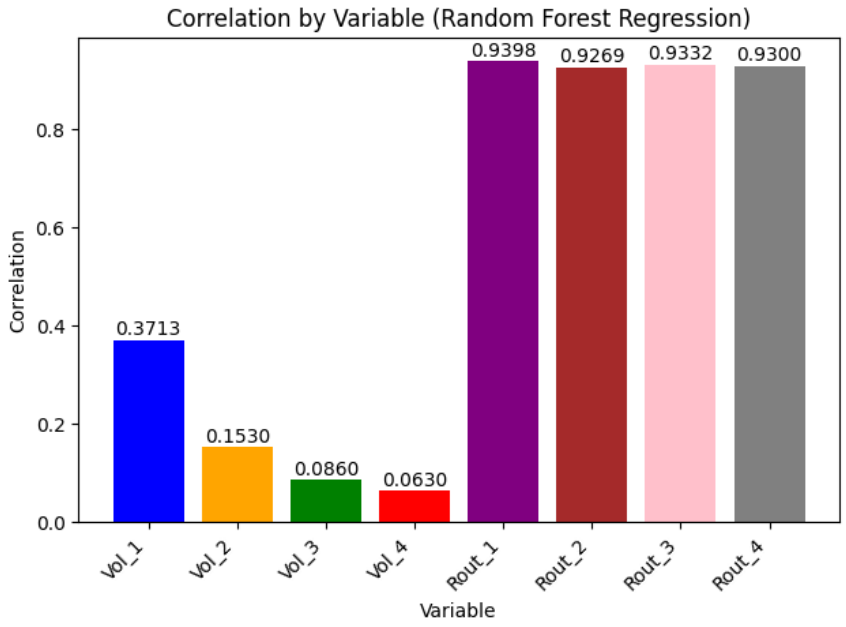


Figure 5.14: Correlation for each variable in the second scenario by Random Forest.

The traffic volumes (Vol_1, Vol_2, Vol_3, Vol_4) have low to extremely poor correlations with the model's expected departures. This could imply that the Random Forest model in this scenario does not well reflect traffic patterns and linkages between entry and vehicle volumes.

The routes (Rout_1, Rout_2, Rout_3, Rout_4), on the other hand, exhibit greater correlations with the model's projected outputs. This implies that the model may be better at capturing the correlations between inputs and the likelihood of taking specific routes. Higher correlations may suggest that the model has discovered data patterns that efficiently relate input properties to outputs. This second scenario looks to be more difficult for the model than the first, where correlations with the Random Forest were higher. The lower correlations for the volume could be attributed to greater complexity in traffic interactions and a greater number of number outputs, making it difficult to predict with few inputs.

5.2.3 *k*NN regression

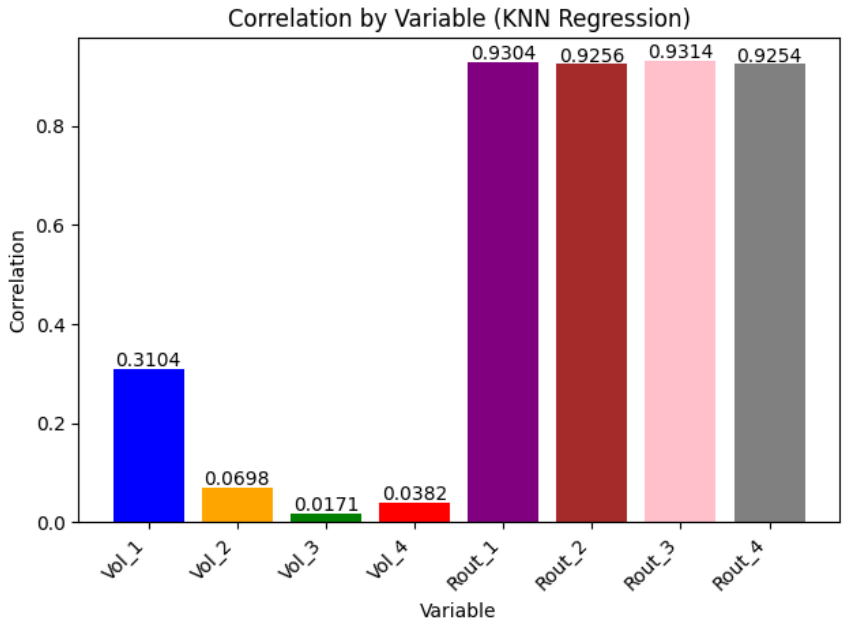


Figure 5.15: Correlation for each variable in the second scenario by *k*NN.

The traffic volumes (Vol_1, Vol_2, Vol_3, Vol_4), like the Random Forest results, have minimal correlations with the departures predicted by the *k*NN model. This could imply that the model in this scenario is not successfully capturing the linkages between vehicle entry and volumes.

As in Random Forest and Neural Networks, the route decisions (Rout_1, Rout_2, Rout_3, Rout_4) present higher correlations with the outputs predicted by the model. This could indicate that the *k*NN model is capturing patterns related to route probabilities based on the input features. Also, the overall correlations appear to be lower compared to the first scenario and also to the Random Forest results.

5.2.4 Summary results

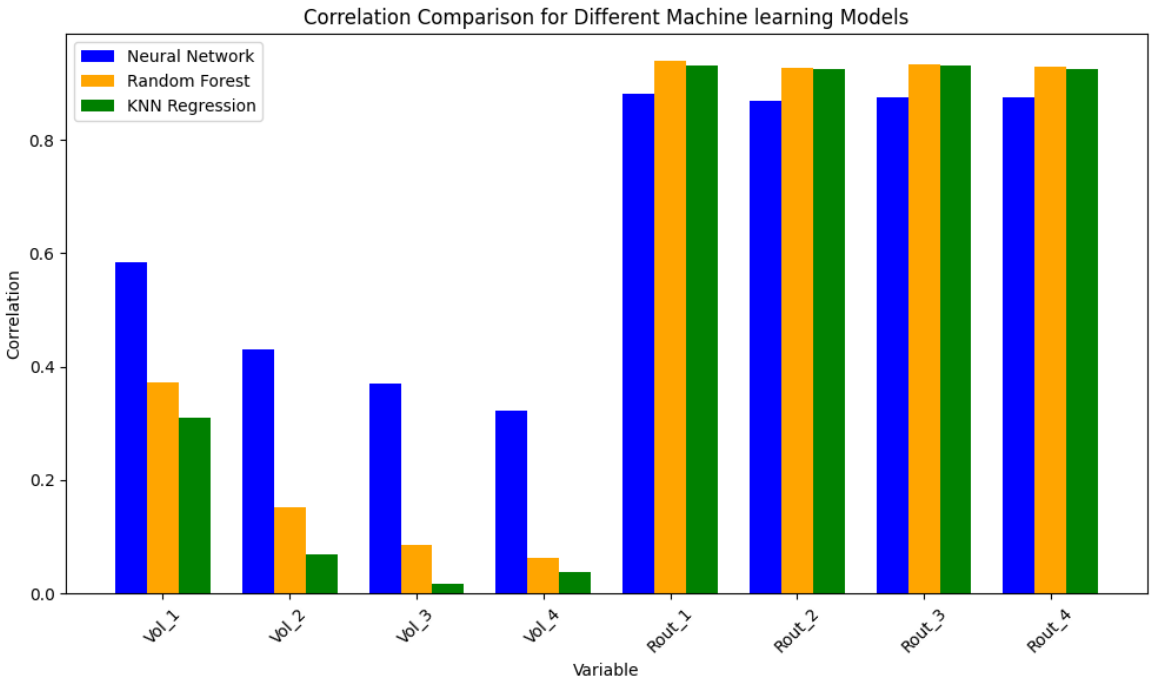


Figure 5.16: Correlation for each variable in the second scenario by each machine learning model.

Compared to the first scenario, the results in the second scenario had weaker correlations. This shows that the linkages between inputs and outputs in this environment may be more complex or less direct. In general, the Neural Network and Random Forest models show stronger correlations than the kNN model. This could imply that the Neural Network and Random Forest are more effective at capturing non-linear relationships and complicated patterns in data.

Even though the correlations are smaller in the second case than in the first, the Neural Network is still the best-performing model in terms of volume prediction. Its reasonably strong correlations imply that it can capture the relationships between inputs and traffic volumes even in more complex contexts.

The results from the second scenario show that all models had difficulty accurately linking inputs to outputs. However, when compared to kNN, the Neural Network and Random Forest models seemed to perform better in this scenario. These differences in performance could be attributed to the variations in the datasets. The first scenario had 12 inputs and 10 outputs, while the second scenario had only 4 inputs and 8 outputs. This meant that the models had to predict more parameters with fewer inputs.

Table 5.5: Summary results correlation and MSE of machine learning models

Machine learning model	**Pearson correlation (r)**	**MSE**
Neural Networks	0.6579	0.5409
Random Forest	0.5504	0.4799
k-Nearest Neighbors	0.5185	0.4398

Finally, according to results in Table 5.5, Neither machine learning model has a Pearson correlation greater than 0.8. As a consequence, following the proposed methodology, we must follow Option 2 presented in Figure 4.4 to perform the calibration.

5.2.5 Option 2: Machine learning model combined with genetic algorithm

Because the results of Scenario 2 were not so good, the traditional way to train machine learning models is suggested. This means that inputs from the traffic simulator will be the inputs to machine learning models, and outputs from the simulator will be the outputs of machine learning models. The objective is to obtain the outputs from the simulator without its execution. Once models have been trained to predict simulator outputs. The machine learning model with the best performance will be chosen as a fitness function in a genetic algorithm defined in Table 4.1 that predicts simulator inputs based on the outputs (vehicle counts).

The setup given below was used to prepare the simulation runs.

- **Number of simulations:** 10.000. The number was chosen empirically. Training and testing are split 80/20, with 20% of the training dataset used for validation.
- **Inputs**
  - 4 volume of vehicles, following a uniform distribution between 0 and 1000 vehicles per hour.
  - 4 route decisions.
- **Simulation duration:** 3600 seconds, chosen empirically.
- **Outputs:** 4 vehicle counts at every fork in the road.

This approach does not require inverting the dataset. Instead, we use the dataset to train machine learning models and generate simulator outputs without executing the simulator. The contrast with the first approach is presented in Figure 5.17. The objective is to find the best machine learning model to be combined with a genetic algorithm as a fitness function.

To train the models it will be done normalization of the inputs (subtraction of the variable's mean and division by its standard deviation) is, therefore, a step that reduces the computational complexity of the iterations. While route decision variables are fractions

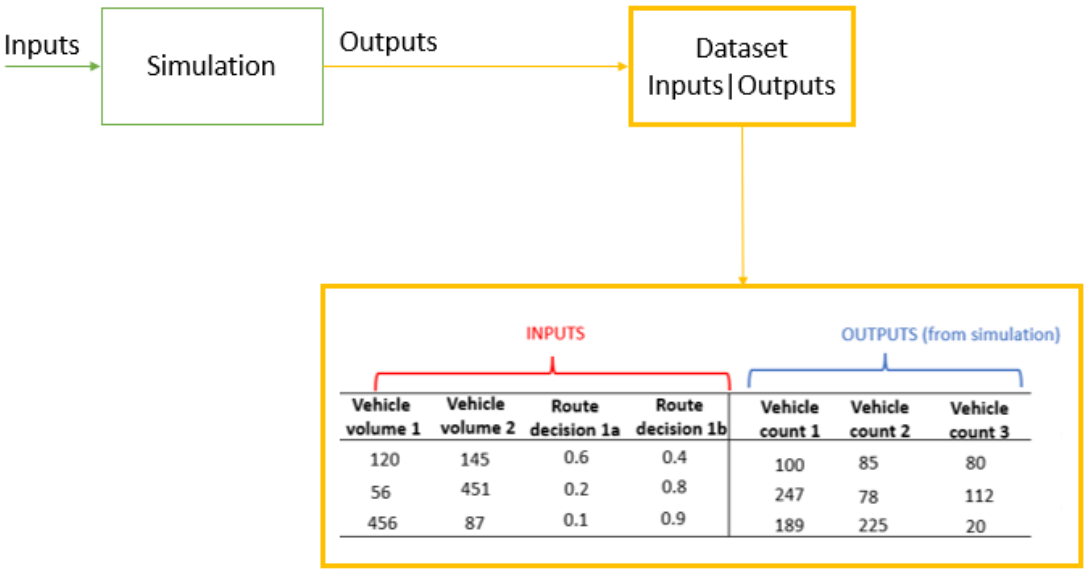


Figure 5.17: Second scenario volumes and route decisions dataset example.

between zero and one, outputs with broader ranges (such as vehicle volume inputs ranging from zero to thousands of vehicles/hour) may bias the training away from fitting the outputs with smaller ranges. As a result, normalizing the output samples is also preferred to maintain the parity of importance of all the outputs in the calibration function modeling.

The Pearson correlation between the model predictions and the desired test outputs is the chosen metric to assess performance in a normalized manner. It is used to determine the correlation.

Neural Network

Based on the results obtained before, we fixed the structure of two hidden layers with 50 neurons each (noted as a 50-50 configuration), and "Nadam" optimizer to train this model.

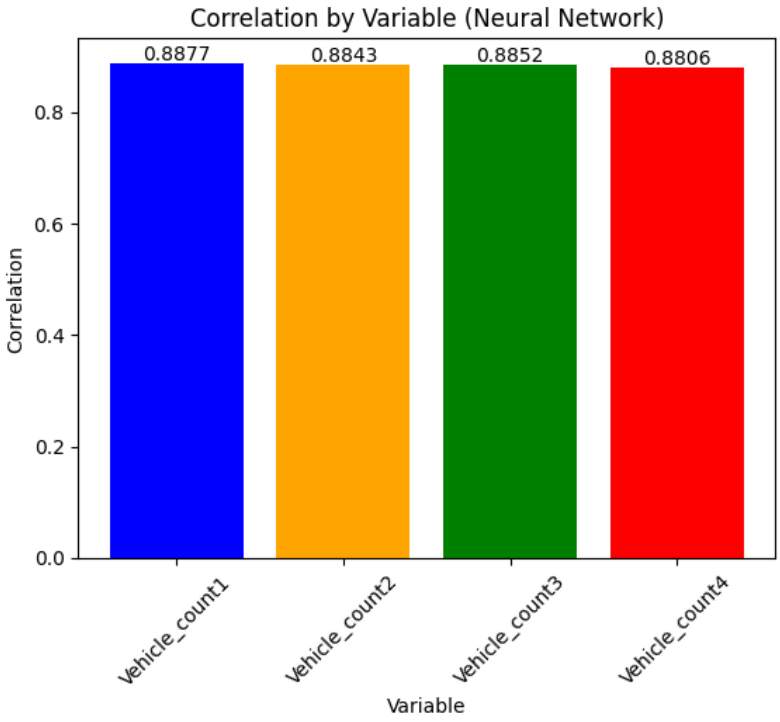


Figure 5.18: Correlation for each variable in the second scenario by Neural Network.

Random Forest

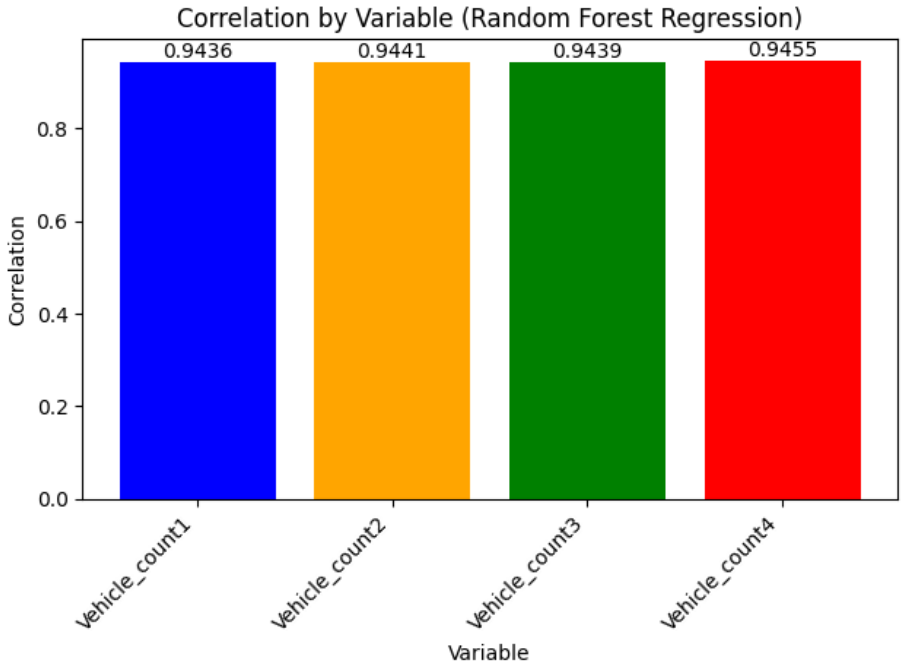


Figure 5.19: Correlation for each variable in the second scenario by Random Forest.

*k*NN regression

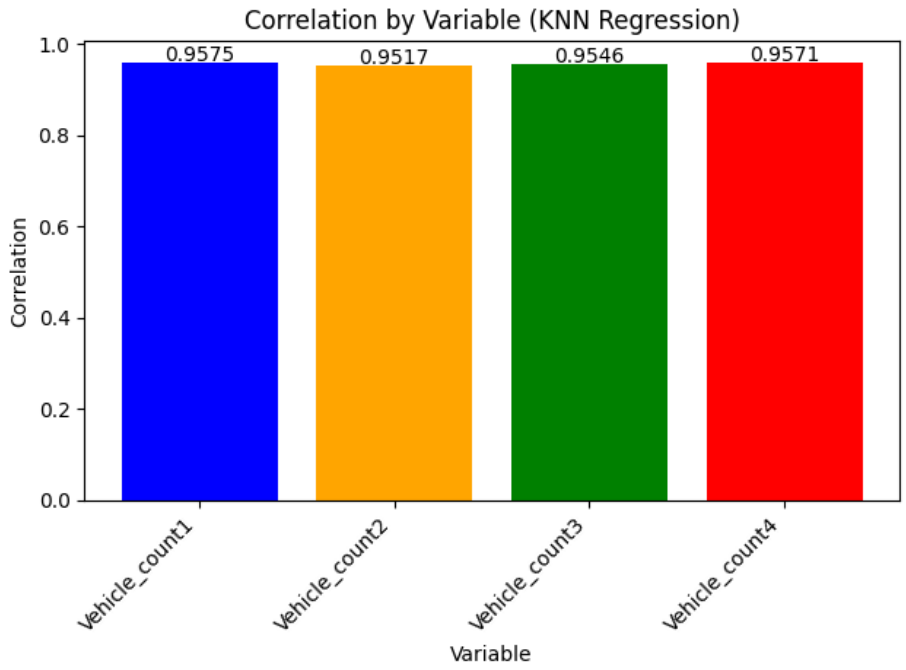


Figure 5.20: Correlation for each variable in the second scenario by *k*NN.

Summary Results

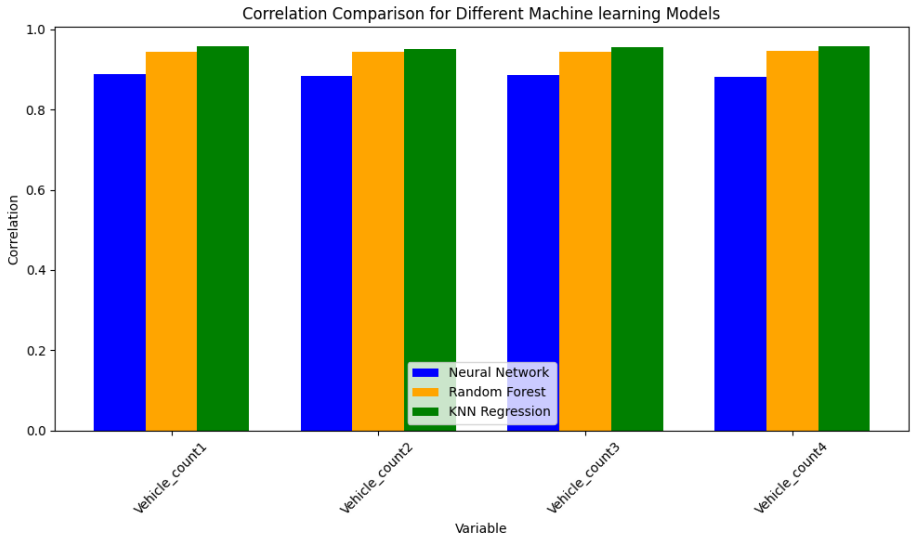


Figure 5.21: Correlation for each variable in the second scenario by each machine learning model.

The analysis of the results obtained for the variables (Vehicle_count1, Vehicle_count2, Vehicle_count3, and Vehicle_count4) reveals interesting patterns in terms of the correlations

established by different machine learning models: Neural Networks, Random Forest, and k-Nearest Neighbors (kNN).

To begin, it is evident that all of the models achieved high correlations for the variables in question. Correlations for Neural Networks range from 0.8806 to 0.8877, for Random Forest from 0.9436 to 0.9455, and for kNN from 0.9517 to 0.9575. These substantial correlations show that the models understood the underlying relationship between vehicle volume and route decisions in order to anticipate counting results. In terms of model comparison, the three models achieve very close correlations with each other in all variables. In terms of model comparison, it is discovered that the three models attain very close correlations with each other in all variables. This suggests that all three models perform similarly in forecasting counting results based on entry volume and routing decisions, implying that the data is informative and representative of each model.

Table 5.6: Summary results correlation and MSE of machine learning models

Machine learning model	**Pearson correlation (r)**	**MSE**
Neural Networks	0.8879	0.2329
Random Forest	0.9434	0.1903
k-Nearest Neighbors	0.9556	0.1038

When we look at the findings presented in Table 5.6, we can see that the correlations for kNN are much greater than the correlations for the other two models (Neural Network and Random Forest). This implies a stronger association and consistency between the input variables (Vehicle_count1, Vehicle_count2, Vehicle_count3, Vehicle_count4) and the kNN outputs. The highest average correlation of kNN (0.9556) demonstrates a strong association between the model's anticipated inputs and outputs, implying that kNN is capable of accurately and consistently capturing the underlying patterns and relationships in the data. This constancy in precision might be traced to the inherent nature of kNN, which makes predictions based on the closeness and similarity of data. Furthermore, while the Neural Network and Random Forest correlations are impressive, kNN outperforms them. Higher correlations with kNN suggest that the model can produce results that are closer to the outputs of the traffic simulator, that is why this model will be selected to be part of the fitness function of the genetic algorithm.

To measure the capacity of the genetic algorithm to find the optimal parameters input in order to obtain the desired output (vehicle count) we performed four experiments. The percentage error was calculated to evaluate the results as is shown in Table 5.7 where column 1 corresponds to the scenario control and column 2 the results obtained by using the best individual given by the genetic algorithm. As a consequence, column 3 has the quality of the simulation using the percentage error.

Table 5.7: Genetic algorithm results

Vehicle count (expected)	**Vehicle count (obtained)**	**Percentage error (%)**
Count_1 = 200	Count_1 = 201	0.5%
Count_2 = 120	Count_2 = 107	10.83%
Count_3 = 95	Count_3 = 103	8.42%
Count_4 = 340	Count_4 = 330	2.94%
Count_1 = 400	Count_1 = 405	1.25%
Count_2 = 100	Count_2 = 125	25%
Count_3 = 300	Count_3 = 294	2%
Count_4 = 23	Count_4 = 41	78.26%
Count_1 = 290	Count_1 = 298	2.76%
Count_2 = 87	Count_2 = 121	38.97%
Count_3 = 149	Count_3 = 141	5.37%
Count_4 = 367	Count_4 = 346	5.73%
Count_1 = 98	Count_1 = 130	32.65%
Count_2 = 475	Count_2 = 465	2.11%
Count_3 = 239	Count_3 = 170	28.87%
Count_4 = 139	Count_4 = 138	0.72%

In general, we can see that the values obtained are pretty near to the expected values in certain cases, while there is a wider disparity in others. The percentage errors in experiment 1 for Count_1 and Count_3 range from 0.5% to 8.42%. This shows that the numbers obtained are quite near to the expected values for these car counters. In experiment 2, there is a percentage error of 78.26% for the vehicle counter Count_4. This indicates a significant discrepancy between the expected value and the value obtained for this case. In experiment 4, the vehicle counter Count_2 has a percentage error of 38.97%. This is also a major discrepancy and might require further review. Overall, these findings provide preliminary insight into how the genetic algorithm works in comparison to expected values.

Chapter 6

Conclusions

The current work has focused on a single essential goal: calibrating a traffic simulator (SUMO) through machine learning in order to provide an efficient and accurate solution for traffic engineers. The acquired results and analysis give useful information to support this goal by demonstrating how alternative models and optimization tactics affect both calibration performance and predictability of input simulator values. This was possible because of the accomplishment of the proposed objectives.

Customized JTRROUTER helped to automate the process for the creation of a dataset and execute a large number of simulations based on the volume of vehicles and route probabilities (route decisions). This was possible thanks to the Traci library that can be implemented in Python and allows a connection with the SUMO traffic simulator. The simulations are carried out without a graphical interface, so their execution time is shorter. Because the processes were carried out by scripts throughout all experiments, one of the validated hypotheses in the proposed approach is that they can be automated in a future software implementation.

The analysis of the literature revealed a trend toward the employment of genetic algorithms for traffic simulator calibration. However, because of their capacity to capture non-linear correlations between variables, fresh contributions have delved into the application of neural networks. As a result, its usage was critical for the creation of this work, and it was supplemented by the use of Random Forest and k-nearest neighbors because each of these models has its own set of strengths and shortcomings. Random Forest is good at maintaining features and dealing with noisy input, but kNN is easy to construct and works well on problems involving local data structures. Using various models enabled these strengths to be harnessed and flaws to be minimized, thereby enhancing project performance.

Patterns and trends have been observed while examining and exploring the results produced by using several machine learning models in two different scenarios, shedding light on the influence of numerous elements on the performance of these models. These discoveries not only lead to a better understanding of the behavior of machine learning algorithms in realistic and complex circumstances, but they also help with the thesis project. Due to their intricacy and regular traffic jams, the two road networks used in the tests were of interest. Heleodoro Ayala Avenue with Mariano Acosta Avenue, and the roundabout at "La Madre". Three machine learning models (Neural Networks, Random Forest, and

kNN) were used to build the calibration function, Neural Network was tested with different optimizers. "Adam" and "Nadam" were the best ones for the two scenarios in terms of the number of epochs needed before early stopping. Random Forest was set in 100 decision trees and kNN with 5 near neighbors.

In the first scenario, with 12 input and 10 output variables, the neural network models established greater correlations than the Random Forest and k-Nearest Neighbors (kNN) models. These findings imply that neural networks are better suited to capturing complicated and non-linear interactions in information-rich, multidimensional data. The increasing number of inputs and outputs may allow models to modify their parameters more precisely, allowing neural networks to adapt to complicated patterns.

In the second situation, where the number of input variables was lowered to 4 and the number of output variables was increased to 8, the overall correlations decreased in all models. This reduction in dimensionality could have resulted in the loss of critical information and the simplification of the data's fundamental linkages. Furthermore, the difference in model performance between the two scenarios may show that the problem's complexity, in terms of the linkages between inputs and outputs, has a direct influence on the effectiveness of each model. Concerning the specific models, it was found that the kNN models demonstrated an unusual behavior by providing high correlations in certain variables, particularly route decisions. This shows that kNN may be a viable option in instances when relationships are more direct and local. Random Forest and neural networks, on the other hand, have proven to be more ideal for more complicated and multidimensional situations due to their ability to model non-linear relationships and capture information at multiple levels. It has been argued that the machine learning model development process produces metrics for individual variables, giving traffic engineers the opportunity to assess if the findings are satisfactory and should be used in their work. The methodology has been confirmed on those grounds. Because of bad results for metrics related to the volume of vehicles in scenario 2, the implementation of a combination of a genetic algorithm with a machine learning model was done. The results obtained from the experiments using the genetic algorithm have significant value in the context of the main purpose of this work: calibrating the SUMO traffic simulator through machine learning. In some cases, the values obtained were remarkably close to the expected values, with absolute errors ranging between 0.5% and 8.42%. This highlights the promise of using genetic algorithms to simplify and speed up the calibration process, freeing traffic engineers from manual and repetitive work.

This extensive examination of various machine learning models in two opposing scenarios demonstrates the necessity of taking into account the dimensionality of the data, the complexity of the problem, and the inherent capabilities of each model when deciding on the best strategy. The results illustrate neural networks' versatility and greater performance in more complicated scenarios, while also identifying areas where kNN and Random Forest shine. This knowledge provides a solid foundation for informed decision-making in the selection and deployment of machine-learning models in real-world scenarios, and it greatly contributes to the growth of knowledge in this research area.

The adoption of machine learning models has significantly improved the repetitious manual calibration procedure traditionally undertaken by traffic engineers. To carry out this procedure, Neural Networks, Random Forest, and kNN algorithms have proven to be an effective and efficient alternative. The combination of genetic algorithms with machine

learning has proved also to be efficient for those scenarios where machine learning models do not work well. Traffic engineers can rely on a more automated and precise process by using these models, which saves time and resources by eliminating much of the manual intervention required in traditional calibration.

This study may be used to complement Yangali [3] earlier research, which aims to develop smart cities in Ecuador. Because a critical component of his study is the simulation of traffic that is comparable to real traffic, the implementation of the approach described in this project would contribute to more exact and realistic outcomes of the surroundings of the smart cities that are intended to be developed.

6.1 Future works

Setting criteria to identify variables that would have a weak connection in the regression model ahead of time is one avenue for future research, and one way is to undertake sensitivity analysis. Another avenue for advancement is to investigate how many measuring entities on the road network (e.g., vehicle counters, time counters, queue counters, and speed measures) are required for the regression models to provide appropriate calibration performance. Finally, because the calibration of this group is also a study topic, a third area of development is the extension of the suggested methodology to macrosimulations.

Bibliography

[1] S. Zhang, J. Li, and Y. Li, "Reachable distance function for knn classification," *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[2] S. B. Thompson, "Simple formulas for standard errors that cluster by both firm and time," *Journal of financial Economics*, vol. 99, no. 1, pp. 1–10, 2011.

[3] Mathematical and computational modeling for the design of smart cities in ecuador. https://repositorio.yachaytech.edu.ec/handle/123456789/496. [Accedido: 20 de septiembre de 2023].

[4] DLR Institute of Transportation Systems. (2023) Sumo traffic simulations documentation. [Online]. Available: https://sumo.dlr.de/docs/Theory/Traffic_Simulations.html

[5] A. Wegener, M. Piórkowski, M. Raya, H. Hellbrück, S. Fischer, and J.-P. Hubaux, "Traci: an interface for coupling road traffic and network simulators," in *Proceedings of the 11th communications and networking simulation symposium*, 2008, pp. 155–163.

[6] S. Raschka, "Stat 479: Machine learning lecture notes (2018)," *URL https://sebastianraschka. com/pdf/lecture-notes/stat479fs18/07_ensembles_notes. pdf. Citado na pág. viii*, vol. 38.

[7] W. Burghout, H. N. Koutsopoulos, and I. Andreasson, "Hybrid mesoscopic–microscopic traffic simulation," *Transportation Research Record*, vol. 1934, no. 1, pp. 218–225, 2005.

[8] I. Vladisavljevic, J. M. Cooper, P. T. Martin, and D. L. Strayer, "Importance of integrating driving and traffic simulations: case study of impact of cell phone drivers on traffic flow," Tech. Rep., 2009.

[9] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "Sumo–simulation of urban mobility: an overview," in *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*. ThinkMind, 2011.

[10] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, "Microscopic traffic simulation using sumo," in *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018. [Online]. Available: https://elib.dlr.de/124092/

[11] Y. Sashank, N. A. Navali, A. Bhanuprakash, B. A. Kumar, and L. Vanajakshi, "Calibration of sumo for indian heterogeneous traffic conditions," in *Recent Advances in Traffic Engineering: Select Proceedings of RATE 2018*. Springer, 2020, pp. 199–214.

[12] R. Balakrishna, C. Antoniou, M. Ben-Akiva, H. N. Koutsopoulos, and Y. Wen, "Calibration of microscopic traffic simulation models: Methods and application," *Transportation Research Record*, vol. 1999, no. 1, pp. 198–207, 2007.

[13] J. Hourdakis, P. G. Michalopoulos, and J. Kottommannil, "Practical procedure for calibrating microscopic traffic simulation models," *Transportation research record*, vol. 1852, no. 1, pp. 130–139, 2003.

[14] T. Alghamdi, S. Mostafi, G. Abdelkader, and K. Elgazzar, "A comparative study on traffic modeling techniques for predicting and simulating traffic behavior," *Future Internet*, vol. 14, no. 10, p. 294, 2022.

[15] F. Malik, H. A. Khattak, and M. A. Shah, "Evaluation of the impact of traffic congestion based on sumo," in *2019 25th International Conference on Automation and Computing (ICAC)*. IEEE, 2019, pp. 1–5.

[16] J. Dargay, D. Gately, and M. Sommer, "Vehicle ownership and income growth, worldwide: 1960-2030," *The energy journal*, vol. 28, no. 4, 2007.

[17] Y. Hollander and R. Liu, "The principles of calibrating traffic microsimulation models," *Transportation*, vol. 35, pp. 347–362, 2008.

[18] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, "Microscopic traffic simulation using sumo," in *2018 21st international conference on intelligent transportation systems (ITSC)*. IEEE, 2018, pp. 2575–2582.

[19] D. Krajzewicz, G. Hertkorn, C. Rössel, and P. Wagner, "Sumo (simulation of urban mobility)-an open-source traffic simulation," in *Proceedings of the 4th middle East Symposium on Simulation and Modelling (MESM20002)*, 2002, pp. 183–187.

[20] D. Krajzewicz, M. Bonert, and P. Wagner, "The open source traffic simulation package sumo," *RoboCup 2006*, 2006.

[21] K. W Axhausen, A. Horni, and K. Nagel, *The multi-agent transport simulation MATSim*. Ubiquity Press, 2016.

[22] A. A. Zafer, "Netedit: A collaborative editor," Ph.D. dissertation, Virginia Tech, 2001.

[23] jtrrouter documentation. https://sumo.dlr.de/docs/jtrrouter.html. Last access: August 17 of 2023.

[24] P. P. Shinde and S. Shah, "A review of machine learning and deep learning applications," in *2018 Fourth international conference on computing communication control and automation (ICCUBEA)*. IEEE, 2018, pp. 1–6.

[25] M. Harth, M. Langer, and K. Bogenberger, "Automated calibration of traffic demand and traffic lights in sumo using real-world observations," in *SUMO Conference Proceedings*, vol. 2, 2021, pp. 133–148.

[26] X. Liu, S. Tian, F. Tao, and W. Yu, "A review of artificial neural networks in the constitutive modeling of composite materials," *Composites Part B: Engineering*, vol. 224, p. 109152, 2021.

[27] S. A. Korai, F. Ranieri, V. Di Lazzaro, M. Papa, and G. Cirillo, "Neurobiological after-effects of low intensity transcranial electric stimulation of the human nervous system: from basic mechanisms to metaplasticity," *Frontiers in Neurology*, vol. 12, p. 587771, 2021.

[28] M. G. Abdolrasol, S. S. Hussain, T. S. Ustun, M. R. Sarker, M. A. Hannan, R. Mohamed, J. A. Ali, S. Mekhilef, and A. Milad, "Artificial neural networks based optimization techniques: A review," *Electronics*, vol. 10, no. 21, p. 2689, 2021.

[29] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2011, pp. 215–223.

[30] A. A. Heidari, H. Faris, S. Mirjalili, I. Aljarah, and M. Mafarja, "Ant lion optimizer: theory, literature review, and application in multi-layer perceptron neural networks," *Nature-Inspired Optimizers: Theories, Literature Reviews and Applications*, pp. 23–46, 2020.

[31] S. Walczak, "Artificial neural networks," in *Advanced methodologies and technologies in artificial intelligence, computer simulation, and human-computer interaction*. IGI global, 2019, pp. 40–53.

[32] S. H. Haji and A. M. Abdulazeez, "Comparison of optimization techniques based on gradient descent algorithm: A review," *PalArch's Journal of Archaeology of Egypt/-Egyptology*, vol. 18, no. 4, pp. 2715–2743, 2021.

[33] P. Netrapalli, "Stochastic gradient descent and its variants in machine learning," *Journal of the Indian Institute of Science*, vol. 99, no. 2, pp. 201–213, 2019.

[34] N. Zhang, D. Lei, and J. Zhao, "An improved adagrad gradient descent optimization algorithm," in *2018 Chinese Automation Congress (CAC)*. IEEE, 2018, pp. 2359–2362.

[35] M. Zeiler, "Adadelta: An adaptive learning rate method. arxiv: 12125701 [cs]. 2012," *arXiv preprint ArXiv:1212.5701*, 2017.

[36] H. Yu, N. Ji, Y. Ren, and C. Yang, "A special event-based k-nearest neighbor model for short-term traffic state prediction," *Ieee Access*, vol. 7, pp. 81 717–81 729, 2019.

[37] D. Yi, J. Ahn, and S. Ji, "An effective optimization method for machine learning based on adam," *Applied Sciences*, vol. 10, no. 3, p. 1073, 2020.

[38] Z. Luo, Y. Chen, and C. Jing, "An enhanced ica based on minimum ber criterion and nesterov-accelerated adaptive moment estimation," *Wireless Personal Communications*, pp. 1–17, 2022.

[39] A. Cutler, D. R. Cutler, and J. R. Stevens, "Random forests," *Ensemble machine learning: Methods and applications*, pp. 157–175, 2012.

[40] M. Bicego, "K-random forests: a k-means style algorithm for random forest clustering," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–8.

[41] X. Dong, Z. Yu, W. Cao, Y. Shi, and Q. Ma, "A survey on ensemble learning," *Frontiers of Computer Science*, vol. 14, pp. 241–258, 2020.

[42] R. Wehrens, H. Putter, and L. M. Buydens, "The bootstrap: a tutorial," *Chemometrics and intelligent laboratory systems*, vol. 54, no. 1, pp. 35–52, 2000.

[43] J. Fox and S. Weisberg, "Bootstrapping regression models," *An R and S-PLUS Companion to Applied Regression: A Web Appendix to the Book. Sage, Thousand Oaks, CA. URL http://cran. r-project. org/doc/contrib/Fox-Companion/appendix-bootstrapping. pdf*, 2002.

[44] K. Kirasich, T. Smith, and B. Sadler, "Random forest vs logistic regression: binary classification for heterogeneous datasets," *SMU Data Science Review*, vol. 1, no. 3, p. 9, 2018.

[45] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, pp. 123–140, 1996.

[46] Y. Song, J. Liang, J. Lu, and X. Zhao, "An efficient instance selection algorithm for k nearest neighbor regression," *Neurocomputing*, vol. 251, pp. 26–34, 2017.

[47] S. Zhang, X. Li, M. Zong, X. Zhu, and D. Cheng, "Learning k for knn classification," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 8, no. 3, pp. 1–19, 2017.

[48] S. Forrest, "Genetic algorithms," *ACM computing surveys (CSUR)*, vol. 28, no. 1, pp. 77–80, 1996.

[49] M. M. Soares and G. E. Vieira, "A new multi-objective optimization method for master production scheduling problems based on genetic algorithm," *The International Journal of Advanced Manufacturing Technology*, vol. 41, pp. 549–567, 2009.

[50] S. Sivanandam, S. Deepa, S. Sivanandam, and S. Deepa, *Genetic algorithms*. Springer, 2008.

[51] S. Mirjalili and S. Mirjalili, "Genetic algorithm," *Evolutionary Algorithms and Neural Networks: Theory and Applications*, pp. 43–55, 2019.

[52] L. J. Eshelman, "Genetic algorithms," in *Evolutionary computation 1*. CRC Press, 2018, pp. 102–118.

[53] L. Chu, H. X. Liu, J.-S. Oh, and W. Recker, "A calibration procedure for microscopic traffic simulation," in *Proceedings of the 2003 IEEE International Conference on Intelligent Transportation Systems*, vol. 2. IEEE, 2003, pp. 1574–1579.

[54] T. J. Cleophas, A. H. Zwinderman, T. J. Cleophas, and A. H. Zwinderman, "Bayesian pearson correlation analysis," *Modern Bayesian statistics in clinical research*, pp. 111–118, 2018.

[55] R. Dowling, A. Skabardonis, J. Halkias, G. McHale, and G. Zammit, "Guidelines for calibration of microsimulation models: framework and applications," *Transportation Research Record*, vol. 1876, no. 1, pp. 1–9, 2004.

[56] R. J. Llanque Ayala, "Procedimento para identificação dos principais parâmetros dos microssimuladores a serem considerados no processo de calibração," 2013.

[57] D. M. Miller, "Developing a procedure to identify parameters for calibration of a vissim model," 2009.

[58] J. Rong, K. Mao, and J. Ma, "Effects of individual differences on driving behavior and traffic flow characteristics," *Transportation research record*, vol. 2248, no. 1, pp. 1–9, 2011.

[59] M. Ben-Akiva, H. N. Koutsopoulos, T. Toledo, Q. Yang, C. F. Choudhury, C. Antoniou, and R. Balakrishna, "Traffic simulation with mitsimlab," *Fundamentals of traffic simulation*, pp. 233–268, 2010.

[60] M. Fellendorf and P. Vortisch, "Microscopic traffic flow simulator vissim," *Fundamentals of traffic simulation*, pp. 63–93, 2010.

[61] T. Tettamanti, A. Csikós, I. Varga, and A. Eleőd, "Iterative calibration of vissim simulator based on genetic algorithm," *Acta Technica Jaurinensis*, vol. 8, no. 2, pp. 145–152, 2015.

[62] E. Borgonovo *et al.*, "Sensitivity analysis," *An Introduction for the Management Scientist International Series in Operations Research and Management Science Cham, Switzerland: Springer*, 2017.

[63] D. Hamby, "A comparison of sensitivity analysis techniques," *Health physics*, vol. 68, no. 2, pp. 195–204, 1995.

[64] M. Mkadziel, "Vehicle emission models and traffic simulators: A review," *Energies*, vol. 16, no. 9, p. 3941, 2023.

[65] B. Ciuffo and C. L. Azevedo, "A sensitivity-analysis-based approach for the calibration of traffic simulation models," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 3, pp. 1298–1309, 2014.

[66] V. Punzo, M. Montanino, and B. Ciuffo, "Do we really need to calibrate all the parameters? variance-based sensitivity analysis to simplify microscopic traffic flow models," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 1, pp. 184–193, 2014.

[67] F. Safranyik, I. Keppler, and A. Bablena, "Dem calibration: a complex optimization problem," in *2017 international conference on control, artificial intelligence, robotics & optimization (ICCAIRO)*. IEEE, 2017, pp. 198–201.

[68] X. Ros-Roca, L. Montero, and J. Barceló, "Notes on using simulation-optimization techniques in traffic simulation," *Transportation Research Procedia*, vol. 27, pp. 881–888, 2017.

[69] H. Wang, M. Zhu, W. Hong, C. Wang, G. Tao, and Y. Wang, "Optimizing signal timing control for large urban traffic networks using an adaptive linear quadratic regulator control strategy," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 1, pp. 333–343, 2020.

[70] M. Jha, G. Gopalan, A. Garms, B. P. Mahanti, T. Toledo, and M. E. Ben-Akiva, "Development and calibration of a large-scale microscopic traffic simulation model," *Transportation Research Record*, vol. 1876, no. 1, pp. 121–131, 2004.

[71] D. Henclewood, W. Suh, M. O. Rodgers, R. Fujimoto, and M. P. Hunter, "A calibration procedure for increasing the accuracy of microscopic traffic simulation models," *Simulation*, vol. 93, no. 1, pp. 35–47, 2017.

[72] K. Aghabayk, M. Sarvi, W. Young, and L. Kautzsch, "A novel methodology for evolutionary calibration of vissim by multi-threading," in *Australasian Transport Research Forum*, vol. 36, no. 1, 2013, pp. 1–15.

[73] F. C. Bethonico, "Calibração de simuladores microscópicos de tráfego através de medidas macroscópicas," Ph.D. dissertation, Universidade de São Paulo, 2016.

[74] O. Kramer and O. Kramer, *Genetic algorithms*. Springer, 2017.

[75] S. Shafiei, Z. Gu, and M. Saberi, "Calibration and validation of a simulation-based dynamic traffic assignment model for a large-scale congested network," *Simulation Modelling Practice and Theory*, vol. 86, pp. 169–186, 2018.

[76] Z.-H. Zhou, *Machine learning*. Springer Nature, 2021.

[77] J. Rzeszótko and S. H. Nguyen, "Machine learning for traffic prediction," *Fundamenta Informaticae*, vol. 119, no. 3-4, pp. 407–420, 2012.

[78] M. Zhou, X. Qu, and X. Li, "A recurrent neural network based microscopic car following model to predict traffic oscillation," *Transportation research part C: emerging technologies*, vol. 84, pp. 245–264, 2017.

[79] J. Tang, Y. Zou, J. Ash, S. Zhang, F. Liu, and Y. Wang, "Travel time estimation using freeway point detector data based on evolving fuzzy neural inference system," *PloS one*, vol. 11, no. 2, p. e0147263, 2016.

[80] C. Chen, H. Xiang, T. Qiu, C. Wang, Y. Zhou, and V. Chang, "A rear-end collision prediction scheme based on deep learning in the internet of vehicles," *Journal of Parallel and Distributed Computing*, vol. 117, pp. 192–204, 2018.

[81] H. Shi, T. Wang, F. Zhong, H. Wang, J. Han, and X. Wang, "A data-driven car-following model based on the random forest," *World Journal of Engineering and Technology*, vol. 9, no. 3, pp. 503–515, 2021.

[82] I. Saadi, A. Mustafa, J. Teller, and M. Cools, "A bi-level random forest based approach for estimating od matrices: Preliminary results from the belgium national household travel survey," *Transportation research procedia*, vol. 25, pp. 2566–2573, 2017.

[83] C. Katrakazas, M. Quddus, and W.-H. Chen, "A simulation study of predicting conflict-prone traffic conditions in real-time," 2017.

[84] H. Park, S. Jung, and C. Oh, "Proactive identification of hazardous traffic conditions caused by reduced visibility using road weather information," Tech. Rep., 2017.

[85] M. Chance Scott, S. Sen Roy, and S. Prasad, "Spatial patterns of off-the-system traffic crashes in miami–dade county, florida, during 2005–2010," *Traffic injury prevention*, vol. 17, no. 7, pp. 729–735, 2016.

[86] A. Iranitalab and A. Khattak, "Comparison of four statistical and machine learning methods for crash severity prediction," *Accident Analysis & Prevention*, vol. 108, pp. 27–36, 2017.

[87] P. Manjunatha, P. Vortisch, and T. V. Mathew, "Methodology for the calibration of vissim in mixed traffic," in *Transportation research board 92nd annual meeting*, vol. 11. Transportation Research Board Washington, DC, United States, 2013.

[88] Q. Ge and M. Menendez, "Sensitivity analysis for calibrating vissim in modeling the zurich network," in *12th Swiss transport research conference*, vol. 5, 2012.

[89] V. Punzo and B. Ciuffo, "How parameters of microscopic traffic flow models relate to traffic dynamics in simulation: Implications for model calibration," *Transportation Research Record*, vol. 2124, no. 1, pp. 249–256, 2009.

[90] M. Bagheri, B. Bartin, and K. Ozbay, "Simulation of vehicles' gap acceptance decision at unsignalized intersections using sumo," *Procedia Computer Science*, vol. 201, pp. 321–329, 2022.

[91] T. Jayasinghe, T. Sivakumar, and A. Kumarge, "Calibration of sumo microscopic simulator for sri lankan traffic conditions," *Proceedings of the Eastern Asia Society for Transportation Studies, Tokyo, Japan*, pp. 12–15, 2021.

[92] M. Rackl and K. J. Hanley, "A methodical calibration procedure for discrete element models," *Powder technology*, vol. 307, pp. 73–83, 2017.

[93] S. Pool, M. Vis, and J. Seibert, "Evaluating model performance: towards a non-parametric variant of the kling-gupta efficiency," *Hydrological Sciences Journal*, vol. 63, no. 13-14, pp. 1941–1953, 2018.

[94] C. Wang, C. Xu, J. Xia, Z. Qian, and L. Lu, "A combined use of microscopic traffic simulation and extreme value methods for traffic safety evaluation," *Transportation Research Part C: Emerging Technologies*, vol. 90, pp. 281–291, 2018.

[95] F. Huang, P. Liu, H. Yu, and W. Wang, "Identifying if vissim simulation model and ssam provide reasonable estimates for field measured traffic conflicts at signalized intersections," *Accident Analysis & Prevention*, vol. 50, pp. 1014–1024, 2013.

[96] N. R. Hoot, L. J. LeBlanc, I. Jones, S. R. Levin, C. Zhou, C. S. Gadd, and D. Aronsky, "Forecasting emergency department crowding: a discrete event simulation," *Annals of emergency medicine*, vol. 52, no. 2, pp. 116–125, 2008.