POLITECNICO DI TORINO

Master Degree in Communications and Computer Networks Engineering

Master Degree Thesis

Edge-assisted Federated Learning for Autonomous Vehicle Trajectory Prediction

Supervisors
Prof.ssa Carla Fabiana Chiasserini
Prof. Claudio Ettore Casetti
Ph.D. Riccardo Rusca
Ph.D. Carlos Mateo Risma Carletti

Candidate
Giuseppe La Bruna

December 2022

Abstract

The evolution of cars has always been driven by the aim of making transport safer for passengers and pedestrians, but also the need to improve the livability of the cities themselves and therefore the driving comfort of all drivers. This has led to the availability of vehicles equipped with a wide set of sensors for the monitoring of internal and external signals. In the near future, the trend is to inevitably increase the number of data collected from the vehicle itself, necessary to provide services as useful and effective as possible. With the born of applications based on the use of neural networks, there is a huge need of a large amount of data and computational power to make their use profitable. The most efficient way to train a neural network dedicated to a vehicular application is to use a server located at the edge of the network, because it can exchange information with vehicles through the mobile network, using low-latency communication. The main problems of this approach are the high computational power required by the training of the neural network and the high amount of data exchanged between vehicles and the server. In order to overcome these problems, a new approach has been envisioned, allowing to exploit a connected network of vehicles sharing their computational power to support the server in the training of a neural network. The work of this thesis focused on the development of a Federated Learning framework to test how a distributed learning approach can help to train a Deep Neural Network (DNN) for trajectory prediction in an urban environment. We leveraged urban mobility traces to select clients to participate in the Federated Learning (FL) process. The proposed framework makes use of three different components: SUMO [1], ms-van3t [2] and Flower framework [3]. The first two components have the role of generating an urban mobility trace file starting from real traffic data. The trace file, describing the behaviour of connected cars under the coverage of a mobile network, results from a simulation of a realistic urban mobility traffic environment made by vehicles communicating with seven LTE eNBs. The third component is a federated learning framework that was properly modified and tuned to leverage the output of the mobility trace file to select at each federated learning round a predefined number of clients reflecting a realistic behavior. Our simulations show that choosing FL clients based on real mobility traces outperforms, in terms of overall simulation

time required to reach a certain accuracy threshold, a simulation with a synthetic scenario where a predefined and less dynamic selection policy for clients is used.

Acknowledgements

A big thanks to prof.Carla Fabiana Chiasserini and prof.Claudio Ettore Casetti for managing this thesis work in a more than professional way and for giving me the possibility to be part of an interesting and challenging work. I thank the PhDs Riccardo Rusca and Carlos Risma Carletti for the continuous support given to me from the beginning of the work until the last day of the submission of the thesis.

A special thanks go to my family, my friends and my Chiara for supporting me in many aspects of everyday life that have allowed the success of this university experience.

Table of Contents

List of Tables

List of Figures

Acronyms

AI Artificial Intelligence

C-V2X Cellular Vehicle-to-Everything

CA Cooperative Awareness

CAM Cooperative Awareness Message

DL Deep Learning

DEN Decentralized Environmental Notification

DENM Decentralized Environmental Notification Message

DSRC Dedicated Short Range Communications

ETSI European Telecommunications Standards Institute

FedAvg Federated Averaging

FedPer Federated Personalization

FedSAE Self-Adaptive Federated Learning

FL Federated Learning

ITS Intelligent Transport Systems

IEEE Institute of Electrical and Electronics Engineers

LSTM Long Short-Term Memory

LTE Long Term Evolution

MED Mean Euclidean Distance

ms-van3t Multi-Stack VANET Framework module developed for ns-3

NN Neural Network

ns-3 network simulator

OBU On Board Unit

RSU Roadside Unit

RNN Recurrent Neural Network

SUMO Simulation of Urban MObility

V2V Vehicle-to-Vehicle

V2I Vehicle-to-Infrastructure

V2P Vehicle-to-Pedestrian

V2N Vehicle-to-Network

VM Virtual Machine

VANET Vehicular Ad-Hoc Network

3GPP 3rd Generation Partnership Project

Chapter 1

Introduction

According to the 2018 World Health Organization report on road safety [4], road accidents account for 1.3 million deaths worldwide. In particular, 43% of these occurred at road crossings. Therefore, it is noticeable that traditional traffic management systems are still ineffective at managing the ever-increasing number of cars, especially within the more developed cities that are increasing their population over time, as reported in [5]. For more than 20 years the research world and companies alike have been in the pursuit of improving the safety and livability of the urban environment by implementing Intelligent Transportation Systems (ITS) [6]. These services combine transportation engineering with the deployment of tons of sensors on-board of vehicles, that are used to collect a huge amount of driving data. This gathered information can be used to create a multitude of helpful machine learning (ML) models to make cars safer, less polluted and improve the driving experience. When we process this data using conventional cloud computing, the real-time service cannot be guaranteed due to the high latency in communication between users and cloud infrastructure. The European Telecommunications Standards Institute (ETSI) tried to solve this problem by defining the Multi-access Edge Computing (MEC) network architecture as the enabler for new vertical real-time services [7] offering an IT service environment and cloud-computing capabilities at the edge of the network. Therefore, the problem of processing driving data, ensuring a low-latency service, can be solved by exploiting the MEC infrastructure for centralized training of ML models. However, two problems, regarding the management of vehicles' collected data, remain unsolved. The first concerns the exchange of a large amount of data between vehicles and the MEC infrastructure, which according to [8] from ETSI, it is a burdensome or unrealistic operation. The second problem is about personal information privacy which could be violated when data, transmitted toward the MEC infrastructure, are sniffed by malicious subjects. The suggested solution from ETSI, in the aforementioned article, is the Federated Learning (FL) approach. A decentralized

ML method that allows training a global ML model using vehicles' computational resources, without the need of uploading driving data from vehicles to the edge network.

The purpose of this thesis aims to build a general framework for Federated Learning in an urban environment, allowing to clearly quantify and comparing the impact of leveraging on real vehicle mobility traces compared to the use of a synthetic policy to select the participants in FL.

The main goal is to build and test an FL framework able to train a trajectory prediction algorithm and quantify the impacts of using different client selection policies.

Federated Learning is used to train a Long Short-Term Memory (LSTM) artificial neural network leveraging vehicles approaching the crossroad. The server at the edge of the network is then used for aggregating the local computational results coming from the vehicles, producing a global model. The implementation of the aforementioned solution exploits three main components:

- *Real mobility traces with ms-van3t framework*
- *Flower framework*
- *Real dataset for LSTM trajectory prediction*

The first component exploits real mobility traces to create a vehicular simulation used to simulate the urban traffic interacting with a mobile network infrastructure. The output file reports the time-referenced interaction between cars and antennas, producing for each timestamp of the simulation and for each vehicle, the remaining dwelling time under the radio coverage of the antennas.

The second component is a, well-known in literature, federated learning framework. During the real-world scenario case, it exploits the previous output file to manage the clients' selection during the whole learning process. While during the synthetic scenario case, a predefined policy is used.

The last component is the artificial neural network model that is trained by the federated clients. The normal FL process is enhanced by introducing two termination criteria, among which the one that stops the process when the accuracy of the algorithm reaches a given threshold, more precisely, when the error between real and predicted trajectory is under a given value.

The accuracy obtained with our framework is comparable with the one training the model in a centralized way [9]. It is noteworthy that, with our solution, we have not loss in model accuracy and at the same time we maintained the training dataset locally in the clients. Privacy preservation and a drastic reduction of the network bandwidth used are two of the main strengths of our solution.

Another significant outcome comes from comparing the results obtained from the real-world scenario with the synthetic one. In the former, the rate of federated client

renewal during an FL cycle is higher than the one in the latter. As a consequence, the algorithm can be trained using a more wide data set in an equivalent time interval, leading to reaching the same accuracy value in significantly less time.

Chapter 2

Background

This chapter describes the state of the art of technologies that connected vehicles and smart cities' environment can use to improve road safety. Starting from the description of modern machine learning methods at the base of some vehicular applications, it is described the IoT-edge-cloud continuum environment that allows a fast and reliable use of these applications. At the end of the section, it is described how trajectory prediction applications can help cities to become safer and more livable.

2.1 Machine Learning

Machine Learning (ML) [10] is a particular branch of computer science and Artificial Intelligence (AI) [11] that includes different mechanisms that allow a machine with embedded electronic logic to improve its capabilities and performance over time, imitating how humans learn things. ML does not use predefined and fixed algorithms but it exploits mathematical methods to improve the machine learning model adaptively, analysing input data. The machine, therefore, will learn to perform specific tasks, improving its responses and functions over time. In short, the machine is no longer programmed but trained. The ML can be done in three main ways:

- ***Supervised learning***
  With this learning method, the system takes as input a data set to test and the corresponding desired output, so that the machine already knows the correct results of the job. The machine's task is indeed to look for a relationship between input and output. This relationship corresponds to the model of the machine that is later used to solve similar tasks. A classic example is the case of image classification in which all the photos in the input are marked with the

corresponding output class so that the model can learn how to autonomously classify unmarked photos received as input.

- ***Unsupervised learning***
  In this second method, the system receives only the input data set without any reference to the desired output. The aim is to let the system discover some hidden patterns in the data so that it can build a classification criterion based on the characteristics of the input data, without the influence of human classification.

- ***Reinforcement learning***
  In this case, the system receives data dynamically from external sensors (e.g. GPS, LIDAR or proximity sensors for cars). The task of the system is to arrive at a predetermined goal. If it succeeds then receives a reward and records the successful action to reinforce its model. In the other hand, he takes an action that leads him to an incorrect goal, he is penalized and therefore will discard the action just made, leaving the model unchanged.

One of the most known sub-categories of ML methods is the NEURAL NETWORK (NN). NN uses mathematical models made by artificial neurons inspired by the structure and logic of biological neural networks. It is an adaptive system because it can modify its interconnection structure between the various neurons based on the data processed over time. Generally, the structure of an NN is composed of three layers, as shown in Fig. 2.1. The first is the input one in which the incoming data is adapted to the demands of the neurons of the network. The middle layer is called "hidden" and is where the actual processing takes place. The last layer is the output layer, with a number of neurons equal to the number of output classes we want, in which the results elaborated by the model are collected.

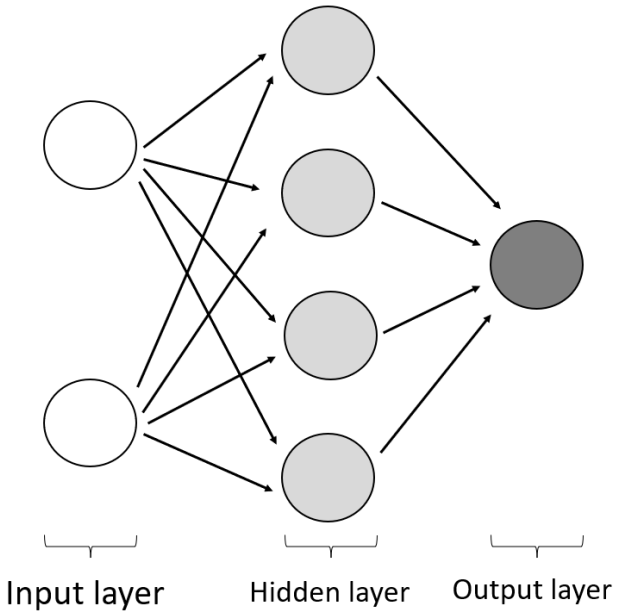

Figure 2.1: Base structure of a neural network.

From the output of an artificial neuron, can be identified two steps. Following Fig. 2.2, the first step is the weighted sum of the inputs, where the weight $\boldsymbol{w_j}$ is represented by the value of the connection between the neuron that performs the calculation and the one from which the input $\boldsymbol{i_j}$ derives. The second step is the generation of the final output by the activation function $\boldsymbol{f}$ which is a function of the output of the first step, $\boldsymbol{z}$.

$$O = f(\sum_{j=1}^{n} i_j * w_i)$$

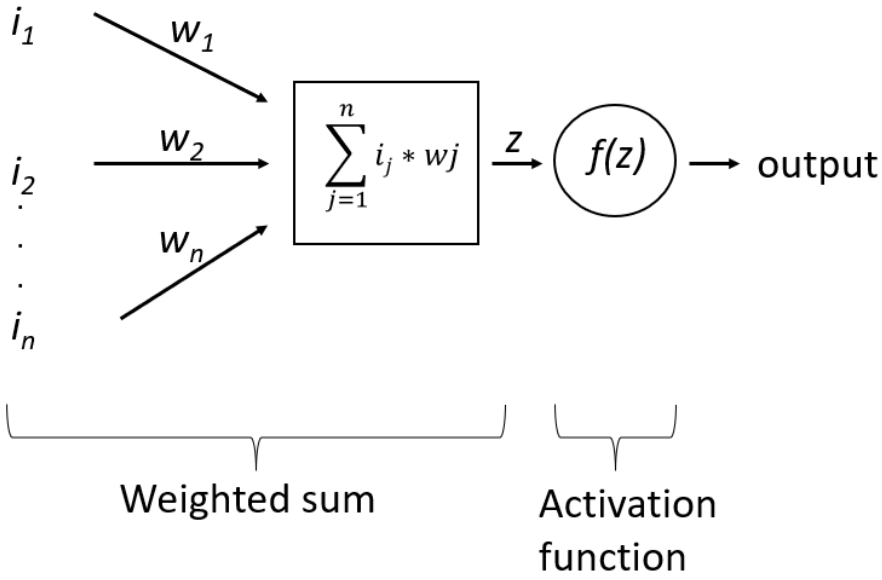

Figure 2.2: Base structure of an artificial neuron.

2.1.1 Deep Learning

Deep Learning (DL) [12] is a sub-class of Machine Learning that creates multi-level learning models, it uses neural networks called Deep Neural Networks (DNN) that have the same skeleton of NN with input, hidden and output layers, but DNNs have a huge amount of hidden layers with respect to NN. The more hidden layers there are, the deeper the network is defined. This allows the Deep Learning system to autonomously identify distinctive features of the data without external human categorizations, distinguishing some categories from others. So it can use direct images, texts or other unstructured data in its raw form. This is a significant difference from Machine Learning, which needs a data set with specific features highlighted by humans. The DL can extract a hierarchy of data based on a categorization logic created by itself, allowing it to select the most relevant data and improve itself continuously. The best-known self-learning method for NNs is the back-propagation one. This method is used to dynamically change the weights of the links between neurons to help improve the model during training. The weights are changed if the current processing was unsuccessful. So when there is another output than the expected one, the error is back-propagated to proportionally change the weights of the network. By repeatedly using back-propagation, the error will change until the desired output is achieved. One approach to figuring out the

correct direction toward the desired output is the one of the descending gradient, which is an iterative method that can be used to minimize the loss of function. Generally, the parameters of the function are modified to find its global minimum. In the case of NN, the weights of the connections between neurons are changed so that the output of the neuron, that is performing the calculation, cancels the error and therefore provides an output as similar as possible to the one expected.

2.1.2 Federated Learning

Federated Learning (FL) [13] is a decentralized ML method that uses the data present on different clients' devices to train a common model. It differs from distributed ML in which a central server divides the data set, used for the training phase, among other servers, thus distributing the workload. Moreover, in distributed ML, the designers of the architecture can manually allocate a portion of the main data set to different servers to distribute the work. Indeed, in FL, the central server cannot access clients' data. This difference in the two learning models usually leads to a non-Independent and Identically Distributed (non-IID) data set in the distributed approach and IID data set in the FL one. There are two types of FL, centralized and decentralized. The first uses a central entity that coordinates the training, aggregating the parameters of the local models trained by each federated client. The second leverage only direct communication between participating clients to coordinate training. Using the FL, the server does not receive the raw data from the clients participating in the training. The clients perform the training of a local model, and then send the results to the server. These results are called "parameters" and represent the weights of the trained local NN. Once the server receives the parameters from the clients, it aggregates them through a strategy decided in advance and creates an updated global model that it sends back to the clients. From here the cycle just described starts again with the local training, performed by the clients, starting from the new global model just evaluated. This cycle is shown in Fig. 2.3. In the beginning, when there is no global model and there are no local models, the server initializes a model through external parameters or parameters of a client chosen randomly among those participating in the training. Once the global starting model is obtained, this is sent to the participating clients, chosen according to a certain policy, so that the FL can start.

FL uses "indirectly" the raw data of the clients because the data set of individual clients never leave the device, so it is never shared with the server. This is one of the strengths of federated training compared to the common centralized ML because by doing so, the privacy of the individual device is protected since no one can have access to its data. The security of this approach is increased by preventing someone from sniffing critical data when they are sent toward the server or by an untrusted server. Another important feature is that, compared to the

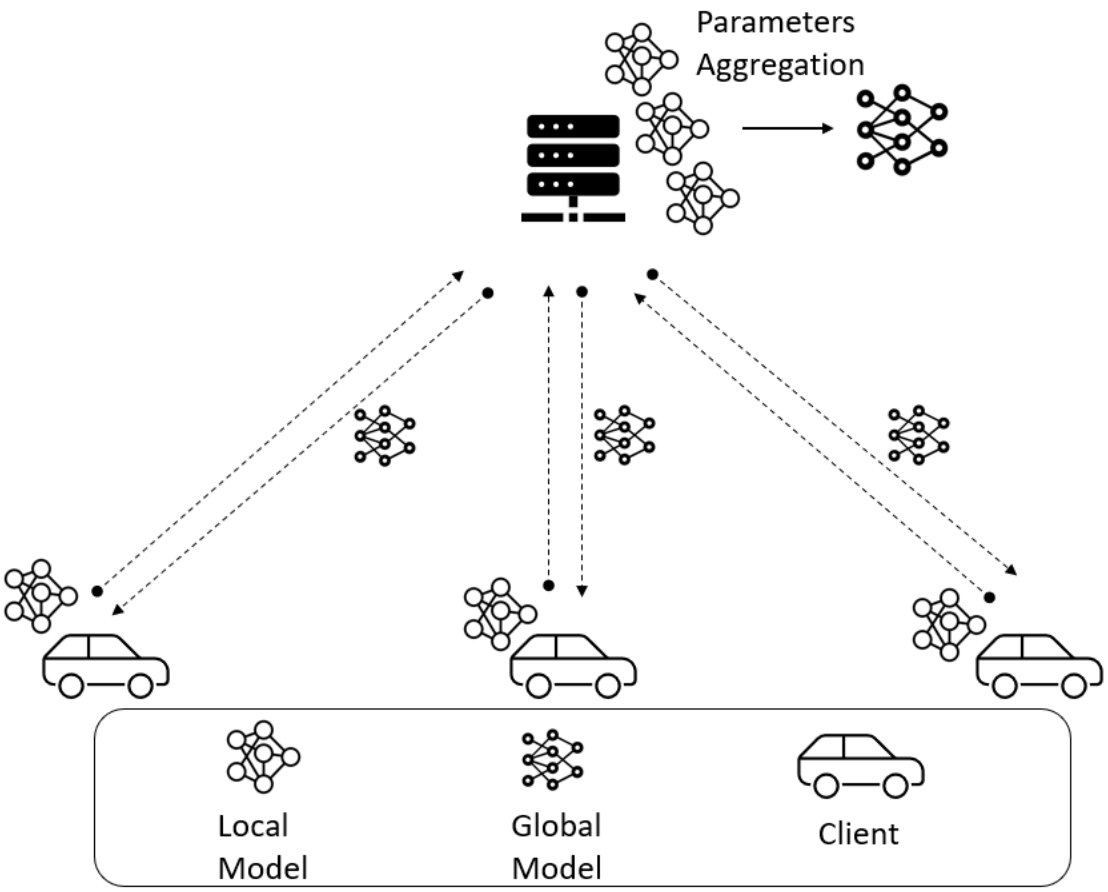

Figure 2.3: Federated Learning mechanism.

distributed machine learning approach, where a large data set is divided across multiple servers into IID subsections, in the FL each device uses its non-IID data set, allowing the use of heterogeneous data for the training of the global model. The client selection policy and the parameter aggregation algorithms are important settings in the FL approach because they can impact the training time and quality of the models trained. Different sampling and aggregation methods were proposed in the literature to solve some of the known problems of the FL approach, below some of those are reported:

Federated Learning strategies

- ***Federated Averaging***
  Federated Averaging [14] (FedAvg) is the most popular optimization scheme in which the clients use the gradient descendent algorithm to improve its model parameters during the local training. More precisely, each client takes N steps

of the gradient descent algorithm on the local model using its data set. Then the server takes a weighted average of the received model parameters, using as weights the sizes of the client data sets. The sampling method is very simple, the server selects the first N responded devices. There is the possibility to select a maximum and a minimum number of participants. If the minimum number of active clients is not reached, the FL cannot start. As shown in [14], this method is computationally efficient due to its simplicity but requires a huge number of training rounds to reach a good accuracy.

- ***Federated Personalization***
  This method, called also FedPer [15], try to collect the personalization aspect of an FL user setting. The principle of this aggregation method is that the model is split in two different layers called "***base***" and "***personalized***" layers. The parameters of the "Personalized" layer are not sent to the server and are trained locally. Only the base layer parameters are sent and aggregated in the federated server as in the FedAvg approach. This method works with Convolutional Neural Network (CNN) and Long Short Term Memory (LSTM) models that are multi-layers neural networks. This approach helps to solve the problem of statistical heterogeneity which is a key aspect of the FL approach. It occurs when in an FL work participate users with different configurations as connection quality and computational power. This inequality is called the statistical heterogeneity problem. With FedPer, the user has the possibility to partially personalize its model according to its setup. Different from the FedAvg approach in which a user, with a very different setup with respect to the other participants, is likely to modify its model according to the characteristics of the majority of users, making the FL useless for him.

- ***FEdCS***
  Federated Learning is mostly used with heterogeneous IoT devices having different computational resources and connectivity conditions. This leads to a longer upload time or longer computational time because the FL round ends when the less performant client ends. The FedCS protocol [16] manages the clients' selection actively, considering their resource state as the choice criterion. In particular, the method sets a resource constraint allowing the client to download, train and upload the model within a certain time frame. This leads to reducing the required time for the training phase and for the overall FL process.

- ***Self-Adaptive Federated Learning***
  One of the known problems of FL is the systems' heterogeneity among the participating devices. This can lead to the need for more resources for some devices to accomplish the same task as other participants. Due to different

training settings, some clients can partially train their models, decreasing the accuracy of the global model. The self-adaptive FL [17] tries to solve this problem by exploiting the server side of the computation. Indeed, the server adapts the workloads assigned to the clients in line with their training history. [17] shows that the self-adaptive approach, in a heterogeneous environment, increases the model accuracy by 26.7% with respect to the FedAvg method.

Federated Learning frameworks

In order to test the results of distributed ML (Federated Learning), it is common to use a platform, as the several described in this paper [18], that simplifies the basic configuration for the client-server communication during the exchange of the partial and global NN models. Below are described two of the most famous frameworks, Flower and IBM federated learning.

- ***Flower***
  Flower [3] is a framework that allows an easy implementation of Federated Learning. It was developed at the University of Oxford from a research project and it has a big community that allows excellent support and documentation for the free self-customization of the framework. Indeed, the user can modify and use the several built-in strategies freely available in Flower. Other characteristics make Flower a noteworthy FL platform. It supports different hardware, operating systems and programming languages. It can support up to 15 million parallel clients and can be deployed on boards like Raspberry or NVIDIA Jetson or in mobile devices with ARM architecture. Flower supports different libraries as Keras [19], TensorFlow [20], PyTorch [21], and scikit-learn [22]. The main structure of this framework is represented by server and client sections. The client section is simpler, it contains the ML model and the training operation.

  The server side contains the file that encompasses the skeleton for the implementation of several functions to manage the clients and the Federated Learning work in general, establishing the strategy used to choose the clients, the maximum and the minimum number of rounds and other important settings.

  The communication between the server and clients is performed through the use of Google Remote Procedure Calls (gRPC) [23], i.e., a framework that allows direct communication between server and client applications located on different machines, allowing easy development of distributed applications.

  The two main functions that ensure the basic functionality of Flower are:

  - ***fit***: it performs the training and validation of the client NN model, exploiting the client's data set
  - ***evaluate***: it computes the difference between the results predicted by the NN model and those taken from the evaluation data set, obtaining the accuracy of the NN model reached after the fit step

- ***IBM Federated Learning***
  IBM federated learning [24] is an enterprise Python framework for FL that allows each party (client) to communicate with the other participants using a learning protocol. As in other frameworks, the aggregation of the parties' parameters is managed by a central entity called an aggregator (server). IBM FL can use any ML framework and supports different learning protocols and topologies and users can create new FL algorithms through the use of APIs. It includes the use of libraries as SK Learn [22], TensorFlow [20], PyTorch [21], RLLib [25] and Keras [19]. IBM FL is organized as a folder containing different modules that the programmer can insert or remove without undermining the proper functioning of others modules. This folder can be managed through a configuration file (.yml file) for the parties and the aggregator. Some of the modules that can be configured are:

  - ***Connection***: includes the types of connections for communication between the parties and the aggregator.
  - ***Local_training***: it includes information to initialize the training class of the parties.
  - ***Model***: it defines the ML model that parties have to train.
  - ***Protocol_handler***: it includes the protocol for message exchange.

2.1.3 LSTM Model

Long Short-Term Memory is a Recurrent Neural Network (RNN). The latter is a NN family specialized in the sequential data processing. The main characteristic of this NN family is the ability to learn from long-term data sequences and save the assimilated information in its memory cell. The key element that allows this function is the feedback "v" in each neuron, as Fig. 2.4 shows. It allows the output to partially go back into the neuron. In this way, the output ***O(t+1)***, depends on the input ***X(t+1)*** and on the feedback ***V***. Furthermore, ***V*** depends on the previous output ***O(t)*** which is function of ***X(t)***. This means that the output ***O(t+1)*** is influenced by the previous input ***X(t)***. This simple cycle in each neuron of the RNN allows for saving the features of the past inputs, creating a reach environment ideal for correctly elaborating long data sequences.

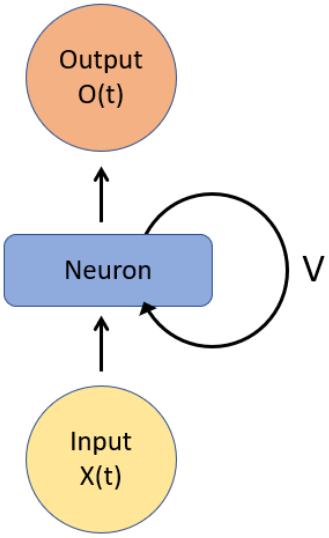

Figure 2.4: RNN neuron with feedback.

The LSTM can be considered an improved RNN because it solves two big problems, described below, of the DNN that uses the "Stochastic gradient descent" in the back-propagation. The gradient value of the activation function of each neuron can be non-linear, in this case, the gradient's value is in an interval between 0 and 1. With the back-propagation, the value of the final gradient is equal to the multiplication of all the gradients (the quantity is equal to the number of layers of the network). Given that a DNN has several hidden layers, the effect of multiplying lots of elements with values in the range (0,1) among them, is that the final result seems to "vanish", so, it is like the gradient disappears, this issue is called "Vanishing gradient problem". The opposite problem happens when the activation functions of the neurons are linear, which means that the value of the gradient is bigger than 1, leading to a gradient value that increases exponentially. This problem is called the "Exploding gradient problem". The LSTM solves these problems thanks to its complex neuron made by different gates that can decide what to save or what to discard and if is convenient to combine the input with the saved internal state.

The encoder-decoder LSTM type is a model designed for "sequence-to-sequence" mapping. The scenario that most needs to map a variable length input sequence toward an output sequence is the forecasting one because needs to exploit past collected data for the output prediction. The LSTM NN is made by two RNNs that work as a decoder-encoder pair. Fig. 2.5 represents, in a very simplified way, the key element of the LSTM model. It shows the LSTM cell, which is divided into 3 main gate vectors that control the input and output flow of information.

Inside this gate vector, there are sigmoid and tanh non-linear activation functions that allow the proper manipulation of the data. Specifically, with the input gate vector, the LSTM cell decides which values in the cell will be updated, with the forget gate vector the LSTM cell decides what information should be thrown away from or kept in the cell state memory, and with the output gate vector, it decides whether to make the output information available.

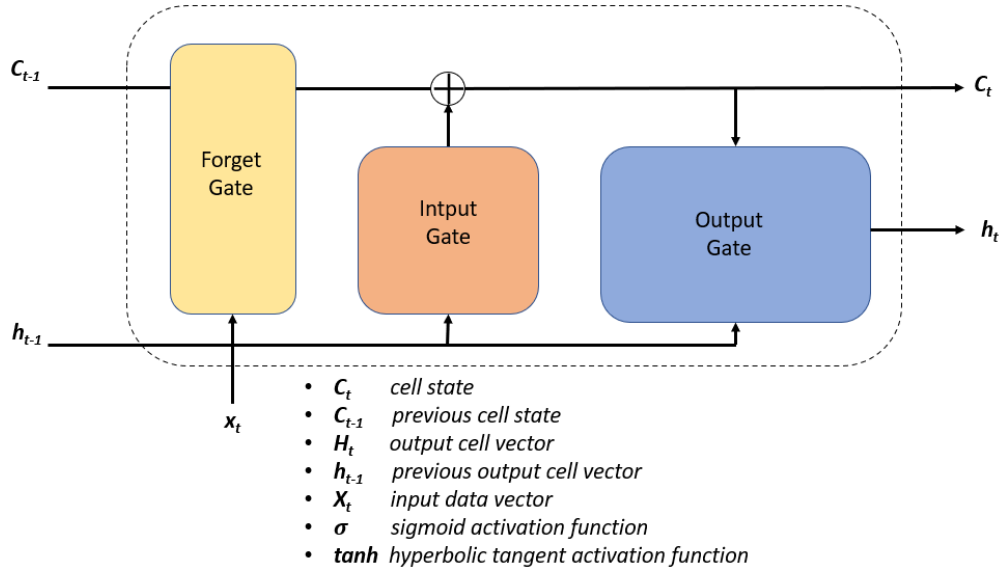

Figure 2.5: LSTM cell.

The work of the LSTM cell starts from the forget gate, Fig. 2.6, that combines the previous hidden state $\boldsymbol{h_{t-1}}$ and the current input $\boldsymbol{x_t}$, passing them through a sigmoid activation function.

The next step is made by the input gate, Fig. 2.7, that combines the previous hidden state $\boldsymbol{h_{t-1}}$ and current input $\boldsymbol{x_t}$ into a sigmoid function and multiplies this result with the output of a tanh activation function whose input is the previous hidden state $\boldsymbol{h_{t-1}}$ and current input.

Adding this last result with the one of the forget gate, the cell state $\boldsymbol{c_t}$ is obtained.

The cell state $\boldsymbol{c_t}$ acts as the cell memory, storing a summary of the past input sequence. It is combined in the output gate, as represented in Fig. 2.8, with the previous hidden state $\boldsymbol{h_{t-1}}$ and the input received $\boldsymbol{x_t}$, so as to obtain the hidden state $\boldsymbol{h_t}$ that represents the output vector of the current LSTM cell and the input of the next one.

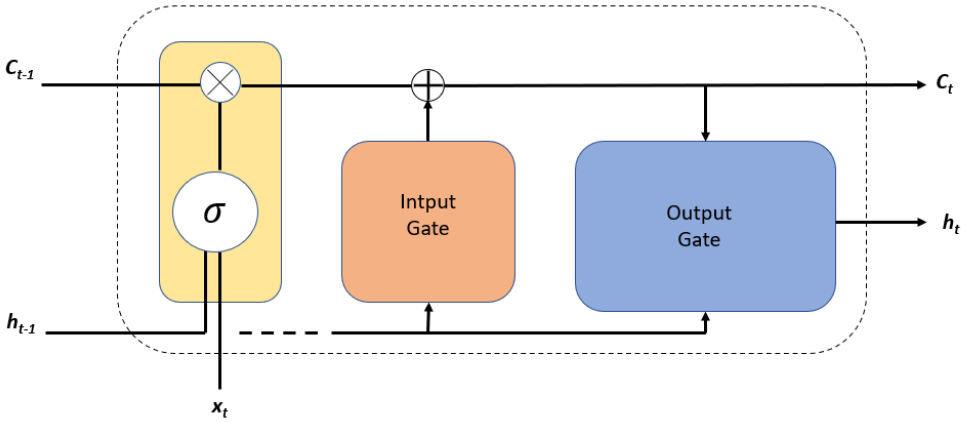

Figure 2.6: LSTM cell with focus on forget gate work.

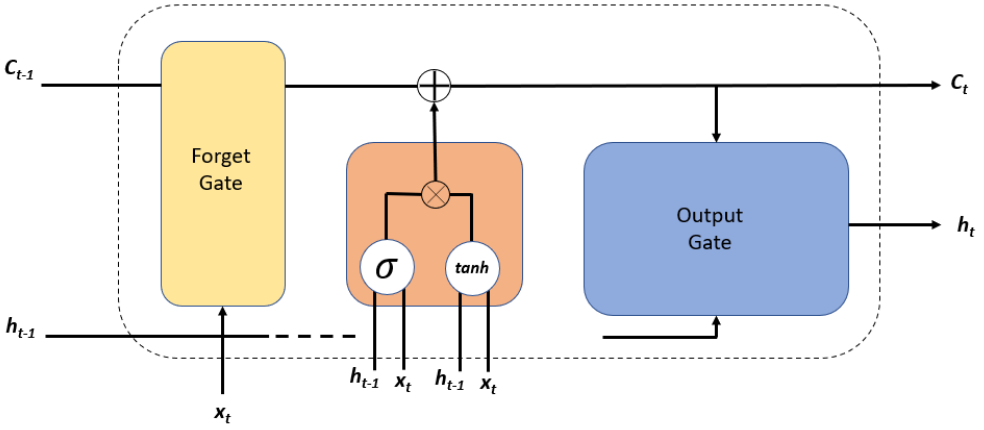

Figure 2.7: LSTM cell with focus on input gate work.

The cooperation of these three gate vectors allows the LSTM model to avoid the vanishing gradient problem, for example, deleting non-important information that can incorrectly modify the value of the gradient.

The LSTM cell is used both in the encoder and decoder group, as shown in Fig.2.9. The encoder reads sequentially the input vector $\boldsymbol{x}$ and summarizes it into the final vectors $\boldsymbol{c_t}$ and $\boldsymbol{h_t}$, what in the scheme is called Encoder state. This mapping between the input sequence and the Encoder state, allows the decoder to be independent of the length of the input sequence, given that it reads just two values every time. Indeed, the decoder starts from the final state of the encoder, which has a fixed dimension, and generates an output trajectory $\boldsymbol{y}$.

This LSTM model can be efficiently used for trajectory prediction applications,

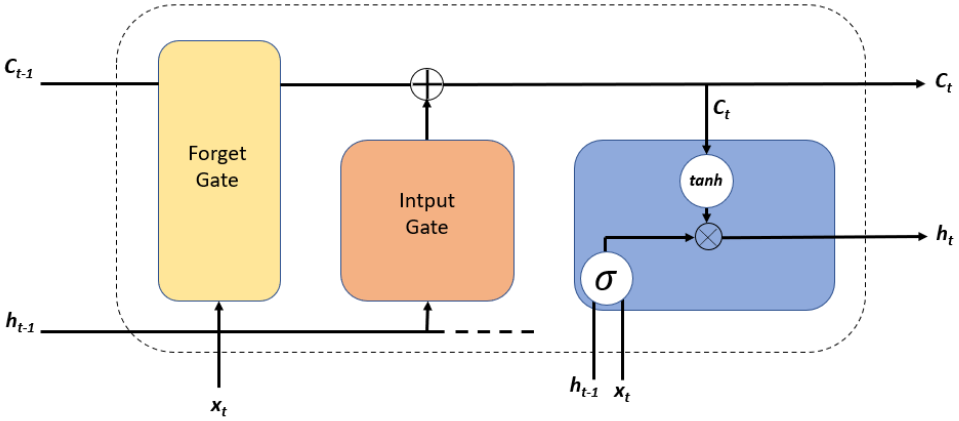

Figure 2.8: LSTM cell with focus on output gate work.

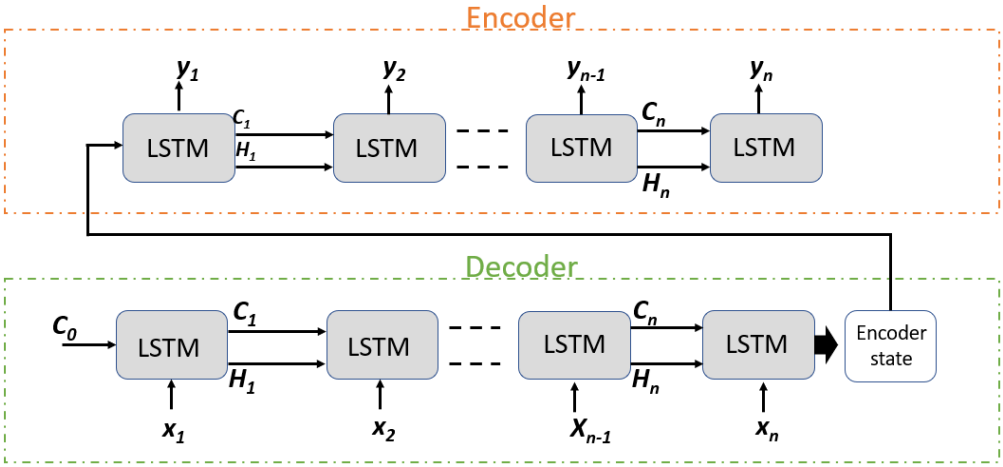

Figure 2.9: LSTM Encoder-Decoder.

which are time-series forecasting problems because using sequential data of past car trajectories, the NN can obtain predictions of the futures trajectories.

2.2 IoT-edge-cloud Continuum

In the past, when the data-driven approach for business and research became a trend, the main approach for data analysis and storage was the cloud one. All data was sent to the cloud infrastructure, for this reason, cloud computing has been the dominant computing paradigm. Today the IoT global market is huge and will increase more and more in the following years, pushed by the concept of Smart Cities that includes a huge quantity of IoT devices. As a direct consequence, the volume of data produced every day will increase as well, requiring a computing paradigm capable of monitoring and analyzing data flows, often with ML (DNN) algorithms, close to their sources. Technically, the need for computational resources close to devices comes from the need for low latency (difficult to achieve with conventional cloud computing, for high amounts of data) for real-time processing. For example, some users' applications have to take quick decisions based on image processing results. In this example, if the elaboration service for these users' applications is in the cloud, the first problem would be due to timing. Since cloud servers are usually far from the users' applications that use them, the exchange of data between the cloud service and the user would suffer a delay that could compromise the correct functioning of the application itself. In the case of the aforementioned application, the decision taken, given the processing of the images, could become useless if the delay to reach the destination exceeds a certain threshold. The second problem of data elaboration in the cloud for real-time applications is that if all current and future services will be developed in the cloud, the network could suffer from congestion due to the data exchange from and toward the user and thus compromise the functionality of many applications. To meet this necessity, the edge computing paradigm was developed to offer computing and storage resources close to IoT devices and end-users. In short, edge computing is the perfect support infrastructure for bandwidth-hungry and delay-sensitive applications. However, the edge infrastructure that is usually located in the cities cannot offer the same amount of resources as cloud infrastructure, which leads to less scalability and less storage capacity. For these reasons, the best future infrastructure to support an end-user environment that needs both lots of storage space and computation resources is made by the combination of cloud, edge and IoT technologies, creating the IoT-edge-cloud continuum [26] represented in Fig. 2.10. Applications that use this complete infrastructure are, for example, those that are made by one or more IoT devices that collect data needing to be processed quickly. The result of the elaboration, together with some relevant data, must be saved every day. In this situation, edge computing can elaborate the data instantaneously and send the results both to the IoT devices and to the dedicated cloud storage. As the aforementioned paper reports, there are different end-edge-cloud computing paradigms in the literature. The most considered by the research community is the Multi-access edge

computing (MEC) paradigm, standardized by the European Telecommunication Standards Institute (ETSI). As the results of [27] [28] reports, MEC can be the key to the development of critical automotive applications. Indeed, the Smart Mobility industry is one of the main IoT market players and it requires strict real-time constraints computation to run as close to the data origin as possible. One of the advanced characteristics of the MEC architecture is programmability. The MEC paradigm helps the transformation of the mobile broadband network from the old static concept to a programmable one. The key to programmability in the MEC architecture is the virtualization of the applications that allow adapting of the local server to the request of the users which can change dynamically and very frequently. This adaptability avoids also the necessity of changing the infrastructure when the market change, and can be sufficient to change just the virtualized application, well accompanying the development of 5G technology. Another important point is that the MEC servers can be allocated in different locations of the Radio Access Network (RAN) such as the 3G radio network controller site, the LTE base station or the site of a large corporate or hospital. The proximity of MEC servers to the user guarantees fast service deployment.

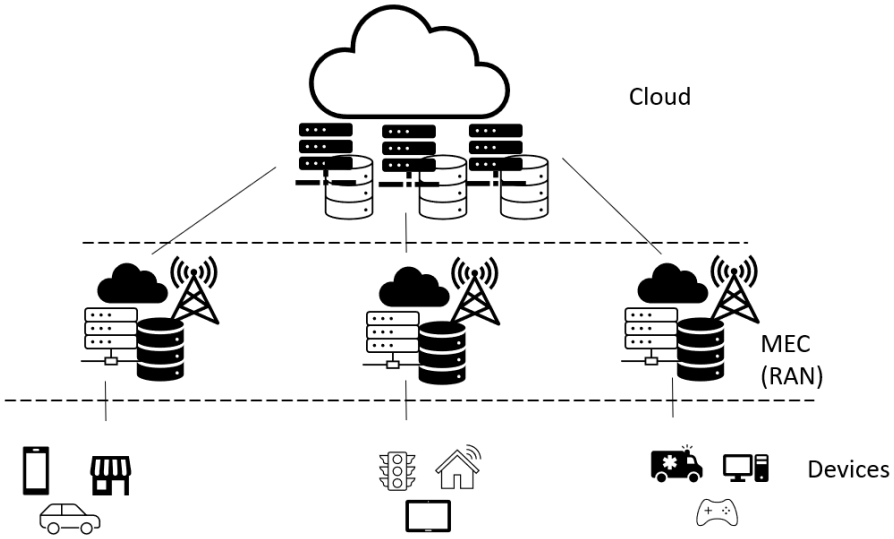

Figure 2.10: IoT-edge-cloud continuum.

2.2.1 Container-based virtualization

With the race toward the virtualization of resources in the IT field, the use of the Virtual Machine (VM) has become a big trend. Nowadays, it is becoming increasingly present the concept of a container, as clearly described in [29], used for the same purposes as a VM but with more flexibility. The VM is a hypervisor-based virtualization technology, which means that the VM can run any operating system (OS), different from the guest OS if desired because the hypervisor emulates the underneath hardware and the VM has its own kernel. Instead, the containers are the result of operating system virtualization, meaning they use the same kernel and operating system of the host machine, as shown in Fig. 2.11. This allows containers to act as very lightweight and fast virtualized machines. Indeed, being small in terms of memory and software resources, starting, updating or installing a container in another server or PC is a very easy and fast operation with respect to the same operations made with the classical VM.

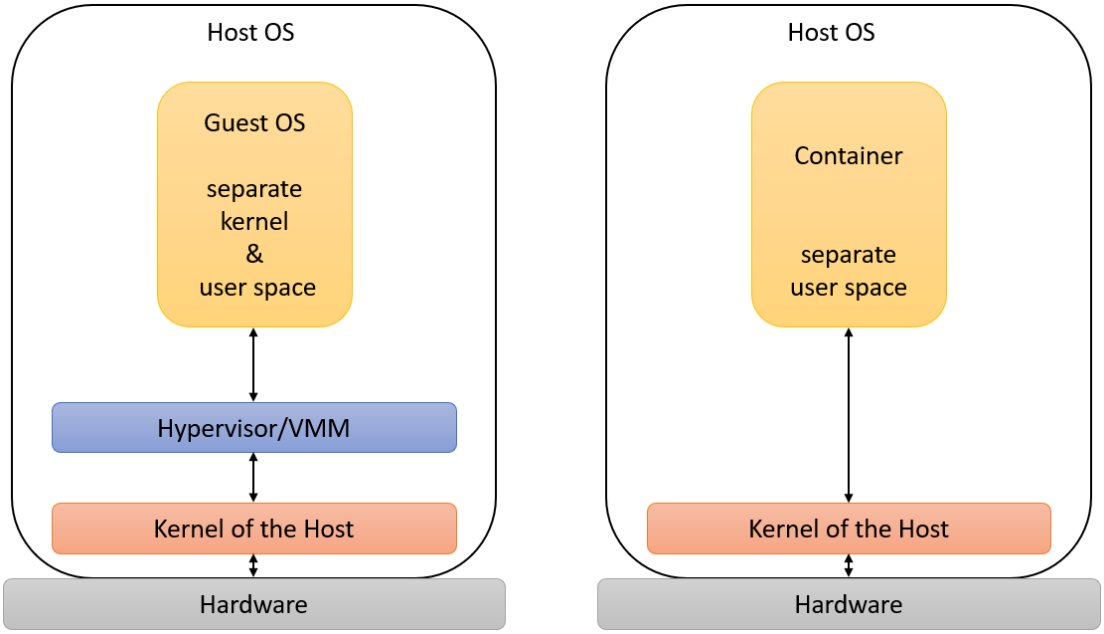

Figure 2.11: VM vs Container.

There are two families of containers, the OS container shown in Fig. 2.12 and the application container represented in Fig. 2.13. The former is a virtualized environment with a concept similar to a VM, ideal to execute multiple applications. The latter exploits the layerization concept of the application in different containers, each one running a single service.

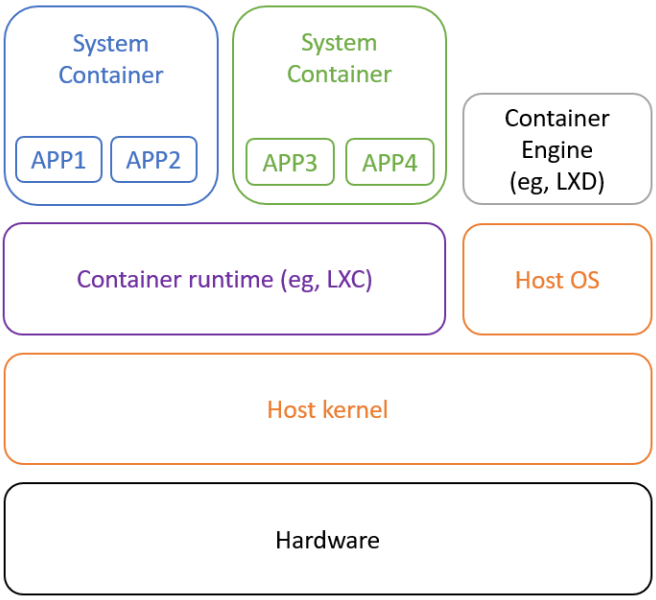

Figure 2.12: Operating System Container.

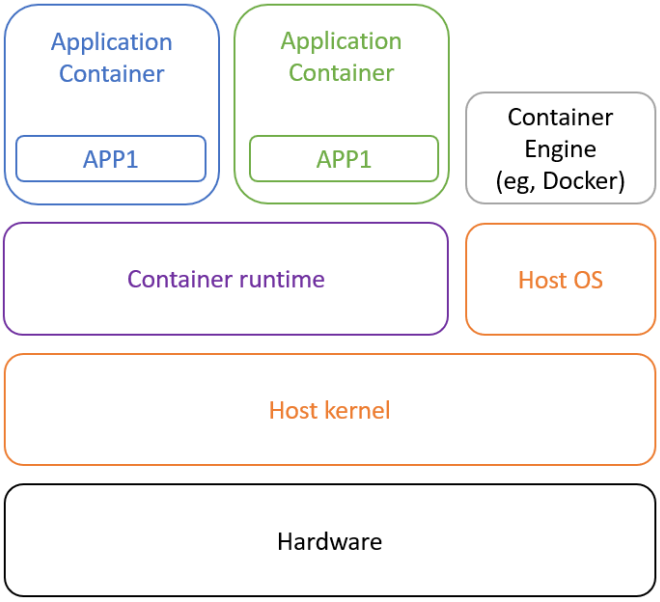

Figure 2.13: Application Container.

An example of an Application and OS container software platform is reported below.

- ***Docker***
  Docker [30] is a software platform that allows to create and manage application containers for the quick development of applications. In Docker, a container is an instance of a precise environment that is built and described by the container image. In a host machine, multiple containers of the same image can be instantiated. The container image is a static file that describes a given environment through system libraries, configuration settings, system tools, workloads and other configuration settings that the container needs to run. The image does not need to save a full operating system because it shares the OS kernel of the host. A common application that runs in a PC or VM can be installed, with all its dependencies, in an image and can be run easily anywhere. This software platform allows us to place together different isolated containers on the same computational resource and split an application into $\boldsymbol{N}$ components allocated in $\boldsymbol{N}$ hosts. These components can have in common the same base image and then different layers can be added to each component to customize it. A Docker image is composed mainly of two files:

  - Dockerfile It is a simple text document in which the user can write all the commands to assemble an image

  - run.sh It is the executable file that is run when the container starts

- ***LXC***
  LXC [31], which stands for Linux Container, is a software platform to develop OS containers in a Linux kernel machine. It allows the segmentation of the system creating different isolated instances. LXC has a low-level approach and can be managed in an easier way by the deamon LXD, which is image-based and can be exploited to run all types of workloads. There is also the possibility to run a Docker container inside an LXD container.

2.3 Road Safety and Connected Cars

Two big problems of the developed cities are traffic congestion and road traffic accidents that decrease the safety, livability and health state of the citizens. One way to mitigate the risk of road accidents and to better control the traffic is to ensure autonomous driving and create an environment where cars are connected to each other, with pedestrians and also other IoT devices, in order to maximize vehicle awareness and improve the user experience.

2.3.1 V2X Communication

The V2X stands for Vehicle-to-everything and is a communication system used by Intelligent Transportation System (ITS) to communicate with any other entity through vehicular communications technologies. The two main communication technologies developed to enable Vehicle-to-Infrastructure (V2I) and Vehicle-to-Network (V2N) services are:

- IEEE 802.11p (WiFi) [32]
- 3GPP C-V2X (LTE) [33]

Both can operate in the band that in Europe is reserved for vehicular applications (5850-5925 MHz).

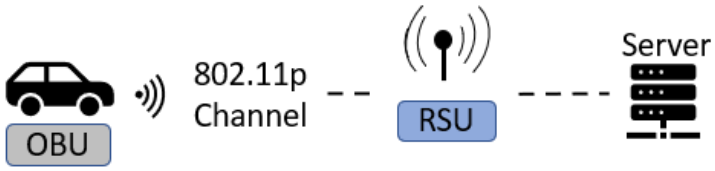

Figure 2.14: V2I 802.11p scheme.

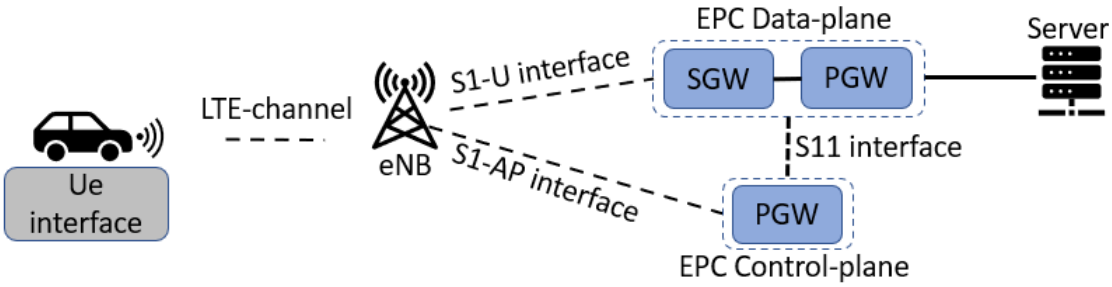

Figure 2.15: V2N LTE-based scheme.

The first one, depicted in Fig. 2.14, is the WiFi-based technology proposed for vehicular networks. The IEEE 802.11p is used by the ITS for the PHY and MAC layers. The PHY layer exploits the Orthogonal Frequency Division Multiplexing (OFDM) with the Carrier Sense Multiple Access mechanism Collision Avoidance (CSMA-CA) for direct communications between vehicles (V2V) and with the infrastructure (V2I). Each channel has a bandwidth of 10 MHz and the supported modulations are BPSK, QPSK, 16QAM, or 64QAM. The vehicle, to communicate

using this technology must be equipped with an *On Board Unit* (OBU) which is exploited both to send messages to other vehicles' OBU and to the *Road Side Unit* (RSU) placed in the proximity of the vehicle. In the range of aforementioned operative frequencies, there are two channel slots, Signaling Channels (SCHs) and Control Channel (CCH). The former is reserved for the exchange of non-safety data, the latter is dedicated to system control and safety-related messages.

The 3GPP C-V2X (LTE), in Fig. 2.15, is the proposal from the cellular world, standardized in 2016 by 3GPP that exploits the eNodeBs to guarantee the connection between the user equipment (UE) and LTE core network. Actually, it presents two interfaces, one is the Uu for communication between the end-user and the cellular infrastructure (V2N) and the other one is the PC5 for the V2V connection. They can be used independently and the PC5 does not mandatory need a network connection to work. The decision of the interface to use for the communication is up to the application layer and the communication channel is managed with the Frequency Division Multiplexing (FDM) technique.

2.3.2 Autonomous Vehicles Data Collection

Connected vehicles and smart cities need to improve road safety using also the support of the internet infrastructure, but not all data generated by autonomous vehicles can be elaborated in the IoT-edge-cloud continuum due to limitations that will be described in this subsection. To realize a hyper-connected environment it is necessary to increase the number of sensors around the cities and also on board cars. To understand the level of automation of a car it is sufficient to look at the classification made by the Society of Automotive Engineers (SAE) [34]. This classification goes from 0, indicating a non-automated car, up to 5, indicating a fully automated car in which there is no steering, no pedals and there is no need for human attention or interaction during the driving. The simple idea is that the higher the level of automation, the more sensors are needed for the car to autonomously operate. This is directly translated into increased production of data. As data production increases, so does the usage of ML (DNN) to extract meaningful results from this huge amount of data in a relatively short time. Nowadays, most of the new automated cars have a level 2 of automation and as the cited article of Tuxera reports [35], connected cars of this level can generate around 25 Gigabytes of data per hour. This quantity will increase proportionally to the increasing autonomous levels of future cars which can be a problem considering a MEC infrastructure that has to support this massive volume of data, together with the ordinary traffic that the network support. Thus, it is necessary to take advantage of the aforementioned IoT-edge-cloud continuum, not only using edge and cloud computing but also exploiting the IoT computing power. In this way, the applications can be stratified based on the data dimension needed and the latency requirements. The server in

the edge can compute important data for low latency required by the IoT devices and save permanent data in the cloud server. If the quantity of end-user data is so huge that the transmission time toward the server needs more time than the processing of the same data, it is better to use the end-user device to partially elaborate the final result, without sending all the data toward the edge server, but just the partial results. In this way, the server is less overloaded from the point of view of the computation and data transmission time, drastically decreasing the band occupation of the edge connection as well.

2.3.3 Trajectory prediction algorithm

One of the most important types of applications that improve the safety and the livability of the urban environment is surely Collision Avoidance (CA). Most of the accidents that happen in the street are due to collisions of vehicles caused by human errors. In particular, a high number of accidents happen at crossroads [36]. A CA application can use the connectivity capabilities of modern car models, and V2X communications, to collect data and generate an efficient response to avoid the collision among vehicles. In general, for collision avoidance, data such as position, direction and velocity of each vehicle approaching the crossroad, are used to check for a possible collision. In order to prevent collisions using this data, there is a need to predict the trajectory of the incoming vehicles. For this reason, trajectory prediction algorithms are the core of the CA. In [37], are described different solutions for trajectory prediction. Each solution differs according to the field of application on which they are employed, for example, the so-called physics-based motion is ideal for short-term predictions because it uses physics's laws to predict the outputs and so, it can be employed to predict basic tasks as check if the vehicle in front is decreasing its velocity. For trajectory prediction in the cross-road scenario, a long-term complex trajectory prediction application is needed. For this task, the learning-based motion prediction model is typically employed. It uses a data-driven approach according to which a NN learns the behaviour of the vehicles during a training phase, and it performs the trajectory prediction when requested. This model, unlike the previous one, allows for predicting tasks in more complex scenarios. The paper [38] describes a trajectory prediction application for a collision detection application. The example uses two types of ETSI messages. The first one is the Cooperative Awareness Messages (CAMs), that is periodically transmitted by vehicles with a typical frequency of 0.1 Hz, carrying information such as acceleration, position, speed, and yaw of the sender. The second one is the Decentralized Environmental Notification Messages (DENMs), which are sent from the eNB or RSU to the vehicle to notify it about generic events or dangerous situations. In a simplified way, all the cars approaching the crossroad send CAMs to the base station near the crossroad, connected to an edge server that collects

all the car's data, elaborating also their future trajectory with the help of an ML algorithm. Having the list of all predicted trajectories of all approaching cars, the server can perform a collision check, identifying if there are trajectories that will collide at the crossroad. In case of a positive response, the server will send, through the base station, an alarm message to the cars with a probability to collide. The applications that use ML methods to provide an advanced trajectory prediction service, need a huge amount of data to train the neural network. In the next future, the data exchanged among vehicles and infrastructures will become huge, generating a problem in the transmission time of the data and in the bandwidth occupation.

2.4 Urban Environment simulation

To reach the goal of improving the livability of the cities, running several tests in different urban scenarios is needed. It is unrealistic thinking to run large-scale tests using real vehicles and drivers on city streets. The main reasons are the safety of the people that normally use the streets and the huge costs that several real large-scale tests require. This problem is solved by using urban traffic and network communication simulators.

2.4.1 SUMO

It is an open-source traffic simulation package [39] that was developed by the German Aerospace Center. SUMO can be adopted to test plenty of urban scenarios e.g. traffic management, route choice, traffic light algorithms, automated driving or simulating vehicular communication. Inside these scenarios, it is possible to simulate things like routes, vehicles, bicycles, railways, pedestrians and many others. The simulation of this tool is considered microscopic because each element has a dedicated customization panel to perform detailed modifications on it. Each vehicle moves individually following a flow, having a characteristic of randomness, or a route decided by the user. The design of the simulation can be performed through the command line or graphical interface. The simulation results can be graphically visualized in real-time or just the result from the command line can be obtained. To simulate vehicular communication (V2X) using SUMO, in conjunction with a network simulator, the Traffic Control Interface (TRaCI) [40] can be used. It gives access to the road traffic simulation and allows retrieval values of simulated objects through a client/server architecture based on TCP protocol, where SUMO is the TRaCI-Server and the communication network simulator is the TRaCI-Client. The connection is real-time so that it is possible to modify the vehicles' behaviour in SUMO, based on the messages from the communication network simulator.

2.4.2 ns-3

One of the most popular network simulators is ns-3 [41]. It is a programmable framework in which the network infrastructure can be built by programming in C++ or Python. Different types of network connections between nodes can be selected, such as point-to-point, wireless and many others. The real or expected behaviour of the connection itself can be simulated, imposing precise parameters such as data rate, error rate, and packet size or choosing the transport protocol (TCP, UDP) and the routing protocol (IPv4, IPv6). ns-3 can also interact with real systems using a real-time scheduler. This framework is based on discrete-event simulation, meaning that each event is linked with its execution time and when the simulation starts, all the events are executed following a precise temporal order. This allows starting some events in a precise time instant in the middle of the simulation. Thanks to its API network simulation traces can be collected and then analyzed through a packet sniffer as real network traces.

2.4.3 Vehicular network simulation framework

A variety of tools are available to perform network simulations, e.g. Shawn [42], JiST/SWANS [43], OMNet++ [44] or the already cited ns3, but none of these allows to evaluate how the networking applications influence the urban mobility. To overcome that need, vehicular network simulation frameworks were developed.

- **Venis**
  Vehicles **i**n **n**etwork **s**imulation is an open-source vehicular network simulation framework. It is based on two simulators, OMNet++ for the communication network part and SUMO for the urban mobility simulation. They are connected using the TRaCI interface so that the results of one simulator can influence in real time the behaviour of the other one and vice versa. Venis is based on completely detailed IEEE 1609.4 Dedicated Short-Range Communications (DSRC)/Wireless Advanced Vehicle Electrification (WAVE) and IEEE 802.11p network layer models. Instead, for 4G/5G cellular networking, it needs to include third-party models.

- **ms-van3t**
  The ms-van3t framework [2], that stands for **M**ulti **S**tack for **V**ehicular **A**d hoc **NET**work applications testing in **ns-3**, can be exploited. It was developed at the polytechnic university of Turin (PoliTo) and is an open-source framework developed in the ns-3 simulator, that in conjunction with SUMO, can be used to build and simulate ETSI-compliant VANET (V2X) applications allowing information transfer between vehicles and infrastructure as well as among vehicles. ms-van3t supports the two main communication technologies (802.11p

and 3GPP C-V2X (LTE)) to develop V2N and V2I applications in which vehicles are set up to periodically broadcast CA messages and receive DEN messages from the server.

Chapter 3

Used Architecture

This chapter will present the overall implemented architecture, describing in detail all the elements of it, showed in Fig. 3.1. In particular, it will be described how the ms-van3t framework is used to get a baseline for the antenna coverage to use in SUMO, how the vehicular Federated Learning scenario is realized with Flower and Docker, and how the data set is used to train the trajectory prediction algorithm.

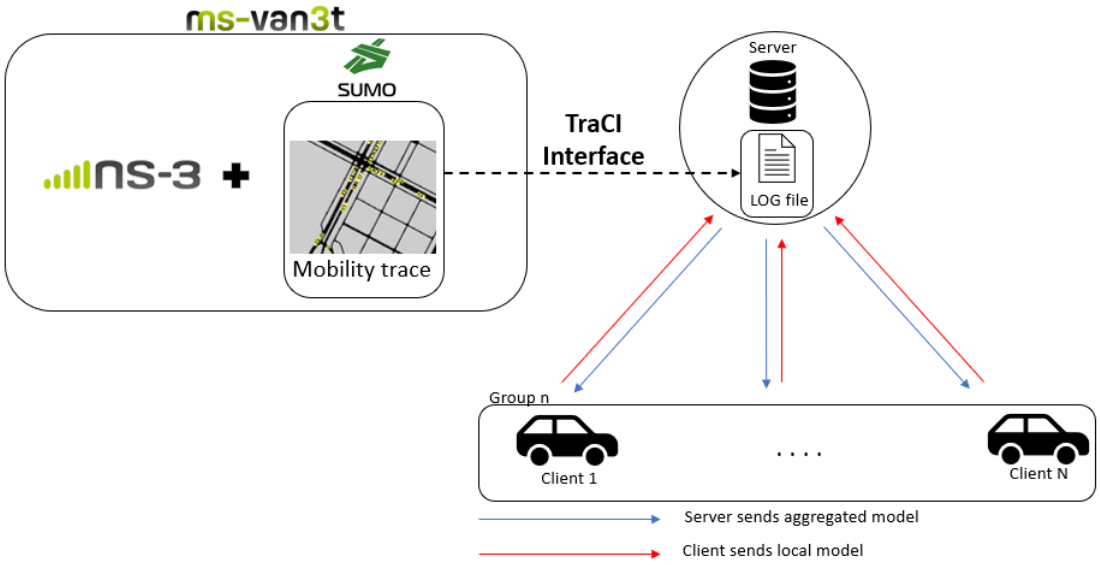

Figure 3.1: Real-world simulation scheme.

The main goal of this work is to compare the performance of FL in two scenarios, the real-world and the synthetic. The former wants to be realized using vehicles in an urban environment as clients of the federated network. The latter is a more static scenario, where clients change with a fixed periodicity and less frequently

compared to clients in a real-world scenario. The purpose of comparing the two scenarios is to understand how an urban mobility scenario can impact a federated learning performance. To test the real-world scenario, we want to link the behaviour of real vehicles entering and leaving a specific area with simulated federated clients. To do that, we got a mobility model using urban mobility and network simulators.

3.1 ms-van3t and SUMO for mobility trace

For the purpose of the thesis, we chose ms-van3t as a vehicular network simulation framework because it comes with LTE support out of the box instead of Veins which needs a third-party module called SimuLTE, in order to implement the same thing. The most useful aspect of ms-van3t is the possibility of using the standard "TraCI", to create a live interaction between SUMO and ns-3 simulation. It allows capturing values of simulated vehicles from SUMO, dynamically updating ns-3 client nodes.

3.1.1 SUMO

To simulate a real urban environment with SUMO we used a data set called TuST (Turin SUMO Traffic), described in [45], it was built to simulate a traffic pattern model of the metropolitan area of Turin and its neighbouring districts, starting from data collected in 24 hours by 5T, the public company of Piedmont that works in the field of the info-mobility. For the thesis work, we considered a portion of the TuST zone, precisely an area of 3.5 km^2 around the Polytechnic University of Turin, shown in Fig.3.2. Specifically, to create the mobility trace were considered only vehicles moving in the part of the city defined by four TAZ (Traffic Assignment Zones), from 7 a.m. to 9 a.m. which is the two-hour morning traffic peak. The area covered is contained in the following coordinates:

- Latitude: [45.054223, 45.073081]
- Longitude: [7.651316, 7.672293]

3.1.2 ns-3

The aforementioned area is covered by the radio signal of seven cellular base stations (eNBs) configured considering the actual real-world cellular deployment, as represented in Fig. 3.3. Using the ns-3 framework, a star network topology was created, its centre was made by the edge-server and the connecting nodes were made by the seven eNBs. As for vehicles, they are represented by simple client nodes equipped with an On-Board Unit (OBU) capable of exchanging packets

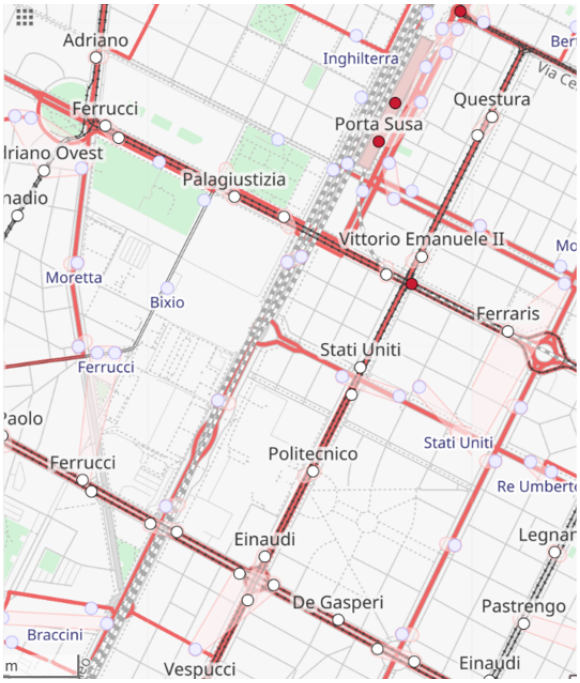

Figure 3.2: Simulation area taken from OpenStreetMap [46].

with the seven eNBs and consequently with the single edge-server. The network protocol used was TCP because reasoning the perspective of a trajectory prediction application, it is necessary to have a reliable and controlled transmission of all the packets transporting the parameters of the local and global NN models.

3.1.3 Mobility trace

It is important to mention that the outcomes described below were only used for real-world scenarios. The idea is to give to Flower clients a behaviour as similar as possible to vehicles in a real urban scenario. Practically, it is necessary to know the average time, known as dwelling time, on which vehicles are covered by at least one of the seven eNBs. With this information, we can know for how long a real vehicle, given real traffic conditions, can be used to train a model in our Federated Learning framework. The first step to obtaining the dwelling time was to decide the radio coverage's dimension for the urban area. Through simulations with ms-van3t was verified that using antennas with 500 m of signal radius, the vehicles can be considered under coverage, with fairly stable bandwidth, as well as low delay and negligible packet loss. The second and last step was performing the urban mobility simulation exploiting SUMO. The settings used were the same as described in 3.1.1 and 3.1.2, imposing the aforementioned antenna radius. The

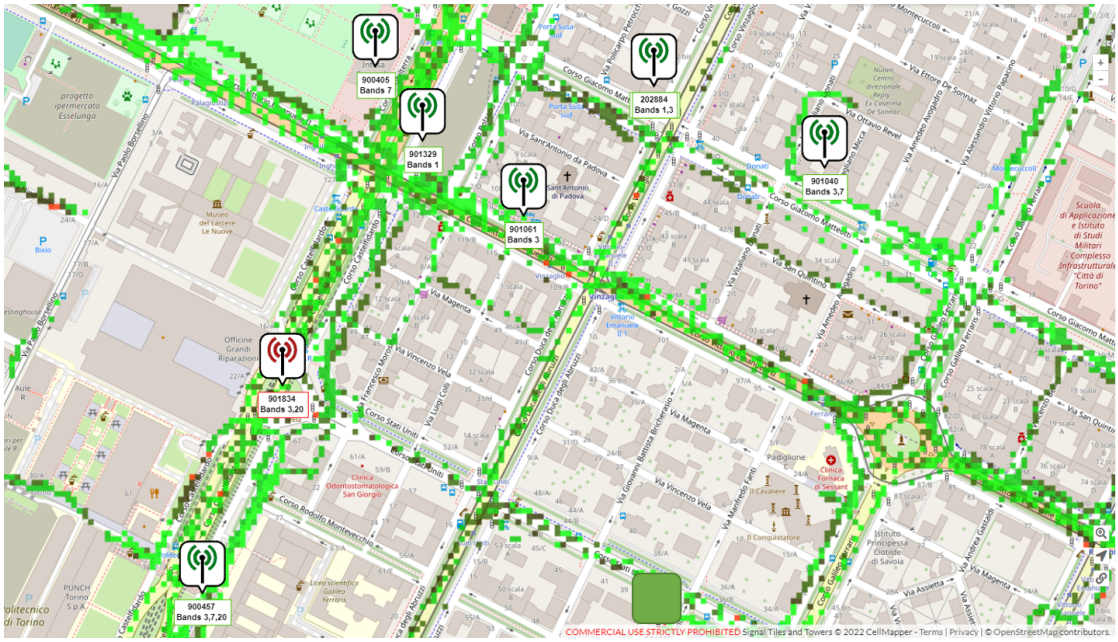

Figure 3.3: Simulation area with network coverage details taken from CellMapper website [47].

outcome of this simulation is a ***LOG file*** containing the mobility trace needed to impose an urban behaviour on the federated clients in Flower. Each line of this file is created collecting the SUMO simulation data every 100 ms, for the two-hour time interval. A single row of the file contains the vehicles' identifier number and their dwelling time. This file is then filtered and adapted to the Federated Learning scenario, considering only vehicles with a dwelling time longer than 60 seconds (the reason for the selection of this time interval is explained in 4.6.2), in order to select only vehicles that can, potentially, finish the FL computation before leaving the coverage area. At each time step, the selected vehicles are ordered in a descendent way, considering the dwelling time, as depicted in Fig. 3.4.

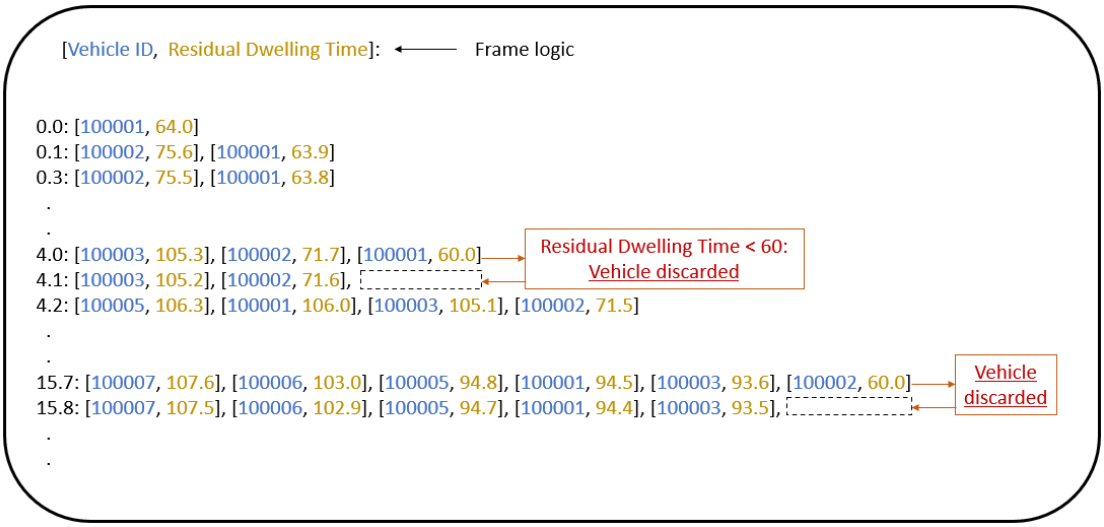

Figure 3.4: ms-van3t LOG file.

3.2 Flower framework

The aforementioned LOG file was implemented in the Federated Learning framework in order to modify the federated clients' behaviour following the one of a vehicle in an urban scenario. Among the different frameworks that can be found in literature, we have tested the IBM Federated Learning software [48] and the Flower framework [3], in the same conditions. The results of the two were comparable, so, the chosen one for the thesis work was Flower, because it is easy to customize and more suitable for our purposes. Indeed, the IBM software has a black box style, the source code cannot be changed following our necessity.

3.2.1 Docker containers

In order to work with a real use case from the point of view of connections between client and server, it was necessary that each vehicle and server were in different locations. Moreover, to make the framework adaptable to a wider number and type of scenarios, it was useful to use a tool that allowed us to vary the network parameters widely. To meet these needs, Docker was used. It allowed the creation of environments isolated from each other through containers and an automatic configuration of the network, with the possibility to vary its parameters. This last function operation would not have been possible without Docker containers. A Linux VM provided with 16 virtual CPUs was used to run the FL simulations where cars and the server were simulated by Docker containers. A Docker image for the server and one for each client was built, mainly installing the Flower software

on them. The communication among containers was possible simply by specifying the IP addresses in the client and server's Flower script that automatically create the needed connections. Given the limit of available CPUs, we decided to assign to each client container a maximum of 2 CPUs with the possibility of simulating six clients at the same time, leaving the remaining computational power for the server container and the VM itself. All the network connections between servers and clients are emulated by Docker containers. Moreover, in the containers were specified the eNB-server links' characteristics:

- 1 Gbit/s bidirectional bandwidth
- No delay
- No packet loss

The communication at the federated learning framework level, between server and clients, is managed by the gRPC, as described in subsection 2.1.2. An example of the global federated learning framework created with Flower framework and Docker is represented in Fig. 3.5.

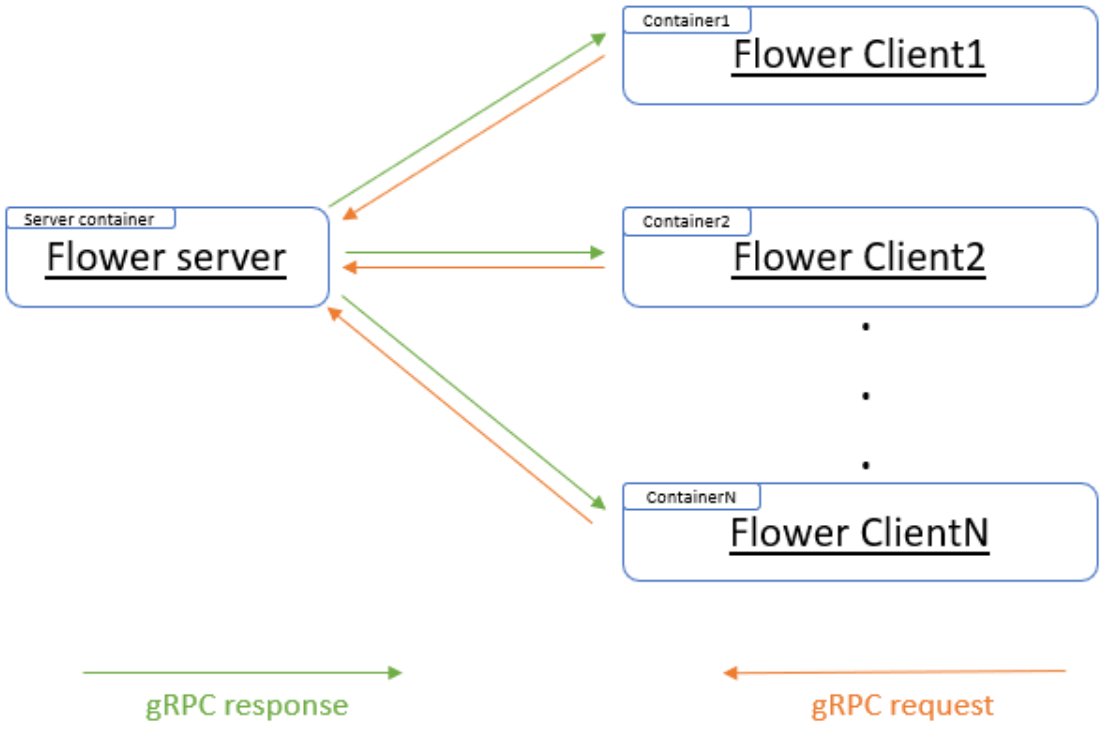

Figure 3.5: Federated Learning framework.

3.2.2 Flower settings

After an in-depth study of the scenario, the maximum number of **epochs** and **rounds** was fixed to 10, leaving the possibility for the algorithm to stop a round or an FL cycle before, using an early stopping mechanism described in 4.4. The value 10 was chosen as the maximum number of epochs and rounds because, by performing several isolated FL cycles, with different lengths of rounds and epochs, we have seen that increasing the number of these parameters leads to an increase in the execution time but not to a clear improvement in the accuracy of the model. It is useless to fix the value of rounds and epochs higher than 10, considering the aforementioned values. The chosen strategy to select and aggregate clients' model parameters was the FedAvg.

3.3 Trajectory prediction data set

The data-set was taken from a real scenario, as shown in Fig. 3.6. It was built starting from an aerial shot taken with a drone and after the post-processing, it were obtained the trajectories of 5105 vehicles. Almost 53% of them, 2699 to be precise, are vehicles travelling straight at the intersection and the remaining part, 2406, are turning vehicles. This almost equal partition of the type of vehicles allows having a model homogeneously trained for both behaviours at the crossroad.

To use the distributed framework with Flower we needed to modify the structure of the dataset because the original one was created for a centralized ML approach. With FL we need to divide the entire dataset into several subsets to assign to all the federated clients. For the evaluation phase of the LSTM NN, a dataset of 200 cars was created using a portion of the 5105 cars. For the training phase, we divided the remaining dataset into 170 subsets, each containing 30 vehicle trajectories. When a new client joins the FL, a subset randomly chosen among the 170 is assigned to it. To not assign the same subsets to consecutive clients that join the FL, at the start of FL a file containing the used data sets was created. When a client joins FL it receives a data set not included in the list of used data sets. This list is emptied when all the 170 data sets are written on it. The criteria for the number and the dimension of the subsets start from the necessity to make the duration of a round, for each client, around 1 minute.

The key element to explain numerically this decision is the simulation timing, directly linked with the dimension of the subset. Considering that vehicles are moving around and are under the coverage of the eNBs for a limited time interval, the timing for the Federated Learning process must be correctly chosen in order to avoid bad behaviours and results. The time interval for an FL round cannot be too long otherwise some vehicles can go outside of the base station's coverage area before finishing the training and their local models cannot contribute to the

aggregation of the global one, possibly reducing its accuracy. On the other hand, the round time cannot be too short because the shorter the round time, the shorter will be the Federated Learning cycle. This leads to a poorly trained model with bad accuracy and behaviour similar to decentralized machine learning with just one server-clients communication round, making useless the usage of the Federated Learning approach. Considering the already described antenna coverage area, the average speed for a vehicle in an urban scenario that is 30 km/h and considering the presence of traffic lights and urban traffic, it was supposed that a car can stay under the antenna's coverage for at most 10 minutes. Taking into account that the maximum number of rounds for a single vehicle in an FL cycle is fixed at 10, we want to ensure that a single round cannot exceed 1 minute. As the last step, considering that a single simulated vehicle has a computational power given by 2 CPUs, the average dimension of a data set to ensure the round time duration of around 1 minute, obtained after several FL simulations, is 30 vehicle trajectories in the original 5105 cars' trajectory data set.

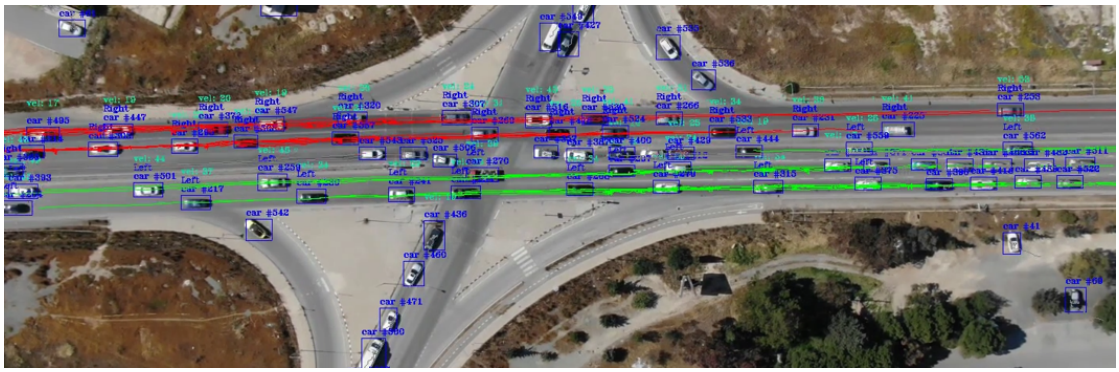

Figure 3.6: Drone view of the real intersection.

3.4 LSTM Trajectory Prediction Algorithm

In this work, the LSTM model described in [9] was used because it shows how this specific LSTM algorithm applied in a trajectory prediction training scenario gives results that outperform other tests' results in a less challenging scenario. The main reason for these results can be found in the cumulative distribution function because almost 60% of the prediction errors are under the values obtained in other papers, this allows the mean value to have competitive and optimal results. To train the algorithm, a data set coming from two different scenarios was used. One crossroad in a simulated urban environment (with and without data coming from traffic lights) and another intersection in a real urban scenario (with and without data coming from traffic lights), precisely in Nicosia, Cyprus, at the peak

hour. The considered scenario for this work is the real one, without traffic light signals (w/o TLS). It is an encoder-decoder LSTM model that is more suitable for trajectory prediction based on data series structure with undefined time gaps. The thesis scenario is characterized by the continuous operation of mapping the input sequence with a non-predefined length (historical data collected by cars) toward an output sequence (vector of future positions). Therefore this model fits perfectly our needs. This algorithm presents two main parts, training and evaluation. This algorithm [9] is written with the support of Keras [19], the Python deep learning API and presents two main parts, training and evaluation. The training part exploits the built-in function ***model.fit()*** from Keras [19]. The evaluating part uses the ***model.predict()*** function from Keras that, using the trained model on an input data set of trajectories, predicts the correspondent output. To obtain the accuracy with which the trained model can predict the results, predicted and correct trajectories are compared. This comparison is obtained by computing the Euclidean Distance between the two types of trajectories. Averaging all the Euclidean Distances acquired by evaluating an entire data set, the Mean Euclidean Distance is computed (MED).

Chapter 4

Framework Implementation

This chapter describes in a detailed manner the implemented architecture, with a focus on the algorithm part.

4.1 Network infrastructure implementation with ms-van3t

The communication architecture shown in Fig. 4.1 was created, in the ms-van3t framework, through ns-3. It helps to verify which radius has to be assigned to the eNBs' signals in order to ensure coverage to all vehicles moving in the test area.

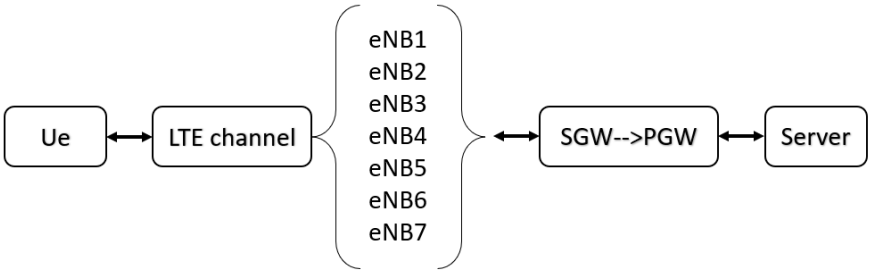

Figure 4.1: Ue-server communication infrastructure logic.

Starting from the right side of Fig. 4.1, a server node was implemented and connected to the EPC node (SGW+PGW). Thereafter, the seven eNBs take place in the scheme with the interfaces necessary to establish V2N communication with clients. The vehicles were represented as client nodes equipped with a V2N interface, with the possibility to modify their position based on SUMO through the TRaCI interface. An important parameter set in the server and client nodes is

the "MaxBytes" that impose the maximum dimension of the packets to be sent. The fixed value was derived by performing several tests of Federated Learning in different configurations (number of clients and dimension of data set) with the Flower framework, using the thesis' work data set 3.3. It was pointed out that the model parameters exchanged between server and clients are fixed to 0.835 MByte. In A.0.2 is represented as an example of traffic capture.

4.2 Flower scheme

Federated learning, by definition, is characterized by a certain number of rounds where clients and the server exchange the NN models' parameters. Before the first round, the server chooses a group of clients and sends them the parameters of the initial model. If clients are fixed objects always connected to the server, the FL goes ahead always selecting the same group of clients until the end of the FL job. In our scenario, the clients are moving around and during the FL simulation, the server chooses different groups of clients that approach the area covered by eNBs. A schematic representation of an FL simulation is represented in Fig. 4.2. It can be seen that the simulation is composed of $\boldsymbol{k}$ FL cycles, necessary to reach the desired accuracy, each one is made by $\boldsymbol{N}$ rounds where the model parameters are exchanged between server and clients and similarly, rounds are made by epochs, as described in the definition of federated learning approach. This scheme is then modified to follow the two tested scenarios, the real-world and the synthetic one.

Before describing the two scenarios in detail, we are going to provide an overview of some common implementations of the FL architecture.

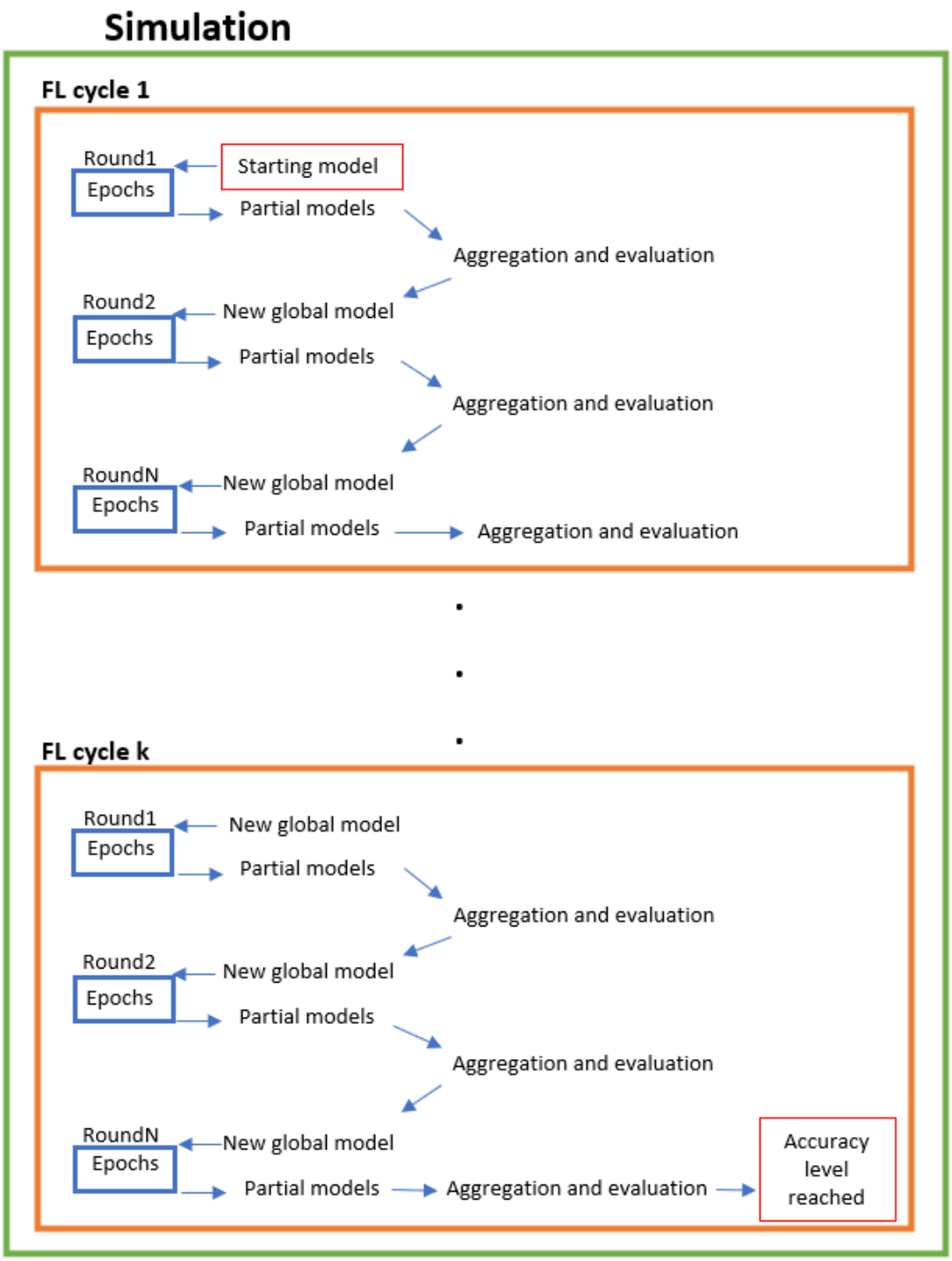

Figure 4.2: Federated Learning logic scheme.

4.2.1 Docker containers implementation

As described in the previous chapter, to simulate distinct locations for clients and the server and to have the possibility to change the network connection parameters, we implemented different Docker containers. For our work, all the commands to install the packages like Flower [3], Keras [19],TensorFlow [20], that we needed to run our FL application, were installed in the ***Dockerfile***. In the same file, the command to run the executable ***run.sh*** is written. This last file contains some instructions to set the transmission environment. The main commands are:

```
'ethtool -K lo tso off'
```

- This command is used to disable the tcp segmentation offload

```
'tc qdisc add dev eth0 root netem rate 1gbit'
```

- The traffic control (***tc***) command helps to configure the kernel packet scheduler, in particular, with ***qdisc***, a standard First-In-First-Out scheduler was assigned to the eth0 interface. Then the network emulator (***netem***) was used to limit the one direction bandwidth to 1Gbit/s

```
'python server.py'  or  'python client.py'
```

- These are the last commands of the run.sh file and are used to run one of the two Flower files (python server.py or python client.py), depending on the container in which run.sh file is

To summarize, when a container is created, all the needed packages are installed on it through the Dockerfile. When the container starts, the executable file is called. It sets the network configuration and runs the Flower file, starting the Federated Learning process.

4.3 LSTM model aggregation in Flower framework

The LSTM algorithm [9] was aggregated in Flower modifying the ***client.py*** and ***server.py*** files. Essentially, the two main parts (training and evaluation) of

the LSTM algorithm, highlighted in section 3.4, were aggregated correspondingly in the fit() function of the client.py file, as the ***Listing 4.1*** shows, and in the aggregate_evaluation() function of the server.py file.

```
1 —— client.py ——
2
3 def fit(self, parameters, config):
4   model.set_weights(parameters)
5   es=EarlyStopping(monitor='val_loss',mode='min',min_delta=1e-3,...)
6   history=model.fit(training_data,validation_data,callbacks=[es])
7   ...
8   result={"val_loss" : history.history["val_loss"][-1]}
9   return model.get_weights(), len(trainingData_X), result
```

Listing 4.1: Client.py code implementation

The fact that the evaluation algorithm was placed in the server.py file is noteworthy. Indeed, after the aggregation of the model, its evaluation is carried out in a centralized way from the server, using the data set of 200 car trajectories, as highlighted in 3.3. A centralized model evaluation, using a large and fixed data set, avoids the presence of bias compared to the evaluation made by clients.

4.4 Early stopping

During an FL round or epoch, the algorithm can stop itself through two early stopping criteria, mainly to avoid an over fitting situation. The first early stopping function, already present in the original LSTM algorithm [9], was implemented in Flower code. Looking at the ***Listing 4.1***, at line 5 there is the definition of the callback ***es*** representing the early stopping function implementation. The main value to look at is ***min_delta=1e-3*** which represents the minimum value of the delta variable needed to go ahead with the current round. This delta represents the variation of the val_loss variable between two consecutive epochs. The val_loss represents the error of the model during the validation phase. If the val_loss variation between two consecutive epochs is less than min_delta, means that there has been no improvement and the round is stopped, as represented in Fig. 4.3.

Modifying the server.py file and the source code of Flower, the second early stop was implemented in order to stop an FL cycle. The concept is the same as the previous early stop method, using the same value of min_delta, but applied at the FL cycle level. The delta value to compare with the min_delta is computed as the difference between the average val_loss of the current round and the average val_loss of the previous one.

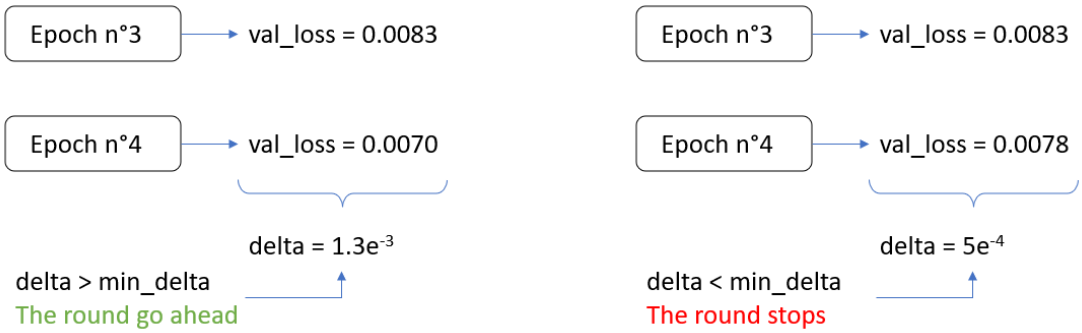

Figure 4.3: Early stopping epoch logic scheme.

4.5 Consecutive FL file

The Flower framework allows to perform only one cycle of FL before stopping the model training. For the purpose of the thesis, performing a single FL cycle is insufficient to reach the required model accuracy. Therefore, the bash script ***consecutive_FL.sh*** to loop the basic job of the FL framework was created. In this way, with a simple script, the entire simulation is automatized, and with a single start, the whole scheme of Fig. 4.2 can be reproduced. More precisely, the script starts the server container, and after sleeping for 30 seconds, it starts the client containers. When a Federated Learning cycle finishes, the script evaluates if to stop or not the simulation. The stop criteria of the bash script is the Mean Euclidean Distance (MED) value, described in section 3.4, provided by the server evaluation. At the end of each FL cycle, the computed MED value is passed to the bash script for comparing it with the fixed threshold of 1.299 m taken from [9]. If the MED value is below the threshold the desired accuracy is reached, and the simulation ends.

4.6 Synthetic vs Real-world scenarios

The comparison of the real-world scenario and the synthetic one, is the core of this work, as highlighted in the introduction of section 3. The synthetic scenario differs from the real-world one mainly because the vehicles in the former remain the same for a whole FL cycle, this means that the datasets change only between one FL cycle and the next one. Instead, in the real-world scenario, the clients change following the vehicles' behaviour taken from the LOG file generated by TraCI. What they have in common is the dataset selection logic described in 3.3.

4.6.1 Synthetic scenario

From Fig. 4.4, representing the synthetic simulation scheme, it can be seen that the training in the whole first FL cycle (FL1) is done with vehicles (datasets) from *group 1.* These datasets are used until the next FL cycle starts (FL2) and other clients (datasets) are chosen. Therefore, the data sets of group 1 are used for all the rounds of FL1. Another characteristic of the synthetic simulation is that during the FL cycle the same vehicles are always present and contribute to the computation. In comparison, in the real-world scenario, it can happen that from one round to the next one, the number of federated clients decreases.

To have a general view of the entire synthetic simulation, the first step is the selection of $\boldsymbol{n}$ random clients among the 170 datasets available and the first FL cycle can start. The first round stops after 10 epochs or when the early stopping takes place. With the same $\boldsymbol{n}$ clients another round starts till 10 rounds take place or the early stopping is activated. At that point, the first FL cycle stops, another group of $\boldsymbol{n}$ clients is chosen and the second FL cycle starts. These operations will be repeated till, at the end of an FL cycle, the resulting MED is under the threshold.

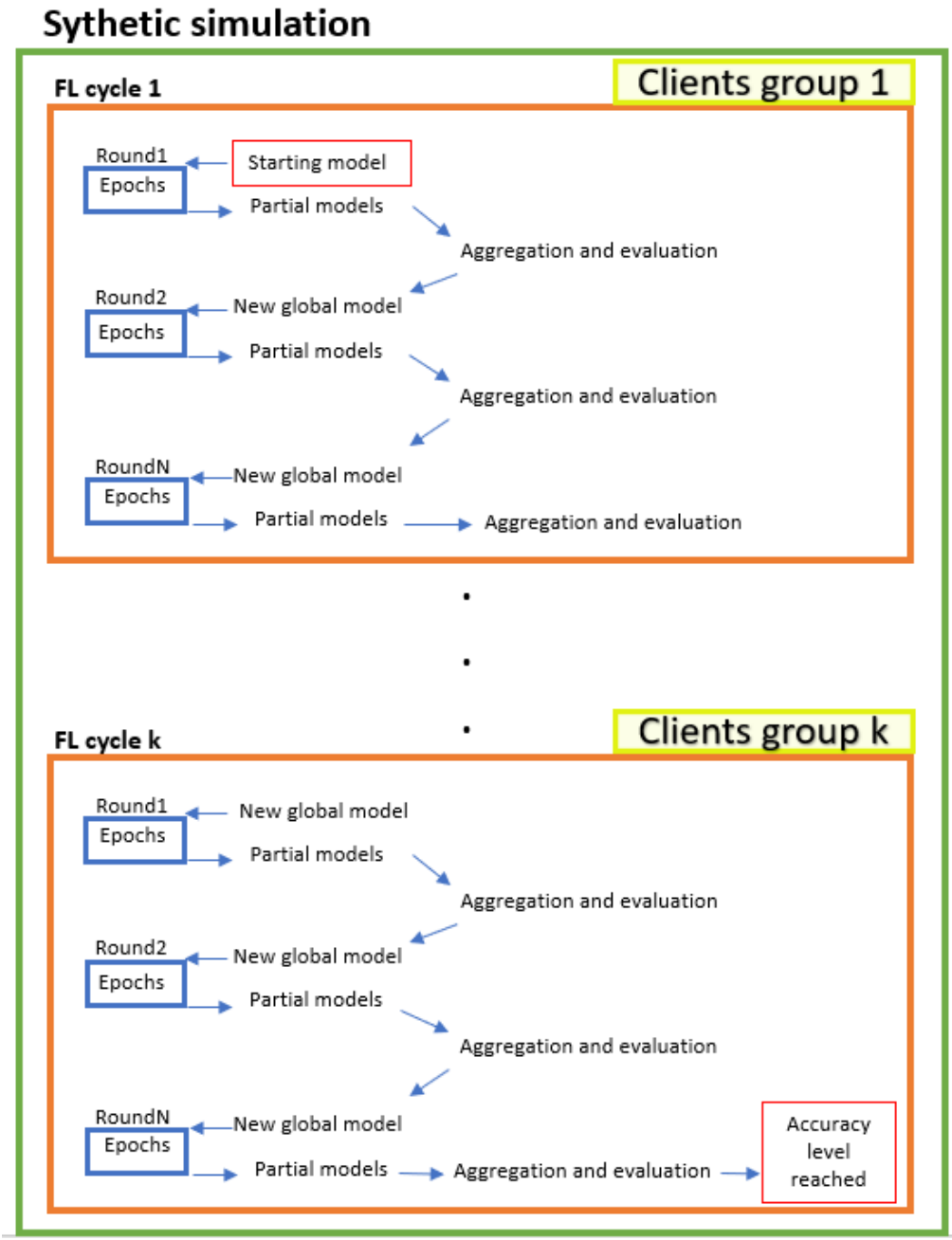

Figure 4.4: Synthetic scheme.

Dataset allocation in synthetic scenario

To test this scenario, a fixed group of clients was used per each FL cycle, without using the LOG file. The allocation was managed by modifying the Flower source

Algorithm 1 Client selection algorithm for synthetic scenario

```
1:  procedure SELECT A NEW GROUP OF CLIENTS(number of vehicles Ω in a line of LOG file)
2:      if length of dataset_file=170 then                ▷ if all the data sets have been used
3:          clear the file
4:      end if
5:      if FL_cycle_enabler = 1 then                      ▷ if a new FL cycle started
6:          FL_cycle_enabler = ∅
7:          while clients found=0 do                      ▷ Search data sets not yet assigned
8:              select a number of new random data sets as Ω
9:              if selected data sets are not in the dataset_file then
10:                 clients found=1
11:             end if
12:         end while
13:         update the NUM*.txt files
14:         update the dataset_file
15:     end if
16:     The server starts the containers
17: end procedure
```

code, more precisely, the ***client_manager.py*** file which contains the policy used to choose clients at each round. The algorithm 1 represents the implemented code. The ***dataset_file*** is the file described in section 3.3 that keeps track of data sets already assigned in previous FL cycles. The file ***FL_cycle_enabler*** is used to understand when an FL cycle ends and a new group of clients must be selected. This enabler file is managed by the ***consecutive_FL.sh*** script and the ***client_manager.py*** files. The former writes '1' on ***FL_cycle_enabler*** when a new FL cycle starts, as described in lines 5 and 6 of the algorithm 1, the latter writes a '0' when a '1' is read before the clients' selection starts.

In line 13, a file with the name ***NUM_*.txt***, where ***** is the number of the respective client (e.g. NUM1.txt), is filled with the identifier number of the data set. This is an easy way to share the data set identifier values between the client_manager.py of the server and all the client.py files of the clients performing the data sets allocation. Indeed, each client, at the start of each FL cycle, reads its NUM_*.txt file to understand which data set to use for the training. Every time a new group of clients (datasets) is chosen, the NUM_*.txt files are updated. The scheme in Fig. 4.5 represents the cyclic usage of the NUM_*.txt files during an FL round.

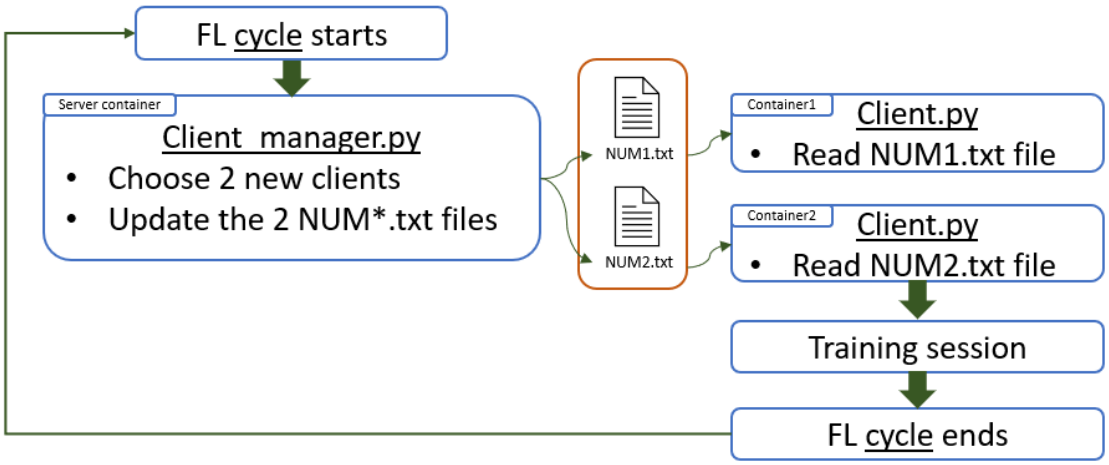

Figure 4.5: Scheme for the NUM.txt file usage in synthetic scenario.

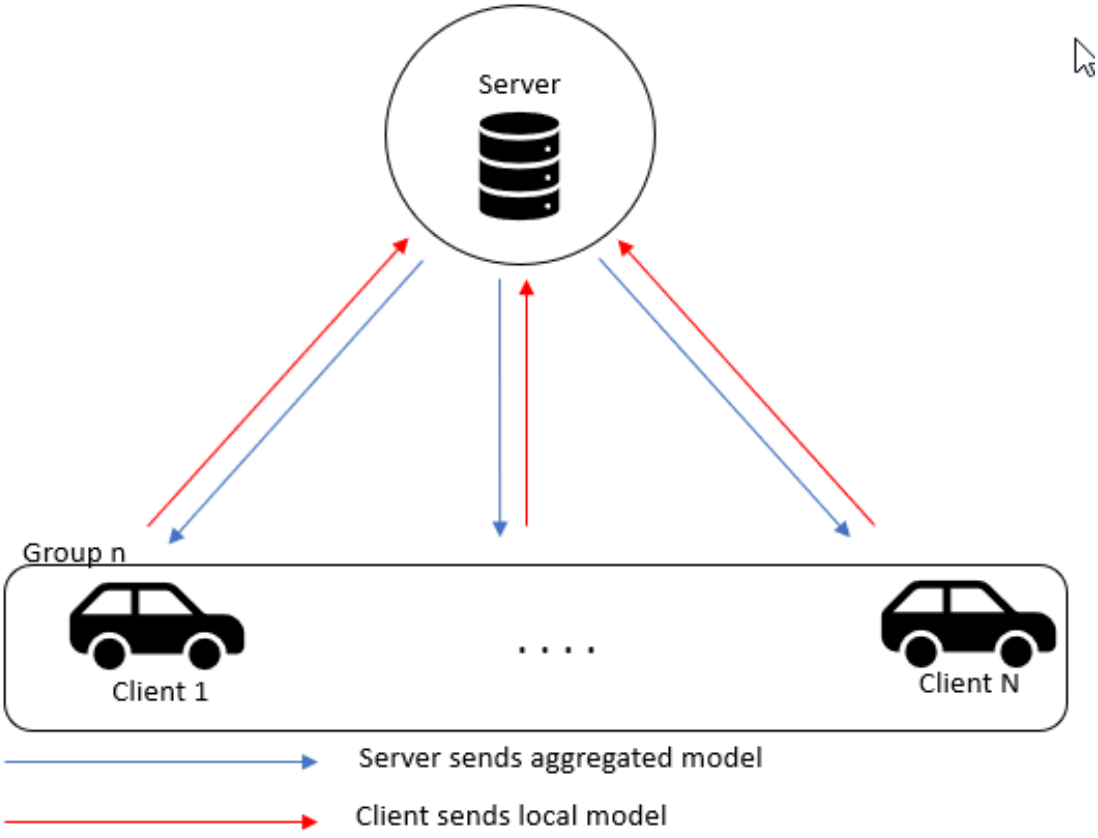

Figure 4.6: Flower scheme synthetic scenario.

This is a simple simulation scenario because it does not consider a real urban mobility condition. The simplicity can be noticed as well by comparing the synthetic scheme 4.6 with the real-world one 3.1. It is an ideal case built to have a baseline comparison with the real-world scenario simulation.

4.6.2 Real scenario

Looking at Fig. 4.11 the sequence of operations performed in the Real-world FL simulation can be seen. The characteristic to highlight is the variation of the clients group at each round. Specifically, the algorithm searches for, at least, the minimum number of clients each round. The clients are chosen considering the progress of the LOG file. This leads to different possibilities:

- Each round can be characterized by a different number of federated participants, because the maximum number of clients is not ensured as in the synthetic case

- Clients do not change between two consecutive rounds, which means that the vehicles were under antenna coverage for more than 120 seconds (Fig. 4.7)

Figure 4.7: Clients do not change.

- All the clients change between two consecutive rounds, which means that the vehicles were under antenna coverage for less than 120 seconds (Fig. 4.8)

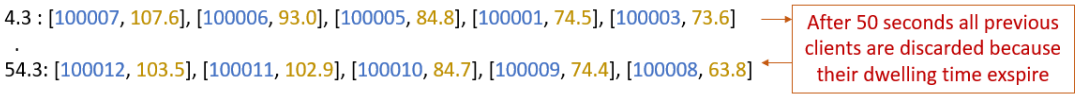

Figure 4.8: All clients change.

- A portion of clients changes between two consecutive rounds (Fig. 4.9)

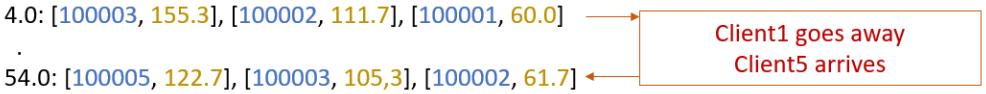

Figure 4.9: Partially change of clients.

- The row of the LOG file contains fewer vehicles than the minimum federated clients needed to start the FL. In this case, the successive rows of the LOG file will be considered until the condition of the minimum number of clients is satisfied (Fig. 4.10)

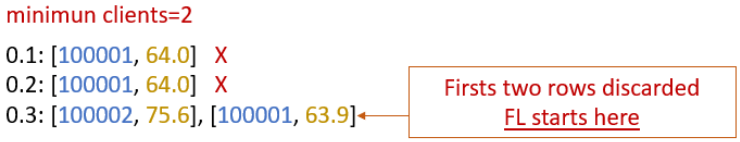

Figure 4.10: Condition on minimum number of clients.

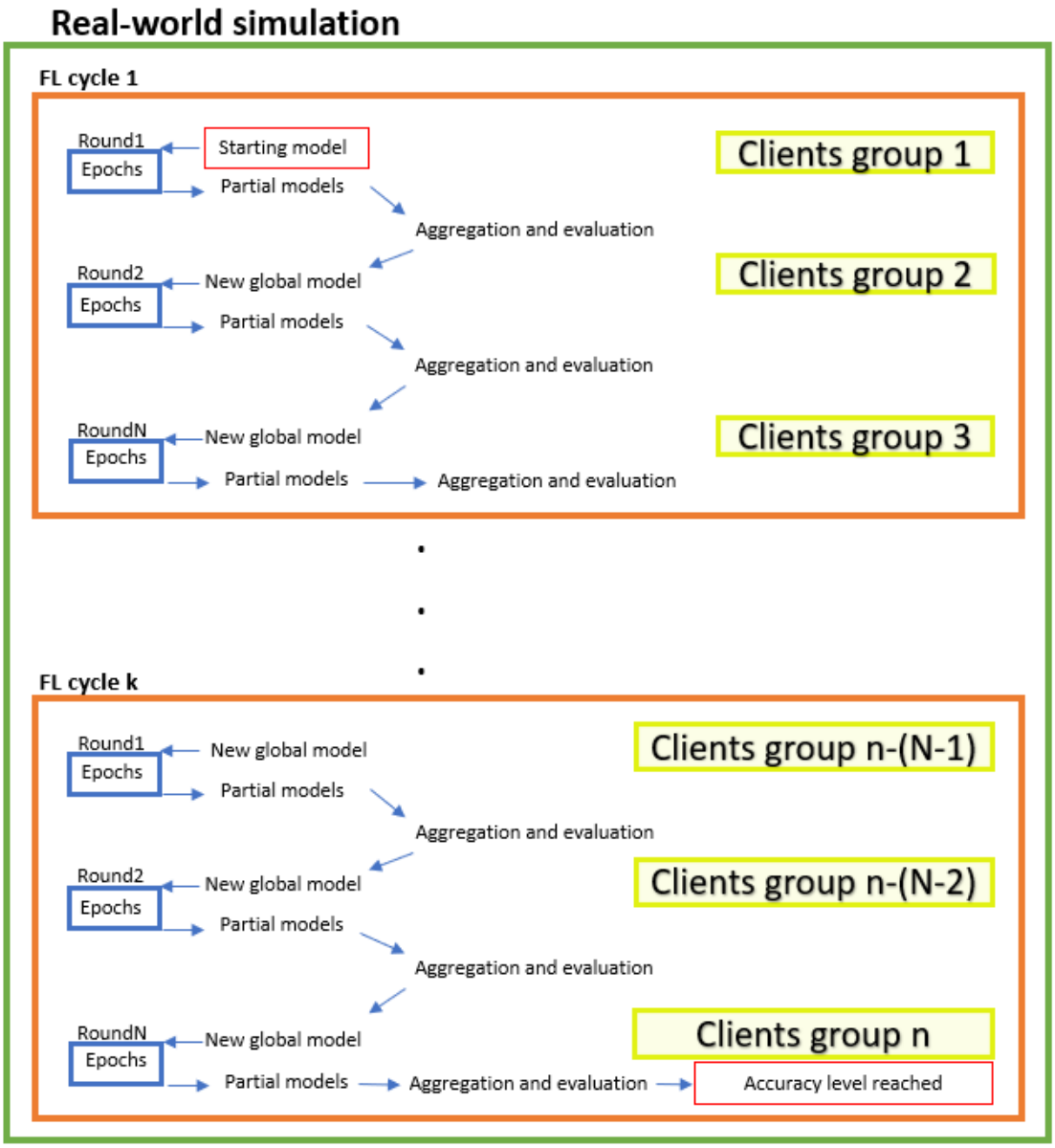

Figure 4.11: Real-world scheme.

Data set allocation in real-world scenario

In this scenario, to properly read the LOG file, the ***client_manager.py*** file was modified. A new strategy selection function named ***traci_selection()*** was created. This function reads the LOG file to properly select the clients at each round. In this sub-section, a general description of the new selection function is reported in the algorithm 2.

Algorithm 2 Client selection algorithm for real-world scenario

```
1:  procedure SELECT A NEW GROUP OF CLIENTS(number of vehicles Ω in a line of LOG file)
2:      let file_index =time_shift                ▷ Initialize the file index with the value of the last read row
3:      let file = LOG_file[file_index : end]     ▷ Put in file variable all the unread lines of the LOG file
4:      for all rows ∈ file do                                                  ▷ Read the file row by row
5:          if time_shift ≥ 77000 then                                          ▷ Check if the index is at the EOF
6:              time_shift = ∅
7:          end if
8:          if length of dataset_file=170 then                                  ▷ if all the data sets have been used
9:              clear the the file
10:         end if
11:         if current Ω ≥ MIN then          ▷ If the condition of the minimum requested clients occurs
12:             let exit = 1
13:             time_shift = time_shift + 500                                   ▷ Increasing time shift of 50 seconds
14:             if dataset_file = empty then                                    ▷ If no data sets were used yet
15:                 select a number of random data sets as Ω
16:                 update the NUM*.txt files
17:                 update the dataset_file
18:             else
19:                 if current vehicles = previous vehicles then
20:                     update the dwelling time
21:                 else
22:                     while clients found=0 do                                ▷ Search data sets not yet assigned
23:                         select a number of random data sets as Ω
24:                         if selected data sets are not in the dataset_file then
25:                             clients found=1
26:                         end if
27:                     end while
28:                     update the NUM*.txt files
29:                     update the dataset_file
30:                 end if
31:             end if
32:         end if
33:         if exit = 1 then
34:             force the exit from the for cycle
35:         end if
36:     end for
37:     The server starts the containers
38: end procedure
```

In this pseudo-code, two things have to be highlighted. The first one is the ***time_shift*** variable, which stores the number of the last row of the file read in the previous round so that in the next round the client manager starts reading from the correct time step and not again from the first row of the file. This is very important to maintain the simulation as close as possible to the real vehicles' behaviour. The second remark to highlight is the shift of 50 seconds in line 13, in case the row with the correct format is found. This time is added because, after several simulations, the computed average time of one round was discovered to be 50 seconds, as reported in Table A.1. Consequently, after one FL round the client manager increments by 500 rows the initial time_shift to simulate the evolution of the mobility during the round time. The first consequence of considering urban mobility as a reference for client selection is line 11, where the clients are selected only if the number

of currently available vehicles is equal to or bigger than the minimum number of clients, fixed in the Flower settings. This leads to the possibility of delaying the start of the next round. Instead, in the synthetic scenario, a group of clients of maximum dimension is always available, avoiding any possibility of delays. The sharing mechanism of the data sets identifier is the same as the one described in the synthetic scenario and represented in Fig. 4.5. The only important difference is that in the real-world scenario this mechanism takes place every FL round when clients are selected, and not only every FL cycle. The logic is represented in Fig. 4.12

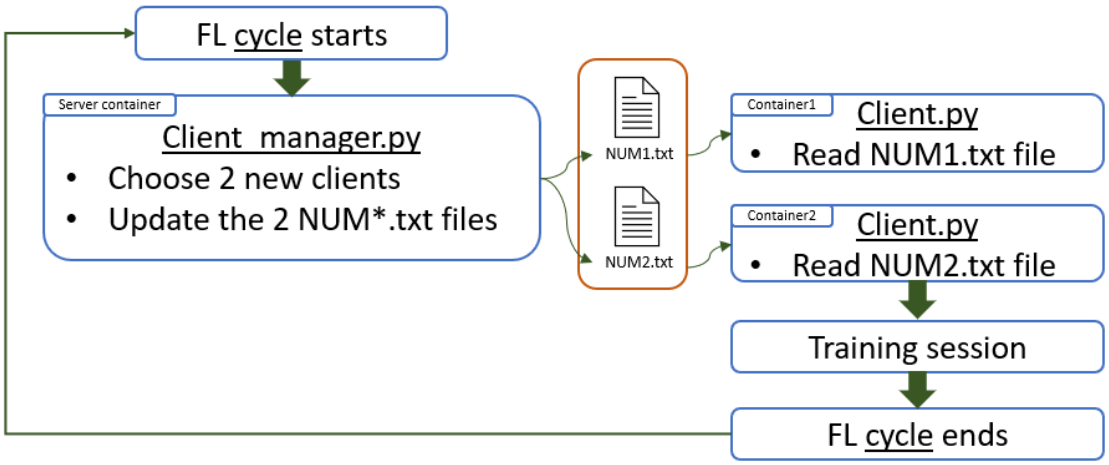

Figure 4.12: Scheme for the NUM.txt file usage in a real-world scenario.

Chapter 5

Results

The thesis work started with the search for the most suitable test environment to simulate the two scenarios described before.

The first step was the creation of the containers for both the servers and the clients with the use of the Flower framework.

The second step was the implementation of the centralized LSTM algorithm, described in 3.4, inside the Flower framework, in order to perform ML in a distributed way.

The third step was one of the most important because it was the key to understand which parameters (number of epochs, number of rounds, dataset dimension) were suitable to build the vehicular scenarios. We started using a single FL cycle to test the number of epochs needed to have a significant improvement on the algorithm. The goal was to find a correlation between the number of epochs and the MED value. As represented in figures 5.1 5.2 5.3, different datasets with different dimensions were tested using rounds with a different number of epochs. The result to highlight is that the goodness of the MED at the end of a round does not improve proportionally with the number of epochs.

These experiments concluded that by increasing the number of epochs, the precision of the algorithm does not increase proportionally. The explanation of this behaviour comes out using the early stopping function described in section 4.4. With this function the round is stopped usually after two epochs, this means that there are no significant improvements in the model after the second epoch. In other words, after the first two rounds, the model has exploited all the features available in that batch of the dataset. Given the aforementioned considerations, was decided to use the early stopping function fixing the maximum number of epochs to 10, as a wide margin.

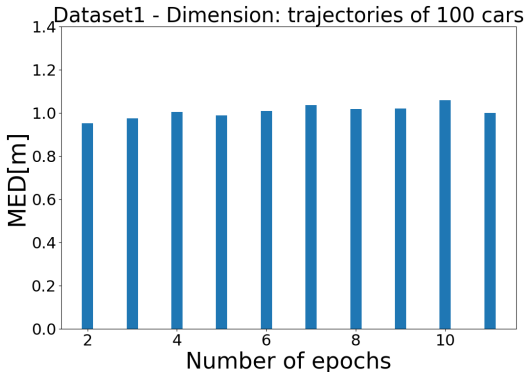

Figure 5.1: Test for the number of rounds: Dataset1

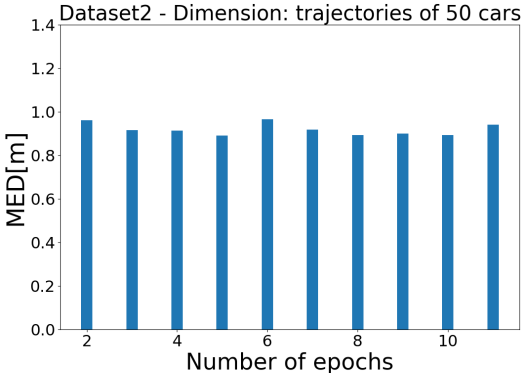

Figure 5.2: Test for the number of rounds: Dataset2

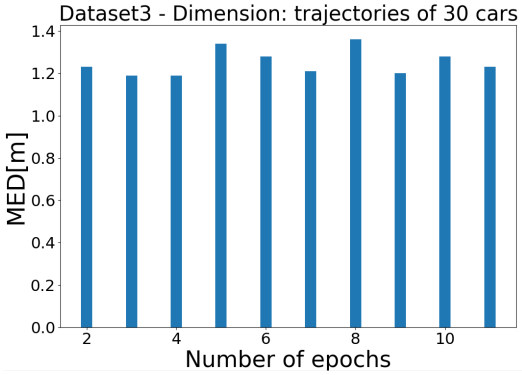

Figure 5.3: Test for the number of rounds: Dataset3

For the maximum number of rounds, the decision criteria were the same and as described in section 4.4, the early stopping was used with a margin of a maximum number of rounds equal to 10.

The last test of the third step was to find the correct dataset dimension. As explained in section 3.3, the suitable number of vehicles' trajectories is equal to 30. In figures 5.4 and 5.5 there are 2 examples of tests performed fixing the same environmental conditions but varying the dimension of the client's dataset. The common environmental conditions were:

- 2 CPUs per client
- Dataset of 5105 vehicles' trajectories to divide into subsets
- Early stopping for epochs and rounds

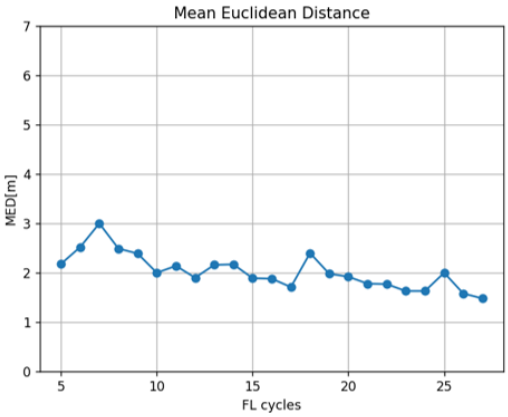

Figure 5.4: FL test with dataset of 30 vehicles' trajectories

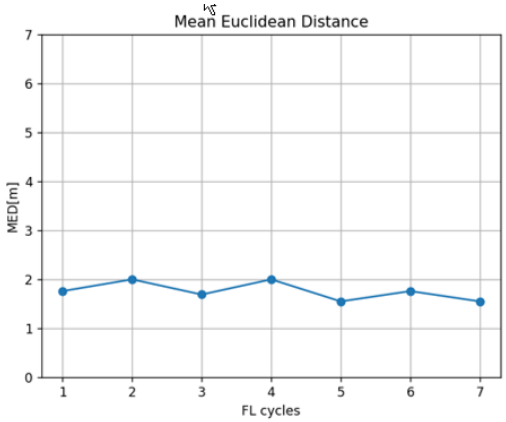

Figure 5.5: FL test with dataset of 120 vehicles' trajectories

Average results' value of simulation 5.4		
Time per round	Number of rounds per FL cycle	FL cycle time
50 seconds	2.3	3 minutes

Table 5.1: Main simulation's results with dataset of 30 vehicles' trajectories

Average results' value of simulation 5.5		
Time per round	Number of rounds per FL cycle	FL cycle time
190 seconds	4.7	15 minutes

Table 5.2: Main simulation's results with dataset of 120 vehicles' trajectories

The first thing to notice in the figures 5.4 and 5.5 is that the number of FL cycles performed is very different because the same amount of data was divided into subsets of 30 vehicles' trajectories in the first case and in subsets of 120 vehicles' trajectories in the second case. Therefore, the first case has more available datasets to perform consecutive FL cycles. However, the two simulation reaches approximately the same MED, 1.48 m for the first case and 1.55 m for the second. The model is trained with the same amount of data and reaches a similar accuracy, then we can choose the dimension of the subsets suitable for our scenario without losing the model accuracy. For the purpose of the work, we are interested in the case in which a single round does not exceed 1 minute, as explained in section 3.3. Looking at the tables 5.1 and 5.2, in particular at the *Time per epoch* section, we can consider the first case, with a subset of 30 vehicles' trajectories per each client, as the most suitable case to satisfy the simulation time requirements. The second one presents a round time and, in general, the simulation's parameters time too long to be adaptable to a dynamic scenario because this timing implies the presence of identical vehicles for a long time under the described coverage area, which is more difficult to obtain, leading to a lower number of vehicles available for the FL training.

The fourth step was the creation of the two scenarios. We started with the simplest one, the synthetic. The Flower source code was modified as described in section 4.6.1 and all the containers were updated with this new version to perform the final tests for this scenario. Then the real-world scenario was created by modifying the Flower open source as described in section 4.6.2 and the same tests of the synthetic scenario were performed.

In this work, both the synthetic and real-world scenarios were tested in three different conditions, varying the maximum number of federated participants in an FL round. Given the computation constraints described in 3.2.1, the three scenarios were characterized by 2,4 and 6 maximum clients per FL round. The first noteworthy result to analyze is the one in Fig. 5.6 where the scenario with the real-world settings spends less time to reach the imposed MED with respect to the scenario with the synthetic settings.

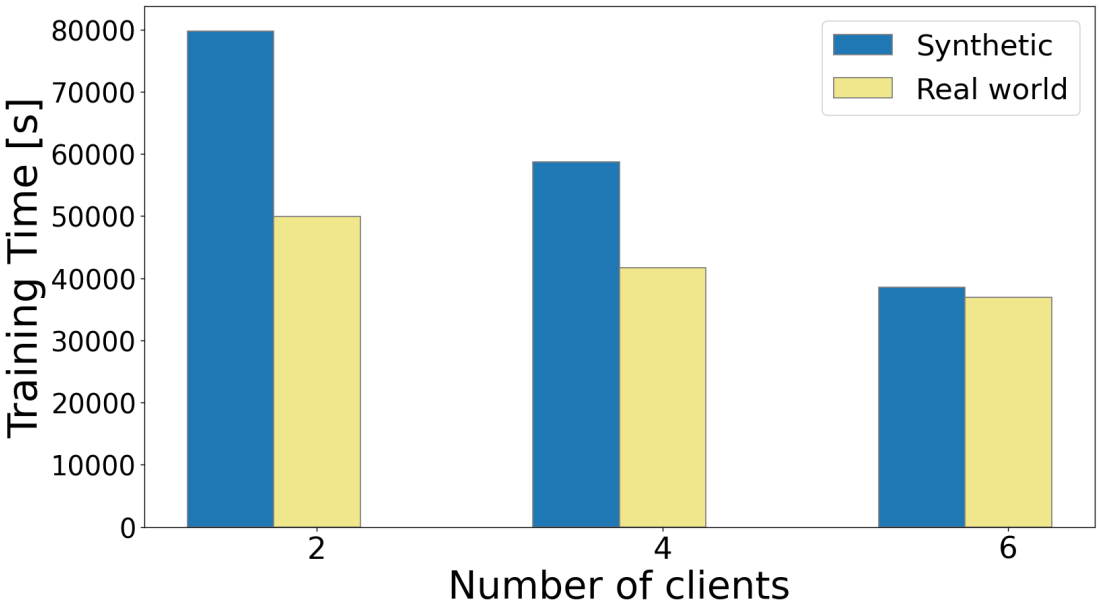

Figure 5.6: Training time vs. number of clients (A.2)

This results become more clear analyzing in detail the statistics regarding the number of FL cycles, FL rounds and the quantity of data used during an FL simulation. Moving to Fig. 5.7 the trend of the Fig. 5.6 can be better understood. Indeed, the cases with real world settings require less time to reach the model accuracy because they perform fewer FL cycles.

The aforementioned time graph 5.6 is not completely justified without specifying how many FL rounds compose an FL cycle. Performing fewer FL cycles does not necessarily lead to a shorter simulation time, because an FL simulation with $\boldsymbol{K}$ FL cycle and $\boldsymbol{N}$ FL rounds can take more time than an FL simulation with less than $\boldsymbol{K}$ FL cycle that performs more than $\boldsymbol{N}$ FL rounds. The Fig. 5.8 shows that, with the real world settings, an FL cycle needs fewer FL rounds than the one with the synthetic settings, further justifying the reason for the lower execution time in the real-world scenario.

The explanation for the fewer FL rounds, FL cycles and execution time needed to the real-world scenarios to reach the imposed MED can be found in Fig. 5.9.

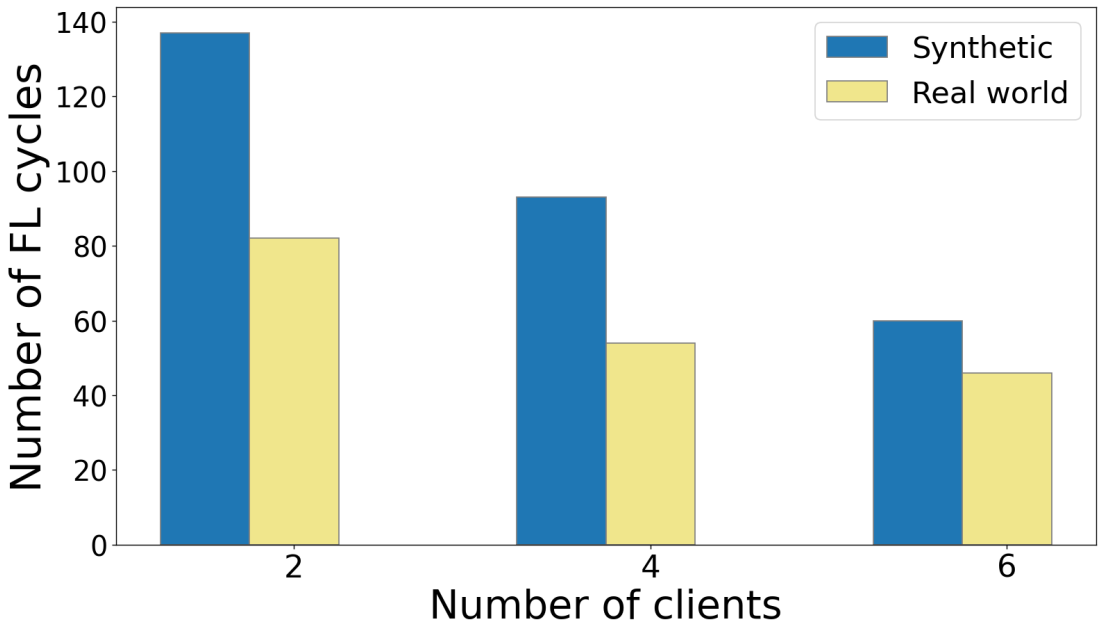

Figure 5.7: Number of FL cycles vs. number of clients (A.3)

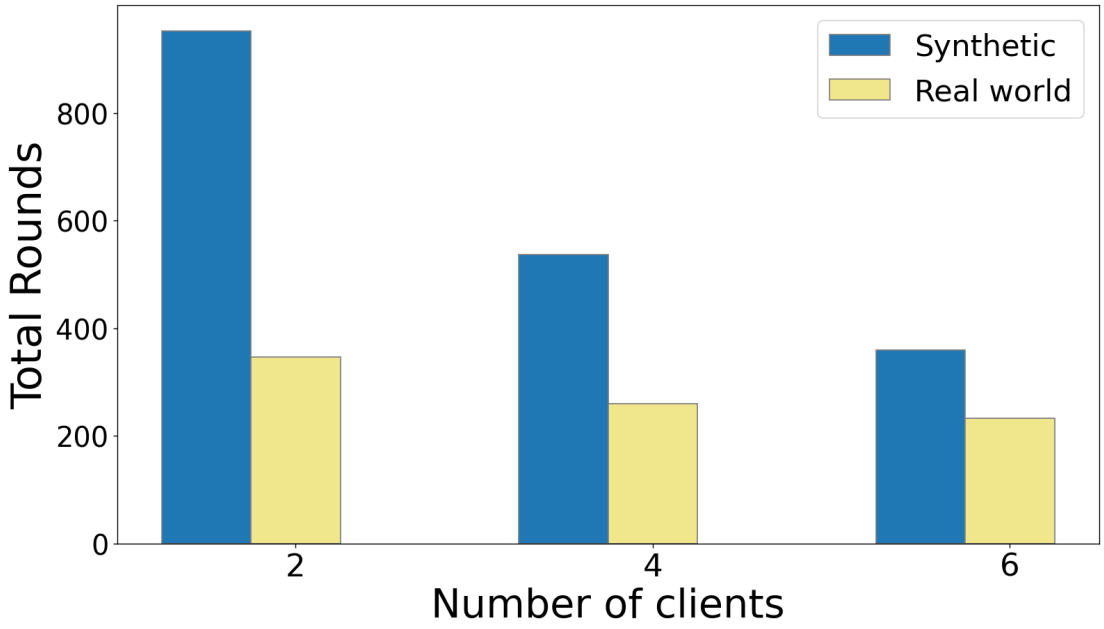

Figure 5.8: Number of FL rounds vs. number of clients (A.4)

This figure represents the number of vehicles that take part in an FL cycle. For the synthetic scenario, this value is equal to the maximum number of participants in

an FL round, because as explained in 4.6.1, a group of clients is used for an entire FL cycle.

In real-world scenarios the number of clients taking part in an FL cycle increase proportionally to the maximum number of federated participants in an FL round. This last aspect can be simply explained considering that for ***N*** consecutive rounds, the first case of FL simulation (maximum number of FL participants fixed to 2) can exploit at most ***N*2** clients with respect to the ***N*6*** of the last case.

The graph in Fig. 5.9 is the most important because contains the reason for the better results of the real-world scenario with respect to the synthetic one. The higher replacement rate of the former scenario is linked to the use of the mobility trace. As detailed described in 4.6.2, at each FL round, a row of the LOG file is read in order to manage the choice of the new clients or to confirm the use of the previous ones. Therefore, given the results in Fig. 5.9, it turns out that, when the mobility trace is followed, the choice of a new group of clients at each round, rather than the confirmation of the one used in the previous round, is a more frequent operation. This means that, in the real-world scenario, the group of clients is chosen more round by round rather than FL cycle by FL cycle. It is important to remember that each vehicle or client corresponds to a data set, changing the client, the data set will change. The higher quantity of different data sets used in an FL cycle in the real-world scenario allows to train of the model with a wider variety of information and this leads to obtaining a lower MED at the end of the FL cycle when the evaluation of the model is performed in the server side. Therefore, the model, being trained with more trajectories examples, can better predict the ones of the evaluation data set, stopping the simulation in less time.

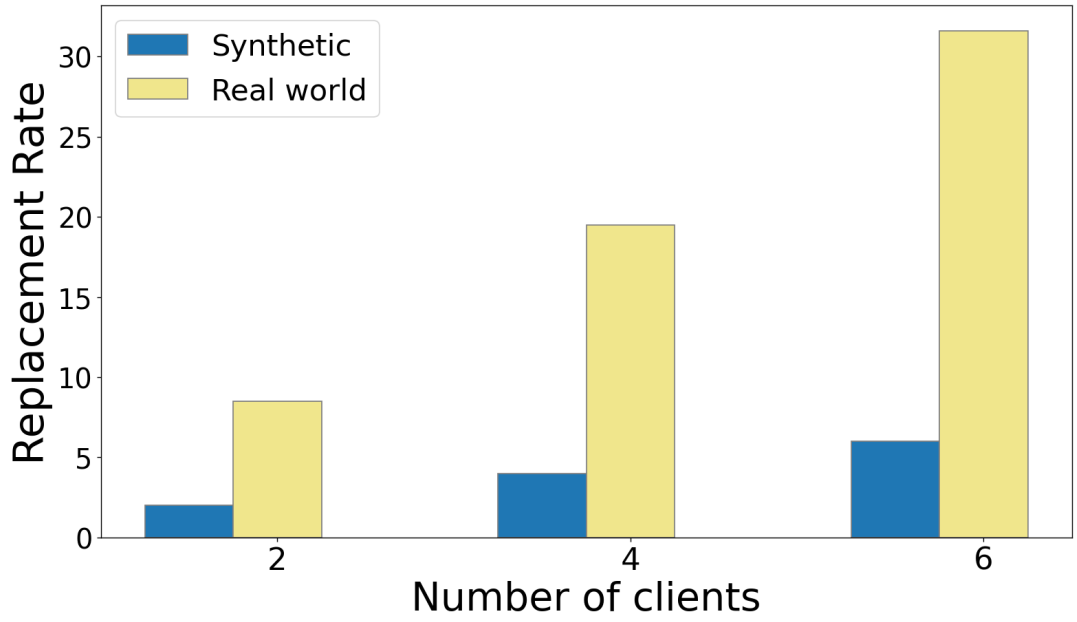

Figure 5.9: Replacement rate vs. number of clients (A.5)

With a general view of all the four graphs just described, the inversely proportional behaviour of the graphs can be seen in 5.6, 5.7 and 5.8 with respect to the replacement rate graph 5.9. This happens because using a bigger number of maximum clients, the model is trained with a wider variety of data samples during the entire simulation, which leads to a more performing and accurate model in fewer FL rounds, fewer FL cycles and then in less time.

The replacement rate motivates the most accentuated gap that is found in Fig. 5.6, in the case with two clients with respect to the case with six clients. This happens because in the case of six clients, even if the data sets are not refreshed every round, the clients' group has enough data to consistently improve the model even during the consecutive rounds. Instead, in the case of two clients happens that the group exploits all the available data sets' features in very few rounds, leading to the need for new data sets to perform consistent model improvement in subsequent rounds. In the synthetic case, the clients' group does not have the opportunity to exploit new features in an FL cycle, then the model improvement is delayed, leading to a bigger simulation time.

Chapter 6

Conclusions

In the next future, several solutions with the aim to improve the safeness and the livability of the urban environment will exploit more direct communication between vehicles and servers located at the edge of the network. These servers can be used to process the data collected by vehicles through ML neural networks, giving back useful ML models to improve the driving experience. The main problems of this solution are the data privacy that can be violated if the information going toward the server is sniffed by malicious subjects and the network congestion considering that in a crowded urban environment several cars need to send a huge amount of information at the same time exploiting the same network.

Chapter 1 consists of an introduction to the main problems and the relative solution analyzed in the thesis. How federated learning can be exploited to remove the need of uploading driving data from vehicles to the edge network. This work examined the effect of real urban mobility behaviour in a federated learning session where a trajectory prediction NN model was trained.

In Chapter 2 the technologies that can give a positive contribution to the urban environment improvement, such as Machine Learning, IoT-edge-cloud continuum, container-based virtualization, Flower, ms-van3t and trajectory prediction algorithm were described in detail.

Chapters 3 and 4 describe the used architecture to build a new FL framework and its implementation. Exploiting ms-van3t [2] and FLOWER [3] frameworks, we had the possibility to modify the behaviour of federated clients considering real urban mobility traces that for this work were taken from the TuST [45] scenario. In order to compare the results coming from the vehicular scenario, parallel tests were made with a different scenario in which clients do not follow mobility traces but have a more static behaviour. Furthermore, in this parallel scenario, the clients'

group changes less frequently and then contributes to federated learning with fewer data samples in an FL simulation.

The results, as described in detail in Chapter 5, clearly show how the dynamic urban mobility environment can be exploited to efficiently contribute to the training of the NN model for a trajectory prediction application, thus decreasing the workload of the edge-server and drastically reducing the amount o bandwidth used by vehicles to exchange data with the edge-server.

6.0.1 Future works

With the fast increase in the population in the cities, it is needed to improve traffic management technologies using new methods such as the creation of a neural network model to predict bad behaviours of the drivers and increase the safety of pedestrians and all drivers.

This thesis work can be considered as a contribution to the creation of a framework to test federated learning in the urban mobility environment, using important frameworks such as Flower and ms-van3t.

In future works, the same scenarios proposed in this thesis work can be tested with a higher number of clients to see if the decreasing trend of the simulation time, seen in Fig. 5.6, is kept constant. Another important aspect to improve is the strategy to select clients and aggregate the model on the server side. In this work was used the simplest and most common strategy, the FedAvg [14]. Other strategies can be more suitable for the vehicular scenario, as the [16] that selects clients based on their resource's state, this can be very useful given that user equipment varies depending on the car maker. Another interesting proposal is the [15] that trains the NN model just partially at the server, avoiding an imbalanced training due to conflicting models' gradients with wide differences in the magnitudes. For what concerns the network simulation, the ms-van3t [2] framework can be exploited to study the dynamic change of bandwidth due to mobility, along the simulation area.

Appendix A

Most significant simulation results of the FL framework

In this chapter are reported important values that were necessary to take the final decision for the implementation of the framework A.0.2 and to create the graphs for the final results analysis A.0.3

A.0.1 Flower parameters

Given the FL framework settings below, were derived the main parameters, reported in the table, to understand how to implement the final version of the FL framework.

- 2 CPUs per client
- Data set dimension of 30 vehicles' trajectories
- Max number of FL rounds and FL cycles equal to 10
- MED threshold 1.3 m

Average results' value				
Epochs per round	Time per epoch	Rounds per cycle	Time per round	Time per FL cycle
2	25 s	3.3	50 s	4 min 20 s
(max: 2) (min: 2)	(max: 65 s) (min: 7 s)	(max: 10) (min: 2)	(max: 130 s) (max: 14 s)	(max: 18 min) (min: 1 min)

Table A.1: Average simulation's value

A.0.2 Network parameters

The figure A.1 represents one of the analyses done with Wireshark [49], to understand the dimension of the packets, containing the model parameters, exchanged between server and clients. Experimentally we have seen that a package, sent by clients, is 0.835 Mbyte/s. The package sent by the server is always N*0.835 Mbyte/s, where N is the number of clients participating to the FL round.

Figure A.1: Network analysis of model parameters exchange.

A.0.3 Numerical values of the final results of FL framework

This subsection presents the values used to build the final results' graphs in 5.

This table presents the simulation time obtained in both real-world and synthetic scenarios, varying the number of clients.

Average results' value			
Number of clients	2	4	6
Synthetic scenario	79740 s	58762 s	38542 s
Real-world scenario	49958 s	41666 s	36947 s

Table A.2: Training time vs. number of clients (5.6)

This table presents the average number of FL cycles needed to reach the given MED. The results are obtained in both real-world and synthetic scenarios, varying the number of clients.

Average results' value			
Number of clients	2	4	6
Synthetic scenario	137	93	60
Real-world scenario	82	54	46

Table A.3: Number of FL cycles vs. number of clients (5.7)

This table presents the average number of FL rounds needed to reach the given MED. The results are obtained in both real-world and synthetic scenarios, varying the number of clients.

Average results' value			
Number of clients	2	4	6
Synthetic scenario	952	537	359
Real-world scenario	346	260	232

Table A.4: Number of FL rounds vs. number of clients (5.8)

This table presents the average number of vehicles that participate in an FL cycle. The results are obtained in both real-world and synthetic scenarios, varying the number of clients.

Average results' value			
Number of clients	2	4	6
Synthetic scenario	2	4	6
Real-world scenario	8.5	19.5	31.6

Table A.5: Replacement rate vs. number of clients (5.9)

Bibliography

[1] Michael Behrisch, Laura Bieker, Jakob Erdmann, and Daniel Krajzewicz. «SUMO – Simulation of Urban MObility: An Overview». In: *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation* (Oct. 2011). Ed. by SINTEF & University of Oslo Aida Omerovic, RTI International - Research Triangle Park Diglio A. Simoni, and RTI International - Research Triangle Park Georgiy Bobashev.

[2] Marco Malinverno, Francesco Raviglione, Claudio Casetti, Carla-Fabiana Chiasserini, Josep Mangues-Bafalluy, and Manuel Requena-Esteso. «A Multi-Stack Simulation Framework for Vehicular Applications Testing». In: Association for Computing Machinery, 2020.

[3] Daniel J. Beutel et al. «Flower: A Friendly Federated Learning Research Framework». In: (2020).

[4] World Health Organization. «Global status report on road safety 2018. Geneva: World Health Organization; 2018. Licence: CC BYNC-SA 3.0 IGO.» In: (2018).

[5] Stéphanie Bouckaert, Araceli Fernandez Pales, Christophe McGlade, Uwe Remme, Brent Wanner, Laszlo Varro, Davide D'Ambrosio, and Thomas Spencer. «Net Zero by 2050: A Roadmap for the Global Energy Sector». In: (2021).

[6] ETSI. *Intelligent transportation systems*. URL: `https://www.etsi.org/technologies/automotive-intelligent-transport`.

[7] ETSI. *Multi-access Edge Computing (MEC)*. URL: `https://www.etsi.org/technologies/multi-access-edge-computing`.

[8] ETSI. *Permissioned Distributed Ledger (PDL); Federated Data Management*. URL: `https://www.etsi.org/deliver/etsi_gr/PDL/001_099/009/01.01.01_60/gr_PDL009v010101p.pdf`.

[9] Dinesh Cyril Selvaraj, Christian Vitale, Tania Panayiotou, Panayiotis Kolios, Carla Fabiana Chiasserini, and Georgios Ellinas. «Edge Learning of Vehicular Trajectories at Regulated Intersections». In: (2021).

[10] IBM Cloud Education. *Machine$_L$earning*. 2020. URL: https://www.ibm.com/cloud/learn/machine-learning.

[11] IBM Cloud Education. *Artificial Intelligence*. 2020. URL: https://www.ibm.com/cloud/learn/what-is-artificial-intelligence.

[12] IBM Cloud Education. *Deep Learning*. 2020. URL: https://www.ibm.com/cloud/learn/deep-learning.

[13] Qiang Yang, Yang Liu, Yong Cheng, Yan Kang, Tianjian Chen, and Han Yu. «Federated learning». In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* (2019).

[14] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. «Communication-Efficient Learning of Deep Networks from Decentralized Data». In: *arXiv* ().

[15] Manoj Ghuhan Arivazhagan, Vinay Aggarwal, Aaditya Kumar Singh, and Sunav Choudhary. *Federated Learning with Personalization Layers*. 2019.

[16] Takayuki Nishio and Ryo Yonetani. «Client selection for federated learning with heterogeneous resources in mobile edge». In: IEEE. 2019.

[17] Li Li, Moming Duan, Duo Liu, Yu Zhang, Ao Ren, Xianzhang Chen, Yujuan Tan, and Chengliang Wang. «FedSAE: A Novel Self-Adaptive Federated Learning Framework in Heterogeneous Systems». In: 2021.

[18] Alexander Brecko, Erik Kajati, Jiri Koziorek, and Iveta Zolotova. «Federated Learning for Edge Computing: A Survey». In: *Applied Sciences* (2022).

[19] Francois Chollet et al. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[20] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[21] Adam Paszke et al. «PyTorch: An Imperative Style, High-Performance Deep Learning Library». In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[22] Fabian Pedregosa et al. «Scikit-learn: Machine learning in Python». In: *Journal of machine learning research* 12.Oct (2011), pp. 2825–2830.

[23] Google. *gRPC*. URL: https://grpc.io/.

[24] IBM. *IBM Federated Learning*. URL: https://www.ibm.com/docs/en/cloud-paks/cp-data/4.5.x?topic=models-federated-learning.

[25] Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. *RLlib: Abstractions for Distributed Reinforcement Learning*. 2017.

[26] Ju Ren, Deyu Zhang, Shiwen He, Yaoxue Zhang, and Tao Li. «A Survey on End-Edge-Cloud Orchestrated Network Computing Paradigms: Transparent Computing, Mobile Edge Computing, Fog Computing, and Cloudlet». In: *ACM Comput. Surv.* (2019).

[27] Giuseppe Avino, Paolo Bande, Pantelis A. Frangoudis, Christian Vitale, Claudio Casetti, Carla Fabiana Chiasserini, Kalkidan Gebru, Adlen Ksentini, and Giuliana Zennaro. «A MEC-Based Extended Virtual Sensing for Automotive Services». In: *IEEE Transactions on Network and Service Management* (2019).

[28] Liang Li, Yunzhou Li, and Ronghui Hou. «A Novel Mobile Edge Computing-Based Architecture for Future Cellular Vehicular Networks». In: (2017).

[29] Vitor Goncalves da Silva, Marite Kirikova, and Gundars Alksnis. «Containers for virtualization: An overview». In: *Applied Computer Systems* (2018).

[30] Dirk Merkel. «Docker: lightweight linux containers for consistent development and deployment». In: *Linux journal* 2014.239 (2014), p. 2.

[31] Linux containers. *Linux containers LXC*. URL: `https://linuxcontainers.org/lxc/introduction/`.

[32] «IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 6: Wireless Access in Vehicular Environments». In: *IEEE Std 802.11p-2010 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, IEEE Std 802.11n-2009, and IEEE Std 802.11w-2009)* (2010).

[33] Apostolos Papathanassiou and Alexey Khoryaev. «Cellular V2X as the essential enabler of superior global connected transportation services». In: *IEEE 5G Tech Focus* (2017).

[34] Society of Automotive Engineers. *SAE Levels of Driving Automation™ Refined for Clarity and International Audience*. URL: `https://www.sae.org/blog/sae-j3016-update`.

[35] Simon Wright Stan Dmitriev. «Autonomous cars generate more than 300 TB of data per year». In: (2021).

[36] M Simon, Thierry Hermitte, and Yves Page. «Intersection road accident causation: A European view». In: *21st International Technical Conference on the Enhanced Safety of Vehicles*. 2009.

[37] Florin Leon and Marius Gavrilescu. «A Review of Tracking and Trajectory Prediction Methods for Autonomous Driving». In: *Mathematics* (2021).

[38] C. Casetti et al. «ML-driven Provisioning and Management of Vertical Services in Automated Cellular Networks». In: *IEEE Transactions on Network and Service Management* (2022).

[39] eclipse. *SUMO*. URL: https://www.eclipse.org/sumo/.

[40] Axel Wegener, Michał Piórkowski, Maxim Raya, Horst Hellbrück, Stefan Fischer, and Jean-Pierre 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.

[41] GNU GPLv2. *Network simulator*. URL: https://www.nsnam.org/about/.

[42] Alexander Kröller, Dennis Pfisterer, Carsten Buschmann, Sándor P Fekete, and Stefan Fischer. «Shawn: A new approach to simulating wireless sensor networks». In: *arXiv preprint cs/0502003* (2005).

[43] *Java in Simulation Time / Scalable Wireless Ad hoc Network Simulator*. URL: http://jist.ece.cornell.edu/.

[44] *OMNeT++*. URL: https://omnetpp.org/.

[45] Marco Rapelli, Claudio Casetti, and Giandomenico Gagliardi. «TuST: from Raw Data to Vehicular Traffic Simulation in Turin». In: (2019).

[46] Open Street Map. URL: https://www.5t.torino.it/.

[47] Cell Mapper. URL: https://www.cellmapper.net/map?MCC=222&MNC=1&type=LTE&latitude=0&longitude=0&zoom=11&showTowers=true&showIcons=true&showTowerLabels=true&clusterEnabled=true&tilesEnabled=true&showOrphans=false&showNoFrequencyOnly=false&showFrequencyOnly=false&showBandwidthOnly=false&DateFilterType=Last&showHex=false&showVerifiedOnly=false&showUnverifiedOnly=false&showLTECAOnly=false&showENDCOnly=false&showBand=0&showSectorColours=true&mapType=roadmap&darkMode=false.

[48] Heiko Ludwig et al. «IBM Federated Learning: an Enterprise Framework White Paper V0.1». In: *CoRR* abs/2007.10987 (2020).

[49] Wireshark. URL: https://www.wireshark.org/about.html.