Politecnico di Torino

Master of science in Energy Engineering

Master Thesis

Politecnico di Torino

An AI-Based Speed Forecasting Algorithm for Efficient Energy Management in Hybrid Electric Vehicles

Supervisor: Prof. Luciano Rolando
Co-supervisor: Ing. Luigi Tresca

Candidate: Mario Robino

November 2025

Index

Index of the Figures

Abstract

Nowadays, powertrain hybridization represents one of the most viable solutions to reduce the carbon footprint of the transport sector. Nevertheless, its potential can be fully exploited only through the design of suitable Energy Management Strategies capable of optimally coordinating the multiple power sources installed onboard. In this context, accurate speed forecasting, enabled by increasing levels of vehicle connectivity, can support the development of predictive control strategies that further enhance the benefits of hybridization through a more efficient optimization of energy flows.

Therefore, this thesis work investigates the use of predictive artificial intelligence for energy management enhancement through vehicle speed forecasting. Specifically, the study employs a Long Short-Term Memory (LSTM) neural network trained on synthetic traffic data generated with the open-source software SUMO (Simulation of Urban Mobility), simulating a typical working day in the city of Turin.

The dataset was derived from SUMO's mesoscopic mode and processed through a Python pipeline for feature extraction and MATLAB scripts for filtering, normalization, and LSTM sequence preparation and for the training of the networks themselves. Several network configurations were trained and evaluated using RMSE and MAE metrics to identify the optimal hyperparameters.

The best-performing model, trained on data from 2000 vehicles, achieved an average RMSE of approximately 0.283 [m/s] and an MAE of 0.240 [m/s] on 80-second sequences, demonstrating the ability to reproduce general speed trends with satisfactory accuracy. Additional tests on 140-second sequences confirmed the model's capacity to capture temporal dependencies, though challenges remain for sequences with weak time correlations. Future developments may focus on extending the dataset to microscopic simulations to better reflect real urban driving behavior and further enhance prediction accuracy for energy optimization purposes.

1 Introduction

Global issues such as global warming and pollution in large urban centers require solutions that make mobility more efficient to reduce emissions, consume less energy, and shorten travel times.

Such need for efficient mobility has led to multiple approaches to modelling traffic dynamics [1]. The aim is to enable researchers to perform experiments without involving real street traffic, thereby reducing costs and risks [2].

Consider relatively recent topics such as autonomous driving or the management of smart traffic lights for congestion control: running experiments in the real world can be risky when solutions are not yet fully validated [1].

Traffic simulations can also be an excellent tool for urban planners, through which they are able to evaluate the impact on road flow of certain choices such as the addition or closure of a road section [2].

Simulating traffic can be a hard task, considering that urban traffic is a very complex system made of thousands of vehicles, each of them driven by different people with their own habits and driving styles.

Also, from an infrastructural viewpoint an urban scenario might be a very large and intricate network made of a huge number of routes and nodes.

To add further complexity, in real urban scenario not all vehicles are of the same kind, considering that aside from private cars there are also buses, cycles, trams and pedestrians; each of them has its own typical way to travel around the network and this should be considered.

1.2 Goal of the thesis

The goal of this thesis is to create an LSTM neural network capable of predicting a vehicle's speed profile. The network will be trained using data extracted from a traffic simulation software called SUMO, which will allow for the provision of data from 2,000 vehicles participating in the simulation.

The following document will be structured to introduce all the elements necessary to connect the two worlds involved in this work. Therefore, the second chapter will provide an overview of the main traffic models, how they are divided, the theoretical framework underlying them, and how they are exploited in today's world. Following this discussion, the third chapter will focus on SUMO, its characteristics, and properties, with a particular focus on the Turin scenario that will be used.

The fourth chapter moves on to the world of Artificial Intelligence and Machine Learning; again, starting from general theory, we will focus on the LSTM network, the protagonist of this study.

The next chapter will explain the methodology used, the decisions made to select the features, the postprocessing phase will be analyzed, and the choice of hyperparameter values will be explained through a sensitivity analysis.

Finally, the results obtained from the training of the LSTM network will be tested, different approaches will be compared, and the concept of autoregressive and multi-step autoregressive prediction will be explored in depth.

2 Traffic Modelling

This complexity has led to different kinds of models over the years, each of these is more suitable depending on the type of application studied. For less detailed cases *Macroscopic models* are usually exploited; these models require less computational costs and can handle large regions like cities or districts. When it comes to simulate more detailed scenarios *Microscopic models* are the best choice; they can handle many dynamics and different kind of vehicles. The obvious downside lies in the computational cost which is high and does not allow to consider large scenarios [3].

A useful trade-off are *Mesoscopic models*, that try to take advantage of both models and minimize the disadvantages; the results are simulation tools and software able to handle large scenarios with enough detail degree. *SUMO, Simulation of Urban Mobility*, is an example of software that implements this model [4].

2.1 Macroscopic models

Macroscopic modelling relies on deterministic relationships between flow, speed, and density on a traffic stream. In this case, the simulation proceeds section by section and does not track the behavior of individual vehicles.

Macroscopic traffic models share many similarities with hydrodynamics [5] and are based on three macroscopic quantities:

- Traffic density ρ $[veh/km]$
- Traffic flow Q $[veh/h]$
- Local speed V $[km/h]$

These quantities are related by the *hydrodynamic relation*

$$Q = \rho V \tag{2.1}$$

The *continuity equation* also applies to traffic

$$\frac{\partial \rho}{\partial t} + \frac{\partial Q}{\partial x} = 0 \tag{2.2}$$

In equation (2.2) are written in differential form with respect to time (t) and space (x) and with the hydrodynamic relation the system can be rewritten accordingly

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho V)}{\partial x} = 0 \tag{2.3}$$

In this framework, two reference viewpoints are common: *Eulerian* and *Lagrangian*. In the Eulerian view, one observes aggregate properties at fixed locations. In the Lagrangian

view, one follows an individual entity (e.g., a vehicle) along its path and monitors how the relevant properties change.

Traffic is said to be *homogeneous* when density is constant over space, and *stationary* when density does not change over time

$$\frac{\partial \rho}{\partial x} = 0 \ (homogeneity\ condition) \tag{2.4}$$

$$\frac{\partial \rho}{\partial t} = 0 \ (stationariety\ condition) \tag{2.5}$$

2.1.1 Kinematic wave model

To solve the system formed by the continuity and hydrodynamic relations.

$$\begin{cases} Q = \rho V \quad (hydrondynamic\ equation) \\ \dfrac{\partial \rho}{\partial t} + \dfrac{\partial (\rho V)}{\partial x} = 0 \ (continuity\ equation) \end{cases}$$

A third *equation* is required because three variables describe the system. Although flow is generally a function of time and space, it can be shown that it is also a function of traffic density. This dependency is known as the *traffic-stream relation* and can be represented by the *fundamental diagram* reported in Figure 1.

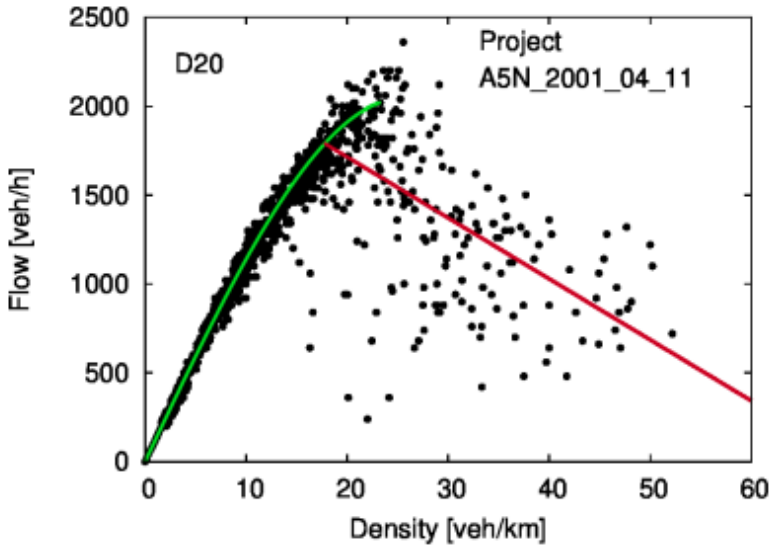

Figure 1 In fundamental diagram traffic flow is shown as a function of density. Along green line traffic conditions are uncongested. On the other hand, red line means congested conditions [6].

With the traffic-stream relation, the system becomes three equations in three variables:

$$\begin{cases} Q = \rho V & (hydrodynamic\ equation) \\ \dfrac{\partial \rho}{\partial t} + \dfrac{\partial (\rho V)}{\partial x} = 0 & (continuity\ equation) \\ Q(x,t) = Q_e\big(\rho(x,t)\big) & (traffic - stream\ relation) \end{cases}$$

leading to the *Lighthill–Whitham–Richards (LWR)* model:

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho V)}{\partial x} = \frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x}\big(Q_e(\rho)\big) = \frac{\partial \rho}{\partial t} + Q_e'(\rho)\frac{\partial \rho}{\partial x} = 0 \qquad (2.6)$$

Differential equations are the mathematical tool par excellence for modeling physical systems and can be of zeroth, first, or second order. The order is established based on the highest order of derivation present in the equation. Leaving aside the zeroth order (which models constant phenomena of little interest in the case in question), in the other two cases the numerical solution strategies change depending on the order.

2.1.2 Model solutions

The kinematic wave model is a first-order partial differential equation and is used to describe propagation phenomena along one dimension. The model assumes that the traffic density ρ in the equation (2.6) propagates like a wave with a speed called *characteristic speed* $c(\rho)$, that in the equation (2.6) is identified as $Q_e'(\rho)$.

Solutions are typically represented on a Cartesian plane with time on one axis and space on the other. Lines of constant characteristic speed are called *characteristic curves* and make it possible to evaluate how a specific traffic feature, such as density, propagates over time and space.

In the kinematic model, traffic speed is always greater than the characteristic speed, which means traffic information travels more slowly than the traffic itself [6]. For example, on a wide road where vehicles run at constant speed, changes in density cannot propagate faster than the vehicles that generate them.

In this model traffic behaves like an anisotropic fluid, events happening behind a vehicle cannot have influence on the vehicle itself.

For *spatial discretization*, two cases are used:

- Downwind finite differences for free traffic, where information propagate downstream
- Upwind finite differences for congested traffic, where information propagates upstream

For *time discretization*, an explicit first-order scheme is often adopted:

- Explicit numerical scheme, where the only information needed are the present and past ones

commonly *forward Euler*

$$\rho_{k,j+1} = \rho_{k,j} - \frac{\Delta t}{\Delta x} * \begin{cases} \left(Q_{k-1,j} - Q_{k,j}\right) & (free\ flow) \\ \left(Q_{k+1,j} - Q_{k,j}\right) & (congested\ flow) \end{cases} \quad (2.7)$$

$$Q_{k,j+1} = Q_e(\rho_{k,j+1}) \quad (2.8)$$

To ensure convergence, the *Courant–Friedrichs–Lewy (CFL)* condition must be satisfied.

$$\Delta t \leq |c|_{max} \Delta x = V_0 \Delta x. \quad (2.9)$$

First-order systems cannot capture some complex dynamics typical of real traffic like acceleration and deceleration of the vehicles; hence *second-order models* have been developed.

A representative equation reads:

$$\frac{dV(x,t)}{dt} = \left(\frac{\partial}{\partial t} + V\frac{\partial}{\partial x}\right)V(x,t) + \frac{1}{\rho}\frac{\partial P(\rho)}{\partial x} = A[\rho(x,t), V(x,t)] \text{ (local form)} \quad (2.10)$$

Where:

- $\left(\frac{\partial}{\partial t} + V\frac{\partial}{\partial x}\right)V(x,t)$ is the acceleration of a vehicle
- $\partial P(\rho)$ is the "traffic pressure" generated by variations of speed
- $A[\rho(x,t), V(x,t)]$ is the aggregated acceleration of vehicles.

This approach resembles one-dimensional flow models in fluid dynamics: it exhibits an *advective* term $\frac{\partial V(x,t)}{\partial t} + V\frac{\partial V(x,t)}{\partial x}$ and the analogue of a *pressure gradient* $\frac{\partial P(\rho)}{\partial x}$.

It is well known that the strategy adopted to solve this kind of problem can lead to different numerical errors, for example adopting an upwind scheme can generate an artificial diffusive term or a numerical dispersion, which anticipate or delay the temporal behavior of a phenomenon.

An incorrect solution can compromise the *anisotropy* of traffic flow; upstream events might unrealistically influence vehicles ahead.

Several models and techniques address these issues:

- Payne's Model

$$\frac{\partial V}{\partial t} + V\frac{\partial V}{\partial x} = \frac{V_e(\rho) - V}{\tau} + \frac{V_e'(\rho)}{2\rho\tau}\frac{\partial \rho}{\partial x} \quad (2.11)$$

Here the differential equation is written explicitly also as a function of the velocity V, unlike the first order case where the latter was deduced by the traffic-stream relation. The derivative of speed over time is clearly an acceleration, which guarantees that the model is of the second order.

$V_e(\rho)$ is the speed defined by the same criteria used to get the traffic-stream relation; it also has its own fundamental diagram, showed in Figure 2.

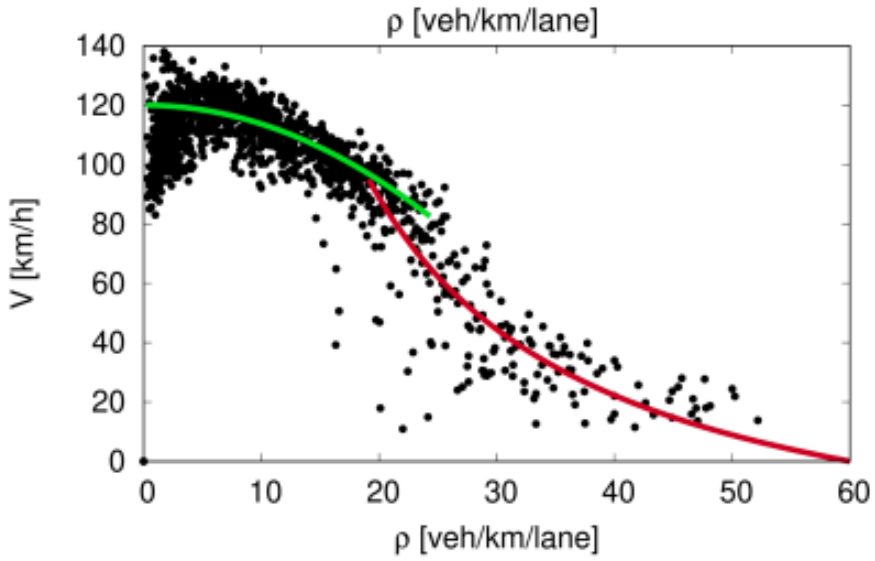

Figure 2 Fundamental diagram of speed [5].

- Kerner-Kohnhauser Model

$$\frac{\partial V}{\partial t} + V\frac{\partial V}{\partial x} = \frac{V_e(\rho) - V}{\tau} - \frac{c_0^2}{\rho}\frac{\partial \rho}{\partial x} + \eta\frac{\partial^2 V}{\partial x^2} \tag{2.12}$$

With respect to Payne's model there is an additional "*speed diffusion term*" η to prevent shock waves; some typical values are

- Relaxation time τ (10-30 s)
- Sonic speed c_0 (15 m/s)
- Speed diffusion η (150 m/s)

When relaxation time is higher than 30 s, flow instabilities are more likely to happen.

- Aw-Rascle Model

$$\frac{\partial}{\partial t}\big(V + p(\rho)\big) + V\frac{\partial}{\partial x}\big(V + p(\rho)\big) = 0 \tag{2.13}$$

This formulation allows for a totally *conservative form*, meaning that under some circumstances there are analytic solutions.

In this case, $p(\rho)$ must not be understood as traffic pressure but as a speed-dependent factor often defined by ARZ model, i.e. $p(\rho) = (V_0 - V_e(\rho))$.

- Gas-Kinetic Based Traffic Flow Model

$$\frac{\partial V}{\partial t} + V\frac{\partial V}{\partial x} + \frac{1}{\rho}\frac{\partial P(\rho)}{\partial x} = \frac{V_e^*(\rho,V,\rho_a,V_a)-V}{\tau} \tag{2.14}$$

Complex but *numerically stable*, in this case an implicit temporal scheme must be implemented.

2.2 Microscopic models

Microscopic approaches track *individual vehicles*, so model variables represent pointwise properties such as position and speed. These models usually provide more detail than macroscopic ones and are particularly effective under *heterogeneous* traffic conditions, which may be classified in increasing complexity as follows:

1. Same model, same vehicle category and same driving style
2. Same model, same category but different driving styles
3. Same model, different categories and different driving styles
4. Different models, different categories and different driving styles

Each driver has a personal driving style and at a modelling level, to try to obtain this variety, the driver's behavior is often considered on three levels:

1. **Operative** level: accelerating, braking and steering
2. **Tactical** level: lane changing, entering priority roads and other choice tasks
3. **Strategic** level: route choice

Numerous microscopic models have been developed, each of which favors a certain dynamics or scale [6]. In this discussion will be explored in depth Car-Following and Driving Strategies Based models.

2.2.1 Car-Following models

Car following models focus on simulating the dynamics between two vehicles traveling in front of each other as realistically as possible.

Among microscopic models, *car-following (CF)* models are fundamental: they focus on longitudinal dynamics and treat the *immediate leader* as the reference, like Adaptive Cruise Control

Key variables include:

- Independent variables: speed v_i, gap $s_i = x_{i-1} - x_i - l_{i-1}$ and leading speed $v_{i-1} = v_l$

- x_i increases with driving direction
- Indices i are considered like in racing logic, so $x_{i-1} > x_i$
- Distance headway d_i
- Time headway $\Delta t_i = t_i - t_{i-1}$

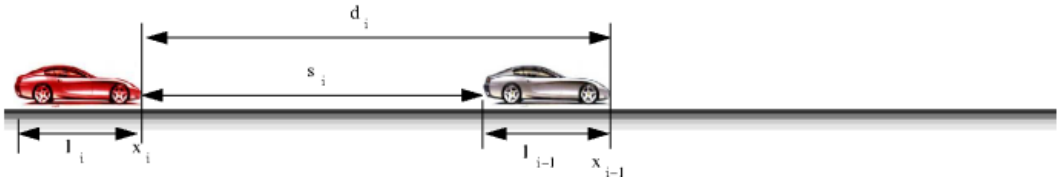

Figure 3 Illustration of spatial parameters [5].

Two useful temporal quantities are the *time gap* $T_i = t_i - t_{i-1} - l_{i-1}/v_{i-1}$ and the *time-to-collision (TTC)* $T^C = s/(v_i - v_{i-1})$, The former is the time interval between leader and follower at a fixed point; the latter considers scenarios where the follower is faster than the leader with no acceleration involved.

A *plausible* and *complete* CF model should exhibit: (i) realistic acceleration in free flow with the desired speed; (ii) a minimum gap in steady state flow (both spatial and temporal); and (iii) a smooth transition to free flow for sufficiently large gaps. In dynamic situations, the model should handle *stopping obstacles* with realistic deceleration and gaps and reproduce *collective phenomena* such as flow instabilities and traffic waves.

Basic Car-Following models alone are not sufficient for effective traffic simulation, but they are useful to illustrate core principles and as a foundation for more sophisticated variants such as the *Optimal Velocity Model (OVM), Full Velocity Difference Model (FVDM), Newell's model*, and *CF cellular automata (CA)* [6].

All these models refine the basic car-following model, for each of them there will be a brief description of their most prominent features and flows.

- **Optimal Velocity Model OVM**

  The model is parameterized by an *optimal-velocity* function

$$v_{opt}(s) = v_0 \frac{\tanh\left(\frac{s}{\Delta s} - \beta\right) + \tanh\beta}{1 + \tanh\beta} \tag{2.15}$$

$$\frac{dv}{dt} = \frac{v_{opt}(s) - v}{\tau} \tag{2.16}$$

  It satisfies plausibility conditions (e.g., it is not directly sensitive to the leader's speed) but can produce unrealistic accelerations and crash responses; τ is the *speed relaxation time*: smaller implies a more responsive driver.

Some elements of equations (2.15) and (2.16) are constants that take conventional values depending on the driving context, some examples are shown in Table 1.

Parameter	City traffic	Highway traffic
Relaxation time τ	0.65 [s]	0.65 [s]
Desired speed v_0	54 [km/h]	120 [km/h]
Form factor β	1.5	1.5
Time gap T	1.2 [s]	1.4 [s]
Minimum distance gap s_0	2 [m]	3 [m]

Table 1 Typical values of the parameters that characterize OVM model

- **Full Velocity Difference Model FVDM**

  Adds sensitivity to *relative speed* $(v - v_l)$:

$$\frac{dv}{dt} = \frac{v_{opt}(s) - v}{\tau} - \gamma(v - v_l) \tag{2.17}$$

  where γ is a *sensitivity factor* (typically ~0.5 s^{-1}).

  Behavior is more realistic than OVM, but the model remains incomplete: as the gap grows large, acceleration still correlates strongly with the leader's speed, which is implausible.

- **Newell's Car-Following Model**

  The follower's position is a function of the leader's position: the follower's trajectory is essentially a *time–space translation* of the leader's trajectory by a vector $\{\tau, \Delta x\}$.

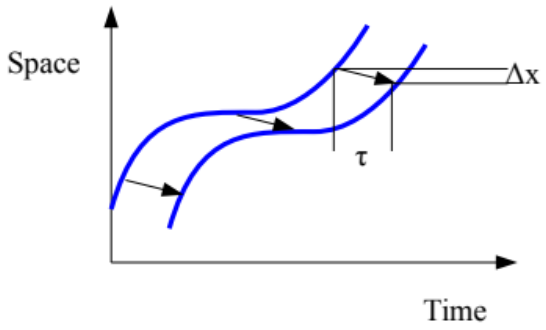

Figure 4 Newell's Car-following model trajectories of leader and follower vehicle [6].

- **Car-Following Cellular Automata**

  Dynamics are expressed via *small integers* on a discretized grid.

Space is discretized into cells and time into intervals Δt and all the other parameters are derived from this first assumption, which means that speed will be $cells/\Delta t$ and acceleration will be $cells/\Delta t^2$. This model can adopt both Euler and Langrangian approach, for this very last case the model falls into Nagel-Schreckenberg model.

2.2.2 Driving Strategies Based Models

CF models alone are limited: e.g., OVM does not avoid collisions; Newell's, CA, and OVM can show unrealistic accelerations ($-9\,m/s^2 \leq \dot{v} \leq 4\ m/s^2$) none distinguishes clearly between *emergencies* and *normal* driving. Models based on driving strategies, such as *Gipps* and *Intelligent Driver Model (IDM)*, address these issues. Models based on driving strategies introduce explicit safety constraints and behavioral rules that prevent unrealistic accelerations and collisions. For instance, Gipps' model imposes a maximum deceleration that guarantees a safe stopping distance even in emergency situations, while IDM uses a continuous desired dynamical gap that differentiates normal following from safety-critical braking. These models incorporate reaction times, bounded accelerations, and anticipation of the leader's dynamics, resulting in smoother trajectories and physically consistent vehicle behavior under both regular and critical driving conditions [6].

- **Gipps' Model**

  Introduces a *reaction time* T for the follower to capture the delay in responding to a braking leader. In highly dynamic situations, the follower's speed depends on T.

  $$v(t+T) = \min\left[v_{free}, v_{safe}\right] \tag{2.18}$$

  Free-flow speed follows $v_{free} = \min\left[v(t) + aT, v_0\right]$, where a is an acceleration term of varying complexity. *Safe speed* is determined from worst-case scenarios to maintain a minimum gap s_0.

  1 Leader brakes until full stop
  2 Follower brakes with deceleration b after a reaction time T. Another "brake hitting time" ϑ is assumed
  3 Constant acceleration starting from $v(t)$ up to v_{safe} during reaction time T, constant speed v_{safe} along ϑ.

  There is a *simplified* version of this model which is based on the following points:

  - *Constant acceleration* a in free-flow regime up to desired speed v_0
  - There is no acceleration during reaction time T and $\vartheta = 0$. By this way it is calculated only the speed value which would avoid a crash in the worst case if

it were adopted instantaneously and held constant during T. Which means that the follower reaction distance is given by $\Delta x_{react} = v(t)T = v_{safe}T$.

- Leader and follower share the *same braking* capabilities $b = b_l$.

All these premises lead to the following quadratic law,

$$v_{safe}^2 + 2bTv_{safe} - v_l^2 - 2b(s - s_0) = 0. \tag{2.19}$$

The final Gipps' model is shown:

$$v(t + T) = \min\left[v + a_{free}(v)T, v_{safe}(s, v, v_l)\right] \tag{2.20}$$

where

$$a_{free}(v) = 2.5a(1 - \frac{v}{v_0})\sqrt{0.025 + \frac{v}{v_0}} \tag{2.21}$$

$$v_{safe}(s, v, v_l) = -b\left(\frac{T}{2} + \vartheta\right) + \sqrt{b^2(\frac{T}{2} + \vartheta)^2 + 2b(s - s_0) + v_l^2\frac{b}{b_l} - vbT}. \tag{2.22}$$

A common value for the braking hitting time is $\vartheta = \frac{T}{2}$, leading to the *Simplified Gipps Model*

$$v(t + T) = \min\left[v + aT, v_0, v_{safe}(s, v_l)\right] \tag{2.23}$$

$$v_{safe}(s, v_l) = -bT + \sqrt{b^2T^2 + 2b(s - s_0) + v_l^2} \tag{2.24}$$

Some typical values for freeway conditions of the main parameters are:

- $v_0 = 35\ [m/s]$
- $a = b = b_l = 1.5\, m/s^2$
- $T = 1.1\ [s]$
- $\vartheta = T/2$
- $s_0 = 2\ [s]$

In urban scenarios the desired speed is reduced.

- **Intelligent Driver Model**

The IDM satisfies most plausibility conditions, supports smooth regime transitions, enables *collision-free* simulations, and uses intuitive parameters mapping to behavioral traits (aggressive/timid, responsive/lazy, slow/fast, anticipative/short-sighted).

For this model, the constitutive equations and most common parameters values are:

$$\frac{dv}{dt} = a[1 - \left(\frac{v}{v_0}\right)^4 - (\frac{s^*(v,v_l)}{s})^2] \tag{2.25}$$

$$s^*(v, v_l) = s_0 + \max(0, vT + \frac{v(v-v_l)}{2\sqrt{ab}}) \tag{2.26}$$

Where $a(1 - \left(\frac{v}{v_0}\right)^4)$ is the *free acceleration* while $a(\frac{s^*(v,v_l)}{s})^2$ is called *repulsive force* which is used to separate two vehicles that are about to collide.

This set of critical points is usually referred to as time-gap deficiency and is solved by adding the following modification:

$$\frac{dv}{dt} = \min[a\left(1 - \left(\frac{v}{v_0}\right)^4\right), a(1 - (\frac{s^*}{s})^2)] \tag{2.27}$$

By this way acceleration function is not smooth anymore but still continuous, this feature led to *double regime* simulation divided into free acceleration (first expression) and interacting acceleration (second expression). Some conventional values used in this model are shown in Table 2.

Parameter	Cars Highway	Cars City	Trucks Highway	Bikes
Desired speed v_0	120 [km/h]	50 [km/h]	80 [km/h]	20 [km/h]
Time gap T	1.0 [s]	1.0 [s]	1.8 [s]	0.6 [s]
Minimum gap s_0	2 [m]	2 [m]	3 [m]	0.4 [m]
Acceleration a	1.5 $[m/s^2]$	2.0 $[m/s^2]$	0.5 $[m/s^2]$	1.0 $[m/s^2]$
Decelaration b	1.5 $[m/s^2]$	2.0 $[m/s^2]$	1.0 $[m/s^2]$	1.5 $[m/s^2]$

Table 2 Typical parameters values for Intelligent Driver Model

2.2.3 Discrete Choice Situations

Drivers take both *continuous* actions (braking, accelerating, steering) and *discrete* actions (lane changes, indicators, etc.). Discrete actions typically correspond to *tactical/strategic* levels, whereas continuous actions are *operational* [6].

Here are some examples of the differences between these definitions; a strategic decision could be determining the route while a tactical one might be related to a lane changing or entering a priority road. As in other fields, *strategy* is usually referred to *long-term result* while *tactics* is a set of decisions aiming at *fulfilling such result*; with a similar logic, tactical decisions are made through operational moves.

At any time, a driver selects from a set of discrete alternatives (e.g., keep lane, change left, change right; pass a signal or stop). Alternatives can be presented at fixed intervals or triggered by events.

Safety and Incentive criteria

Every active decision must satisfy two criteria:

- **Safety** criterion, where it is evaluated how the manouver affects vehicles around and how relevant an accident risk is
- **Incentive** criterion, where the advantage of the manouver is established

Whenever there are more actions which are both advantageous and safe, the most advantageous one is made. It may happen that there are not totally safe decisions; in that case the safest one is chosen.

Safety criteria take into consideration *microscopic* Car-following *acceleration* by all the vehicles that are not the decision maker; if, because of a choice, none of those accelerations are larger than a *limit safe deceleration*, safety criterion is respected.

$$a_{mic}^{(j,k)} > -b_{safe} \; \forall \; affected \; vehicle \; j \; by \; a \; decision \; k$$

Incentive criteria start from a function (travel time, comfort, etc.) and try to optimize such function. From this starting point, two kinds of driver behaviours are implemented, being these *egoistic* and *general drivers*.

An egoistic driver basically takes all the decisions that only *optimize its own benefit*, while a general driver behaves following default values or *statistical distributions* aiming at simulate a typical driver.

The way a decision affects the surroundings is weighted by a *politeness factor* $p \leq 1$.

Let's see an application of these criteria on a lane changing scenario, like the one displayed on Figure 5:

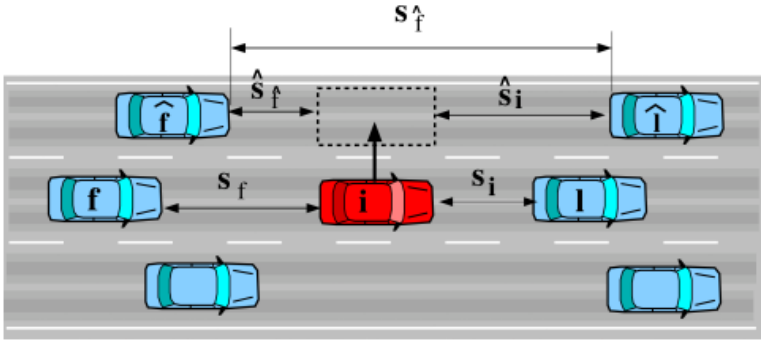

Figure 5 Lane changes parameters [6].

Where:

- $i = ego\ vehicle$
- $f, l = first\ follower\ and\ leader\ \ vehicles$
- $\hat{f}, \hat{l} = new\ follower\ and\ leader\ vehicles$

$$General\ safety\ criteria \begin{cases} \hat{a}_{\hat{f}} = a_{mic}(\hat{s}_{\hat{f}}, v_{\hat{f}}, v_i) < -b_{safe}\ (safety\ criterion) \\ \hat{a}_{\hat{l}} = a_{mic}(\hat{s}_{\hat{l}}, v_{\hat{l}}, v_i) < -b_{safe}\ (self - protection) \end{cases}$$

$$Egoistic\ incentive\ \hat{a}_i - a_i = a_{thr} + a_{bias}$$

Where a_{thr} and a_{bias} are respectively a *threshold acceleration* preventing frantic mind changes that lead to minimum utility gain and a *bias acceleration* adopted where there are some tactical/strategic constraints and traffic regulations.

Polite drivers' law is reported below:

$$\hat{a}_i - a_i + p\left(\hat{a}_{\hat{f}} - a_{\hat{f}} + \hat{a}_f - a_f\right) > a_{thr} \pm a_{bias}$$

Politeness factor equal to 1 corresponds to the **MOBIL** principle, where the acronimous means **Minimizing Overall Braking** by **Intelligent Lane-changing** decisions, where a vehicle changes lanes only if the acceleration advantage itself is sufficient and does not force vehicles in the target lane to brake excessively.

Another context in which incentives and safety criteria are applied is that of a vehicle approaching a traffic light. Clearly, this is an extremely likely scenario in an urban context and must be properly simulated; there are obviously three main cases: green, yellow, and red lights. The most delicate case is undoubtedly the yellow light, where one must decide whether the safest course of action is to cross the light or stop in anticipation of a red light. The following example briefly illustrates the logic adopted with the help of the illustration in Figure 6.

Approaching a Traffic Light

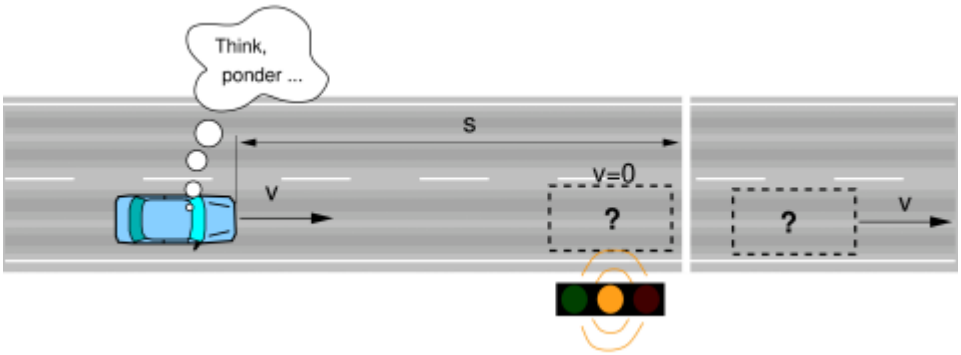

Figure 6 Schematic scenario of a vehicle approaching a yellow traffic light [6].

Stopping criterion for a car approaching a yellow traffic light:

- Speed of the leading vehicle is equal to zero
- *Safe deceleration* is a little larger than comfortable one or the safe deceleration for the lane-changing safety criterion
- Distance between leader and follower is larger than safety distance

Other circumstances can lead to a lane change, and the traffic light is ignored.

Not all intersections are controlled by traffic lights; many are governed by priority rules. Normally, right-of-way is given to those approaching from the right, but some minor roads merge into more important and typically busier roads. These are called priority roads, and vehicles traveling on them have right-of-way regardless of direction. A vehicle approaching this type of intersection faces two choices: either enter the priority road or cross it to stay on the minor road.

In Figure 7 and Figure 8 there are two ways to enter a priority road; the details are explained better in the first of the two figures..

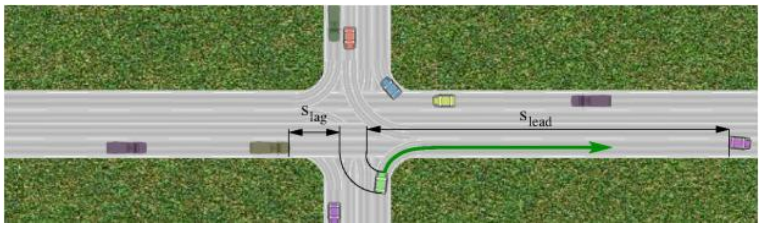

Figure 7 Priority road entering logic for a stopping intersection [6].

While entering a priority road, the incentive criterion is usually positive, the safety criteria mirror those of lane changes, with *lead* and *lag* gaps computed along the arc length to the nominal merge point.

Two cases occur:

- **Stop intersections**: vehicles start when the leading gap at the entry is positive, and the lag gap is equal to the critical gap at traffic lights.
- **Running start intersections**: In this case, a decision point is set where the gaps estimation is calculated. If safety is guaranteed, vehicles accelerate to enter the priority road; otherwise, vehicles decelerate to stop to behave like a classic stop intersection.

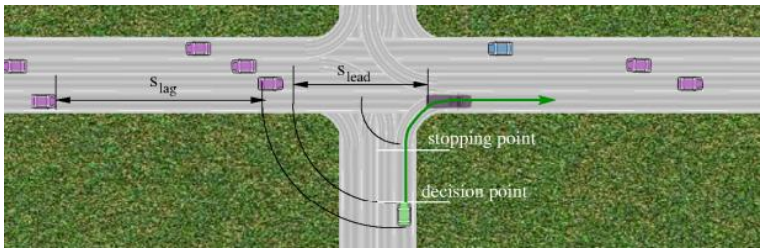

Figure 8 Priority Road entering logic for a running start intersection

For what concerns crossing a priority road, the vehicle approaching the road decelerates up to a stopping point, which is the spot where all the safety criteria checks are made. These criteria are based on time-to-collision calculations; when all criteria are satisfied, vehicles accelerate and pass.

If there are no adequate conditions, checks are regularly re-made up until it is safe to cross. Time-to-Collisions calculations are mostly based on trivial cinematic equations [6].

2.3 Mesoscopic Models

Mesoscopic traffic models emerged as an intermediate approach between microscopic modeling, which has been shown to focus on describing the behavior of individual vehicles, and macroscopic modeling, which is based on aggregate variables such as traffic flow, density, and average speed [7]. This category of models is developed to balance accuracy and computational costs, enabling the simulation of extensive networks over long timescales while maintaining a satisfactory level of detail that captures local phenomena, such as the formation and spread of congestion waves, the effects of accidents, and the consequences of road openings or closures.

From a theoretical perspective, mesoscopic models treat vehicles as discrete entities, similarly to microscopic models, but describe interactions between them in a statistical or simplified form, without resorting to complex car-following or lane-changing models [8]. This results in an agent-based simulation in which each vehicle has individual properties (such as desired speed, class, path, and position along the network), but the collective dynamics emerge from probabilistic distributions, state rules, or models derived from statistical mechanics [7].

Mesoscopics models are essentially divided into three categories:

- **Queue based models**
- **Discrete Choice models**
- **Gas Kinetic models**

In the first category, road segments are modeled as a queue with finite capacity, service time, and a fixed length [9]. Traffic evolves depending on the queue's capacity and the

interaction with surrounding infrastructure (traffic lights and intersections). Vehicles move from one queue to another based on available capacity, and speed depends on the degree of congestion [9].

The second model, like its microscopic counterpart, has vehicles make decisions based on individual optimization and iterative learning of travel strategies.

As for the discrete choice model, the third model is also very similar to a previously mentioned macroscopic model derived from the kinetic theory of gases.

The advantages of using a mesoscopic model include its ability to simulate very large scenarios, such as cities or regions, with low computational costs. They are also capable of representing heterogeneous vehicle samples like cars, buses, bikes and pedestrian [10].

Thanks to this versatility, these models lend themselves to implementation in various software like:

- **Aimsun Next**, an industrial software that supports a hybrid micro/meso model
- **PTV Visum/Vissim**, widely used by urban mobility agencies
- **TRANSIMS**, devoted to large scale simulations and statistical analysis
- **METROPOLIS (INRIA)**
- **MATSim**, an agent based often used in academic environments
- **SUMO**, which implements a queue model that is quite simplistic but is a powerful tool for simulations at large scale and for the generation of large datasets.

In the current context, characterized by the increasing availability of real-time data, the integration with machine learning techniques and the development of intelligent transport systems, mesoscopic models remain a central element in simulation and planning strategies, acting both as application tools and as a theoretical bridge between micro and macro approaches.

3 SUMO

SUMO is the traffic simulation software that will generate the dataset for training and testing the LSTM network. It plays a fundamental role in this work, and for this reason, this chapter will describe the software's main features. This will be followed by an in-depth look at the Turin scenario, explaining how the scenario was generated, the measures taken to make the traffic flow smoothly and realistically similar to reality, and how the model was verified and validated.

3.1 General Overview

SUMO (Simulation of Urban MObility) is an open source and multi-modal traffic simulation tool. Initially it was intended to implement microscopic model only; the latest versions can perform also mesoscopic simulations, like the one exploited for this work.

Open source means that the source code is public, and any user can deliberately update and modify the code, share the update code and in general provide its own contribution to improve the quality of the software.

Multi-modal traffic simulation refers to the vehicle types that can be used during the simulation, which in this case are not limited to private cars and bikes; since SUMO is able to simulate public transport, bikes, pedestrians, heavy duty vehicles and taxis.

Simulations can implement a large set of traffic management topics (traffic lights control, public transport, sustainable mobility, etc.) and are deterministic by default, even though there are various options for introducing randomness.

This last task is performed to make the simulation more realistic, and it’s implemented by means of the Mersenne Twister algorithm for the generation of random numbers [8]; a random number is initialized by a seed, and each seed gives a deterministic sequence. Seed changing criterion can be time dependent, to have random output along the simulation.

The simulation uses multiple Random Number Generation instances to decouple different simulation aspects, like:

- Loading vehicles
- Probabilistic flows
- Vehicle driving dynamics
- Vehicle devices

The road infrastructure on which vehicles travel is modeled as a network, meaning that the key elements are nodes and edges. Nodes are what are commonly referred to as intersections, while the connections between nodes are called edges. A road is typically

composed of multiple edges, considering that there is usually at least one intersection along each street.

SUMO is designed to perform microscopic simulations, however by using the MESO option it is possible to implement a mesoscopic queue based model where vehicles are aggregated in platoons characterized by a common speed [10].

3.2 Main Features

For what concerns simulation

- Space-continuous and time-discrete vehicle movement
- Different vehicle types
- Multi-lane streets with lane changing
- Different right-of-way rules, traffic lights
- A fast OpenGL graphical user interface
- Manages networks with more than 10.000 edges (streets)
- Fast execution speed (up to 100.000 vehicle updates/s on a 1GHz machine)
- Interoperability with other applications at run-time
- Network-wide, edge-based, vehicle-based, and detector-based outputs
- Supports person-based inter-modal trips

Routing

- Microscopic routes, where each vehicle has its own one
- Different Dynamic User Assignment Algorithm [10].

3.3 Turin Scenario

Anyone can create a virtual road network using SUMO; many users and researchers also make their generated scenarios available online. Among these scenarios, it is possible to download one that reproduces Turin and its surroundings. The availability of this scenario is an excellent opportunity to work on a large network. The following pages will explain the main features of this simulation, how it was produced, and how similar it is to the real world.

Having a huge amount of data is fundamental to training a neural network and, to create a dataset, a full large scale 24 hours long model of the Turin area with SUMO has been exploited.

The model must clearly mirror the real traffic flow in Turin as closely as possible, and it's not enough to simply ensure the simulation works from start to finish. It's therefore necessary to have real data available on which to base the model.

To obtain this information, it is necessary to rely on data collected by third-party entities such as the 5T company, which makes its findings available to the public.

5T is a Turin company that operates in ITS and Info-mobility field and collects real traffic data (in the interest of Turin municipality, Metropolitan city of Turin and Piemonte region) by means of a huge set of sensors (approximately 1700) placed in strategical points all over the city. Where these sensors are not present, information is integrated with:

- floating car data
- 26 panels for displaying traffic data
- 71 traffic cameras
- an adaptive traffic control system able to manage more than 300 urban traffic lights
- a traffic SuperVision system (SV), which is a system that estimates real time traffic-state

This system has been used to generate the Origin/Destination (O/D) matrix; this matrix is a table that shows how many departures and arrivals happen over a zone during a certain time interval and it's a fundamental tool in traffic simulation field.

This matrix has been originally developed in 2011 and, from that moment on, periodically updated with the growing availability of information and the content of the matrix is computed using Macroscopic kind of data like traffic flow, travel times and traffic density.

Different traffic behavior might result in considering which day it's being investigated, i.e. a weekday differs from a typical weekend day or a festive one; the simulation involved for this work considers a school weekday.

Origins and Destinations in the matrix are associated with Traffic Assignment Zones (TAZs) that are aggregated Census Cells defined in the 90's for planning activities and statistical purposes [11].

The Scenario, which is called TuST (Turin SUMO Traffic), covers an area of 600 Km^2 that includes the city of Turin itself with its own suburbs the total number of TAZs counts 257 units for a population of 1.200.000 people.

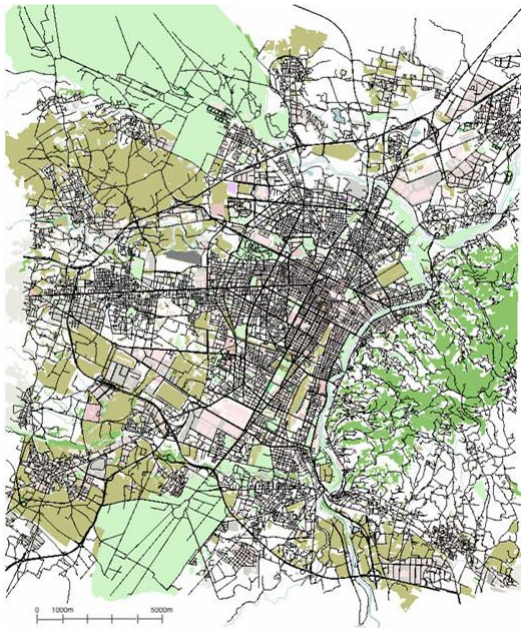

Figure 9 Map of the whole scenario [11].

3.3.1 Road Graph Generation

The road network has been generated using JOSM tool, an extensible editor for OpenStreetMap in Java 8+, yielding an OSM file [11]. The OSM file contains all the geo-referenced information needed to link and build a two-dimensional map containing highways, roads, railways and bike paths.

SUMO has a helper tool called NETCONVERTER that allows to produce a network file in SUMO-readable XML format from the OSM file. The ultimate network is made of almost 33.000 nodes, 66.000 edges, and more than 6.500 Km of roads.

Obviously, the result shows some differences between the real road network of the city and the digital copy, anyway such approximations are negligible when compared to the whole scenario.

- **Speed Limits**

  An API of Google Maps called Distance Matrix has been exploited in order to extract values for Free-Flow conditions; such conditions are used to identify a situation where a vehicle is the only one on the street and starting from this condition the speed limit of such edge has been set. In the context of the TuST simulation, some random vehicles are allowed to slightly exceed the limits if there is no congestion on the edge.

- **Streets Width and Lanes**

There are a certain number of streets in Turin that allow two vehicles to run side by side even though there is a single lane and SUMO cannot handle this kind of road; this problem forces the simulation to detect some congestions on roads where the traffic is fluid. Due to a lack of online databases, more than 100 roads between the most important ones have been manually edited to reduce congestion.

- **Intersection Management**

  For what concerns intersection management, the standard criterion followed by SUMO is a right-before-left law for normal crossings and a left-before-right law in roundabouts; in the real world this logic might be too simplistic so more than 500 roundabouts were manually fixed to fit better with reality.

  Also, traffic lights phasing represents a quite demanding task, since the only use of NETCONVERTER does not provide realistic output; for this reason, more than 900 traffic lights have been corrected by the developers using data from 5T. Further problematic rises by considering that traffic lights phases change during the day thanks to an algorithm called UTOPIA, that evaluates live traffic patterns. To avoid complex behavior, phase durations through 24 hours have been averaged and applied to more than 800 intersections regulated by traffic light.

Figure 10 Traffic lights distribution over the network [11].

By means of a Python script SUMO can coordinate traffic lights to have the so called green-wave effect. The combined actions of all the previous interventions lead to excessive congestion and for this reason it has been implemented an option that activate traffic lights only for traffic-saturated edges, implementing a priority-based logic for low traffic density cases.

- **Traffic Assignments**

  Traffic Assignment is performed by means of an algorithm called Incremental Traffic Assignment (ITA); in this algorithm fractions of traffic volumes are assigned in steps thanks to Dijkstra algorithm, which finds out the shortest path

inside a network. In SUMO, ITA is implemented by means of DUAROUTER, which applies the algorithm to every vehicle. The result is that, at the beginning of a ride, the shortest path is selected considering the actual traffic conditions, like a human would do.

- **Further Improvements**

  All the corrections performed previously are not enough to have a perfect working simulation, because only a fraction of the total traffic can be handled, and further improvements are needed.

  It has been noticed that very few vehicles started their journey in secondary edges, which are edges that constitute roads with little traffic because they are peripheral or do not connect important places in the city, because the ITA algorithm is induced to prioritize primary roads; even if it may be true that people tend to exploit primary edges, it is also true that the number of journeys that start or end in residential areas is large.

  To fix such problems, a Python script has been developed that couple DUAROUTER benefit with a random Origin/Destination choosing criterion that prioritizes residential and peripherical spots. It is obvious that applying this criterion to all the 2.200.000 vehicles that take part in the simulation would generate a specular problem; so only a fraction of the whole fleet is edited in this way.

3.3.2 Mesoscopic Implementation

The most impactful modification involves the adoption of Mesoscopic approach on the simulation, such option is implemented by means of MESO. By this way, the vehicle on every edge is aggregated in traffic queues, where the leader vehicle of such queue can join another one present on another edge. The system treats each queue individual element, simplifying the management of the simulation. The difference in terms of time gain can be compared, and it results that a Mesoscopic simulation might be 100 times faster than a Microscopic one [11].

MESO option has some flaws too, considering that queue-based model is optimal for large scale simulations but lacks accuracy in terms of microscopic dynamics, lane changing behavior and intersections handling.

3.3.4 Scenario in numbers

To comprehend the magnitude of the scenario, in Table 3 there are some numbers displayed:

Area	602.61 Km^2
Traffic Assignment Zones	257
Total nodes	32.936
Total edges	66.296
Total junctions	22.209
Traffic lights	856
Roundabouts	501
Total length edges	6.570,28 Km
Total length lanes	7.723,40 Km
Vehicle length	4,3 m
Total vehicle trips	2.202.814
Ended vehicle trips	2.197.672
Final waiting vehicles	0
Final running vehicles	5142
Teleports	334
Collisions	0
Halting	19

Table 3 Values of the main parameters of the simulation

3.4 Model Verification

When simulating a physical phenomenon, it's obviously important to ensure everything is working properly. During the verification phase, the goal is to ensure the simulation doesn't encounter any internal problems. In this case, unexpected congestion could arise, preventing vehicles from flowing, or the number of vehicles entering the system could grow dramatically, preventing the balance between the vehicles exiting the simulation and the number of vehicles exiting the system.

3.4.1 Global Output

TuST provides different output files that sum up data and metrics of various nature and allow to evaluate the performance of the simulation; by means of the Summary files it is possible to get aggregated data from the whole scenario and plot those data as a function of time, like in Figure 11 where, by means of a stability plot, which is a graph that shows if the simulation tends to diverge, it is assessed that there are no phenomena like cumulation of vehicles that makes the simulation unstable.

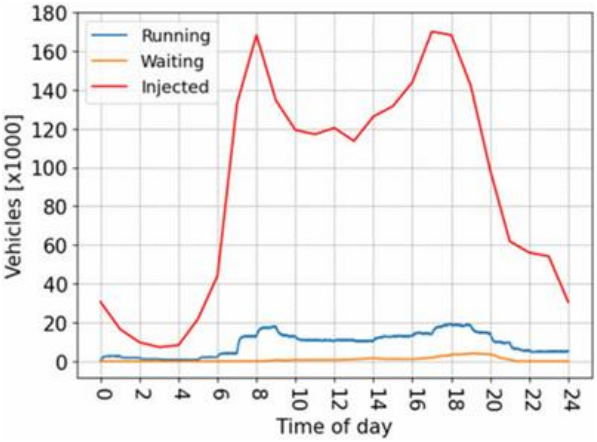

Figure 11 Stability plot of the simulation [8].

The stability plot in Figure 11 shows correct working day traffic behavior, where two main peaks are clearly visible around 8:00 and 5:00 pm; those peaks are present for both the injected (red line) and the running (blue line) vehicles. Waiting (yellow line) vehicles are those that, due to congestion, cannot enter the simulation and for this reason are kept waiting for the best moment to be injected. Exiting vehicles are not shown in the graph, but they are obviously present and make the balance between running and injected vehicles true.

3.4.2 Road Level Inspection

It is possible to investigate aggregated data on road level by means of detectors, a solution implemented to emulate street sensors; results are provided at the end of the simulation into a file named Detectors Output.

Detectors' Output allow to study vehicles distribution on the map and show results as in Figure 12.

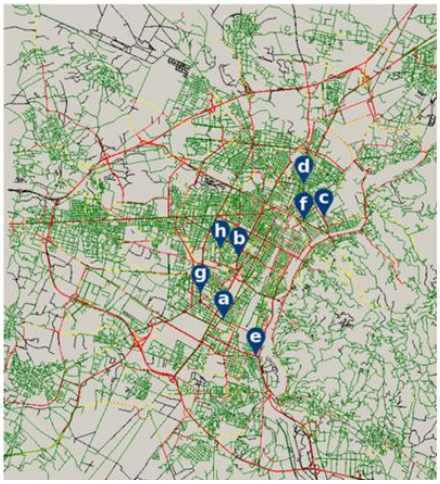

Figure 12 Traffic flow distribution is displayed directly on the map with the help of detectors [11].

Each edge has a color that indicates the traffic density that characterizes it, where

- Red roads are those in which more than two vehicles per minute are observed
- In yellow roads one vehicle per minute is detected
- Black roads are unused
- Green roads are residential streets

Blue icons are the detectors used for validation of the model.

Thanks to the map and the colors of the edges it is possible to evaluate if there is congestion, still it is necessary to exploit detectors to locate the exact point where this phenomena are occurring and intervene to eliminate or reduce it.

3.5 Model Validation

In verification phase it has been assessed the correct working of the simulation; still TuST must be validated, which means that it must be verified that the simulation emulates well enough the real traffic of Turin.

To perform such validation, a set of detectors highlighted in Figure 12 have been located in the city of Turin and compared with results provided by the Detectors of SUMO. Data collected from the virtual and real detectors have been plotted.

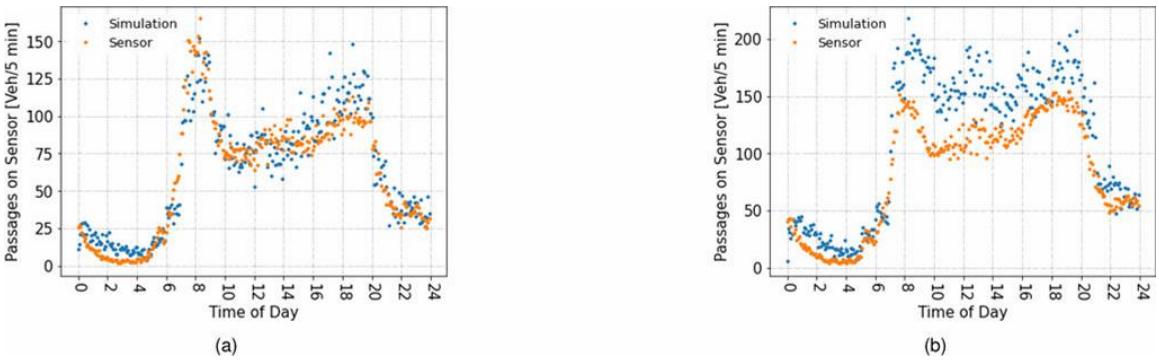

Figure 13 Validation by means of detectors (a) and (b) [11].

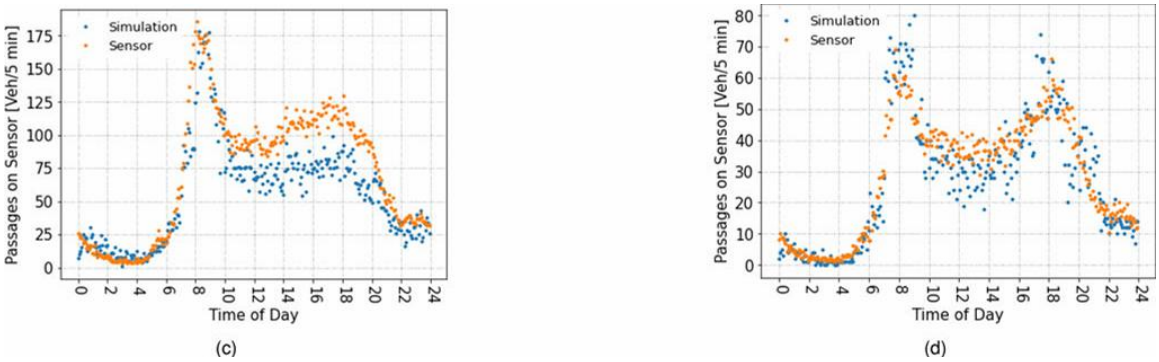

Figure 14 Validation by means of detectors (c) and (d) [11].

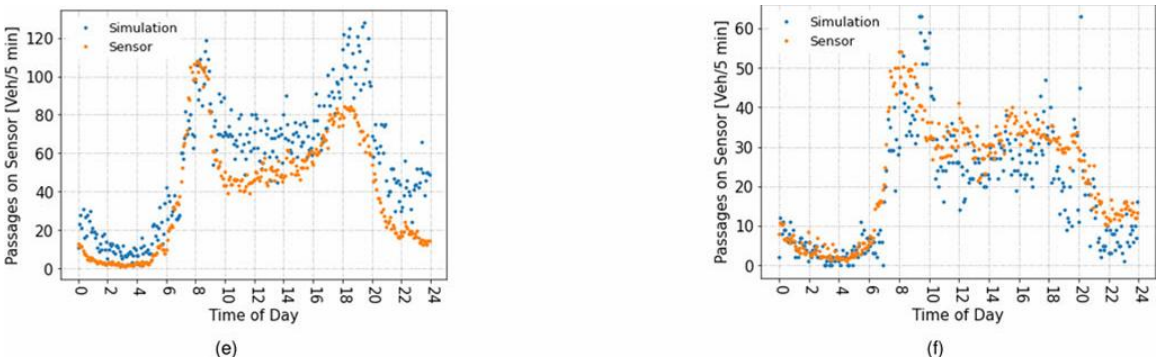

Figure 15 Validation by means of detectors (e) and (f) [11].

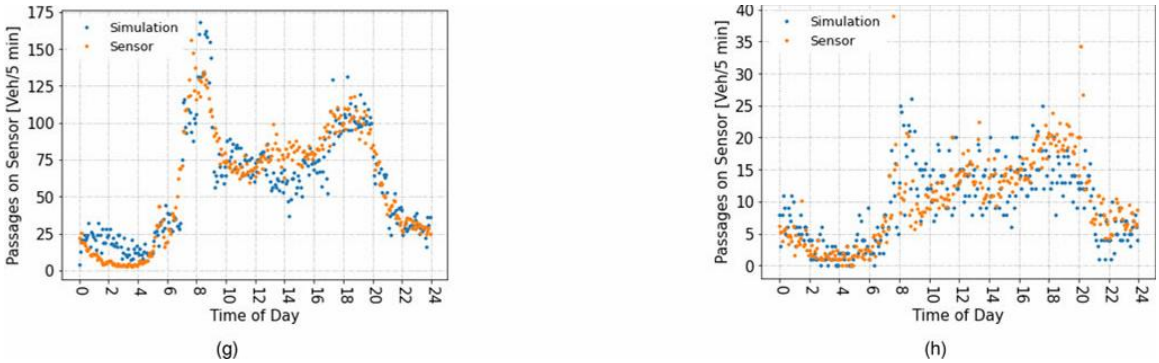

Figure 16 Validation by means of detectors (g) and (h) [11].

Blue dots, related to simulation, overlap with satisfying accuracy the orange ones, which are referring to the actual readings taken by the sensors [11], although there are some distinctions to be noted; the simulation appears to consistently overestimate the traffic density detected by sensors b and e. The peaks are generally well captured although it is difficult for the h sensor to capture a real shape, the data are very dispersed, however quite overlapping. To have a more precise idea of the accuracy of the simulation, in Figure 17 it is shown that 80% of th sensors have more than 75% affinity with data from the simulation [11].

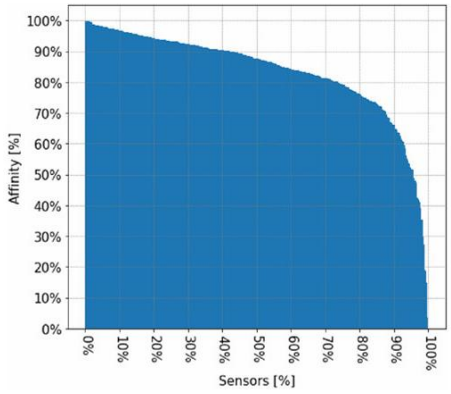

Figure 17 Percentage of sensors vs affinity with simulation data

Key features underlying the SUMO software have been explored together with how a simulation of a real, large-scale city environment is constructed. However, using the NETCONVERTER option alone isn't enough to achieve a fluid simulation, which is why manual intervention was necessary to manage some aspects of the simulation, such as traffic light operation and priority at roundabouts. The model has been successfully verified and validated, allowing the simulation to be used as an effective tool for creating a credible dataset to train a neural network.

4 Artificial Intelligence Fundamentals

4.1 Machine Learning

Artificial intelligence is a discipline of computer science and engineering that aims to make technology capable of carrying out typically human activities related to learning and the use of senses such as image recognition, speaking ability, outcomes prediction, etc. [12] [13].

Among the various forms of artificial intelligence, one is called Machine Learning (ML). As its very definition suggests, the goal of this discipline is to enable a computer to learn how to perform a tasks.

The learning style of a ML algorithm is inductive [14], therefore based on the analysis of a large amount of data, organized by *features*; the objective is to analyze the input features and grasp the implicit correlations that these have with the outcome of the task it has to perform.

This approach is possible for phenomena where a theoretical approach is difficult to implement due to system complexity, poor accuracy, or incomplete data [15]. Machine Learning is currently widely used in a variety of fields, from targeted advertising to finance to voice assistants like Siri to hospital applications for classifying X-ray and CT scan results [16].

The one above was just a very short overview of the concept of Machine Learning. Still, it is important to highlight that the task for which the program was trained, the nature of the data and the architecture of the algorithm determine the category of the machine learning model it has been developed.

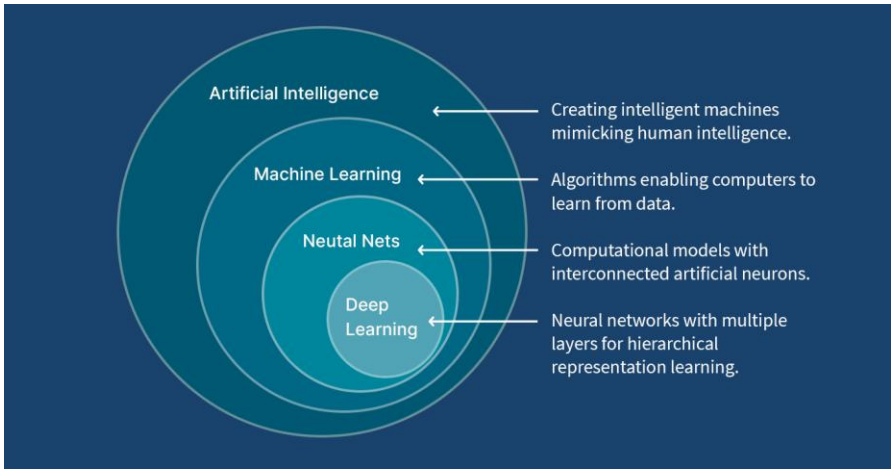

Figure 18 A.I. hierarchy

A very common way to distinguish Machine Learning techniques is by dividing them into three main categories based on how data are handled and organized:

- **Supervised Learning**: there are two dataset, an input one and an output one; the ratio here is to establish which is an outcome of a certain scenario and it is widely used for predictive AI [17]; i.e. load following algorithms in electrical grids try to foresee electric consumptions by the users in order to optimize production from the generation side.
- **Unsupervised Learning**: In this case there is not a labelling of the data at disposal and usually the aim of such approach is to find hidden relationships between the data. For this reason, unsupervised learning is often exploited in clustering problems [18]; i.e. many computer vision applications rely on this kind of learning.
- **Reinforcement Learning**: here the dataset is collected by the machine itself through interaction with a dynamic environment following a trial-and-error logic in which learning is reinforced by a reward that occurs when the action performed is considered correct [19]. This approach is often used to develop walking robot and autonomous vehicles.

There are many other ways to divide Machine Learning techniques, for example based on the output form.

- **Classification**: Supervised programs that provide an output made of discrete values fall into this category; each value is called class, and the aim of these models is to discriminate input values and associate each value to a class [20].
- **Regression**: Remaining in the context of supervised models, when the output is continuous, we fall into this category [15].
- **Clustering**: This is a typical example of an unsupervised model in which output elements are grouped based on common features that are not previously labeled. It is up to the user to understand the nature of the grouping [21].

The SUMO software provides output files containing numerous data points related to the vehicles participating in the simulation and aggregated traffic information collected by sensors placed at the edge inputs and outputs. These data points are easily labeled and can be defined as continuous functions over time. Given that the goal of this thesis is to develop a model for predicting the speed profile of a vehicle in an urban context, and considering the premises listed above, the most obvious choice is a supervised model oriented towards processing time series [11].

4.2 Neural Networks

Neural Networks are fundamental since they are the basis on which *Deep Learning* is built; this kind of AI can perform demanding tasks and it has been possible to work on it since recent time, with the growing capability to handle big data [22].

Human neurons are cells linked together and able to transmit electrical signals [23]; these signals are then processed and translated into information that are our interpretation of reality. In Figure 19 it is possible to see the illustration of a human neuron.

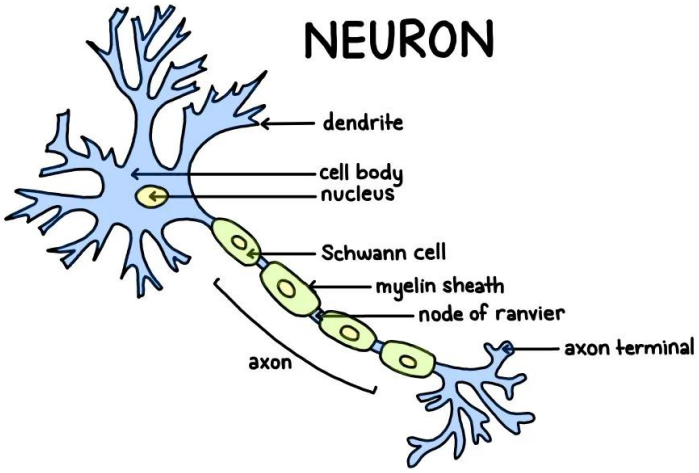

Figure 19 Human neuron structure

Here there are the main components of a neuron and their main objective:

- Dendrite: it is the input channel of the neurons which receives
- Cell body
- Nucleus, control center of the cell
- Schwann cell, increases conductivity of the axon
- Myelin Sheath, isolation function
- Node of Ranvier, place where the signals pass
- Axon
- Axon terminal

The neurons that make up a neural network, like the one in Figure 20, try to imitate the operating principle of a human neuron; however, they work following a more simplistic logic and process different information. In the biological case, as previously mentioned, neurons handle electrical signals while artificial neurons deal with numerical functions [24].

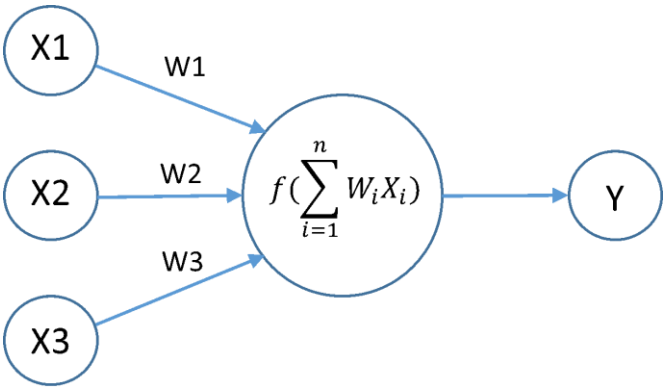

Figure 20 Artificial neuron structure

The cell of the neuron is called *Perceptron,* and it realizes the following operation:

$$g = \bar{W} \cdot \bar{x} + b = \sum_{i=1}^{n} W_i x_i + b \tag{4.28}$$

Where:

- $\bar{x} \in \mathbb{R}^{\mathbb{N}}$ is called **input vector**, which is the collection of the output coming from antecedent linked neurons.
- $\bar{W} \in \mathbb{R}^{\mathbb{N}}$ is the **weight vector**, it establishes the impact that a certain input has on the output.
- $b \in \mathbb{R}$ is the **bias** associated with the pre-activation function f.

Downstream of the cell there is the *activation function* $Y = f(g)$

$$Y = f(\sum_{i=1}^{n} W_i x_i + b) \tag{4.29}$$

The main purpose of the activation function is to process the output of the perceptron depending on whether it should be in the range from -1 to 1 or from 0 to 1. Activation functions are many, yet there are three of them which are widely exploited:

- **Hyperbolic Tangent Function**: $\tanh(f) = \frac{e^f - e^{-f}}{e^f + e^{-f}}$

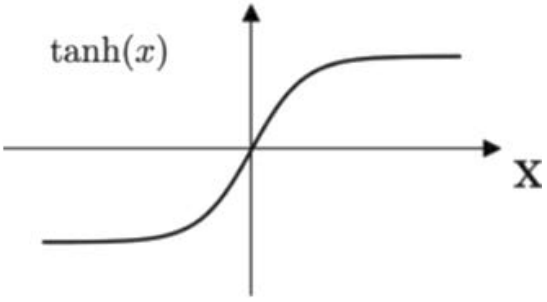

Figure 21 Hyperbolic tangent function plot

The use of this activation function is recommended when data are centered around zero, input and output values are both negative or positive but for deep network it can lead to vanishing gradient easier than other functions.

- **Sigmoid Function**: $\sigma(f) = \frac{e^f}{1+e^f}$

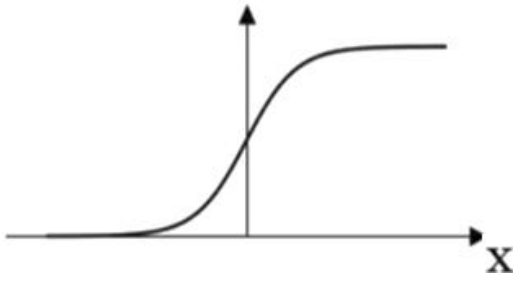

Figure 22 Sigmoid function plot

It is often used for binary classification problems and outputs are only positive; it is not recommended for deep networks because of vanishing gradient.

Moreover, it may be quite costly from a computational point of view, considering that performing an exponential is more complex than a linear function.

- **Rectified Linear Unit** (ReLU): $\max(f) = \frac{f+|f|}{2}$

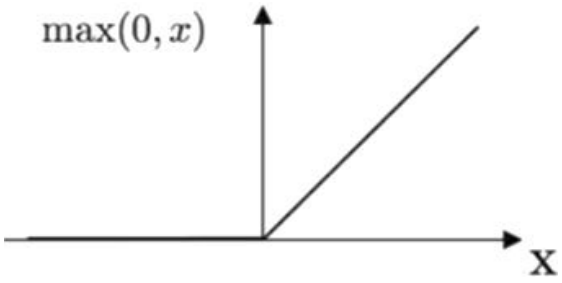

Figure 23 Rectified Linear Unit

The usage of this last activation function is usually recommended for deep learning and deep networks because it is not that heavy from a computational point of view, and it is harder to get vanishing gradient, a concept that will be explained later in the discussion [25].

Weights and biases are parameters of fundamental importance and neural networks are designed to select the optimal values for them to catch complex relationships between training data [23].

4.2.1 Neural Network Architecture

An artificial neural network is composed of two main elements: neurons and links. Neurons are organized into parallel layers that can contain one or more elements. Neurons within the same layer are not connected to each other but only to those in the layers before and after them.

Layers are defined differently depending on their location and role in the network architecture. The first layer is called the Input Layer and is responsible for receiving data to be processed by the network.

The actual mathematical function characteristic of neurons occurs in the so-called hidden layers, which can be one or more depending on the depth of the network. This last aspect is crucial for the quality of the result, as a network that is too deep risks overfitting and high computational costs [26], while a less deep network could, conversely, lead to underfitting [27].

The Output Layer is responsible for collecting and final reorganizing the data. In Figure 24 it is shown the scheme of a neural network made of three hidden layers, each containing five hidden units; at the beginning of the network there is the input layer and at the end the output layer.

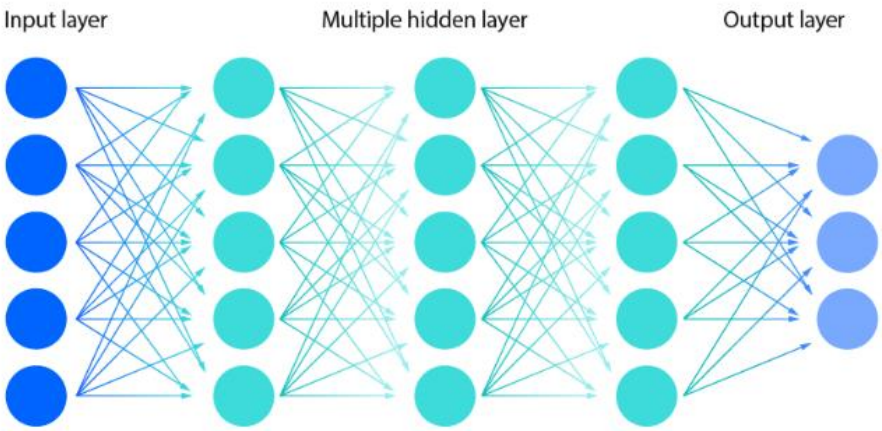

Figure 24 Feed Forward Neural Network architecture [23].

Regarding data, the *quantity* provided to the network for training is important, but *quality* should not be overlooked. Inaccurate data selection could mislead the network, which risks processing more data than necessary, unnecessarily increasing the computational cost of the training phase [28].

Another phenomenon that could occur is *overfitting*, which is an overly specific adaptation to the training data that prevents sufficient generalization of the phenomenon being predicted, it will be better explained further in the thesis. From a quantitative

perspective, overfitting can occur when there is a lack of data; from a qualitative perspective, however, the phenomenon can occur if the dataset presents an incorrect predominance of certain dynamics that occur less frequently. In this case, the network could be induced to not learn the underrepresented dynamics; the result is that during the testing phase, there is an imbalance between some dynamics that are predicted with excessive accuracy and others that return completely inaccurate results, like a sort of localized overfitting [29].

The network itself receives input data without being able to manipulate it before training, meaning that the most delicate phase from the programmer's perspective is data *preprocessing*, which requires in-depth evaluation to optimize data quantity and quality [26]. Tools such as *Primary Component Analysis* (PCA), an artificial intelligence technique that helps reduce the dimensionality of the dataset [31], can be used. It's also good practice to analyze *data distribution* to ensure there are no anomalies, excessive presence of certain values, or absence of others. These latter evaluations can be made based on adequate knowledge of the physical model the programmer is working with, making the implementation and training of a neural network not merely operational but the result of careful study.

Training Algorithm

The name Machine Learning already suggests that a neural network is designed to learn how to perform a task, but this learning occurs through the training phase.

A neural network can obviously be supervised, and its training phase is of primary importance. This phase usually consists of two blocks: *backpropagation* and optimization using an *optimization method*.

Backpropagation aims to update the weights so that the network's predictions are increasingly accurate to the real data and consists of 3 steps:

- Forward pass, where input data are provided to the net and processed by means of bias, weights and activation functions; at the end of this step an output vector is delivered.
- Via Loss Function calculation the real output and the output of the network are compared using metrics like RMSE, MAE, etc.
- Backward pass, by applying the chain rule of differential calculus, we find the derivative of the error as a function of each weight in the network [32].

The optimization method serves to update the weights in the direction of the negative derivative of the error, so that with each iteration the Loss Function should decrease, and learning should improve [33]. There are several methods available to the designer; the

most classic is called Gradient Descent and is simple; however, it is subject to problems of gradient explosion or vanishing and from a computational viewpoint is expensive.

Other methods exist and some of them are known as *Stochastic Gradient Descent*, *Root Mean Square Propagation* or *Adaptive Moment Estimation Method* (ADAM) [33].

In this study, the last-mentioned method will be used, so it is worth explaining its main characteristics.

As mentioned, ADAM updates the neural network's weights during training more efficiently and faster than the classic Gradient Descent Method (GDM). The key weight updating equation in GDM is given by:

$$w_{t+1} = w_t - \eta \cdot \nabla L_t \tag{4.30}$$

Where

- w_{t+1} is the updated weight
- w_t is the weight of the last iteration
- η is the learning rate
- ∇L_t is the gradient of the Loss Function

GDM algorithm risks stopping at a local minimum and is too dependent by the learning rate and for these reasons ADAM introduces automatic adaptive learning rate mechanisms based on statistical evaluation of past gradients.

The main steps in such algorithm are:

- $g_t = \nabla_w L_t$ (gradient calculation) (4.31)
- $m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (momentum update) (4.32)
- $v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (variance update) (4.33)
- $\widehat{m}_t = \frac{m_t}{1-\beta_1^t}, \widehat{v}_t = \frac{v_t}{1-\beta_2^t}$ (initial bias correction) (4.34)
- $w_{t+1} = w_t - \eta \cdot \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t}+\epsilon}$ (final weight correction) (4.35)

where

Symbol	Parameter name	Typical value	Meaning
η	Learning rate	0.001	Updating intensity
β_1	Momentum decay	0.9	Past gradient weight
β_2	Variance decay	0.999	Variance weight
ϵ	Stability term	10^{-8}	Avoid zero at the denominator of the fraction

Figure 25 Parameters and typical values of Adaptive Moment Estimation Method

Using Adam is recommended to reduce computational cost while maintaining the ability to handle large data sets without having to excessively tune the hyperparameters.

It has been said that using ADAM prevents the Gradient Vanishing problem from occurring; however, it remains to be defined precisely what this problem consists of. In Gradient Descent optimizers, the weights are updated according to the previously cited equation and according to a law proportional to the partial derivative of the loss function, which is itself a function of the weights.

Most activation functions take values in the range [0,1], and consequently, the associated gradient falls within the same range. If the gradient approaches 0, the weight update becomes meaningless, and the network stops learning. This problem can occur before the training data has been effectively fitted, and the network stops learning not because there is nothing left to learn, but because no weight update occurs [34].

The chain rule calculates partial derivatives by multiplying n functions together, where n is the number of neurons in the network. This shows that the deeper the network, the more likely this problem is to occur, since the probability that some activation function will yield results close to zero increases [35].

Exploding Gradient Problem is analogous but it happens when the gradients tend to infinite.

Two problems that should never occur are overfitting and underfitting. Overfitting has been briefly discussed previously; essentially, it's the type of problem that arises when training data is scarce or poorly represents the nature of the phenomenon you want to teach the network. This situation creates a tendency for the network to over-adapt to the training data, rendering it incapable of operating outside of that comfort zone. Essentially, a predictive model could make completely incorrect predictions because, once put to work in the context for which it was trained, it encounters a situation it has never processed before.

Underfitting is the opposite phenomenon that leads to the same result; in underfitting, the network doesn't over-adapt to the training data; in this case, the network simply learned less than expected from the data it received. In both cases, the performance offered by the network is simply unacceptable [26].

Not all available data should be used to train the network, since the quality of the resulting network needs to be tested. For this reason, the starting dataset must be divided into three sub-datasets: the *training dataset*, the *validation dataset*, and the *testing dataset*. The first dataset should be the largest, between 70% and 90% of the total, and is used to teach the network the relationships between the data. Between 20% and 5% of the starting data is used to create the validation dataset, which is used to evaluate how the

network performs during training on data that has not been used to update the weights, but which comes from the same training domain. This is essential for controlling learning and preventing overfitting. The testing dataset, made up of what remains of the original data, is a set of sequences never seen by the network during the training phase. It is used to evaluate the accuracy of the training and therefore the network's ability to make predictions on previously unseen scenarios.

Given the same training, validation, and testing datasets, what makes the difference between an accurate network and a less accurate one is the setting of the Hyperparameters, which are parameters set by the designer before training and have an impact on various aspects of this phase.

By tuning the hyperparameter values, you can influence the optimization method, the backpropagation algorithm, the network architecture, and dozens of other aspects of it. This concept will be revisited in subsequent chapters as it will be the subject of sensitivity analysis.

4.3 Long Short-Term Memory Neural Networks

The architecture analyzed so far assumes that the data flow is unidirectional, starting from the input layer, passing through the hidden layers, and ending in the output layer. This type of behavior is called feed-forward and is typical of simpler neural networks.

However, many physical realities have complex dynamics in which temporal correlations require a comparison between the updated data and the data recorded in previous iterations. For these types of situations (which include urban traffic dynamics, driving cycles, and energy management strategies), Recurrent Neural Networks (RNN) have been invented, where a neuron can interact with itself [36] [37].

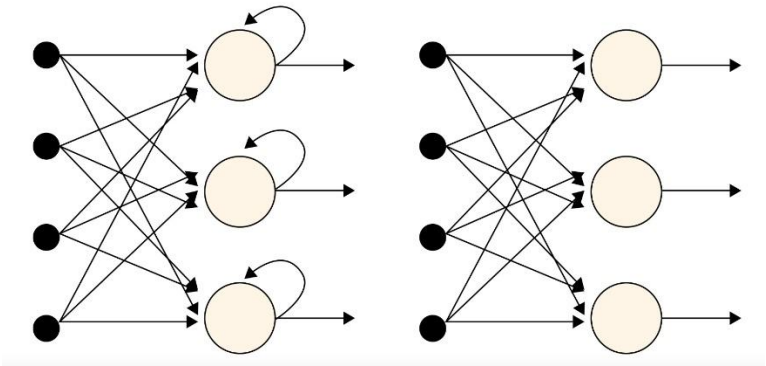

Figure 26 Difference between Recurrent Neural Network (left) and Feed Forward Neural Network

This type of structure, the looping of neurons, produces a state memory; since recurrent neural networks are excellent at capturing temporal dependencies, this makes them ideal for predictive problems such as speed forecasting [38].

The neural network used in this work falls into this category and is defined as Long Short-Term Memory, precisely because it is specialized in capturing relationships between data even over the long term, although with some limitations.

An LSTM unit has a more complex structure than a neuron from a simple recurrent neural network; in fact, it contains a cell state having four gates, called *forget*, *cell*, *input*, and *output* [39]. In Figure 27 it portrays the structure of a LSTM; it is possible to see respectively the forget gate (f), the cell gate (g), the input gate (i) and the output gate (o).

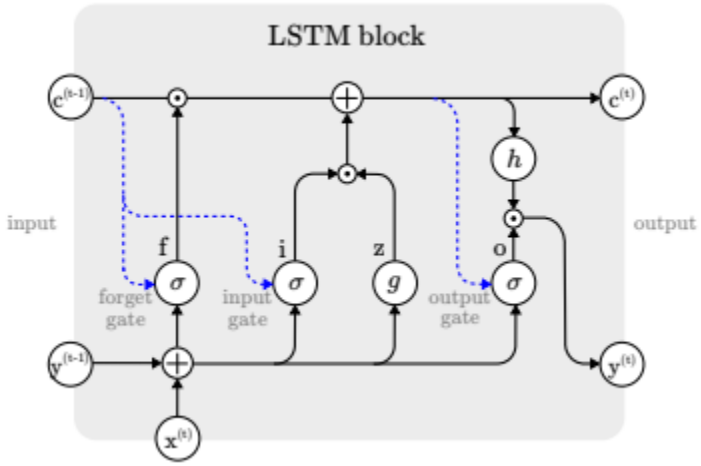

Figure 27 LSTM unit structure [39].

Aiming to clarify how the LSTM model works, let us assume a network comprised of N processing blocks and M inputs. The forward pass in this recurrent neural system is described below.

***Block input**. This step is devoted to updating the block input component, which combines the current input x(t) and the output of that LSTM unit y(t−1) in the last iteration. This can be done as depicted below:*

$$z^{(t)} = g(w_z \cdot x^{(t)} + r_z \cdot y^{(t-1)} + b_z) \qquad (4.36)$$

where w_z and r_z are the weights associated with $x^{(t)}$ and $y^{(t-1)}$, respectively, while b_z stands for the bias weight vector.

***Input gate**. In this step, we update the input gate that combines the current input $x^{(t)}$, the output of that LSTM unit $y^{(t-1)}$ and the cell value $c^{(t-1)}$ in the last iteration. The following equation shows this procedure:*

$$i^{(t)} = \sigma(w_i \cdot x^{(t)} + r_i \cdot y^{(t-1)} + p_i * c^{(t-1)} + b_i) \qquad (4.37)$$

*where * denotes point-wise multiplication of two vectors, w_i, r_i and p_i are the weights associated with $x^{(t)}$, $y^{(t-1)}$ and $c^{(t-1)}$, respectively, while bi represents for the bias vector associated with this component. In the previous steps, the LSTM layer determines which information should be retained in the network's cell states $c^{(t)}$. This included the selection of the candidate values $z^{(t)}$ that could potentially be added to the cell states, and the activation values $i^{(t)}$ of the input gates.*

***Forget gate**. In this step, the LSTM unit determines which information should be removed from its previous cell states $c^{(t-1)}$. Therefore, the activation values $f^{(t)}$ of the forget gates at time step t are calculated based on the current input $x^{(t)}$, the outputs $y^{(t-1)}$ and the state $c^{(t-1)}$ of the memory cells at the previous time step $(t-1)$, the peephole connections, and the bias terms bf of the forget gates. This can be done as follows:*

$$f^{(t)} = \sigma(w_f \cdot x^{(t)} + r_f \cdot y^{(t-1)} + p_f * c^{(t-1)} + b_f) \quad (4.38)$$

Where w_f, r_f and p_f are the weights associated with $x^{(t)}$, $y^{(t-1)}$ and $c^{(t-1)}$, respectively, while b_f denotes for the bias weight vector.

***Cell**. This step computes the cell value, which combines the block input $z^{(t)}$, the input gate $i^{(t)}$ and the forget gate $f^{(t)}$ values, with the previous cell value. This can be done as depicted below:*

$$c^{(t)} = z^{(t)} * i^{(t)} + c^{(t-1)} * f^{(t)} \quad (4.39)$$

***Output gate**. This step calculates the output gate, which combines the current input $x^{(t)}$, the output of that LSTM unit $y^{(t-1)}$ and the cell value $c^{(t-1)}$ in the last iteration. This can be done as depicted below:*

$$o^{(t)} = \sigma(w_o \cdot x^{(t)} + r_o \cdot y^{(t-1)} + p_o * c^{(t)} + b_o) \quad (4.40)$$

where w_o, r_o and $p_o$are the weights associated with $x^{(t)}$, $y^{(t-1)}$ and $c^{(t)}$, respectively, while $b_o$denotes for the bias weight vector.

***Block output**. Finally, we calculate the block output, which combines the current cell value $c^{(t)}$ with the current output gate value as follows:*

$$y^{(t)} = g\left(c^{(t)}\right) * o^{(t)}. \quad (4.41)$$

In the above steps, σ, g and h denote point-wise non-linear activation functions [39].

As for the network and the data flow, Figure 28 can help to understand better its working.

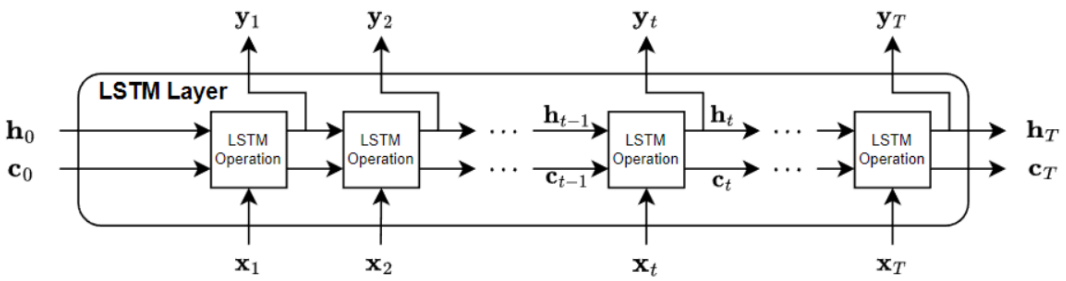

Figure 28 LSTM layer structure [37]

In this scheme it is possible to appreciate a "vertical" flow of data that goes from the input matrix $\bar{x}$ to the output vector $\bar{y}$ and an "horizontal" flow made of cell and hidden states. If the input matrix contains multiple features and the output consists of a single feature, the problem structure is defined as multivariate-input, univariate-output. Other possible configurations are multivariate-to-multivariate, univariate-to-univariate and univariate to multivariate mapping [42]. In this work the output matrix will contain many traffic features providing a speed profile as the only output [43].

5 Methodology

This chapter describes the procedure followed in generating the LSTM network, starting from the generation of the dataset, moving on to the postprocessing of the data extracted from SUMO and ending with a sensitivity analysis aimed at selecting the optimal hyperparameters.

5.1 Dataset Generation

Training an LSTM network requires the collection of an adequate amount of data relating to fundamental characteristics in the description of the physical problem and of the phenomenon to be predicted, in the case of this thesis the speed profile of the vehicle.

A vehicle's speed profile is certainly partly influenced by driver behavior; however, as seen in the chapter on traffic models, traffic characteristics can play a crucial role in this respect.

Several simulations were run to extract the data; each time, the simulation duration, time slot, and percentage of vehicles providing data were varied. The goal was to achieve a trade-off between an acceptable amount of data and reasonable processing times. Finally, it emerged that accurate results, which will be shown in the next chapter, could be obtained from a simulation that considered the time slot from 8:00 a.m. to 9:00 a.m., extracting data from 5% of the vehicles participating in the simulation. Figure 29 shows the simulation scenario. The main urban area, Turin, is located in the center, where most of the vehicles in the simulation are concentrated. Other smaller urban centers are visible in the peripheral areas. The cities are obviously connected by high-speed roads.

Figure 29 TuST scenario.

By zooming in on a road segment, as done in Figure 30, it is possible to observe the vehicles and the queue structure in which they travel. It is also possible to interact with the simulation and extract data in real time, but this possibility was not exploited in this work due to the computational cost and time required.

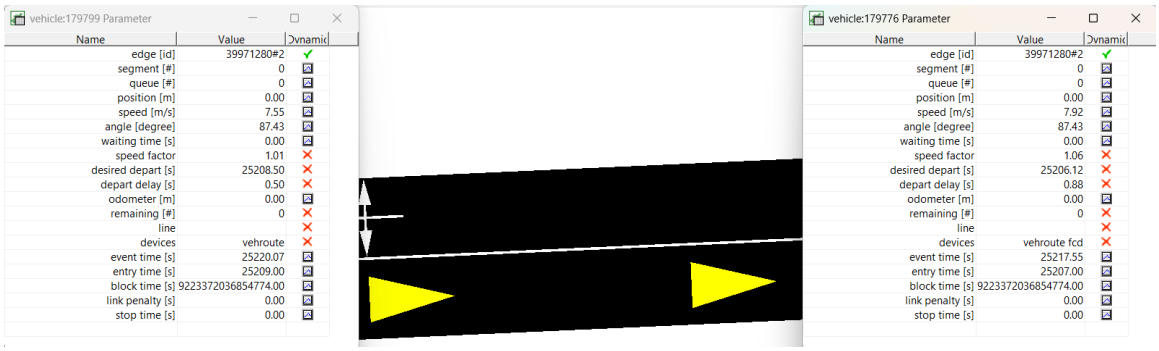

Figure 30 Detail of two vehicles with their data displayed.

This data is saved in an .xml file which contains:

- time instant [s]
- the vehicle ID
- x coordinate [m]
- y coordinate [m]
- speed [m/s]
- angle [°]
- type [passenger]
- position
- edge
- slope [°]

Traffic conditions information is collected in the form of aggregated data collected by dummy sensors placed at the edges of the roads. In this case, the information is saved to another .xml file and aggregated with a frequency of one detection every ten seconds; this is used to calculate

- traffic density on the relevant edge in [vehicles/km/lane]
- average speed [m/s]

Both files contain information about the time step and the current edge, which allows to establish a correlation between the different information contained in them and associate them with the ID of a specific vehicle.

TuST.net is a configuration file for the SUMO network, meaning it's provided as input before the simulation, determining some aspects of its operation. For example, each edge

is associated with a *maximum speed [m/s]*. This value is important because it establishes the speed limit for vehicles traveling on that stretch of road, which are allowed a slight speed limit using the "speed factor" parameter.

The vehicles participating in the simulation tend to reach the maximum speed compatible with the environmental conditions in which they operate. Thus, a comparison between the average speed on an edge and the maximum speed on the same edge becomes an indirect indicator of the degree of congestion on that road.

Care must be taken not to confuse traffic density with congestion; although the two are often related, there are cases where a road can be very densely trafficked without any vehicles being stationary, especially in mesoscopic simulations.

While FCD data are mandatory for 1 $[Hz]$ sampling due to SUMO settings, which cannot be adjusted without compromising the stability of the simulation, data collected via detectors are sampled at $[0.1\ Hz]$. This choice is based on a compromise between the need to reduce file size and considering traffic dynamics, which by nature are slower than those of a single vehicle. This difference therefore does not degrade the quality of the final dataset for training purposes of the LSTM network.

Furthermore, the sampling frequency of the detectors allows for an average speed calculation with an error no greater than 2% compared to the results obtained with a frequency of one measurement per second; this deviation was considered acceptable given the gain in computational cost and file size to be processed.

The .xml files contain several features, many of which are not strictly necessary for training the network for speed forecasting. The final choice fell on ten features, all defined as a function of time, because LSTM is optimal for processing time series. The selected features are:

- Speed
- Coordinate X
- Coordinate Y
- Edge
- Average Speed on the edge
- Maximum Speed on the edge
- Edge traffic density
- Edge traffic density +1
- Edge traffic density +2
- Edge traffic density +3
- Edge traffic density +4

Over the course of twenty-four hours, over 2.000.000 vehicles participate in the simulation, each providing data every second. This scenario produces output files too large to be processed by standard computers due to the computational cost of handling information contained in files that can exceed a TB in size. Furthermore, the LSTM network does not require this many data to be adequately trained.

Furthermore, since the model is mesoscopic, the simulation logic groups many vehicles into "queues," which, for large stretches, ensure that the components share the same characteristics. In other words, many vehicles exhibit the same behavior, adding unnecessary redundancy.

For this reason, it was decided to build the dataset based on a simulation shorter than the total possible and by selecting a small sample of randomly chosen vehicles. The aim is to get close to the minimum number of vehicles necessary to obtain accurate predictions while reducing the computational cost and data processing time.

It was also decided to look for a time interval where the number of vehicles in the simulation was high but local congestion was not yet present, so as not to have long sequences of constant speed profiles of stationary vehicles.

It was decided to extract data over a one-hour interval starting from 8 am to 9 am, obtaining data from 8000 vehicles; from this entire fleet, 2000 vehicles were randomly selected for the generation of the training, validation and testing sequences of the LSTM network. Thanks to a Python script, the data from the different .xml files were cross-referenced and a .csv file was produced for each vehicle for which the information was saved.

5.2 Postprocessing

Data provided by SUMO is raw and presents several issues, and their use is not recommended since it presents several critical issues that will be analyzed with the help of the Figure 31.

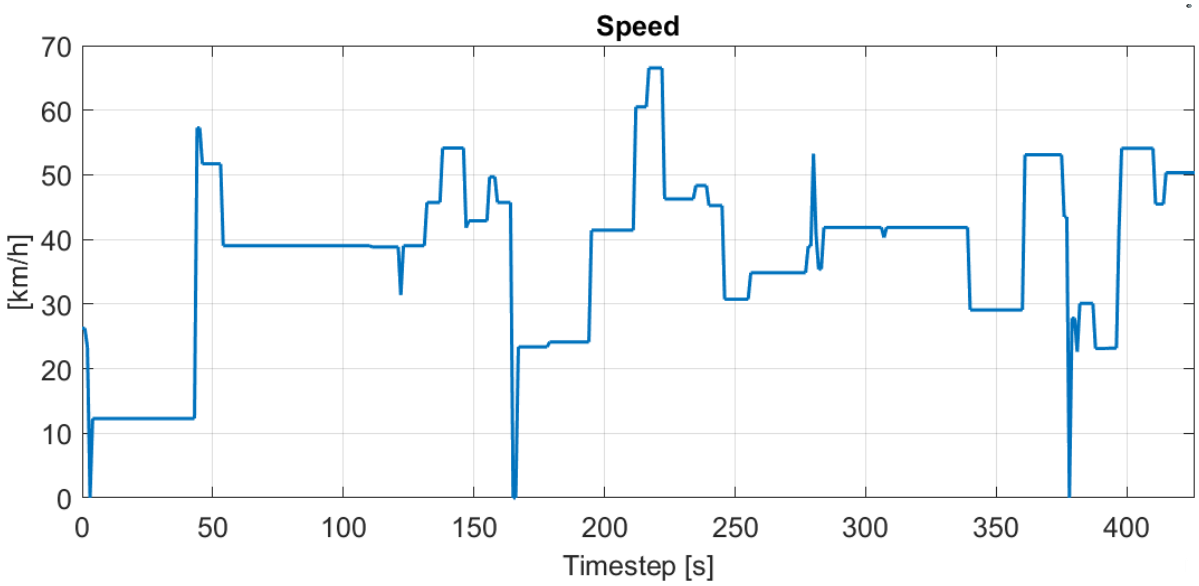

Figure 31 Speed profile of a vehicle

The speed profile shown in the figure is a SUMO Output and summarizes many of the issues with the available sequences. First, note that the profile is a set of sequences of constant strokes with little temporal dynamics and is completely unrealistic. Training a network in this way is difficult because LSTM requires time-varying sequences, and from a realistic perspective, no driver is capable of being so constant while driving. While it's plausible that the driver will maintain a constant speed, it's also true that the vehicle will travel at a speed that fluctuates around the constant value; in practice, it's necessary to add a $\pm 1/2$ % noise to the signal to simulate the modulation the driver uses on the accelerator while driving.

Another rather obvious problem is the drastic change in velocity that occurs between one instant and the next at some points along the profile. The accelerations presented are specific and exaggerated, essentially devoid of any physical meaning, since credible values do not exceed +-4 m/s^2 [6], while in the case in question, peaks of ± 13 m/s^2 are reached, as possible to observe in Figure 32 in the red circles.

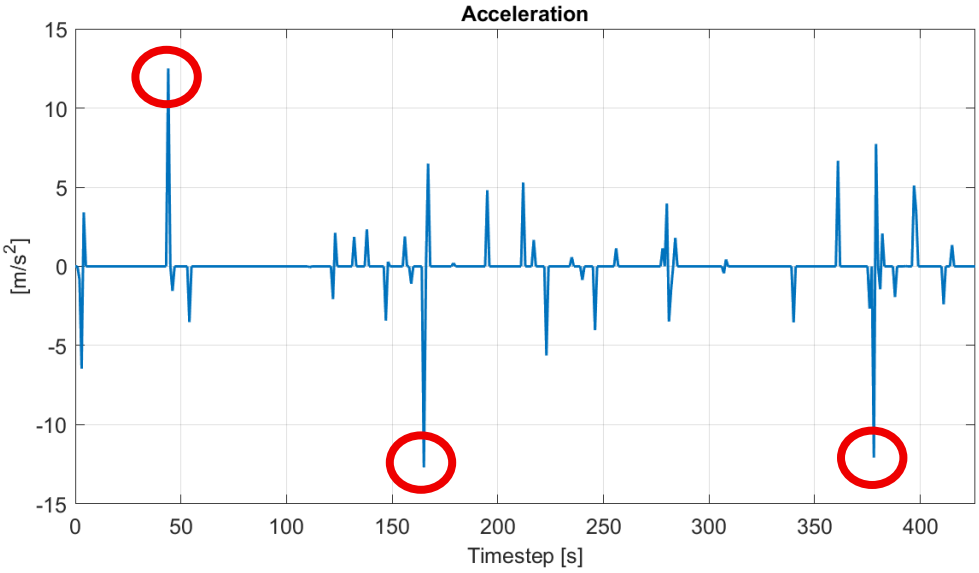

Figure 32 Acceleration profile of the vehicle from Figure 27

A final problem arises from unavoidable transcription errors that appear as spurious values excessively distant from those close to them in the sequence, such as peaks in which the velocity suddenly becomes 0 and then immediately stabilizes on values slightly lower than 5 m/s.

Combining the application of a moving average filter with the superimposition of noise, we obtain velocity profiles that are more realistic, and we cancel spurious values due to incorrect transcriptions and this way, LSTM training will also be more effective.

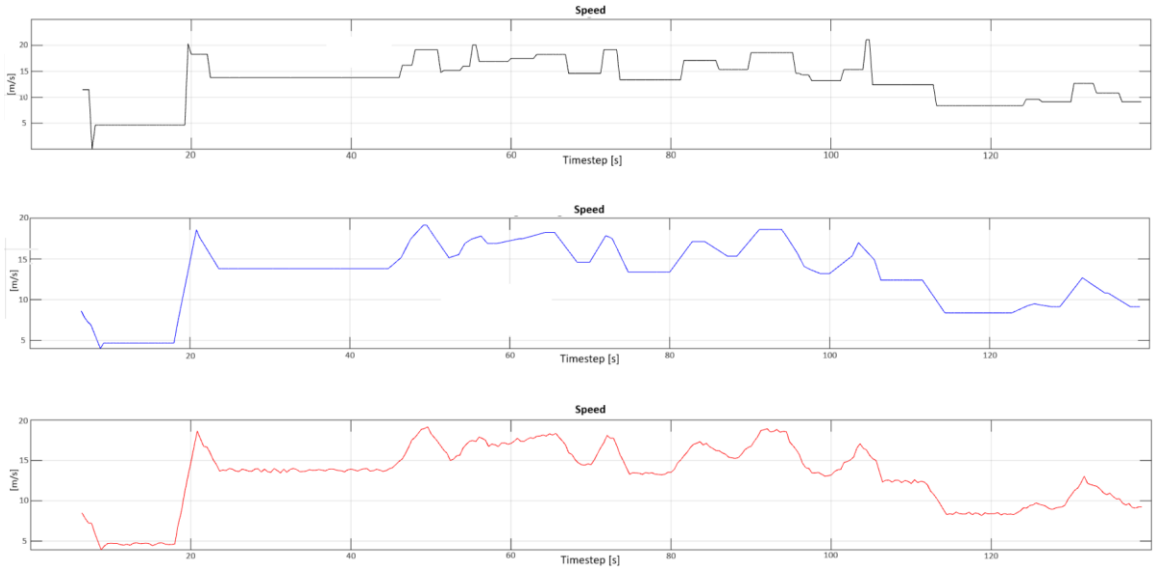

Figure 33 Comparison between raw speed profile (black line), speed profile filtered (red line) and filtered speed profile with noise (red line)

5.3 LSTM Training

Once the data has been filtered, it must be prepared for use in training the LSTM neural network. First, it's important to remember that the thesis is focused on a supervised network model, which is why the training data must be divided into input and output data. The idea behind this setup is that training is performed by showing the network the "premises" (inputs) and the "consequences," i.e., the evolution of the target feature shifted by one second compared to the input data.

Since we are working on a multivariate-to-univariate configuration, the input data must be organized according to a matrix [F x T] where F refers to the features and T are the timesteps, in this case each timestep is equivalent to one second. Also, the network requires the values to be normalized and centered around zero. As for the output, being univariate, it is sufficient to provide a vector of the velocity profile, as previously said translated by one second with respect to the input matrix.

Of the total sequences generated from the 2.000 vehicles, 70% were selected as the training set, 20% as the validation set, and 10% were used for testing. This means that the final sequences are never shown to the network during training and are used to verify the accuracy of the LSTM predictions.

Naturally, the more features used for training, the longer the training takes, and the learning quality doesn't necessarily improve. For this reason, there are numerous techniques that allow selecting the most significant features, reducing the dimensionality of the problem. In the case study, the features making up the dataset are relatively few, and nine are initially selected: speed, acceleration (which was derived posthumously from the speed sequences), maximum speed, average speed, and the traffic densities of the edges.

5.4 Sensitivity Analysis of the Hyperparameters and the Architecture of the network

Neural networks are characterized by numerous parameters, called hyperparameters, which have a fundamental influence on the quality of the network's training. It is necessary to evaluate the optimal values of these parameters through a sensitivity analysis process.

The goal of this preliminary procedure is to identify the set of hyperparameters that promise the most accurate training. Once identified, some predictions will be shown and commented on in the results.

Figure 34 shows the result of a forecast made with random values of the hyperparameters. The figure's title reads "autoregressive prediction," a concept that will

be explored further later; it's essentially a technique for making forecasts over very long time horizons. What's interesting to note for now is the solution's poor accuracy, which highlights the need for sensitivity analysis.

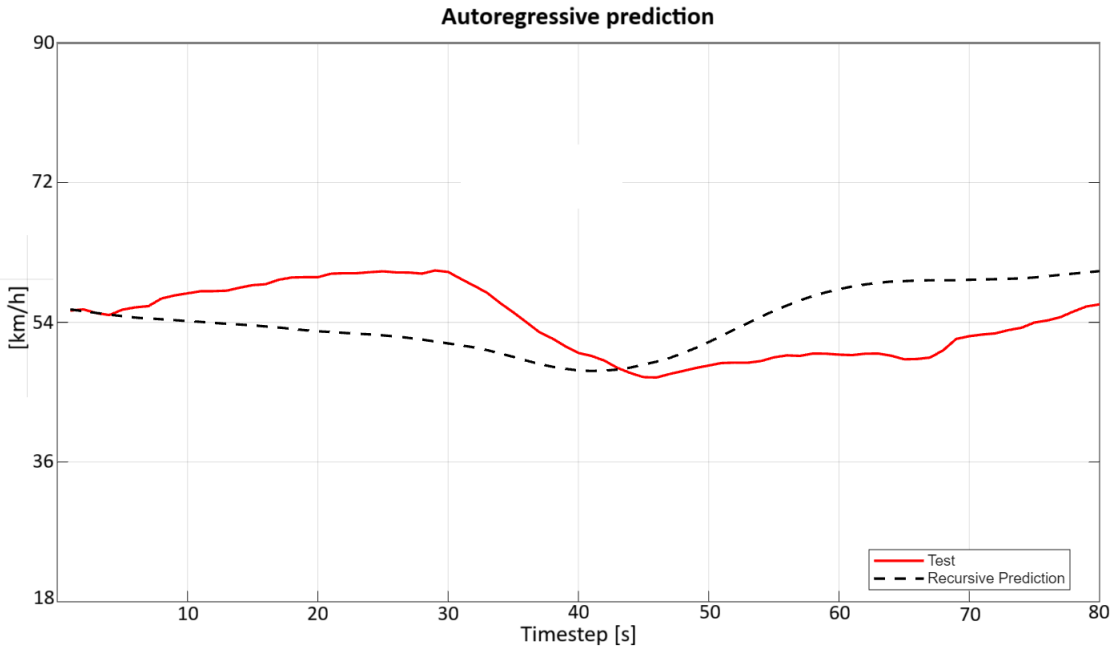

Figure 34 Sequence Prediction before the sensitivity analysis

The metrics used to compare the networks are two, Root Mean Square Error (RMSE) and Mean Absolute Error (MAE), which compare the predictions with the real test sequences according to the following equations:

- $RMSE = \sqrt{\sum_{i=1}^{N} \frac{1}{N} (y_i^{true} - y_i^{prediction})^2}$ (5.42)
- $MAE = \frac{\sum_{i=1}^{N} \left| y_i^{true} - y_i^{prediction} \right|}{N}$ (5.43)

The variables appearing in equations 5.42 and 5.43 are:

- y_i^{true} = i-th element of the vector representing the real velocity profile
- $y_i^{prediction}$ = i-th element of the vector representing the predicted velocity profile
- N = number of elements that form the previous vectors.

The first sensitivity phase involved the following parameters:

- *Number of vehicles used to generate training and testing sequences*

  As already mentioned, the total number of vehicles used is 2000; this figure appears to be a good compromise between final prediction accuracy and computational cost.

- *Window size*

This number identifies the timestep length of the training sequences; since each timestep is equivalent to one second, *window size* coincides with the number of columns of the input matrix and the output vector. In this analysis the windows chosen are 5, 10, 15, 20 or 25 timesteps.

- *Layers number*

This value identifies the depth of the neural network; networks that are too deep risk overfitting the data and increasing computational cost of the training. A low depth network can underfit data.

It was decided to analyze networks with 1, 2, or 3 layers.

- *Number of hidden units*

This number represents how many neurons are contained in each layer; again, the ideal would be to find a trade-off between network size and training time. In this thesis, the choice was made between 100 and 200 units.

- *Dropout value*

To avoid overfitting, it's good practice to silence a percentage of the network's neurons at each iteration of the training phase. The technical term for the percentage of deactivated neurons is dropout, and in this case, it will vary between 10%, 20%, and 30%.

The combination of these parameters produced eighteen networks, which were evaluated based on the test sequences. There are more than 25,000 of these and evaluating them all requires significant computational effort. For this reason, the approach to calculating the metrics was incremental; starting with ten random sequences, the metrics were calculated by increasing the number of sequences at each cycle. Once the number of sequences reached approximately one thousand, the results stabilized, yielding the five best networks, always the same ones and always in the same order.

Table 4 and Table 5 will show the hyperparameter combinations on which the first phase of the analysis was performed.

Network	Number of vehicles	Number of layers	Window size	Number of input features	Number of hidden units	Dropout value
LSTM	2000	1	20	9	100	0.1
LSTM	**2000**	**1**	**20**	**9**	**100**	**0.2**
LSTM	2000	1	20	9	100	0.3

LSTM	2000	1	20	9	200	0.1
LSTM	2000	1	20	9	200	0.2
LSTM	2000	1	20	9	200	0.3
LSTM	2000	2	20	9	100	0.1
LSTM	2000	2	20	9	100	0.2
LSTM	2000	2	20	9	100	0.3
LSTM	2000	2	20	9	200	0.1
LSTM	2000	2	20	9	200	0.2
LSTM	2000	2	20	9	200	0.3
LSTM	2000	3	20	9	100	0.1
LSTM	2000	3	20	9	100	0.2
LSTM	2000	3	20	9	100	0.3
LSTM	2000	3	20	9	200	0.1
LSTM	2000	3	20	9	200	0.2
LSTM	2000	3	20	9	200	0.3

Table 4 Hyperparameters of the compared networks for the first step of the sensitivity analysis. The highlighted row is the best one based on RMSE and MAE

Once this starting point was established, the discussion was explored in depth by assessing the window sizes.

Network	Number of vehicles	Number of layers	Window size	Number of input features	Number of hidden units	Dropout value
LSTM	2000	1	5	9	100	0.2
LSTM	2000	1	10	9	100	0.2
LSTM	2000	1	15	9	100	0.2
LSTM	**2000**	**1**	**20**	**9**	**100**	**0.2**
LSTM	2000	1	25	9	100	0.2

Table 5 Networks compared during the second phase of the sensitivity analysis. Highlighted row set the best one based on RMSE and MAE

The comparison metrics are calculated on the predicted series compared to the actual solution. There are more than 25.000 test sequences available, making testing them all very computationally expensive. The comparison was performed on 5.000 sequences randomly selected from the testing set.

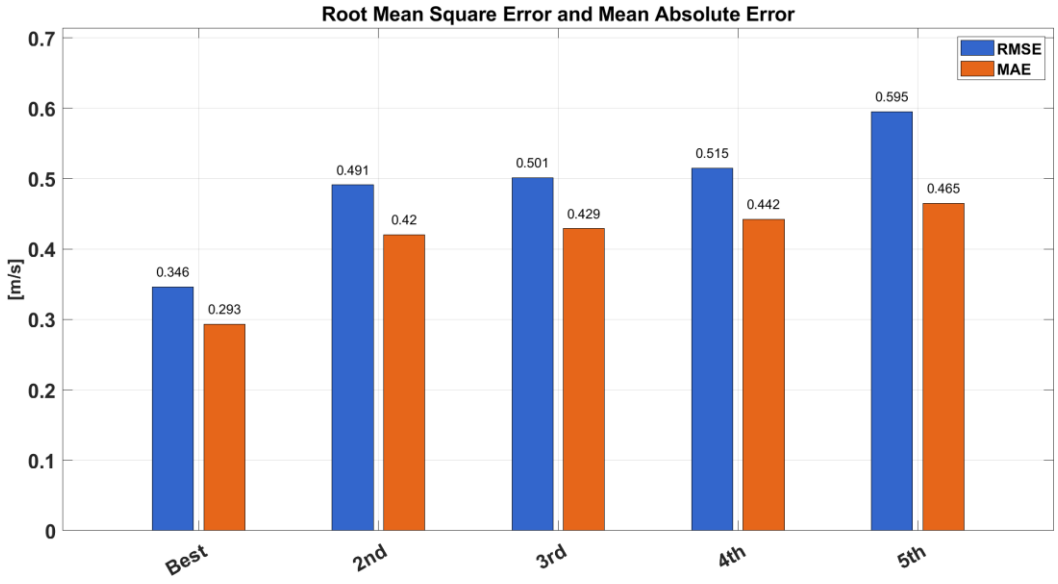

Figure 35 Metrics of the five most performing networks of the first half of the sensitivity analysis

The graph in Figure 35 shows the results of the first phase of sensitivity analysis, with a particular focus on the five best networks. The blue columns identify the Root Mean Square Error, while the orange columns are the Mean Absolute Error.

As expected, the deepest three-layer networks do not appear among the best networks, as they risk overfitting the training data. The most accurate network has an intermediate dropout and 100 hidden units, which is the minimum value among those considered.

Networks are named using a numeric code that identifies the hyperparameters on which the analysis was performed. The logic is as follows: LSTM_[Number of vehicles]_[Number of layers]_[Window size]_[Number of hidden units]_[Dropout value].

Once the first hyperparameters were chosen, a second phase has been performed to determine the final value of another set of parameters, listed and briefly explained below.

- *Number of epochs*

  The network is trained through an iterative process in which, at each step, all the update operations described in the theoretical chapter on LSTM neural networks occur. Each iteration is called an *Epoch*, and too few of them risk underfitting, while too many risks overfitting. The values compared are 5, 10, 20 and 40 epochs.

- *Mini batch size*

  The network isn't trained by processing the entire dataset on a single batch, but rather by handling groups of information that are processed in parallel. These samples are

called *mini batches*, and their size impacts the training time and the accuracy of the final predictions. The values compared are 64, 128, 256 and 512

- *Initial learning rate*

Initial value used by the Adaptive Momentum Estimation (ADAM) optimization algorithm to update weights and biases. It has been opted for values to be compared equal to 0.0005, 0.001, 0.002 and 0.005

- *Gradient threshold*

The LSTM network is optimized and designed to avoid vanishing gradients; however, this does not prevent the opposite problem, namely gradient explosion, from occurring in certain circumstances. Choosing value for this parameter forces the network to never exceed that gradient during training. Choosing a Gradient Threshold value equal to infinite means that there is no intervention on gradient during the training of the LSTM neural network. In this case the values of the Gradient Threshold selected are 1,2,5 and Inf.

Through the Table 6 it is possible to observe the configurations of the LSTM networks that have been compared.

Network	Optimizer	Initial Learning Rate	Mini Batch Size	Maximum Epochs	Gradient Threshold
LSTM	ADAM	5e-3	64	10	Inf
LSTM	ADAM	5e-3	128	10	Inf
LSTM	ADAM	5e-3	256	10	Inf
LSTM	ADAM	5e-3	512	10	Inf
LSTM	ADAM	2e-3	64	10	Inf
LSTM	ADAM	2e-3	128	10	Inf
LSTM	**ADAM**	**2e-3**	**256**	**10**	**Inf**
LSTM	ADAM	2e-3	512	10	Inf
LSTM	ADAM	1e-3	64	10	Inf
LSTM	ADAM	1e-3	128	10	Inf
LSTM	ADAM	1e-3	256	10	Inf
LSTM	ADAM	1e-3	512	10	Inf
LSTM	ADAM	5e-4	64	10	Inf
LSTM	ADAM	5e-4	128	10	Inf
LSTM	ADAM	5e-4	256	10	Inf
LSTM	ADAM	5e-4	512	10	Inf

Table 6 Hyperparameters of the compared networks for the first step of the sensitivity analysis. The highlighted row is the best one based on RMSE and MAE

Given the large number of networks compared, calculating the metrics on all the test sequences requires more than a day of calculations, which is why an incremental approach was adopted.

Starting from a limited number of randomly chosen sequences (10), these were gradually increased until, by repeating the calculation, the ranking of the five best networks remained unchanged each time.

Starting from 1000 sequences the result converges as shown in the following figure.

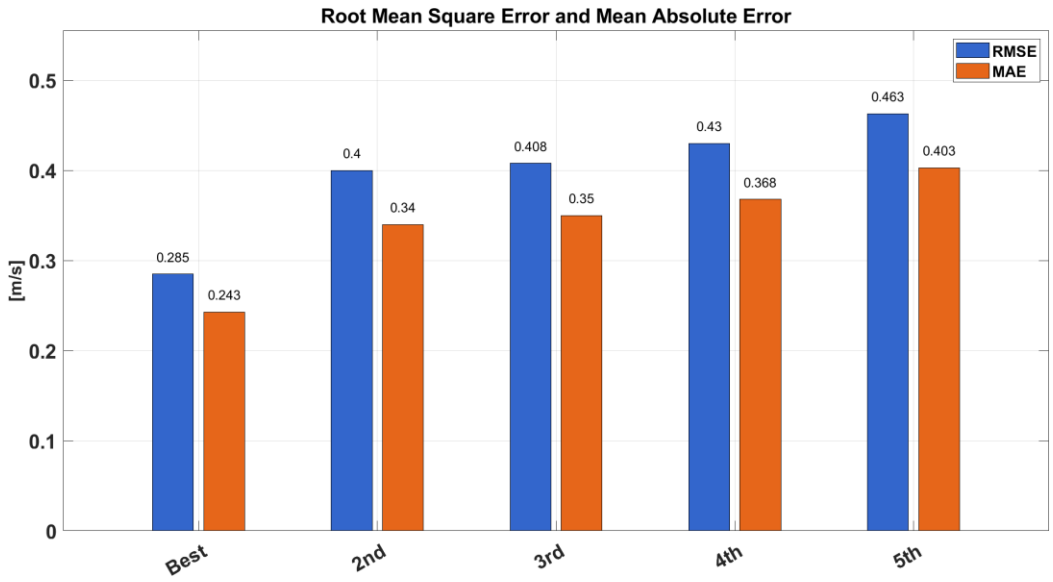

Figure 36 RMSE and MAE values

The graph in Figure 36 shows the results obtained by continuing the analysis. The column color scheme remains consistent with Figure 35; what changes is the network naming scheme.

In this case, in fact, the numerical values must be interpreted in the following way: LSTM_[Number of epochs]_[Mini Batch Size]_[Learning Rate]_[Gradient Threshold].

Comparing the two networks with the best metrics, at the second pass the RMSE value is reduced to 82% compared to the initial value, going from 0.346 [m/s] to 0.285 [m/s].

A similar reduction, equal to 83%, occurs when comparing the MAE of the two networks, which goes from 0.293 [m/s] to 0.243 [m/s].

In the previous step, the Gradient Threshold was kept constant and equal to INF. However, it is a parameter that can have a certain impact on the quality of the training, which is why, using the most accurate network in the graph in Figure 30 as a basis, this last parameter was calibrated.

Network	Optimizer	Initial Learning Rate	Mini Batch size	Maximum Epochs	Gradient Threshold
LSTM	ADAM	2e-3	256	10	1
LSTM	ADAM	2e-3	256	10	2
LSTM	ADAM	2e-3	256	10	5
LSTM	**ADAM**	**2e-3**	**256**	**10**	**Inf**

Table 7 Hyperparameters of the compared networks

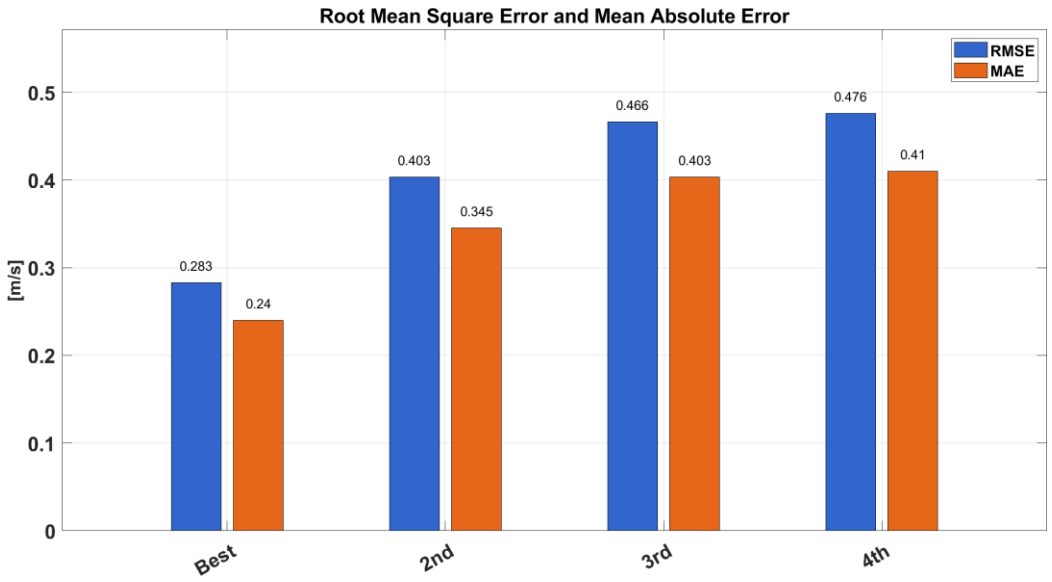

Figure 37 Gradient Threshold focused plot of RMSE and MAE of four compared LSTM networks

The graph in Figure 37 confirms the superiority of the network in Figure 36; it is possible to observe a slight improvement in the metrics resulting from the fact that, being a comparison between 4 networks versus the 16 of the previous step, in this case it was possible to calculate RMSE and MAE on more sequences, obtaining a more refined result.

In the case of 16 networks, as it was for the first comparison with 18 networks, the calculation of the metrics was iterative and incremental until the top 5 settled on the same networks.

A last calibration concerns the number of Epochs; this tuning confirmed the most accurate configuration used from now on.

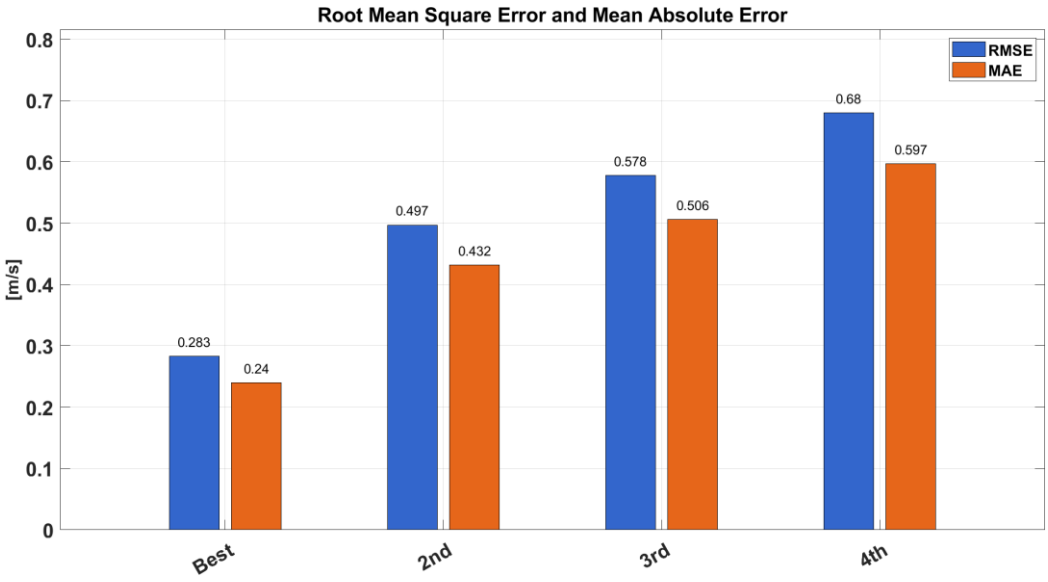

Figure 38 Epochs-focused plot of RMSE and MAE of four networks

The following table summarizes the hyperparameters selected following the sensitivity analysis.

From now on it is understood that any other LSTM neural networks that will be generated will have parameters in table 8, unless otherwise specified.

Hyperparameter	**Value**
Number of vehicles generating dataset	2000
Number of hidden layers	1
Window size	20
Number of hidden units	100
Dropout	0.2
Optimizer	Adam
Number of epochs	10
Mini Batch sizes	256
Gradient Threshold	Inf
Initial Learning Rate	0.02

Table 8 Hyperparameters chosen from the sensitivity analysis

6 Results

Once the most accurate LSTM network has been trained and selected, it is possible to look at its predictions to understand its strengths and weaknesses, further experimenting with configurations and leveraging the network in different ways to structure the predictions to work best for optimizing energy management strategy of the hybrid powertrain.

First, it should be noted that the forecasting approach used is a sliding windows approach, meaning that the input features are provided to the network in time-dependent sequences of 20 timesteps, or 20 seconds, in length. This information is used by the network to predict the target quantity (in this case, velocity) at the immediately subsequent timestep. This procedure can be iterated by shifting the input feature matrix by one timestep at each step.

A key aspect lies in the nature of the input features; these can be composed entirely of real data (taken from the testing set) or progressively replaced by predicted data. The first is a "standard" use case for the LSTM network, meaning it was designed and optimized specifically for this task. Its practical utility, however, is very limited, considering that the prediction horizon is limited to one second, which is not functional for optimization. However, it can be useful to observe some predictions of this type, because low accuracy at this stage would indicate significant errors in the network's training phase.

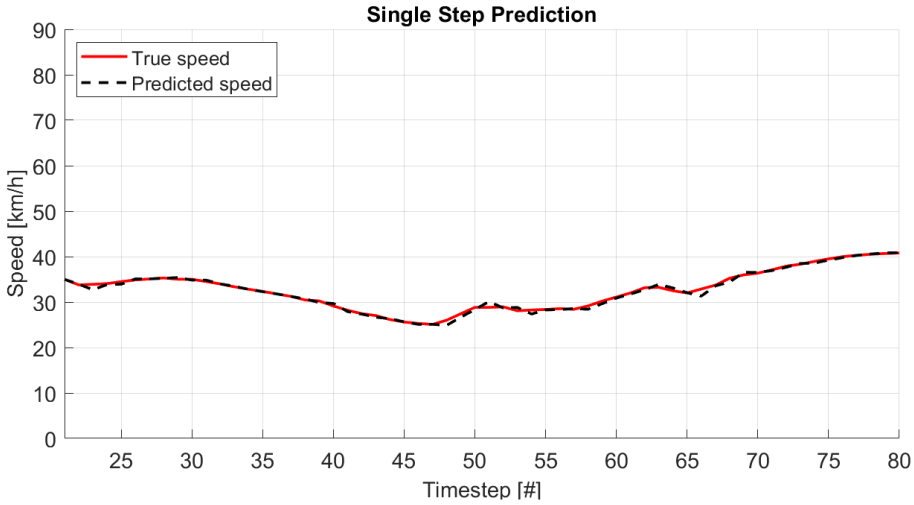

Figure 39 Comparison between a randomly chosen real test sequence and predicted one in a single step configuration

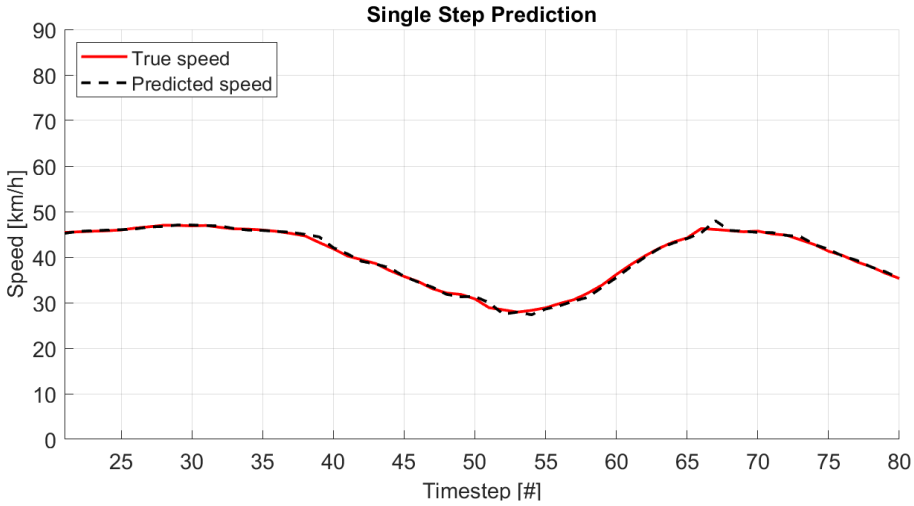

Figure 40 Comparison between a randomly chosen real test sequence and predicted one in a single step configuration

From the previous figures it is clearly seen that the profile derived from the forecasts (dotted line) and the real speed profile (red line) overlap with an excellent degree of accuracy, this fact is not surprising, but it is nevertheless an indication of a good starting point. Prediction in Figure 40 have a RMSE that values 0.18 [m/s] and a MAE of 0.14 [m/s].

The gap between the two profiles is minimal, but the most noticeable criticality is a slight delay in the prediction at the local and absolute minimum and maximum points. It was perfectly predictable that the two profiles would not be identical, since a complete overlap would indicate an error, perhaps due to the coupling between an overfitted training and the presence of the test sequence in the training dataset.

As previously established, forecasting over a 1-second time horizon has little practical relevance in this context, however the good compatibility between 1-step forecasts and test sequences allows us to hypothesize an algorithm that progressively inserts the predicted speed values into input, so that starting from a certain instant it is possible to make a prediction over a longer time horizon.

This type of approach will be defined as autoregressive approach and at this point the input features will be the usual 9; however, since the real velocity profile is progressively replaced by the predicted one, there will inevitably be an accumulation of error which in the long term will lead to a drift of the two sequences.

Strictly speaking, acceleration should also be recalculated based on the forecasts, but always with a view to a progressive approach, the acceleration values derived from the real profile will initially still be used, an inaccuracy that will be corrected in the future.

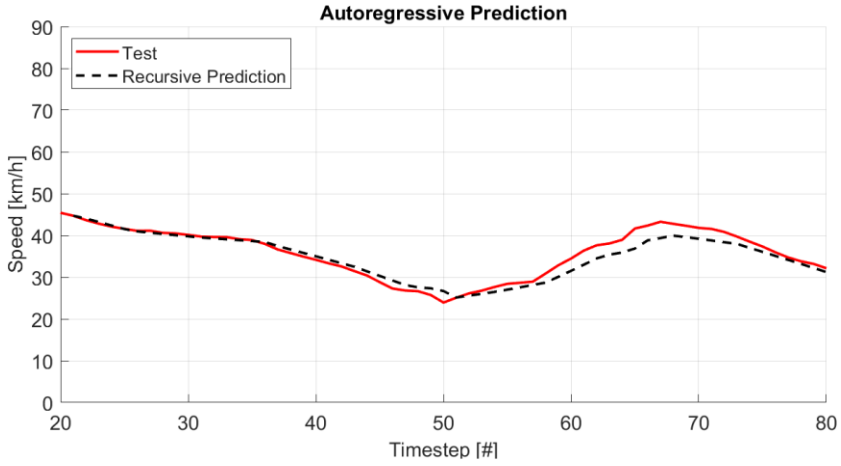

Figure 41 Comparison between a randomly chosen real test sequence and predicted one in an autoregressive configuration

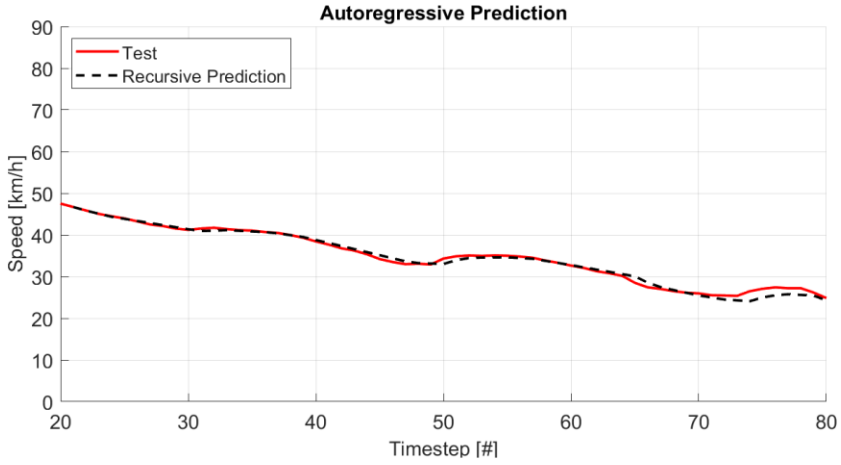

Figure 42 Comparison between a randomly chosen real test sequence and predicted one in an autoregressive configuration

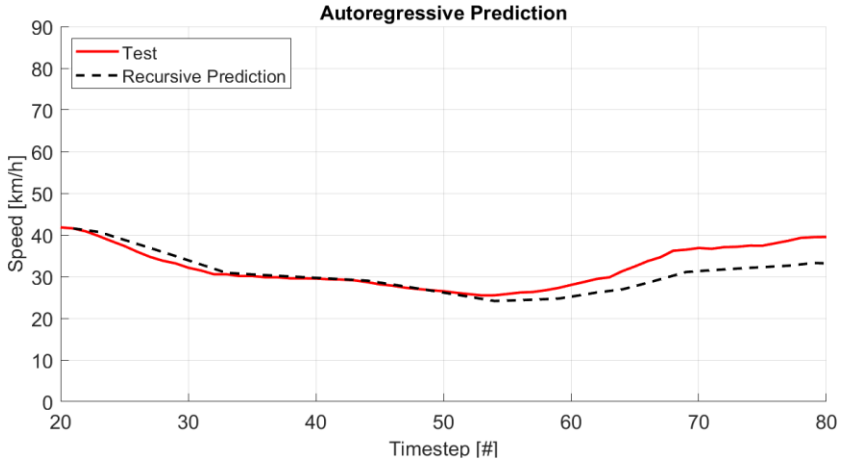

Figure 43 Comparison between a randomly chosen real test sequence and predicted one in an autoregressive configuration

The sequences shown in Figure 41, Figure 42 and Figure 43 are randomly selected and represent some of the most accurate results found. The predictive horizon extends to 60 seconds, and it is interesting to note that the predicted drift does not appear to occur significantly.

This behavior is not common to all available test sequences, in fact in some cases the prediction is less accurate, although still acceptable as it can capture many of the main characteristics of the real profile in terms of trend, maximum, minimum and concavity points, as can be observed in Figure 44.

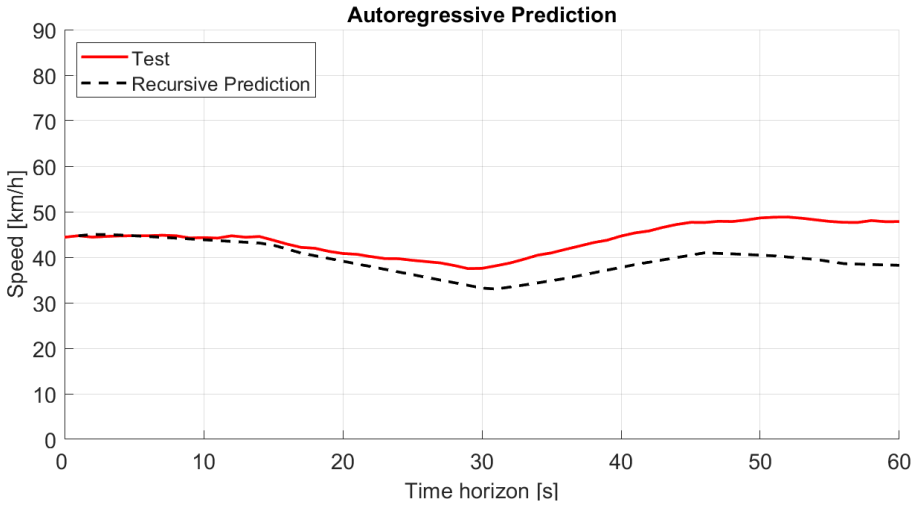

Figure 44 Comparison between a randomly chosen real test sequence and predicted one in an autoregressive configuration

In certain sequences it is possible to observe a much more accentuated drift, especially in cases where there are no temporal dynamics, as in Figure 44.

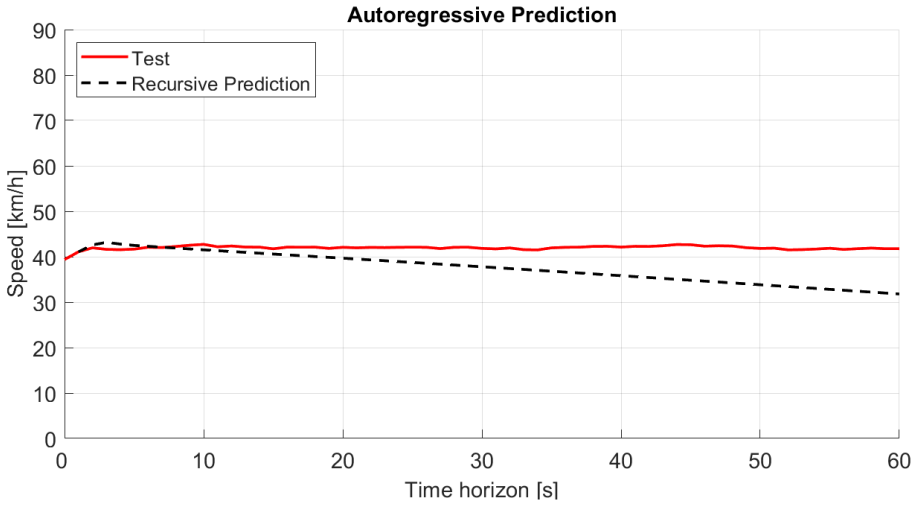

Figure 45 Comparison between a randomly chosen real test sequence and predicted one in an autoregressive configuration

The results obtained so far are encouraging, but this approach presents some critical issues. First, velocity is a feedback loop, while acceleration is still based on real data. This is problematic because, at the first instant in which the prediction and actual velocity do not match, the next instant the input acceleration is provided that does not match what would be deduced from the prediction.

Furthermore, it should be noted that the time selected for data extraction is the one with the highest number of vehicles present before the congestion that SUMO suffers from occurs (small but still present). However, this is a transition time during which many vehicles are inserted into the simulation. The result is that for many vehicles, the density of the edges following the current one is quite low, since the simulation tends to place vehicles in the least congested areas.

For this reason, a five-input feature LSTM network was trained,

- Speed $[m/s]$
- Acceleration $[m/s^2]$
- Max speed $[m/s]$
- Average speed $[m/s]$
- Edge Density $[\#vehicles/lane/km]$

The choice of hyperparameters was made consistently with the results of the sensitivity analysis while in the autoregressive prediction stage the acceleration given as input is derived from the predicted profile and not from the real one.

In the following figures some predictions are shown:

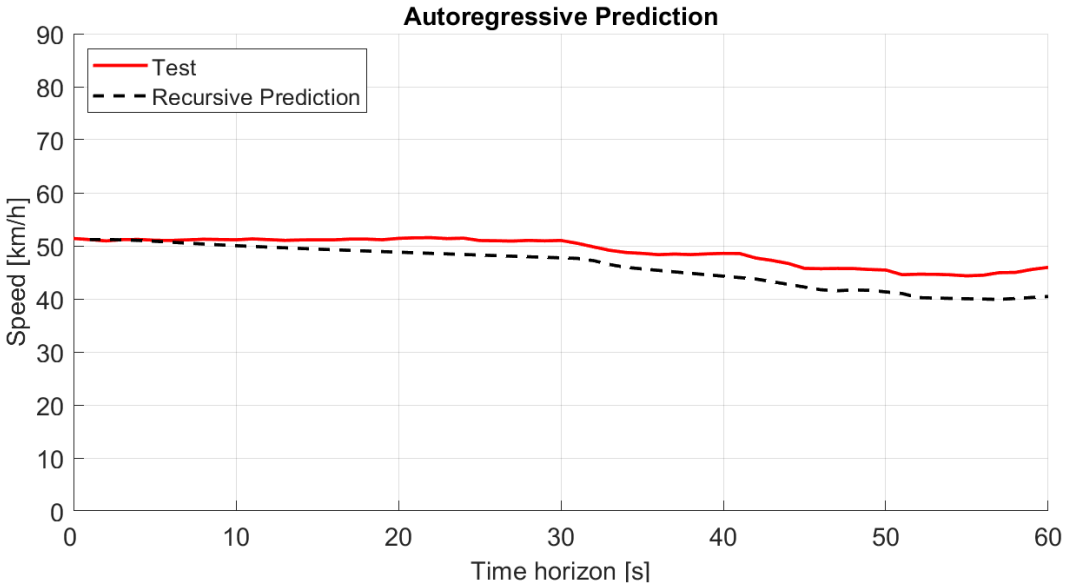

Figure 46 5-input LSTM network prediction compared with test sequence.

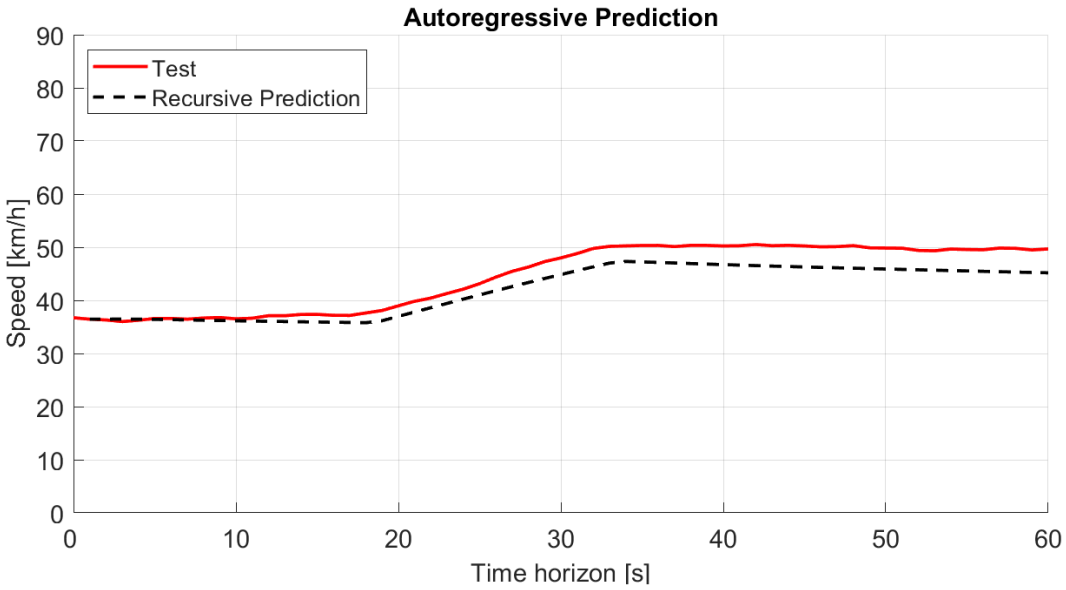

Figure 47 5-input LSTM network prediction compared with test sequence.

The two previous figures allow us to appreciate a more evident drift as expected even for sequences with a more evident temporal dependence. However, there remain some sequences in which the accuracy seems to improve along the predictive horizon; this phenomenon is due to the combination of a slightly accentuated drift and the temporal delay of the prediction which causes the two trajectories to converge at certain time segments, as highlighted in Figure 48, Figure 49 and Figure 50.

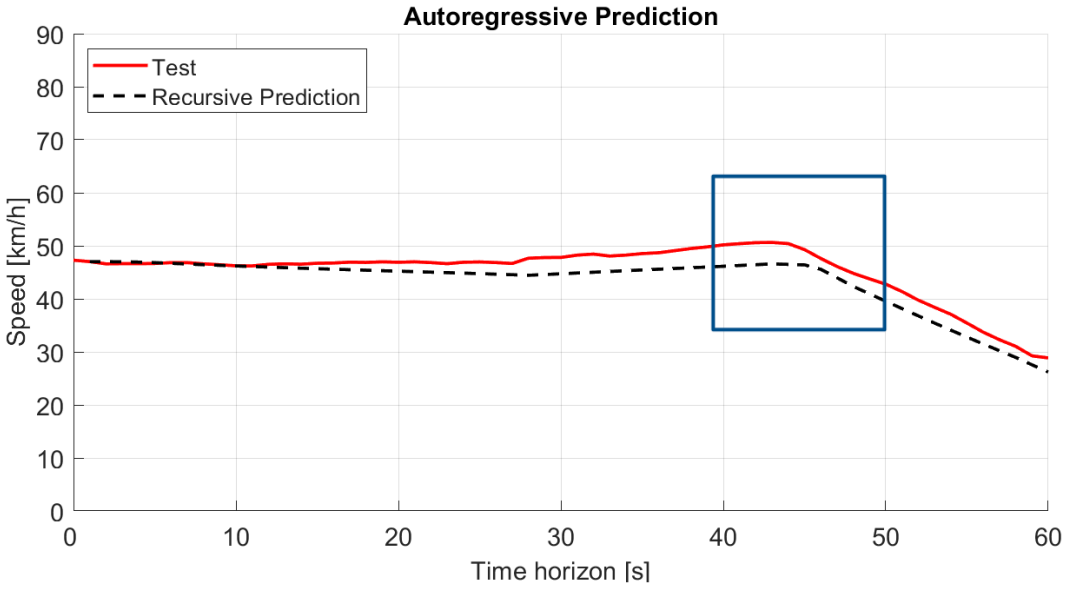

Figure 48 5-input LSTM network prediction compared with test sequence.

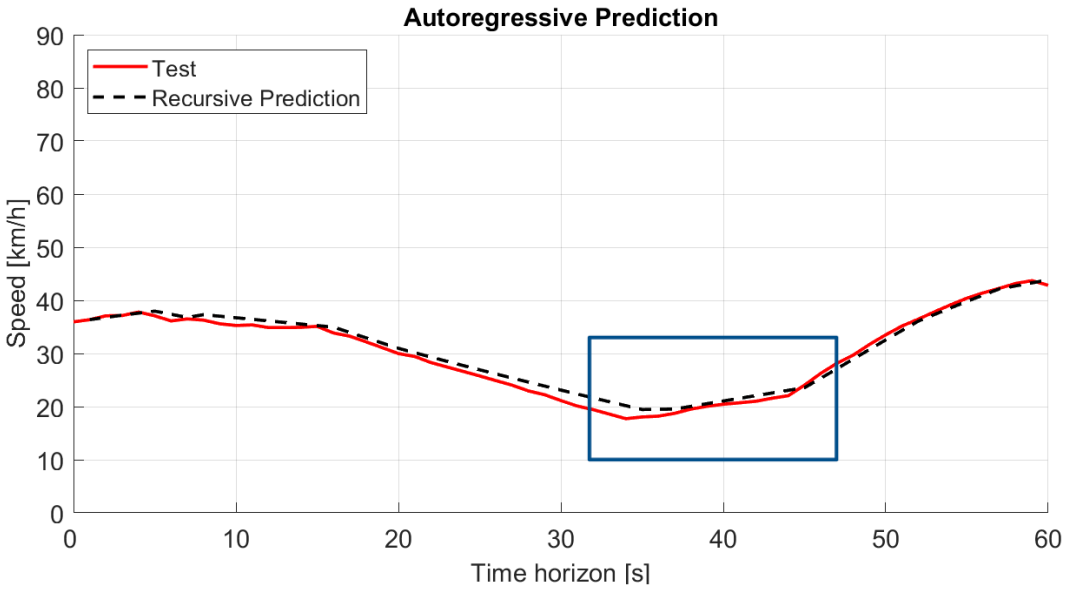

Figure 49 5-input LSTM network prediction compared with test sequence.

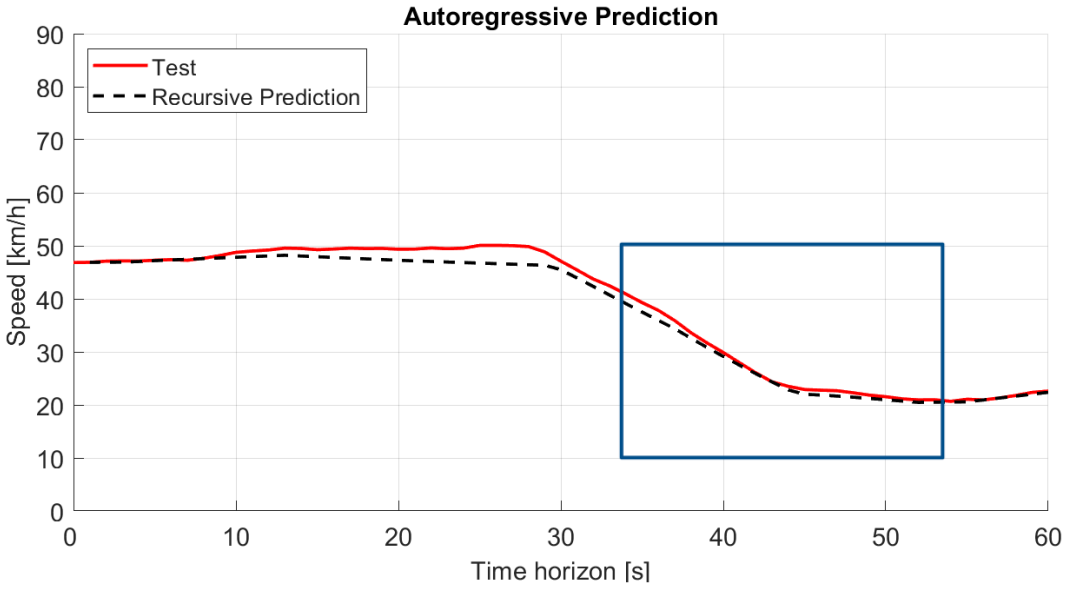

Figure 50 5-input LSTM network prediction compared with test sequence.

The blue rectangles in Figure 48, Figure 49 and Figure 50 show some of the cases in which the phenomenon just mentioned occurs, the result is that the divergence between the two sequences can occur later than in other sequences; obviously this effect is obtained in cases in which the forecast is an underestimation but the sequence presents a decreasing trait or, vice versa, if the forecast overestimates the profile and an increasing profile is presented.

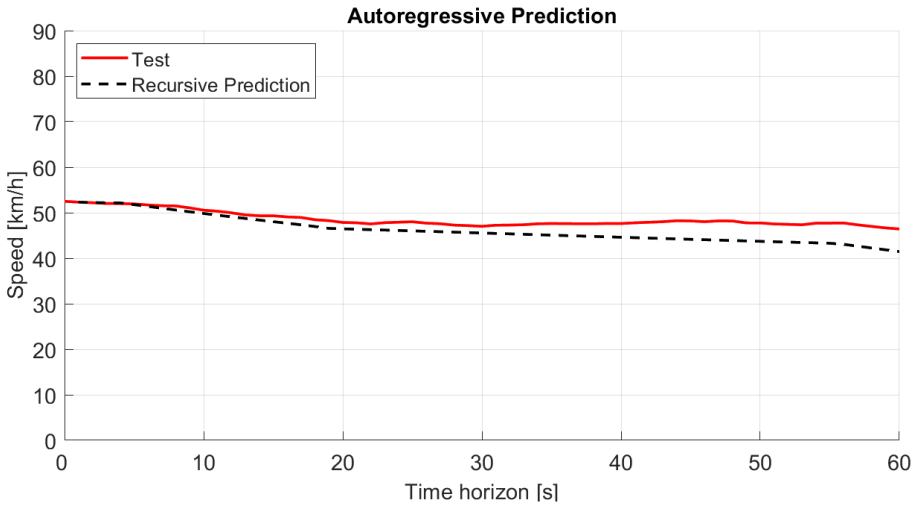

Figure 51 5-input LSTM network prediction compared with test sequence.

Overall, a slightly less rounded forecast profile is also noted compared to the 9-input case, which may be due to the lack of some features in the training phase; however, the forecast accuracy does not generally appear to be compromised, and the predictions obtained can be useful for managing the energy strategy.

Regarding this last application, which has the most practical relevance, it should not be forgotten that nothing prevents the vehicle from tracking its actual speed profile. This makes the actual profile a valuable ally in constantly correcting the forecast, so that the energy optimization strategy can always work on the most accurate version available.

The logic at this point becomes as follows: in the first twenty timesteps, the actual velocity profile data is collected and used to display the first forecast over a time horizon longer than a second. Every second thereafter, the forecast is remade, each time using the actual velocity values from the immediately preceding twenty timesteps.

This is, in a certain sense, a hybridization of the initial purely closed-loop approach and a purely autoregressive forecast. Indeed, if we take the first value of all the forecast profiles we obtain at the end of this operation, the profile will mirror that of a pure closed loop on the same sequence. The fundamental difference is that in the second case, the sequence can only obtain a posteriori at the end of the journey, while in the initial forecasts like those in Figure 39 and Figure 40, this reasoning was not applied.

All the forecast profiles that start at each instant of the test sequence are instead obtained in an autoregressive manner according to a sliding window logic and in fact tend to diverge in the long term, as will be clearly visible in the following figures.

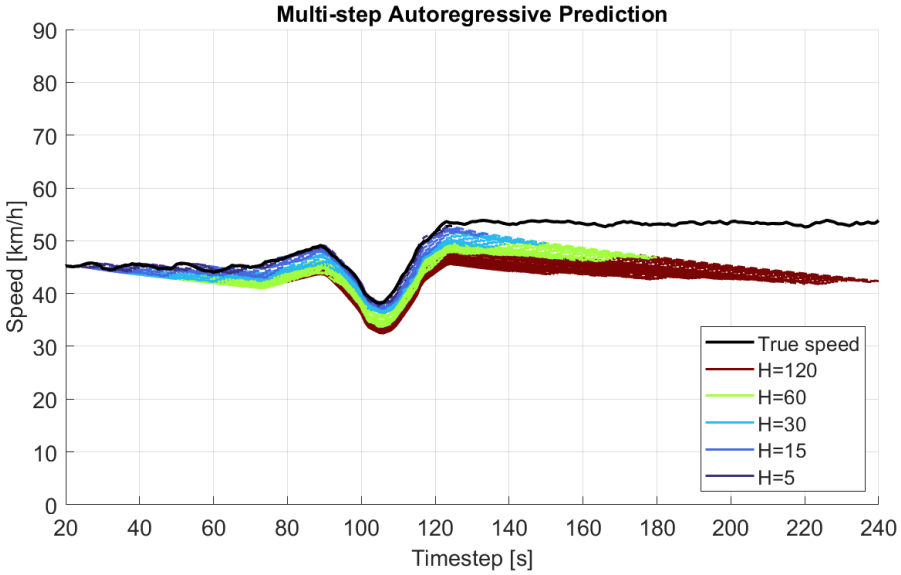

Figure 52 Multi-step and multi-horizon prediction plot of a random sequence

The Figure 52 shows a visual example of the concept expressed above. In this case, multiple forecasts were calculated at each instant, each with an increasing time horizon. Each horizon is characterized by a different color, ranging from blue shades for the shortest horizons (5-15 timesteps). The green shade indicates a forecast that reaches one minute, as in many of the previous plots, while the red line reaches up to two minutes into the future.

The reason these test sequences are 240 timesteps long, rather than the 60, 80, or 100 seen previously, is to allow the last prediction produced, the one originating at timestep 120, to be visible in its entirety, having a predictive horizon of 120 timesteps.

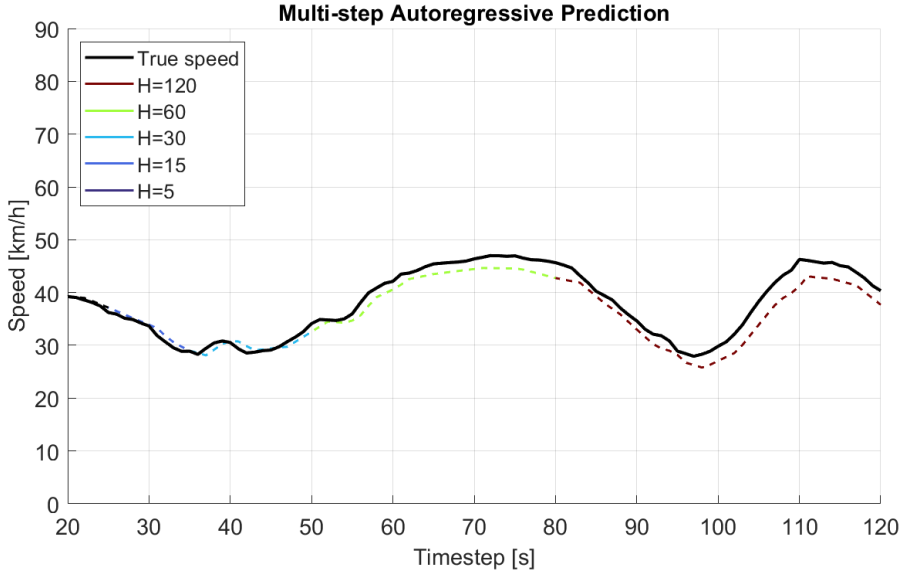

Figure 53 Detail of a single multi-horizon prediction

In Figure 53 the set of forecasts generated starting from a single instant is shown. Obviously, each sequence initially follows the shortest-term forecasts.

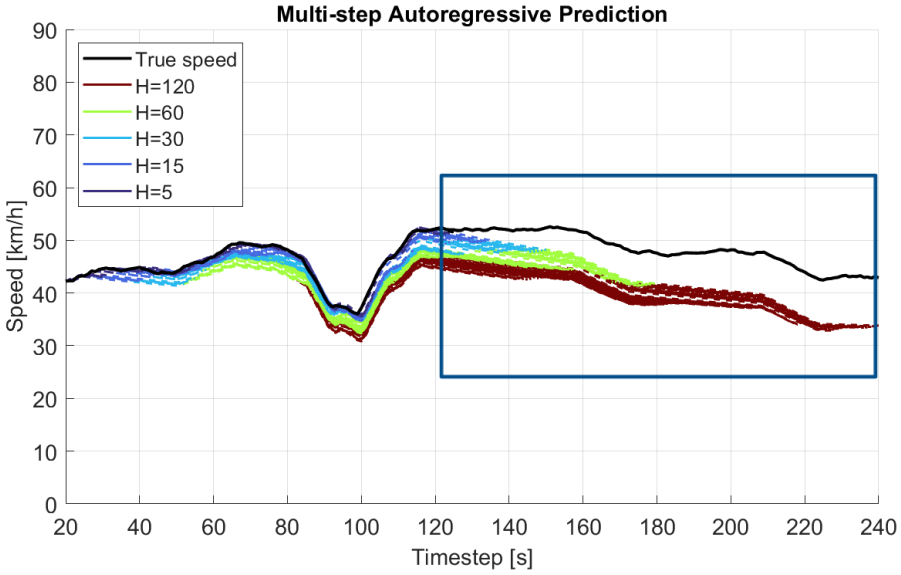

Figure 54 Multi-step and multi-horizon prediction plot of a random sequence

All graphs show, with a different degree of amplitude, the inevitable drift to which the forecasts are subject, as highlighted by the blue rectangle in Figure 54.

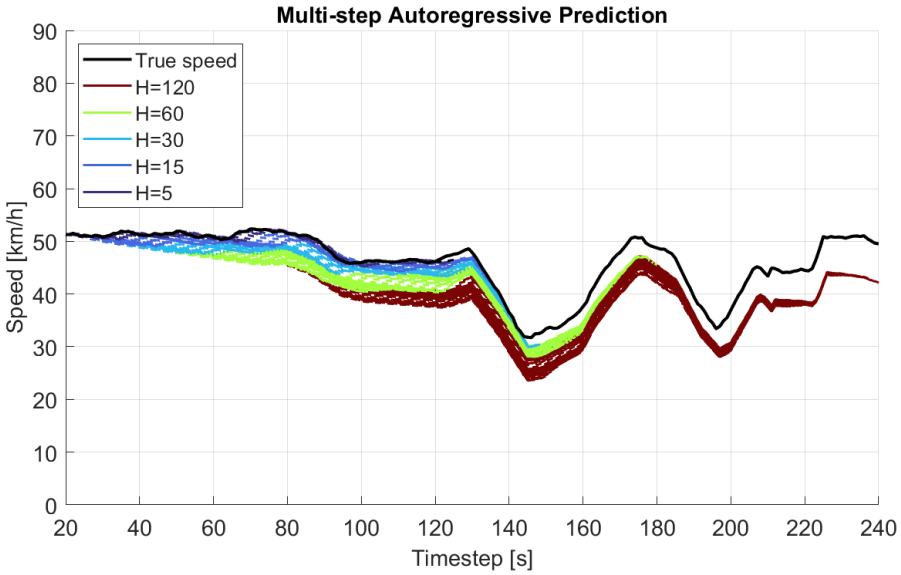

Figure 55 Multi-step and multi-horizon prediction plot of a random sequence

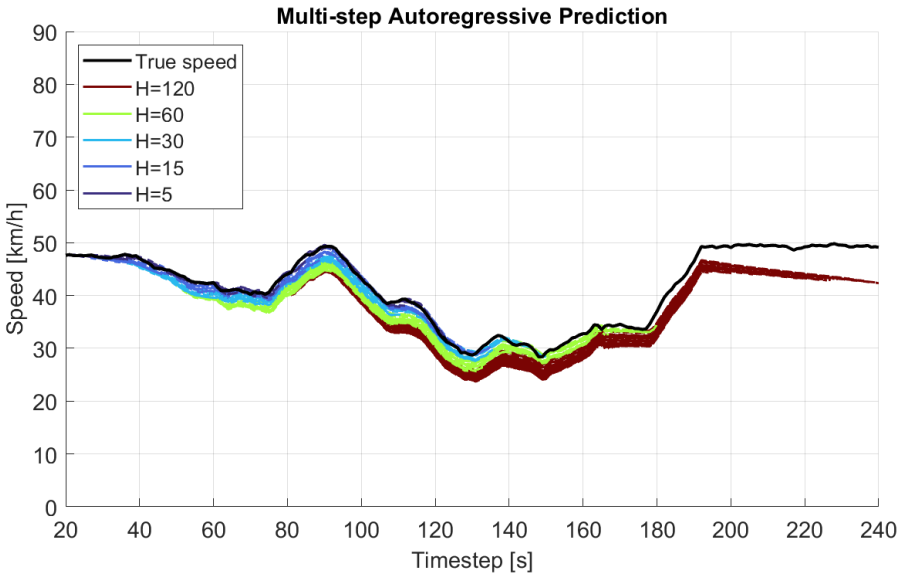

Figure 56 Multi-step and multi-horizon prediction plot of a random sequence

Sequences similar to those shown in Figure 55 and Figure 56, characterized by a more marked temporal dynamics, show more accurate forecast trends.

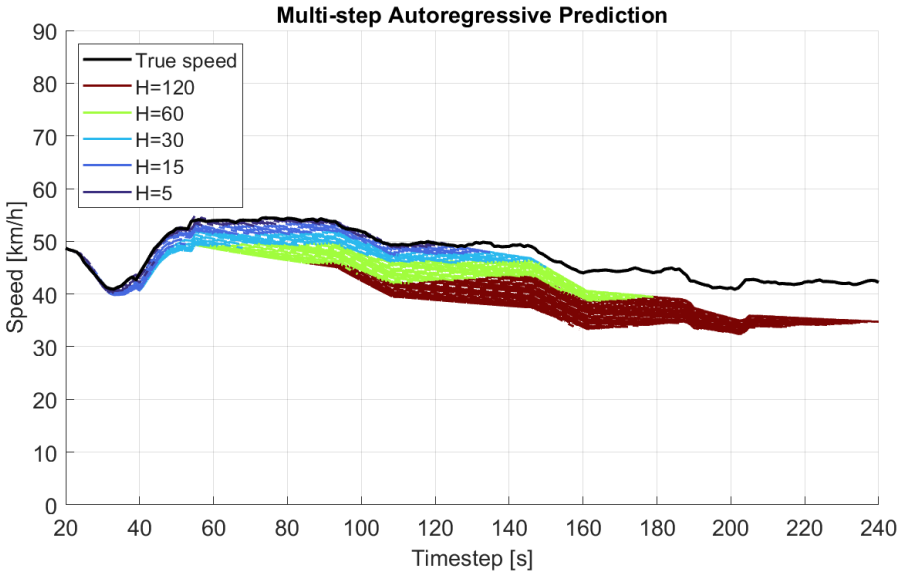

Figure 57 Multi-step and multi-horizon prediction plot of a random sequence

The choice to extract sequences randomly is made considering the large number of possible test sequences, which makes it extremely difficult to evaluate all sequences. By extracting randomly, we try to capture the quality of the results more generally, avoiding optimizing the network for a single specific sequence.

The comparison between different cases can be performed again by means of Root Mean Square Error and Mean Absolute Error metrics. Computational costs are dependent on

the size of the testing dataset, the number of prediction sequences associated with a testing speed profile, the amount of horizons considered and the length of them.

As previously written, testing dataset counts more than 25.000 sequences, making a global evaluation a hard task. Still it is possible to make comparisons between single scenarios considering sequences strongly characterized by significant features.

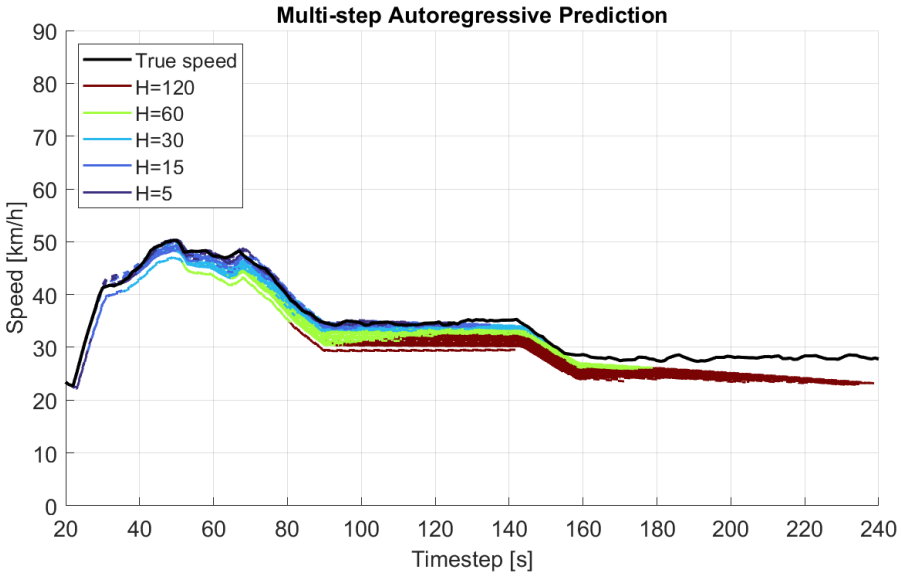

Figure 58 First sequence considered for metrics calculations

In Figure 58 a testing sequence with a medium temporal dynamic is considered. The general trend looks well fitted and as expected, for longer prediction horizons (red lines) drift increases. The sequence presents two semi-constant segments and a first segment highly time dependent.

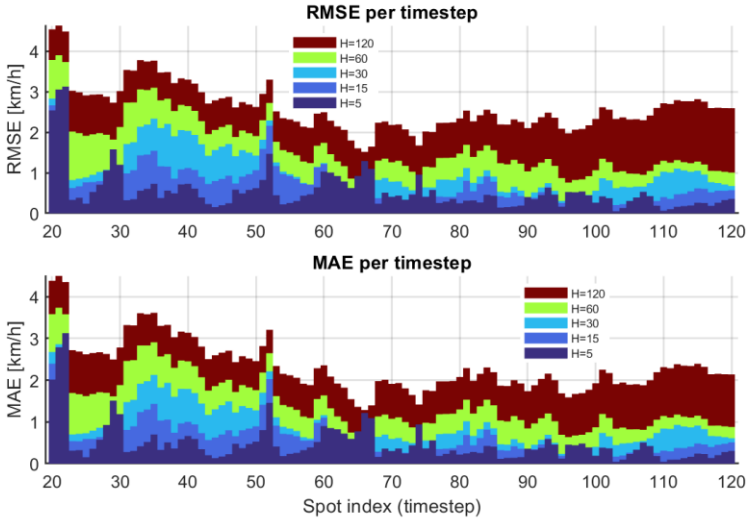

Figure 59 Distributed RMSE and MAE in [km/h] for the sequence in Figure 58.

Figure 59 shows RMSE and MAE in [km/h] related to all the prediction sequences calculated in Figure 58. On the x axis, called *Spot index*, there are the timesteps where the predictions start and the columns show the values of the metrics associated with the prediction originating from the correspondent timestep. Each color is associated with a temporal horizon according to the choice made for the plots of the sequences.

A deeper look at the graphs shows erratic behavior and the following features:

- In the first three timesteps the mean value of the metrics is larger than the rest of the sequence, highlighting a lack in accuracy which is visible also in Figure 58 by the set of detached predictions at the beginning of the sequence.

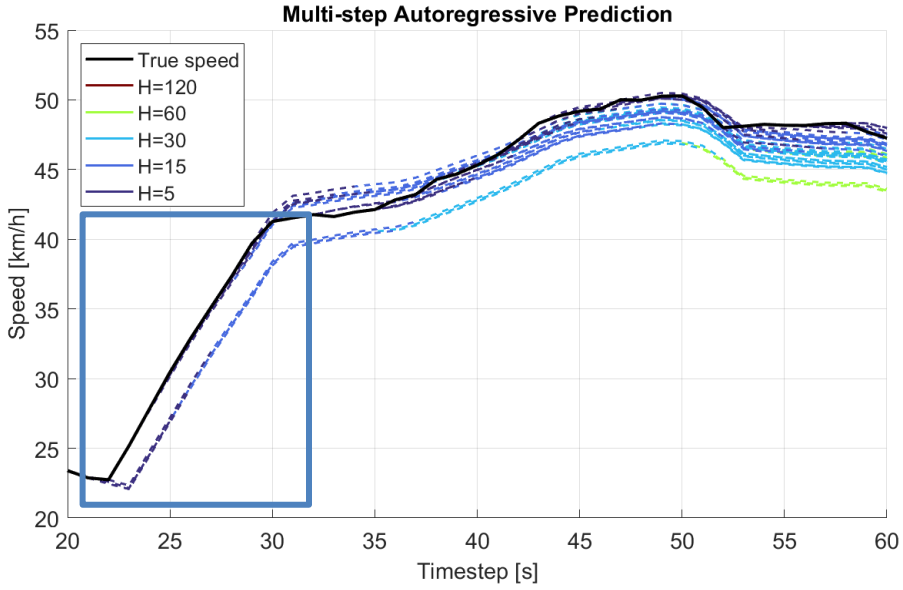

Figure 60 Detail from the multi-step and multi-horizon prediction in Figure 58

- The general trend shows that as the horizon increases; the predictive accuracy decreases and on average for most of the horizons the mean error does not exceed 4 $[km/h]$.

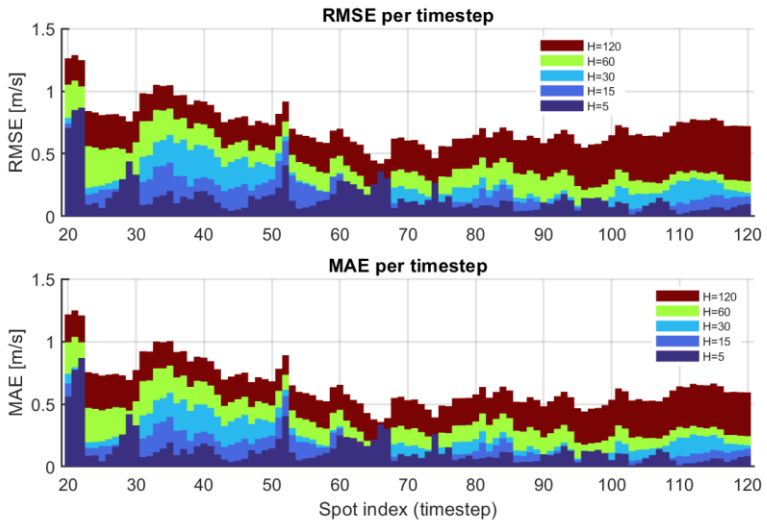

Figure 61 Distributed RMSE and MAE in [m/s] for the sequence in Figure 58.

To maintain consistency with the chapter on sensitivity analysis, the same results are shown in Figure 61 with the units of measurement translated into [m/s].

To reduce the dispersive effect of the error calculated according to the criteria described above, it has been decided to furtherly concentrate the results by calculating the average RMSE and MAE by horizon. A further step consists of calculating the average of the metrics referred to the horizons (which will be called Global RMSE and MAE) so that for each sequence there is a single parameter that allows a first rough assessment of the accuracy of the set of forecasts.

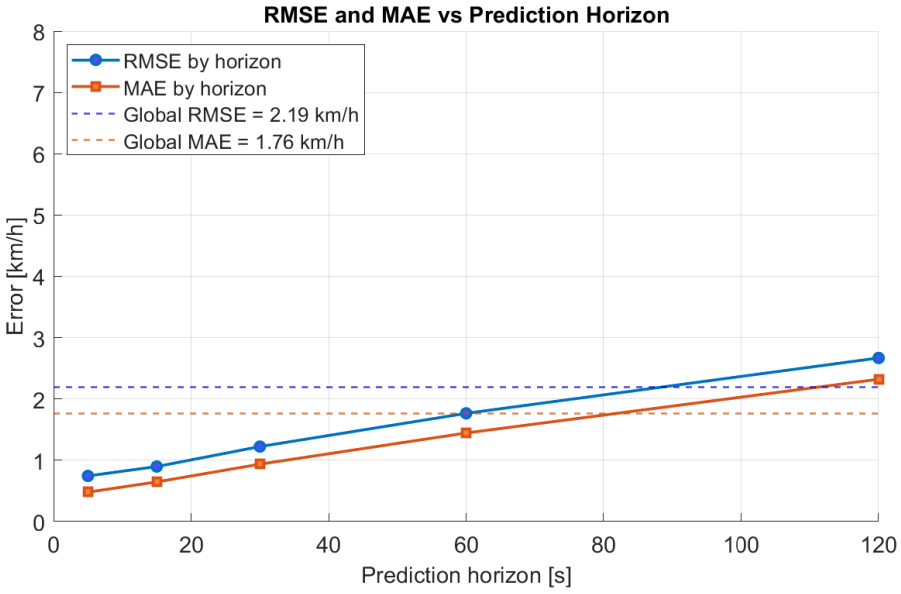

Figure 62 Horizon and Global metrics [km/h] for the sequence.

The previous figure shows the comparison between global RMSE and MAE (dashed line) and the separate averages of the metrics calculated based on the reference prediction horizon (solid lines). The trend of the two metrics are increasing for both and parallel as can be intuited from the observation of Figure 59.

Previously, it was noted that predictions for constant sequences tend to be less accurate due to a more pronounced drift and the impossibility of correcting the accumulation of error that occurs in more time-varying sequences. For this reason, a constant sequence was taken and examined with the same approach as the sequence in Figure 58.

Note that by constant we mean a sequence in which the only temporal dynamics is given by the noise produced by the artificially imposed accelerator modulation to imitate the behavior of a real driver.

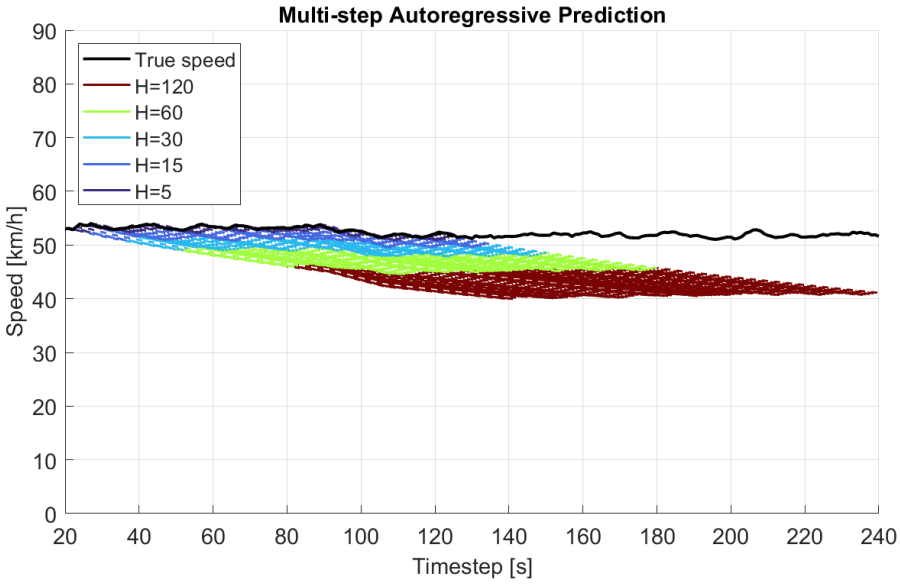

Figure 63 Constant sequence set of predictions.

Looking at the sequence, the lower accuracy of the predictions is immediately noticeable. Therefore, a higher and more constant average value for the metrics is expected, considering that the predictions differ only slightly from each other.

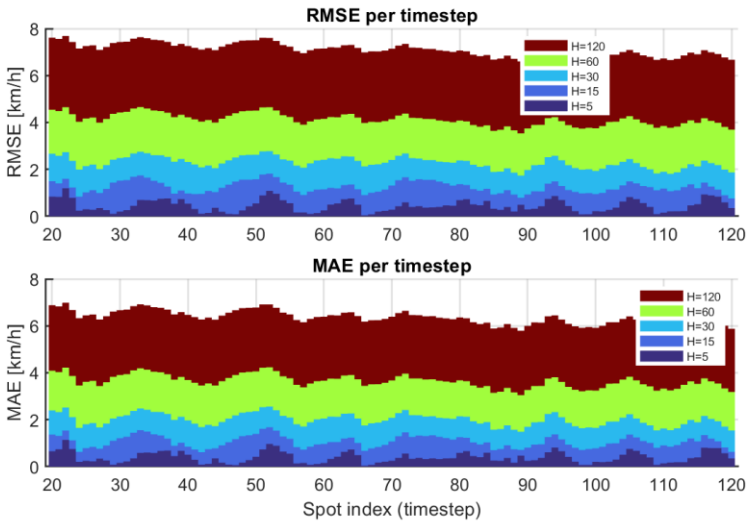

Figure 64 Distributed RMSE and MAE in [km/h] for the sequence in Figure 63.

The expectations are confirmed by the plot in Figure 64, where the error metrics assume higher average values than in the previous case, furthermore the structure is decidedly more defined in terms of stratification and less oscillating.

In cases like this one might choose to ignore certain time horizons if the mean error exceeds a certain tolerance threshold and one notices that this is consistently exceeded by a certain time horizon.

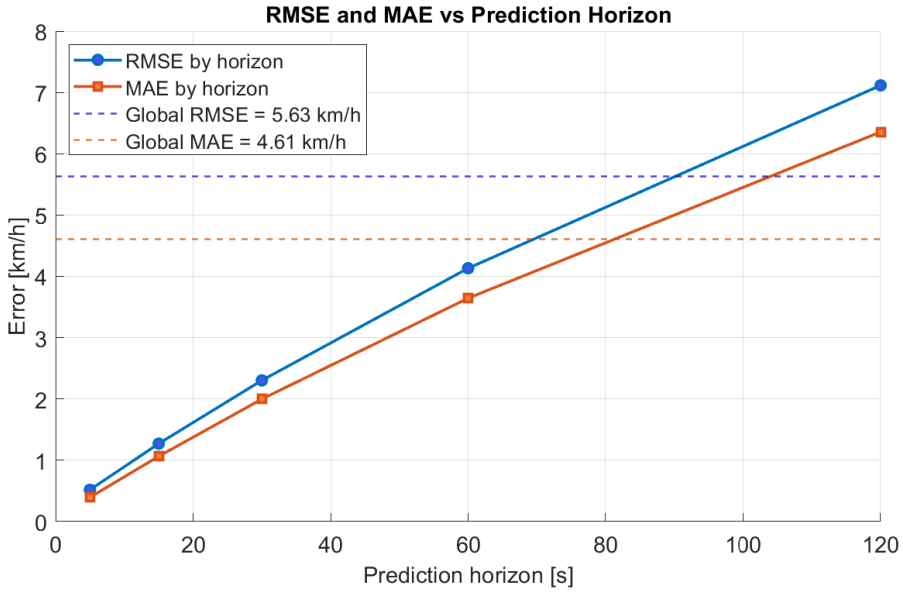

Figure 65 Horizon and Global metrics [km/h] for the sequence in Figure 63.

From Figure 65 it is possible to see how global metrics are sensibly higher, global RMSE in this case is equal to 5.63 $[km/h]$ with respect to the previous situation where global

RMSE was equal to 2.19 $[km/h]$, a value approximately equal to 39% of the worst scenario.

For what concerns global MAEs, in the last sequence the computed value is equal to 4.61 $[km/h]$ compared to the one of the first case that has a value of 1.76 $[km/h]$, equal to 38% of the worst scenario.

In percentage terms, the difference between the average value of MAE and RMSE for the first and second case does not differ significantly, since in the first sequence the MAE is approximately 80.4% of the RMSE, while in the second the MAE is approximately 81.9% of the RMSE. This might be index of a good stability of the network with respect to data variation.

Regarding the average metrics by horizon, the trends are strongly increasing and, unlike previously, a slight divergence between RMSE and MAE is noted as the prediction horizon increases.

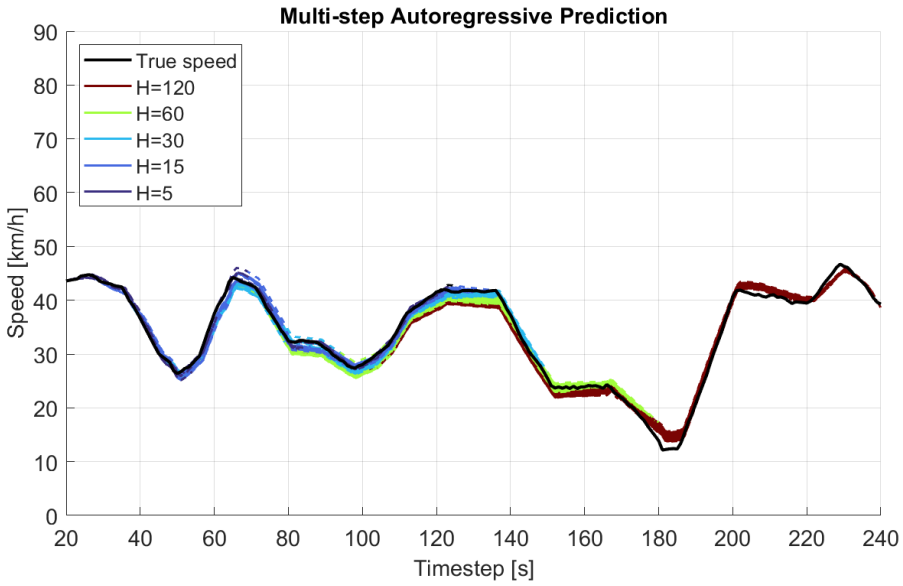

Figure 66 Time-dependent sequence set of predictions.

In Figure 66 a sequence is shown characterized by a set of particularly accurate forecasts and an extremely accentuated temporal dynamic. At first glance, it is difficult to establish which time horizon is the most accurate, and at times it even seems that long-term forecasts (120 seconds) are better than those with a more limited time horizon.

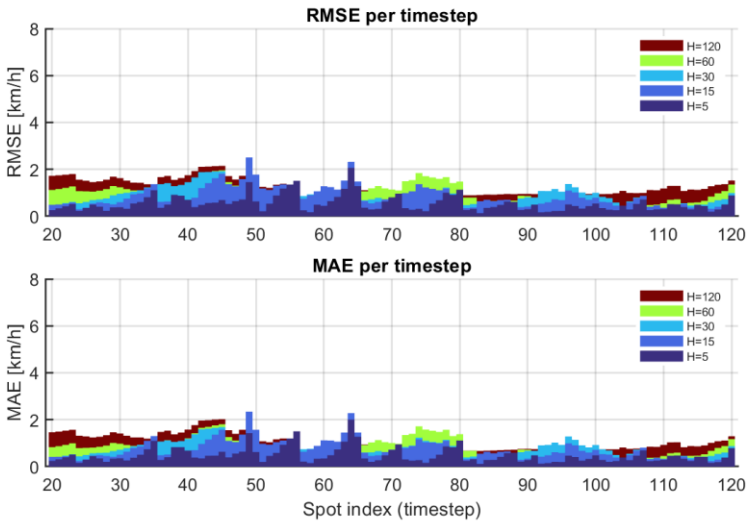

Figure 67 Distributed RMSE and MAE in [km/h] for the sequence in Figure 66.

The Figure 67 chart confirms what was observed previously, the RMSE and MAE are more compressed and for large sections the columns relating to medium-short time horizons cover those of the metrics with a longer horizon.

The above-mentioned characteristics prevent us from having a complete vision of the accuracy related to that sequence and the doubt could arise that for short horizons in general the error metrics are higher than for longer horizons.

It is observed that forecasts starting in the range between timesteps 50-80 seem to have a lower accuracy on the short-term horizons, which is also noticeable when looking at detail in Figure 66.

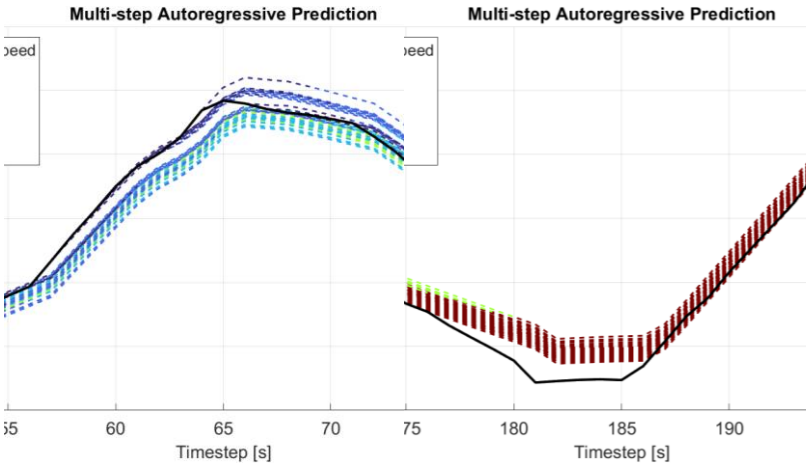

Figure 68 Detail from the sequence in Figure 66

In Figure 68 predictions for two time intervals are shown; the first ranges from 50 to 80 timesteps, and the blue hues indicate that they are short-term forecasts. The second range

covers timesteps between 170 and 200, thus showing the long-term extension of the predictions generated in the first range.

Since the red lines generally seem to fit the real velocity profile better, it is understandable why in Figure 67 the error metrics on the short-term horizons overwhelm the long-term ones.

As in the previous cases, it is possible to graphically observe the aggregated metrics in the Figure 69.

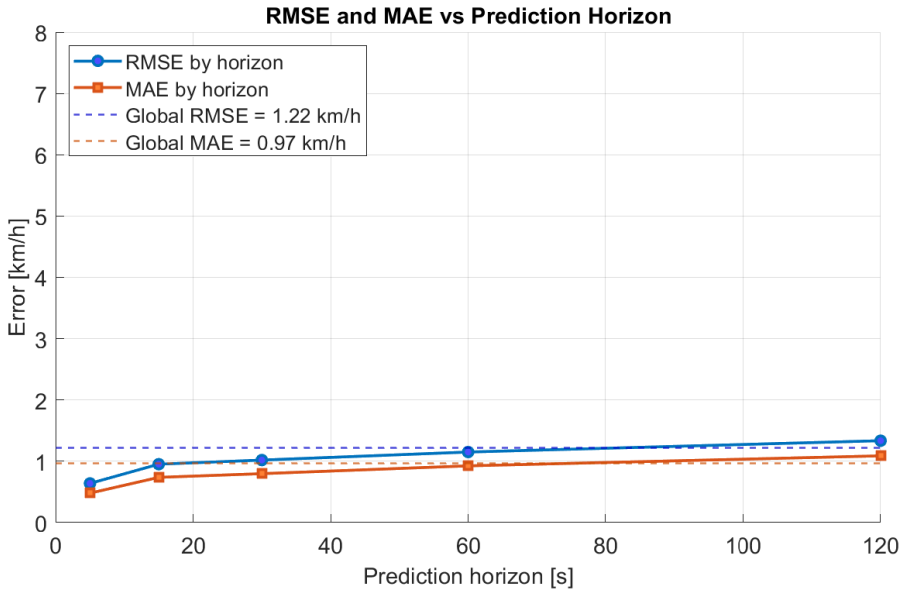

Figure 69 Horizon and Global metrics [km/h] for the sequence in Figure 66.

In reality, Figure 69 shows that the error tends to increase on average, albeit slowly. In this case the global metrics are the lowest recorded, being respectively equal to 1.22 $[km/h]$ and 0.97 $[km/h]$ for RMSE and MAE, reaching the value of 21.7% of the RMSE related to the worst scenario evaluated and 21% for what concerns MAE.

Also, in this case global MAE is equal to 79% of the global RMSE, remaining consistent with the trend seen in the two previous sequences.

It could also be interesting to evaluate the percentage variation of the mean values relating to the horizons, the results of which are shown in Figure 70 and Figure 71.

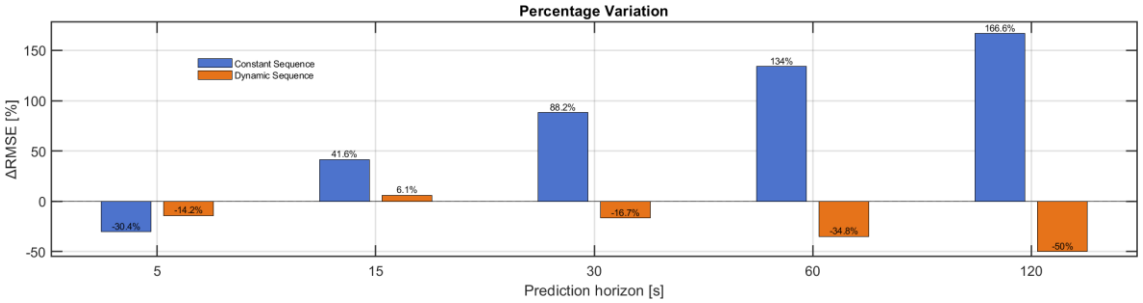

Figure 70 Percentile variation of the RMSE for prediction horizons.

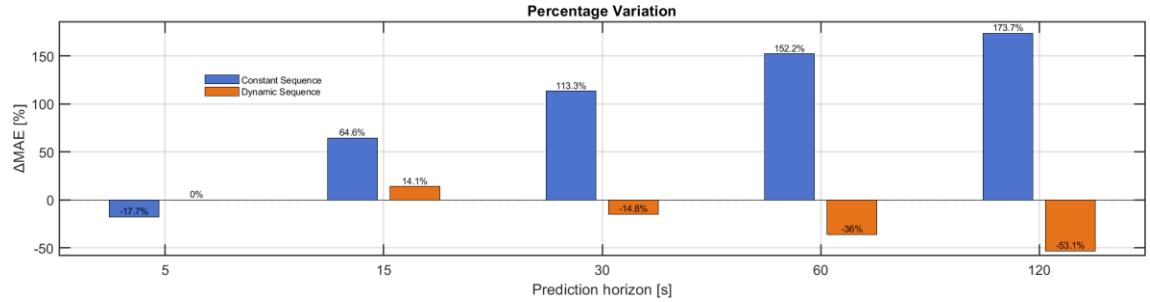

Figure 71 Percentile variation of the MAE for the prediction horizons.

The figures above show the variation in the metrics per horizon of the second test sequence (blue columns) and the third (orange columns) in terms of both RMSE and MAE compared to the first sequence. Interestingly, for the 5-second prediction horizon, RMSE and MAE are more accurate in the second sequence, while the MAE is identical for the medium- and high-dynamic-time sequences. Unexpectedly, the RMSE and MAE worsen for the 15-second horizon in the third sequence. For the other time horizons, which are of greater practical interest, expectations are met, as the third sequence improves, achieving a 50% improvement in RMSE and 53% in MAE for the 120-second horizon. Conversely, the constant sequence increases its inaccuracy, reaching peaks of 166% (RMSE) and 173.7% (MAE).

The result is somewhat encouraging considering that in urban environments vehicle speed profiles tend to be very dynamic compared to other situations.

7 Conclusions

In this thesis, a SUMO scenario simulating traffic in the city of Turin was used to collect data for training, validation, and testing a specialized LSTM neural network capable of predicting vehicle speed profiles in an urban context. Following postprocessing, a sensitivity analysis was performed to identify the hyperparameters to use for training the network. The model was then tested to evaluate its accuracy, and several experiments were conducted. Specifically, the possibility of making long-term predictions by feedback loops using the same predicted speed values was explored. Furthermore, an approach based on multiple corrective forecasts made instant by instant has been implemented, with a view to integrating this system with algorithms designed to manage the energy strategy of a hybrid vehicle.

It has been demonstrated how artificial intelligence, and in particular Long Short-Term Memory (LSTM) neural networks, can provide a reliable prediction of future driving conditions and thus improve the performance of the MES.

The LSTM models developed have shown the ability to forecast vehicle speed profiles with satisfactory accuracy, achieving an average RMSE of 0.283 [m/s] and an MAE of 0.240 [m/s] on 80-second sequences. These results indicate that the network successfully captured the main temporal dynamics of urban driving. Moreover, the extension of the forecast horizon and the adoption of multi-step and autoregressive strategies confirmed that the model can reproduce both short- and medium-term trends, though with increasing uncertainty as the horizon grows.

From a broader perspective, the work validates the potential of combining traffic simulation tools and deep learning algorithms to develop predictive systems that can later be integrated into energy management strategies for hybrid or electric vehicles. Accurate speed forecasting enables more efficient power-split decisions, ultimately reducing energy waste and emissions.

Nonetheless, some limitations remain. The mesoscopic simulation approach simplifies certain microscopic dynamics such as lane-changing behavior and driver variability, which may limit the realism of specific sequences. Furthermore, the network was trained on synthetic data only, which—while consistent and scalable—might not fully reflect the variability of real-world driving.

Future developments could therefore focus on:

- Expanding the dataset to microscopic simulations or real-world floating car data;
- Introducing additional features such as acceleration prediction, traffic light phase data, and inter-vehicle interactions;

- Testing hybrid architectures, integrating LSTM layers with attention mechanisms or convolutional preprocessing blocks;
- Embedding the forecasting module within model predictive control frameworks for real-time energy management.

Future developments for this activity could be:

- Exploiting of a Microscopic traffic model.
- Implementation of the predictive model using Transformer.
- Validation of the methodology through SUMO scenarios of other urban centers

Bibliography

[1] L. Lach e D. Svyetlichnyy, «Comprehensive Review of Traffic: Towards Autonomous Vehicles.,» *Applied Science,* vol. 8456, n. 14, 2024.

[2] F. Qiao, T. Liu, S. Haochen, G. Lingzhong e C. Yifan, «Modelling and simulation of urban traffic system: present and future,» *Int. J. Cybernetics and Cyber-Physical Systems,* vol. 1, n. 1, pp. 1-32, 2021.

[3] M. &. V. P. Fellendorf, Fundamentals of Traffic Simulation, Springer, 2021.

[4] D. e. a. Krajzewicz, «Recent Development and Applications of SUMO – Simulation of Urban MObility.,» *International Journal On Advances in Systems and Measurements,* n. 5, pp. 128-138, 2012.

[5] B. R. Munson, T. H. Okiishi, W. W. Huebsch e A. P. Rothmayer, Fundamentals of Fluid Dynamics, 7th ed. Wiley, 2013.

[6] M. T. a. A. Kesting, Traffic Flow Dynamics: Data, Models and Simulation., Berlin: Springer-Verlag, 2013.

[7] S. P. Hoogendoorn e P. H. L. Bovy, «State of the art of vehicular traffic flow modelling.,» *Transportation Research Part B: Methodological.,* vol. 34, n. 5, pp. 283-303, 2001.

[8] I. Prigogine e R. Herman, Kinetic Theory of vehicular Traffic, Elsevier, 1971.

[9] C. F. Daganzo, «The cell transmission model, part II: Network traffic.,» *Transportation Research Part B,* vol. 28, n. 4, pp. 269-287, 1995.

[10] G. A. C. (DLR), «SUMO at a Glance – Simulation of Urban Mobility,» March 2025. [Online]. Available: https://sumo.dlr.de/docs/SUMO_at_a_Glance.html. [Consultato il giorno 21 October 2025].

[11] M. Rapelli, C. Casetti e G. Gagliardi, «Vehicular Traffic Simulation in Turin From Raw Data,» *IEEE Transactions on mobile Computing,* vol. 21, n. 12, pp. 4656-4666, 2022.

[12] C. Stryker e E. Kavlakoglu, «What is artificial intelligence (AI)?,» IBM Think, 2024. [Online]. Available: https://www.ibm.com/think/topics/artificial-intelligence?. [Consultato il giorno 15 October 2025].

[13] B. J. Copeland, «Artificial intelligence,» Encyclopaedia Britannica, [Online]. Available: https://www.britannica.com/technology/artificial-intelligence. [Consultato il giorno 19 October 2025].

[14] N. J. Nilsson, The Quest for Artificial Intelligence: A History of Ideas and Achievements, Cambridge, UK: Cambridge University Press, 2010.

[15] P. A. Micheli, *Appunti dalle lezioni,* Pisa: Università di Pisa, A.A. 2017/2018.

[16] L. Tresca, F. Millo, L. Rolando e L. Pulvirenti, *Development of a Neural Network based Energy Management Strategy (EMS) for a plug-in Hybrid Electric Vehicle,* Torino: Politecnico di Torino, 2021.

[17] MathWorks, «Supervised Learning,» [Online]. Available: https://www.mathworks.com/discovery/supervised-learning.html?utm_source=chatgpt.com. [Consultato il giorno 20 October 2025].

[18] MathWorks, «Unsupervised Learning,» [Online]. Available: https://it.mathworks.com/discovery/unsupervised-learning.html. [Consultato il giorno 21 October 2025].

[19] MathWorks, «Reinforcement Learning,» [Online]. Available: https://it.mathworks.com/discovery/reinforcement-learning.html. [Consultato il giorno 22 October 2025].

[20] IBMThink, «Classification Machine Learning,» [Online]. Available: https://www.ibm.com/think/topics/classification-machine-learning. [Consultato il giorno 21 October 2025].

[21] Winland, E.; Kavlakoglu and V. «What is k-means clustering?,» IBM Think, [Online]. Available: https://www.ibm.com/think/topics/k-means-clustering. [Consultato il giorno 22 October 2025].

[22] Y. He; L. Li; X. Zhu and K. L. Tsui «Deep learning,» *Nature,* vol. 521, pp. 436-444, 2015.

[23] IBM, «What is a Neural Network?,» IBM Think, [Online]. Available: https://www.ibm.com/think/topics/neural-networks?. [Consultato il giorno 25 October 2025].

[24] G. Luo; K. Chou and M. T. Colonnese, «Recent Advances at the Interface of Neuroscience and Artificial Intelligence,» *Frontiers in Computational Neuroscience,*

vol. 16, n. 945275, 2022.

[25] Dubey, S. R.; Singh, S. K.; Chaudhuri, B. B., «Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark,» *Neurocomputing,* vol. 503, pp. 92-108, 2022.

[26] C. Xu; P. Coen-Pirani and X. Jiang, «Empirical Study of Overfitting in Deep Learning for Predicting Breast Cancer Metastasis,» *Cancers,* 2023.

[27] C. Isaac; K. Zareinia, «Effect of Excessive Neural Network Layers on Overfitting,» *WJARR,* vol. 16, n. 2, 2022.

[28] S. Mohammed; L. Budach; M. Feuerpfeil; N. Ihde; A. Nathansen; N. Noack; H.Patzflaff; F.Naumann and H. Harmouch «The effects of data quality on machine learning performance,» Cornell University, [Online]. Available: https://arxiv.org/abs/2207.14529. [Consultato il giorno 19 October 2025].

[29] X. Ying, «An overview of overfitting and its solutions,» *Journal of Physics: Conference Series,* vol. 1168, 2019.

[30] K. Maharana et al., «Data pre-processing and data augmentation techniques,» *Computers & Electrical Engineering,* vol. 99, n. 107917, 2022.

[31] I. T. Jolliffe, «Principal component analysis: a review and recent developments,» *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences,* vol. 374, n. 2065, 2016.

[32] Z.hou; B. Gu; Q. Yang and H. Huang, «Decoupled Parallel Backpropagation with Convergence Guarantee,» *arXiv preprint,* 2018.

[33] M. G. M. Abdolrasol, «Artificial Neural Networks Based Optimization Techniques,» *Electronics,* vol. 10, n. 21, 2021.

[34] Dubey, S. R.; Singh, S. K.; Chaudhuri, B. B., «Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark,» *Neurocomputing,* vol. 503, pp. 92-108, 2022.

[35] Bengio, X; Glorot, Y., «Understanding the Difficulty of Training Deep Feedforward Neural Networks,» *Proc. 13th Int. Conf. Artificial Intelligence and Statistics,* pp. 249-256, 2010.

[36] Y. He; L. Li; X. Zhu and K. L. Tsui, «Multi-Graph Convolutional-Recurrent Neural

Network (MGC-RNN) for Short-Term Forecasting of Transit Passenger Flow,» *IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS,* 2021.

[37] Y. He et al, «In-Depth Insights into the Application of Recurrent Neural Networks for Traffic Prediction,» *Algorithms,* vol. 17, n. 9, 2024.

[38] I. Malashin; V. Tynchenko; A. Gantimurov; V. Nolyub and A. Borodulin, «Applications of Long Short-Term Memory (LSTM) Networks in Polymeric Sciences: A Review,» *Polymers,* vol. 16, n. 18, 2024.

[39] G. Van Houdt; C. Mosquera and G. Napoles, «A Review on the Long Short-Term Memory Model,» *Neural Computing & Applications,* 2020.

[40] Mathworks, «What is a Long Short-Term Memory?,» [Online]. Available: https://it.mathworks.com/discovery/lstm.html. [Consultato il giorno 1 November 2025].

[41] H. Sak; A. Senior; F. Baeufays, «Long Short-Term Memory Recurrent Neural Network Architectures for Large-Scale Acoustic Modeling,» *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing (ICASSP),* pp. 338-342, 2014.

[42] S. Siami-Namini; N. Tavaroli and A. Siami-Namini, «A Comparison of ARIMA and LSTM in Forecasting Time Series,» *IEEE 17th International Conference on Machine Learning and Applications (ICMLA),* 2018.

[43] N. Chung and C.T., «A Review of Deep Learning Algorithms for Time Series Forecasting,» *IEEE Access,* vol. 9, p. 91896–91912, 2021.

[44] «The history of artificial intelligence,» IBM Think, [Online]. Available: https://www.ibm.com/think/topics/history-of-artificial-intelligence. [Consultato il giorno 20 October 2025].

[45] D. Hassabis; D. Kumaran; C. Summerfield and M. Botvinick, «Neuroscience-Inspired Artificial Intelligence,» *Neuron,* vol. 95, n. 2, pp. 245-258, 2017.