ABSTRACT During the past ten years, worldwide efforts have been pursuing an ambitious policy of sustainable development, particularly in the energy sector. This ambition was revealed by noticeable progress in the deployment and development of infrastructures for the production of renewable electrical energy. These infrastructures combined with the deployment of wired and wireless communications could support research actions in the field of connected electro-mobility. Also, this progress was manifested by the development of electric vehicles (EV), penetrating our transportation roads more and more. They are considered among the potential solutions, which are envisaged to further reduce road transport’s greenhouse gas emissions, relying on low-carbon energy production. However, the uncertainty caused by both external road disturbances and drivers’ behavior could influence the prediction of upcoming power demands. These latter are mainly affected by the unpredictability of the electric vehicles’ speed on transportation roads. In this work, we introduce an energy management platform, which interfaces with in-vehicle components, using a developed embedded system, and external services, using IoT and big data technologies, for efficient battery power use. The platform was deployed in real-setting scenarios and tested for EV speed prediction. In fact, we have used driving data, which have been collected on Rabat-Salé-Kénitra (RSK) urban roads by our Twizy EV. A multivariate Long Short Term Memory (LSTM) algorithm was developed and deployed for speed forecasting. The effectiveness of LSTM was evaluated against well-known algorithms: Auto Regressive Integrated Moving Average (ARIMA), Convolutional Neural Network (CNN) and Convolutional LSTM (ConvLSTM). Experiments have been conducted using two approaches; the whole trajectory dataset and segmented trajectory datasets to train the models. The experimentation results show that LSTM outperforms the other used algorithms in terms of forecasting the speed, especially when using the trajectory segmentation approach.

INDEX TERMS Electro-mobility, intelligent transportation systems, speed forecasting, time-series forecasting, deep learning.

I. INTRODUCTION Nowadays, electric vehicles (EVs) are penetrating more and more transportation roads. There are three main categories of EVs: Hybrid Electric Vehicles (HEV), Plugin Hybrid Electric Vehicles (PHEV), and Battery-powered Electric Vehicles (BEV). BEVs are fully-electric vehicles with rechargeable batteries. HEV and PHEV are, however, powered by both gasoline and electricity. Unlike PHEV, which can be recharged through both regenerative braking and external...
electricity sources, HEV is charged from the electricity generated by the car’s braking system. However, both of them are still using gasoline, which is one of the sources of greenhouse gas.

BEVs have been getting more attention in the last years, as they are environment-friendly while having an acceptable driving range, particularly in urban areas. BEV driving range can vary between 90 to 500 Km [1], [2]. Although, the driving range depends mainly on the battery capacity, the bigger it is, the longer the driving range is. However, BEVs’ batteries are highly expensive as their cost per kWh is very high [2]. Thus, other solutions have been investigated to maximize the driving range without increasing the battery capacity, mainly, by minimizing the power consumption while maintaining the driver’s comfort [3].

Rerouting applications could provide to the driver better routes with low traffic leading them to minimal power consumption. The development of in-vehicle applications and services could help in predicting the remaining driving range according to the upcoming routes and driver’s behavior as well. The rapid evolution in the field of Internet of Things (IoT), embedded systems, wireless communications and big data technologies are the main assets which could lead to the development of this kind of applications [4]. Furthermore, the number of sensors, which are deployed in road infrastructures, is increasing year after year. For instance, sensor data could be used in Intelligent Transportation System (ITS) applications to improve traffic efficiency and road safety [5]. Moreover, control strategies can be deployed in BEVs in order to maximize their driving range by, for example, controlling the speed and other in-vehicle systems (e.g., HVAC) [6]. All these technologies could be combined in a holistic way to develop an intelligent in-vehicle battery energy management system (BEMS). This latter relies mainly on several factors, such as vehicle characteristics, driving behavior (e.g., speed), and road status (e.g., slope).

Recent studies showed that the major component that consumes more power from the battery is the electric motor, which is related to the vehicle’s speed profiles [7], [8]. Generally, knowing in advance the vehicle’s speed could help in developing many ITS applications, e.g., traffic management, driving range prediction, rerouting, and energy management [9], [10], [11], [12], [13]. However, electric vehicle speed prediction is still a challenging and tedious task. It requires further research investigations and it depends on a lot of parameters, which can be categorized into two families, endogenous and exogenous. Endogenous parameters are related to the electric vehicle characteristics or internal context (e.g., mass, auxiliary equipment), whereas exogenous parameters are related to the vehicle surrounding or external context (e.g., weather, traffic, and road conditions). It is worth noting that, endogenous parameters are, in general, easy to obtain, as they remain constant during a given trip trajectory. However, exogenous parameters are hard to obtain, as they are dynamic or semi-dynamic during a given road trip.

In this paper, we introduce an IoT/Big data platform, named HELECAR, which has been used to deploy deep learning multivariate algorithms for real-time speed forecasting of electric vehicles. The proposed platform was tested in real-setting scenarios on Rabat-Salé-Kénitra urban roads. In summary, the main contributions of this work are three folds:

I. Development of an IoT/Big data platform for data collection and processing.
II. Development and deployment of an embedded multi-variate LSTM algorithm for speed prediction.
III. Extensive experiments were conducted to compare the LSTM algorithm with other ones from the literature, e.g., Auto Regressive Integrated Moving Average (ARIMA) algorithm, Convolutional Neural Network (CNN), and ConvLSTM (Convolutional LSTM).

The remainder of this paper is structured as follows. The next section presents existing works from literature by focusing mainly on those related to speed prediction. The HELECAR platform is introduced in Section III. Materials and methods are presented in Section IV. Section V presents experiments we have performed together with obtained results. Conclusions and perspectives are given in Section VI.

II. RELATED WORKS

Vehicle speed prediction is one of the topics that has drawn many researchers’ concerns. Numerous approaches have been proposed to predict vehicle speed, which can be classified into three main categories: model-based approaches, data-driven approaches, and hybrid approaches [14]. Model-based approaches depend mainly on analytical models, which are very hard to design as they require intensive modifications and calibrations for being used in real-setting scenarios. On the other hand, data-driven approaches are easy to implement and have shorter development and deployment cycle [15]. Hybrid approaches can integrate both model-based and data-driven approaches [16].

Model-based approaches can be divided into two main families: parametric and non-parametric. Generally, parametric approaches assume the structure of the prediction model. Thus, massive and representative training data are required to achieve good accuracy. The Constant Speed (CS), Constant Acceleration (CA), SUMO (Simulation of Urban MOBility) model, and Intelligent Driver Model (IDM), are four typical models that apply the parametric approaches [17]. Unlike parametric approaches, non-parametric methods do not require prior fixing of the model structure, as it could be computed from historical data. For instance, Authors in [18] have conducted a comparison study between parametric and non-parametric approaches for vehicle speed prediction. In their work, they evaluated six models by implementing both parametric and non-parametric approaches, and they run 10-fold cross-validation for different prediction horizons to test the models’ accuracy. Reported results showed that parametric models perform well for short-term prediction,
while advanced parametric models are more accurate for long-term prediction.

Recently, more attention has been given to machine learning techniques for vehicle speed forecasting. It is worth noting that, these techniques can be classified under data-driven approaches, as they rely mainly on the historical data of model training. For instance, the authors in [19] proposed a deep learning approach to predict the future short-term speed values using the collected dataset. This latter includes historical vehicle speed values, acceleration, steering information, locations, and driving dates. In order to test the generalization ability of their approach, the authors used four standard driving cycles and analyzed the influence of driving factors on the accuracy of vehicle speed prediction. Similarly, in [20], the authors proposed a deep learning algorithm for vehicle speed prediction featuring big data analytics and Adaptive Neuro-Fuzzy Inference System (ANFIS). The proposed algorithm can predict the vehicle’s speed on both freeways and urban traffic roads. The big data analytic program is used to extract the data of a specific trip, and the ANFIS system is utilized to extract the features of the speed dynamics through supervised learning. Other studies investigated Recurrent Neural Networks (RNN) by implementing the Long Short Term Memory algorithm (LSTM). For instance, the authors in [21] proposed an ego-vehicle speed prediction model based on LSTM. This model is used as a microscopic and non-parametric approach to apply various urban driving conditions. The proposed model can predict up to 15 seconds of vehicle speed values. They mainly compared three driving scenarios to determine the impact of the inputs’ combinations, which are: the internal information of the ego-vehicle, the relative speed, the distance to the vehicle ahead, and the GPS coordinates. Also, the authors in [22] discussed several machine learning algorithms for speed prediction, and they proposed an algorithm, which is based on Back Propagation Long Short Term Memory (BP-LSTM) for on-road individual long-term average speed prediction. Furthermore, the authors in [23] proposed a novel speed predictor that is based on a K-means algorithm and radius basis function neural network (RBF-NN). The proposed method contains various predictive sub-models to cope with different input drive states. First, the principal component analysis (PCA) is applied to the data to extract the key factors. The dataset is then preprocessed into sub-datasets using the K-means clustering technique. Finally, the RBF-NN is trained upon the sub-datasets in order to predict future speed values.

Certainly, the use of machine learning has become very popular for speed prediction due to its ability to link multiple parameters. These parameters represent the inputs to be used by the speed prediction models. In this context, the authors in [24] proposed a deep learning-based solution for speed prediction, which exploits the vehicle speed history and takes advantage of the availability of data, which are related to the geometric nature of the roads, such as curvature, altitude, and the number of lanes. These data are further enriched by the road traffic provided by the Traffic Message Channel (TMC). The proposed solution achieves accurate forecasts despite occasional overshoots caused by sudden speed changes. Based on the importance of road traffic data and how it impacts individual vehicle speed, the authors in [25] proposed a two-level data-driven prediction model. The first one consists of a neural network for the prediction of the average speed in the road network segments, while the second one exploits a Hidden Markov Model (HMM) to establish a statistical relationship between the average speed in a segment and the individual vehicle speed.

The relevance of the parameters applied for speed prediction is of great importance to build accurate prediction models. Starting from the fact that speed is an emergent feature of the vehicle operating context, the authors in [26] investigated its most influential parameters. They proved that their model is able to make accurate predictions and remains robust to the non-detection of a stop sign or the misestimation of road traffic conditions. However, the prediction errors become intolerable when the road profile is misidentified or when there is a compounded error of key parameters. To alleviate these issues and particularly the abrupt changes in vehicle speed, the authors in [27] investigated the integration of vehicle lateral dynamics for in-curve vehicle speed prediction, in a way to maximize the speed, and reduce the energy demand while ensuring the driver’s safety.

Beyond dealing with the selection of the key parameters impacting prediction quality, other research works have focused on developing better approaches to predict vehicles’ speed by restructuring the input data. From a perspective of extending the prediction horizon, the authors in [28] investigated two approaches by combining the Markov chain and Monte Carlo methods. The first approach considers the speed as a single-scale time series and makes the prediction in multi-steps to extend the forecasting horizon, while the second is based on a multi-scale time series and makes the forecast in a single step. The reported results show that the second approach outperforms the first one by an average improvement of 7% due to its ability in avoiding accumulative prediction errors. Authors in [29] proposed a method by using a Convolutional Neural Network (CNN) for large-scale speed prediction. It is mainly based on the representation by Spatio-temporal matrices of the speed in different segments of the road network. These matrices are used to generate images that will feed the CNN model for speed prediction. The reported results show that this method outperforms the LSTM and random forest methods. A convolution technique, hybridized with an LSTM model, was also applied by Authors in [30] for large-scale speed prediction.

Moreover, ITS applications can also play a role in the process of speed forecasting. Many researchers have used vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication information in order to forecast the EVs’ speed. For instance, the authors in [31] proposed a
data-driven multi-range speed prediction strategy, which provides a preview of three different ranges; short, medium, and long-range. The authors used the V2V and V2I communications to obtain the short-range, a neural network (NN) for medium-range, and a Bayesian Network (BN) for long-range speed prediction. Simulation results show that the medium-range speed prediction using NN yields better results than the short-range and long-range. Furthermore, the authors in [32] designed a predictor that outputs short to medium-range speed trajectory forecasting using a locally weighted polynomial regression algorithm (LWPR). The proposed method takes as input the real-time V2V and V2I information and outputs the short-range and medium-range vehicle’s speed forecasting. The authors stated that by integrating the V2I information into their approach, they can also predict the speed trajectory of the preceding vehicle with better accuracy at locally nonlinear points.

Despite the extensive work from literature related to speed prediction, BEMS efficiency for EVs is still of keen interest, for extending vehicle driving range and improving drivers’ experience. However, it is a tedious task as it depends on various parameters that are hardly predictable and that emerge from the vehicles’ context, notably, the speed, which has a great impact on battery SoC and energy consumption. To the best of our knowledge, speed prediction works, as previously cited, tackle this problem exclusively from an accuracy perspective, and seeks to improve it based on the data availability using GPS, ITS information, traffic lights, vehicles communication, a fleet of vehicles, and the deployed sensors in roads’ infrastructures. Certainly, accurate speed prediction for an extended horizon is the basis to anticipate energy needs and to better recover kinetic energy for sustaining battery charge. Nevertheless, these research works omit the challenges of embedding these solutions at the vehicle level. More precisely, speed prediction is a process that needs to be integrated in order to establish a near real-time predictive energy management. However, this process is constrained by the limit of the onboard computing power and the recurrent need for retraining the prediction models to fit the newly collected data. Furthermore, speed prediction is made over a short horizon, given its high variability, which needs to be improved according to the available road and in-vehicle data (e.g., traffic conditions, the speed of leading vehicles).

In a previous work [33], we studied the use of multivariate LSTM to forecast vehicle speed and compared it against univariate LSTM. Mainly, we found that the multivariate algorithm outperforms the univariate one. It is mainly due to the fact that the speed relies on a lot of factors and, therefore, multivariate models are suitable for this case. The study was performed using a dataset collected from the Vehicle Simulation Runtime Infrastructure (VSimRTI) simulator [34], which is, a simulator dedicated to developing ITS applications. To further investigate the development and deploy these algorithms in real-setting scenarios, we have investigated, in this work, recent advances in artificial intelligence (AI), IoT, and Big Data technologies. In fact, an IoT/Big data platform is first introduced to enable the development and deployment of data-driven approaches for vehicle speed prediction. A multivariate LSTM algorithm is then developed and deployed together with other algorithms from the literature, mainly ARIMA, CNN, and ConvLSTM. Extensive experiments were conducted in real-setting scenarios to compare these algorithms in terms of accuracy and performance.

III. THE HELECAR PLATFORM

In order to enable, as stated previously, the development of speed prediction approaches, a platform, named HELECAR, was first developed by combining recent IoT and Big data technologies. The platform is built upon a scalable architecture taking into consideration future expansion, including charging infrastructure integration. Consequently, it could be extended to consider the monitoring of BEV charging demands and charging station occupancy (e.g., charging time, waiting time, and battery charging rate) [35].

The proposed platform enables potential predictive analytics of received data, either from the in-vehicle embedded system (e.g., acceleration, battery features) or from the surrounding environment (e.g., traffic, road profiles), for better battery energy management. For instance, predicting the remaining driving range according to the upcoming routes will allow assisting BEVs’ drivers to decide on rerouting suggestions, driving mode (economical, comfort), and charging operations planning (when, where, charging rate). This enables less restrictive exploitation of BEV through minimizing battery consumption and charging time and increasing the vehicle’s range taking into consideration the passengers’ comfort (e.g., air-conditioning). Fig.1 depicts an overview of the HELECAR platform deployment whereas Fig.2 presents an overview of its general architecture. HELECAR subsystems are described in the rest of this section.

![FIGURE 1. Mains steps for HELECAR platform deployment.](image)

A. DATA MONITORING SUBSYSTEM

Data availability is the ultimate requirement for performing predictive analytics, which is required to develop BEMS for electric vehicles. The data monitoring subsystem deals with data gathering through...
modular embedded devices, which support data streams from different sensors. Both server-side and client-embedded monitoring devices are equipped with communication capabilities, which could be performed when required. This strategy reduces network traffic. However, it requires embedded intelligence to guarantee autonomy in case of network breakdown. The deployed devices cover both BEV monitoring and the recharging infrastructure exploitation as well.

BEV monitoring sub-system consists of an onboard device, which fed back, in near real-time, data streams of BEV endogenous parameters. The values of these parameters, such as BEV load, SoC, and velocity, are required for developing energy management techniques. The onboard device is also equipped with GPS and GPRS/4G shields. The GPS module is used to maintain BEV tracking/tracing including detailed information about trips. This enables enriching our data models with information, which is related to roads’ parameters, in particular, type (urban/freeway), slope, and speed limitations. The GPRS/4G ensures the transfer of data streams to the server side of the platform. In case of a lack of network coverage, the monitoring device switches to local archiving mode and executes deferred transfer accordingly. Similarly, charging station monitoring relies on an embedded device, which reports information about the charging station’s exploitation. Gathering fine-grained data about the charging time, station occupancy, demands distribution, and exploitation patterns are required to enable efficient charging request scheduling.

**FIGURE 2.** Holistic architecture of the HELECAR platform.

**B. DATA PROCESSING SUBSYSTEM**

The data processing subsystem is composed of four main components axed on IoT nodes interconnections, streams and batch data processing, and AI-based techniques for forecasting BEV endogenous and exogenous parameters. The IoT component provides requirements, software, and hardware to support the interconnection of both BEV and charging station onboard devices. Apache Kafka and Node-red are combined in order to support batch and streaming data flows. This component helps gather the required data from electric vehicles, charging stations, and also from other services, such as weather and traffic data. This component comes on top of the deployed monitoring subsystem. As this latter collects required data, the IoT component ensures that collected data are first preprocessed and transferred to related components for further processing. More precisely, data reported by IoT nodes are first processed as streams using Apache Kafka streams. The data streams are then analyzed to detect data drift and to forecast important features, such as SoC, remaining driving range, and charging stations’ occupancy. These features are keys for performing BEV predictive control to achieve efficient energy management. Data streams are also archived in the Mongo database for further batch processing.

It’s worth noting that batch processing comes in handy when, for example, the accuracy of the used in-vehicle forecasting model decreases. In this case, a new model is generated accordingly (i.e., trained based on the stored data). The decision of requesting a new model is not obvious,
in fact, the accuracy of the forecasting model depends on the data history as well as on the newly collected data. Hence, the accuracy decreases dramatically when the gap between historical and newly received data is important. Relying only on real-time processed error metrics to decide the perfect time to request a new model is not ideal, as the collected data is dynamic and depends on a lot of factors. One way to overcome this problem is to use a strategy that monitors the change in the error metrics. These latter are used to request a new model, in particular, when errors are increasing and not getting better for a certain period.

C. DECISION SUPPORT SUBSYSTEM

This subsystem uses the output from the data processing subsystem to perform the required decisions. These latter are generated to achieve efficient BEV energy management. Decisions could be rerouting suggestions (i.e., best driving routes) to the driver, charging schedule, or even control actions (e.g., heating, ventilation, and air-conditioning control) [6]. Compliance with these suggestions enables a less restrictive BEV exploitation and extends its remaining driving range. In a fully automated BEV, predictive control could be deployed for controlling the speed accordingly.

IV. MATERIALS AND METHODS

In this section, we focus on materials and methods we have followed to enable BEV’s speed forecasting. The aim is to deploy a low-cost embedded software/hardware (Sw/Hw) subsystem for speed forecasting. This subsystem is a part of the HELECAR platform, it was developed and deployed for collecting needed data and for real-time speed forecasting as well. Speed forecasting was performed using a neural network algorithm following two steps; training a model upon historical data for being used to predict future speed values. However, model training can be a tedious task, which needs a powerful computer and a huge historical dataset. In fact, due to the computational limitations of the low-cost existing embedded devices, the deployed subsystem focuses only on data gathering/communication as well as the execution of the established pre-trained model for speed forecasting. Consequently, the model’s training is performed in a distant server, which hosts other subsystems of the HELECAR platform. Fig.3 depicts the operational architecture of the deployed platform’s prototype. It is worth mentioning that, for this study, Message Queuing Telemetry Transport protocol (MQTT) is used to transmit the data instead of Apache Kafka or Apache Flume. MQTT was proposed for developing IoT applications and services by including the publish/subscribe principle, i.e., allowing bi-directional communication between IoT devices. The choice of using this protocol over Apache Kafka is based on the fact that, for this study, the amount of transmitted data, from the in-vehicle subsystem to the distant server, is not massive. More precisely, the MQTT protocol is widely used in IoT applications for data transmission and inter-communication between devices, however, when working with a large-scale and massive amount of data, the use of Apache Kafka comes in handy.

A. DATA COLLECTION

In order to forecast the speed, a pre-trained model is needed. However, historical data of the vehicle’s speed, the weather, the roads’ traffic/conditions, and other information are required to train the model. Unfortunately, at the time of this study, there is no such historical data in the Moroccan context. Therefore, the first step of our study was about collecting data. The used vehicle in the conducted experiments was the Renault Twizy, which is a fully electric vehicle with a maximum speed of 80 km/h. The Twizy uses a Lithium-Ion battery with a capacity of 6.1 kWh, with a maximum theoretical range of 100 Km. The EV’s battery takes about 3.5 hours to be fully charged. However, the Twizy comes with no accessories, such as the FM radio or an air conditioner, which makes its electric motor the only major energy consumer. To capture different traffic and road conditions, three main trajectories have been selected in Sala Al Jadida and Rabat region, Morocco, as shown in Fig.4. Each trajectory is between 30 min and 2 h long, and each trajectory is repeated many times in different periods, mainly, morning, noon and afternoon. The aim is to mimic a typical real-life scenario of a person going from home to work and from home to another place in the city. Also, all the trajectories have the same starting and end point (S), which is our university, and sometimes, the electric vehicle is recharged in point (D). An example of the speed profile of each trajectory is depicted in Fig.5. The second and third trajectories, as depicted in Fig.5b and Fig.5c are done on an urban road, which explains the stops and many slowdowns in the dataset due to the traffic lights, speed bumps, and heavy traffic. However, the first speed profiles shown in Fig.5a, are related to a trajectory that is done mostly on a ring road, i.e., with high speed and few stops compared to the other two trajectories. It is worth noting that the speed profiles in Fig.5 represent all the data, which were collected during the conducted experiments.
As mentioned above, each trajectory is repeated many times, thus, in Fig.5, the gray lines represent the data collected and the red line represents the average speed profile for each trajectory. Also, each trajectory is divided into several segments, the first and last segments are almost identical for the three trajectories, as in this segment, the vehicle is near the starting/end point. However, we noticed that the other segments differ depending on the chosen trajectory. In the trajectory, as depicted in Fig.5a, the second segment is done on a ring road in which the speed is limited to 80 Km/h, thus the high-speed profile. Also, in some cases, the speed drops to 20 Km/h in this second segment due to the permanent police traffic control points.

In trajectory B (Fig.5b), segment 2 was realized between the city of Sala al Jadida and Rabat, Morocco (urban areas). The speed limit in this segment is 60 Km/h. However, it is shown that there are a lot of stops and slowdowns due to the dense traffic, traffic lights, and speed bumps. However, a part of trajectory B was realized on another ring road, which explains the speed spikes in segment 3, since on this ring road the speed limit is 80 Km/h and the traffic is mostly low. Nevertheless, segment 3 has many roundabouts and the road slope is high, which explains the speed spikes since the driver needs to slow down when entering a roundabout and the speed decreases depending on the slope. This is because the electric motor needs to generate more torque when going uphill than on a straight road or downhill. Furthermore, since the electric vehicle used in this experiment has a small battery (compared to the other existing BEVs), it cannot deliver enough current to the electric motor to generate the needed torque, therefore, the speed decreases. The last trajectory C (Fig.5c) was performed mostly in the city of Sala Al Jadida. Segment 2 is closely identical to the one from trajectory B, which is normal since it was realized on an urban road with heavy traffic and many speed bumps and traffic lights.

Fig.6 represents the in-vehicle embedded device, which was used to collect the data from the Twizy. An Arduino nano development board is used with a Controller Area Network Bus (CAN Bus) module to read the data from the electric vehicle through the On-board Diagnostics (OBD) connector. The CAN protocol, widely used in the automobile industry, allows embedded components to communicate with each other without the need for a host computer or server. The CAN bus is a message-based protocol, each message is transmitted in a frame and each frame has a unique identifier. When a component transmits a message, all other components receive it including the message emitter.

The data collected is raw data, which is encoded in Hex values. This data is transmitted to the NVIDIA Jetson nano, in which a developed program, using a Node-red framework, performs raw data decoding. After this decoding phase, useful information is filtered and then saved into a Comma-separated values (CSV) file and MongoDB database for later use. The embedded component contains also a user interface for real-time visualization of vehicle information, such as the actual speed, driving mode, position on the map, and also predicted speed. More precisely, the collected dataset is composed of the following information: current vehicle speed, driving mode, battery SOC, GPS coordinates, weather conditions, speed limit, and road traffic. It is worth noting that the vehicle speed, SOC, and GPS are collected directly from the vehicle using the CAN Bus protocol, or from the GPS module, which is connected to the in-vehicle device. As for the weather data, the Open Weather Map Application Programming Interface (API) is used. Each minute, the program, deployed in the embedded component, sends a Hypertext Transfer Protocol (HTTP) request to the API and retrieves weather information, which is related to the current location of the vehicle. Likewise, TomTom API
is used to collect the traffic and speed limit data in each road segment. However, as the TomTom API needs very precise GPS coordinates, the speed limit is, therefore, not available in some road segments due to the low precision of the used GPS module. To overcome this problem, the speed limits on those segments are introduced manually by the driver using the given user interface, which we have mainly developed for that purpose.

The in-vehicle embedded device collects data and stores it in real-time, however, since the collected data does not change during a certain amount of time, two main collection steps have been used: five seconds and ten seconds. The choice of these two collection steps is done upon some observations of the collected data as well as some factors, which are related to the used hardware, such as refresh time and communication delays. A collection’s step, which is less than five seconds, resulted in having frequent redundant and constant values of some factors, such as GPS coordinates. However, a collection step greater than ten seconds resulted in a lack of some useful information, such as speed values, specifically during acceleration and deceleration. This led to a big difference between two successive speed values, which, however, affect the prediction accuracy. Therefore, collect step between five and ten seconds is likely to be fit for our case study. The application deployed into the NVIDIA Jetson nano collects the data and saves two types of datasets, one every five seconds and another every ten seconds. Aside from that, another dataset is saved using a collect step of one second, this dataset is used mainly in the data processing phase.

B. DATA PREPROCESSING

During the data collection process, some errors may occur, mainly missing values, unwanted values, or other errors. Therefore, the collected data need to be first preprocessed to be used for models’ training. Data collection is performed, as mentioned above, in the NVIDIA Jetson nano, using a CAN bus transceiver with a time-step equal to one (TM1), five (TM5), and ten (TM10) seconds. The transceiver receives many messages in one second. In our study, we were only interested in a few messages, however, filtering those messages may take more than one second. Furthermore, some in-vehicle sensors emit messages with an interval greater than one second. Moreover, the CAN messages are in hexadecimal format. Some values, such as vehicle driving mode, are coded in one bit, and other values are coded in two or more bits. However, we do not have all the instructions to convert, from hexadecimal to decimal, all the values in each message. Thus, strange values are sometimes received; those values are ignored and replaced with previous ones. This leads to missing values in the dataset. To overcome this problem, two methods are used. Regarding TM5 and TM10, for any given second, if a missing or strange value is received, the value of the last/next second is used as a substitute. However, in the case of TM1, we assume that during one second the value of the sensor will not change very often, and, therefore, the missing value is replaced with the value, which was collected in the previous second.
After this data preprocessing step, another process was launched for preparing the data. This process prepares data for being used in the training process. In this latter, some dataset transformations have been performed, mainly, encoding some values from string to integer, date and time format, grouping the data in batch sequences, and slope processing from the altitude. This process is mandatory in order to prepare the data for models’ training. Furthermore, the dataset is divided into multiple sets depending on the segmentation of each trajectory.

C. MODEL TRAINING AND TESTING

After cleaning and preparing the collected data, we split it into a training dataset and a testing dataset. In literature, there are many prediction methods, which could be applied, using common computing hardware, for speed forecasting. However, the model training process is mainly performed on the in-vehicle embedded system. For our real testbed, selecting a prediction method is constrained by its suitability for being deployed in an in-vehicle embedded device, such as a low computing power embedded board. Hence, the selected method should manage the trade-off between forecast accuracy and computing time, which is needed for predicting new values and for retraining the forecasting model as well. For this purpose, we have selected four methods respectively, each from a forecasting methods category [4]: LSTM, ARIMA, CNN, and ConvLSTM.

LSTM represents an improved version of neural networks. It can deal with multivariate time-series (TS) and stands out by its ability to handle dependencies among data over long and short periods. This method dynamically decides the dependency of the closest or distant past on the future TS values to be forecasted [36]. ARIMA represents a classic statistical method in the field of prediction, which is capable of making accurate forecasts. This method is chosen for its simplicity and low computing requirements. CNN, initially conceived for image recognition and classification, was recently adapted for TS prediction. This method goes beyond the temporal dependency of TS values and deals with 2D structures. ConvLSTM has been widely used in various applications, such as traffic accident prediction [37], short-term wind power forecasting [38], pedestrian attribute recognition in videos [39], and flood forecast [40]. The basics and principles of these four methods, LSTM, ARIMA, CNN and ConvLSTM, are briefly discussed in the following subsections.

1) LSTM MODEL

LSTM was introduced by Sepp Hochrieter and Jurgen Schmidhurber [41] in 1997. It was proposed to alleviate the shortcomings of conventional Recurrent Neural Networks (RNN) used for time series forecasting, mainly the problem of long-term data dependencies. RNN has proved its ability to process current data by laying on previous observations. However, it lacks identifying patterns upon long term dependencies and whether it is better to rely on the closest or distant past data to achieve better forecasting accuracy [36]. For this purpose, the LSTM model has been proposed to address this problem and control error flow.

From a structural perspective, LSTM relies on the use of a memory cell with three gates, namely, the forget gate, the multiplicative input gate, and the multiplicative output gate. These gates are used to learn how to control the error flow while maintaining it constant in order to avoid exploding and vanishing error problems. Fig. 7 depicts, in-depth, the LSTM gates where $C_{t-1}$ represents the previous cell state, $h_{t-1}$ represents the previous hidden state, $C_t$ and $h_t$ represent respectively the next cell state and next hidden state, whereas $x_t$ and $y_t$ represent the input and the output respectively.

**FIGURE 7. LSTM gates that compose an LSTM cell.**

- The forget gate $f_t$ controls the amount of information, which must be forgotten from state cell at the previous time step.
- The multiplicative input gate $i_t$ protects the memory cell from irrelevant data, which may disturb the error flow control. It decides, at the current moment, about the amount of the network input $x_t$ to be saved in the unit cell.
- The multiplicative output gate $o_t$ aims to protect other unit cells against irrelevant data stored in the cell.

These gates are formulated as follows [42]:

\[
\begin{align*}
    f_t &= \sigma(W_f[h_{t-1}, x_t] + b_f) \quad (1) \\
    i_t &= \sigma(W_i[h_{t-1}, x_t] + b_i) \quad (2) \\
    o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \quad (3)
\end{align*}
\]

where $(W_f, b_f), (W_i, b_i), (W_o, b_o)$ represent the couple weight matrix and the bias of, respectively, the forget gate, the input gate, and the output gate.

2) CNN MODEL

CNN is a class of Artificial Neural Networks (ANN). It is used mostly in image and speech recognition applications. Unlike LSTM, CNN relies on spatially-organized data, which $i$) enables recognizing new patterns in 2D structure and $ii$) supports sequential data (1D) processing [43]. Moreover, CNN shows interesting performances in terms of accuracy, which can challenge classical prediction methods, LSTM and ARIMA, and stands out for its performance in terms of computation time.
CNN is based on three main layers: convolutional layer, pooling layer and fully connected layer. Fig. 8 provides a simplified view of CNN functioning. The first layer of the CNN is the convolutional layer, in which, the filters are applied to the original data. Also, this layer takes as parameters the number and the size of the kernels as well as the activation function. The second layer of a CNN network is the pooling layer. It is similar to the convolutional layer, but the dimensionality of the network is reduced by applying either the max pooling function to the data, which takes the maximum value in a certain filter region, or the average pooling function, which takes the average value. The last layer is the fully connected layer, which aggregates the information from the previous layers and generates the final classification.

![CNN Functioning Diagram](image)

FIGURE 8. Overview of CNN functioning.

3) ARIMA MODEL
ARIMA is a well-founded statistical method for time series modeling and prediction. It is mainly suitable for linear time-series. It represents an enhanced version of ARMA model, which relies on a mash-up of the Auto-Regressive (AR) model and the Moving Average (MA) enriched by an integration process. This latter aims to set the right differencing order to make time-series stationary. The ARIMA model can be written as follows:

$$X_t = \sum_{i=0}^{p} \varphi_i X_{t-i} + \sum_{i=0}^{q} \theta_i \epsilon_{t-i}$$  \hspace{1cm} (4)

where $p$ is the lag order for the auto-regressive part of the model, $q$ is the lag order for the moving average model, $d$ is the differencing order, which is required to make the time-series stationary, $\varphi$ is the coefficient of the lagged variable, and $\theta$ is the coefficient of the lagged error term. Fitting the forecasting model to time-series consists of identifying the right parameters ARIMA($p,d,q$) and estimating ($\varphi$, $\theta$). For this purpose, Box and Jenkins [44] methodology proposed a solution based on the significance of Auto Correlation Function (ACF) and Partial Auto-Correlation Function (PACF).

It is worth noting that ARIMA has been proposed to tackle some specific situations, such as ARIMAX, SARMA, FARMAX, which are used respectively to process exogenous variables, manage the seasonality in time-series data and integrate fuzzy logic. ARIMA’s accuracy relies mainly on the selection of the model through the identification of the parameters $p$, $i$ and $q$, where $p$ is the lags count for the AR model, $i$ is an integration order, which is needed to make the data stationary, and $q$ is the lags count for the MA model [45].

However, as the three methods, LSTM, CNN, and ARIMA, are tested in this study, the training process is done either in the in-vehicle embedded device or in an external infrastructure (e.g., a distant server). Since the aim is to forecast future vehicle speed, error metrics are processed, when those errors exceed the tolerable threshold, the model will be retrained either in-field, for example, when the vehicle is disconnected (i.e., connection loss to the infrastructure), or in a distant server by requesting the remote training process. In both cases, the model is trained for a certain number of epochs and stops when it reaches an acceptable accuracy accordingly (i.e., computed evaluation metrics).

4) ConvLSTM MODEL
ConvLSTM was introduced by Shi et al. [46] in 2015 for precipitation nowcasting, as a modification and an extended version of the LSTM and Fully connected LSTM (FC-LSTM). In fact, instead of matrix multiplication in LSTM, ConvLSTM uses convolutional operators for the input state and state-to-state transition. This enables it to handle the Spatio-temporal features in data and therefore, determine the upcoming state of the cell using inputs rom local neighbors and previous states.

The mathematical equations of the convLSTM algorithm are given as follows [46]:

$$i_t = \sigma(W_{ci} \ast X_t + W_{hi} \ast H_{t-1} + W_{ci} \circ C_{t-1} + b_i)$$  \hspace{1cm} (5)

$$f_t = \sigma(W_{cf} \ast X_t + W_{hf} \ast H_{t-1} + W_{cf} \circ C_{t-1} + b_f)$$  \hspace{1cm} (6)

$$C_t = f_t \circ C_{t-1} + i_t \circ \tanh(W_{xc} \ast X_t + W_{hc} \ast H_{t-1} + b_c)$$  \hspace{1cm} (7)

$$o_t = \sigma(W_{co} \ast X_t + W_{ho} \ast H_{t-1} + W_{co} \circ C_t + b_o)$$  \hspace{1cm} (8)

$$H_t = o_t \circ \tanh(C_t)$$  \hspace{1cm} (9)

where $i$, $f$ and $o$ represent respectively the input, forget and output gates. $W$ represents the weighted connections between states and $\ast$ denotes the inputs. $\mathcal{H}$ and $\mathcal{C}$ are respectively the hidden state and the cell state, $\circ$ denotes the convolution operation, where $\circ$ denotes the dot operation. $\sigma$ and $\tanh$ are the activation functions.

V. PERFORMANCE EVALUATION
In this section, we first introduce the experimental set-up and metrics we have used to evaluate the performance of LSTM against ARIMA, CNN and ConvLSTM, and then present the obtained results following two main scenarios.

A. SET-UP AND PERFORMANCE METRICS
It is well known that the accuracy and performance of each forecasting model depend mainly on its parameters (hyper-parameters, lags, coefficients). Therefore, before models’ training, it is mandatory to fine tune them using suitable parameters. Unfortunately, this process seems to be time...
consuming, as it requires testing different combinations in order to find the best parameters.

We used the Keras python library for LSTM, CNN and ConvLSTM and the statemodels python library for ARIMA. The models are presented as follows:

- ARIMA: the values of the parameters $p$, $d$ and $q$ are: 1, 0, 1 respectively.
- LSTM: A model with three layers is used. The first layer is an LSTM layer with 16 neurons, and the second and third layers are both dense layers. The model is trained using 500 epochs.
- CNN: the model is composed of five layers. The first one is a convolution layer with 64 filters and a kernel size of 2. The second layer is a max pooling layer with a pool size of 2. Then the output of the second layer is flattened and passed to a dense layer.
- ConvLSTM: The model used for ConvLSTM is an encoder/decoder model. The encoder is the ConvLSTM with 64 filters and a kernel size of (1,3) to extract the Spatio-temporal data from the dataset. Then the output of the encoder is fed to the LSTM layer, which plays the role of a decoder. The model is trained using 200 epochs.

It is worth noting that different epoch sizes were tested and the best ones are selected to avoid underfitting or overfitting phenomena. The activation method used is $relu$ and $adam$ as optimizer.

Regarding the performance evaluation of the abovementioned algorithms, three different processing units have been used; a personal computer (PC), a Raspberry Pi 3 B+ and an NVIDIA Jetson Nano development board. The training is done in each of those processing units and the training time was computed. The computer used is a Dell Inspiron 157000 gaming, with 16 GB RAM, 64bit @ 2.80 GHz i7-7700hq CPU; and NVIDIA GeForce GTX 1050 Ti GPU with 4GB RAM. The NVIDIA Jetson nano is a development board with a Quad-core ARM Cortex A57, 4GB LPDDR4 RAM and NVIDIA Maxwell GPU with 128 CUDA cores. The Raspberry Pi 3 B+ is a single-board computer, which is equipped with a Broadcom BCM2837B0, cortex-A53 processor (ARMv8, 64bit, @1.4GHz), and 1GB LPDDR2 SDRAM memory and 400 MHz Video Core IV GPU. NVIDIA Jetson nano was chosen as a processing unit according to its capabilities, i.e., computational power and price.

Three metrics are used: Symmetric Main Absolute Percentage Error (SMAPE), Root-Mean-Square Error (RMSE), and R-squared ($R^2$). The SMAPE, RMSE, and $R^2$ are computed as specified in [47], where $n$ is the number of observed values, $A_t$ are the real values and $F_t$ are the forecasted values. $SS_{res}$ is the residual sum of squares and $SS_{total}$ is the total sum of squares. Also, the Diebold-Mariano (DM) [48], [49] a test is conducted to compare the prediction errors of the used models.

### B. EXPERIMENTAL RESULTS

In order to study the effectiveness of the multi-variate LSTM forecasting algorithm against ARIMA, CNN, and ConvLSTM, several experiments have been conducted following two evaluation methods. In the first one, the whole trajectory is used to train the model (WT), while in the second, the trajectory is divided into segments (ST) and each segment’s dataset is used to train a related model.

1) WT: TRAINING UPON THE WHOLE TRAJECTORY

The first method (WT) was tested by training the models upon the whole dataset of a trajectory. One of the driving cycles is chosen for the test, the algorithm uses the pre-trained model to forecast the future speed values. The results of the forecast are then stored and error metrics are processed to compute the models’ accuracy. It is worth mentioning that this process is repeated in each time step (i.e. TM5 and TM10) for the considered algorithms and each forecast horizon as well. Fig.9 shows the forecast result of the LSTM algorithm, whereas Fig.10, Fig.11, and Fig.12 present the forecast results of ARIMA, CNN, and ConvLSTM respectively. From these figures, we can clearly see that LSTM and ConvLSTM outperform ARIMA and CNN in forecasting the vehicle’s speed. Error metrics are given in Table.1; thereby ConvLSTM shows the lowest errors for almost all the trajectories followed by LSTM, ARIMA and CNN. On the other hand, ConvLSTM

![FIGURE 9. LSTM forecast result for different horizons for trajectories: a) trajectory A, b) trajectory B and c) trajectory C.](image_url)
performs poorly than LSTM for trajectory B. The same results have been found when the same pre-trained model was used to forecast the speed using the NVIDIA Jetson nano and the Raspberry Pi 3 B+. Forecasting and training time have been computed and presented in Table.3. The training time is processed for each algorithm using the same conditions (e.g., epochs, layers, steps, forecast horizon) in PC, NVIDIA Jetson nano and also a Raspberry Pi 3 B+. It is clearly seen that, as expected, the PC takes less time, followed by the NVIDIA Jetson nano, and last, the Raspberry Pi 3 B+. Also, the time to train a model, using a collect step of five seconds, is higher compared to a time step of ten seconds. This result is due to the fact that the dataset collected with a time-step of five seconds is twice bigger than the dataset collected with a time-step of ten seconds.

On the other hand, it is worth mentioning that, in the first test, the Raspberry Pi 3 B+ outperforms the NVIDIA Jetson nano regarding the training time. This was due to the power supply issues. In fact, the NVIDIA Jetson nano requires a power supply with five Volts and a maximum current between
TABLE 1. Error metrics using a single model for the whole trajectory.

| Trajectory | Collect step (s) | Forecast horizon (s) | SMAPE LSTM (%) | RMSE LSTM (%) | $R^2$ LSTM (%) | SMAPE ARIMA (%) | RMSE ARIMA (%) | $R^2$ ARIMA (%) | SMAPE CNN (%) | RMSE CNN (%) | $R^2$ CNN (%) | SMAPE ConvLSTM (%) | RMSE ConvLSTM (%) | $R^2$ ConvLSTM (%) |
|------------|------------------|----------------------|----------------|--------------|--------------|----------------|--------------|--------------|--------------|--------------|--------------|----------------|----------------|------------------|
| A          | 5                | 30                   | 42.54          | 11.49        | 80.41        | 49.87          | 12.14        | 80.6         | 61.04        | 32.01        | 14.64        | 30.22          | 7.49          | 91.94            |
|            |                  | 60                   | 46.61          | 15.59        | 69.5         | 55.84          | 14.67        | 73.61        | 92.75        | 36.56        | 0.04         | 46.64          | 13.03        | 75.3             |
|            | 10               | 30                   | 55.49          | 15.81        | 64.34        | 43.43          | 10.88        | 83.58        | 64.68        | 28.46        | 17.17        | 28.81          | 5.07          | 96.34            |
|            |                  | 60                   | 51.47          | 15.43        | 65.28        | 46.87          | 12.98        | 76.72        | 77.63        | 38.12        | 6.83         | 32.61          | 7.52          | 91.60            |
| B          | 5                | 30                   | 66.27          | 15.47        | 45.61        | 72.38          | 16.79        | 44.94        | 88.42        | 24.46        | 1.04         | 39.3           | 9.94          | 77.41            |
|            |                  | 60                   | 72.33          | 18.44        | 23.37        | 86.29          | 21.52        | 21.21        | 86.44        | 23.23        | 2.38         | 65.82          | 13.77        | 50.42            |
|            | 10               | 30                   | 68.80          | 16.60        | 38.14        | 63.13          | 14.25        | 57.87        | 85.77        | 24.17        | 2.47         | 106.96         | 25.51        | 2.26             |
|            |                  | 60                   | 73.20          | 18.74        | 22.91        | 67.72          | 17.08        | 37.04        | 84.08        | 22.34        | 2.02         | 84.2           | 23.1         | 5.54             |
| C          | 5                | 30                   | 70.53          | 13.99        | 49.83        | 72.87          | 15.70        | 45.87        | 113.11       | 50.97        | 1.30         | 62.12          | 12.3         | 60.92            |
|            |                  | 60                   | 72.43          | 17.01        | 27.26        | 80.53          | 19.92        | 22.04        | 87.53        | 41.23        | 4.82         | 76.59          | 19.19        | 6.38             |
|            | 10               | 30                   | 69.00          | 14.88        | 41.29        | 61.79          | 13.54        | 53.51        | 90.93        | 37.47        | 0.89         | 57.27          | 11.86        | 62.78            |
|            |                  | 60                   | 77.67          | 18.5         | 17.49        | 66.95          | 16.04        | 32.50        | 92.11        | 48.25        | 1.16         | 65.83          | 14.63        | 44.19            |

FIGURE 13. LSTM forecast results for different trajectories using 5s collect step: a) trajectory A, b) trajectory B and c) trajectory C.

FIGURE 14. LSTM forecast results for different trajectories using 10s collect step: a) trajectory A, b) trajectory B and c) trajectory C.

three and four Amps. However, the used power supply in the first try delivers five Volts with two Amps max current, which resulted in a degradation of the computational power of the NVIDIA Jetson nano; only two cores were used instead of four. After using an adequate power supply, the performance of the NVIDIA Jetson nano has increased, and thus, the training time has decreased as depicted in Table 3. This latter presents also the average forecasting and training time taken by the algorithms to train the model, in each device, and forecast the specified number of steps (forecast horizon). Only LSTM and ConvLSTM are presented in the table as they have fewer error values compared to CNN and ARIMA. Also, from Table 2, which represents the DM test results for all the used models (p-values are between Parentheses). The table shows that, the LSTM and ConvLSTM outperform the other used models in terms of forecasting the speed. The Raspberry PI 3 B+ shows less time to forecast the speed, followed by the PC and NVIDIA Jetson nano. From the Table 3, two conclusions can be drawn. The first one is, that the suitable embedded device, to be used in our platform, is the NVIDIA Jetson Nano. It shows less time for models' training compared to the Raspberry PI 3 B+. Furthermore, the Raspberry PI 3 B+ halts every time the ConvLSTM is running and thus there is no training/forecasting time...
for the ConvLSTM in the Raspberry Pi 3 B+. The second conclusion is, despite that the ConvLSTM presents fewer errors compared to LSTM, the ConvLSTM takes too much time for models’ training compared to LSTM in both devices; PC and NVIDIA Jetson nano.

For instance, to forecast 30 steps ahead, using a collect step of 5 seconds, LSTM takes, on average 147.59 seconds (about 2 minutes and a half) for models’ training and 0.04 seconds for forecasting in PC, while in the NVIDIA Jetson Nano, it takes 819.55 seconds (about 14 minutes) for training and 0.19 seconds for forecasting. This time can be reduced significantly by using 10 seconds as a collect step, as shown in Table 3. ConvLSTM, for the same forecast step, takes 380.55 seconds (about 6 minutes) for training and 0.04 seconds for forecasting using PC, however, when using the NVIDIA Jetson Nano, the time rises to 2023.84 seconds (about 34 minutes) for training and 0.33 seconds for forecasting. Therefore, LSTM is chosen to be used in the platform, and the NVIDIA Jetson nano might be a good choice to use for training and forecasting the speed in real-time in case of Internet disconnection or the instantaneous no-availability of the distant server.

2) ST: TRAINING UPON SEGMENTS OF TRAJECTORY

As an attempt to improve the accuracy of speed forecasting, another method is used for models’ training. In fact, instead of using the whole trajectory dataset to train one model for this specific trajectory, the trajectory is split into segments, and the model is trained upon each segment separately. This approach can reduce the training time significantly, as shown in Table 3. For instance, the training time for LSTM on the whole trajectory is 819.55 seconds (about 14 minutes) in PC, while using the NVIDIA Jetson Nano, it takes 2023.84 seconds (about 34 minutes) for training. However, when using the NVIDIA Jetson Nano, the time rises to 2023.84 seconds (about 34 minutes) for training and 0.33 seconds for forecasting. Therefore, LSTM is chosen to be used in the platform, and the NVIDIA Jetson nano might be a good choice to use for training and forecasting the speed in real-time in case of Internet disconnection or the instantaneous no-availability of the distant server.
TABLE 4. Error metrics using a model for each trajectory segment.

| Trajectory | Collect step (s) | Forecast horizon (s) | SMAPE (%) | RMSE (%) | R² (%) |
|------------|-----------------|----------------------|-----------|----------|--------|
| A          | 5               | 30                   | 17.72     | 6.47     | 99     |
|            | 10              | 30                   | 22.09     | 7.78     | 98     |
| B          | 5               | 30                   | 61.34     | 12.50    | 88     |
|            | 10              | 30                   | 80.92     | 17.94    | 76     |
| C          | 5               | 30                   | 62.43     | 12.58    | 61.19  |
|            | 10              | 30                   | 73.48     | 15.72    | 37.11  |

and a model is trained for each segment. Therefore, for each trajectory, there are one or many forecasting models depending on the type of trajectory. Fig.15 presents the diagram flows of the used algorithms. Fig.15a depicts the forecasting process. It starts with reading data streams from the in-vehicle embedded subsystem and determines the current road segment in which the vehicle is located. The current segment is determined using the GPS coordinates and a Keyhole Markup Language (KML) file in which the boundaries of all the trajectory segments are stored. The algorithm loads then the corresponding model for the given segment and starts using it to forecast the vehicle speed. Error metrics are then processed. When these latter exceed some maximum threshold, a request is sent to the server to train a new model. At the same time, the deployed algorithm in the distant server is launched manually or when a request is received from a client (system deployed on the EV). The training process, presented in Fig.15b, starts with loading the stored historical data, and loads, from a KML file, the segments’ information for the given trajectory. The algorithm then trains a model for each segment and processes the error metrics for each model, the training is repeated until the errors are under a certain threshold. The trained models are then sent back to the client.

It is worth noting that, as depicted in Fig.5, each trajectory is divided into several segments depending on the type of the road. A full trajectory can take place in different road types. For instance, trajectory (A), depicted in Fig.4, takes place in three different road types: (a) urban Low, (b) highway, and (c) urban dense, as also shown in Fig.16.

Table.4 presents the error metrics for the LSTM multi-model forecasting results. It’s clear that the use of multi-models slightly improves the forecasting results. However, the forecasting accuracy greatly depends on the type of data, i.e. the metric errors for trajectory A seem to be better than the errors of the trajectories B and C, which is due to the fact that the speed profile of trajectory A does not contain a lot of changes unlike trajectories B and C. Therefore, the likelihood of having nearly the same speed profile on each road trip on trajectory A is higher. On the other hand, and as mentioned before, vehicle speed depends on many dynamic factors. Combining all these factors increases the complexity of finding the relationship between them, thus, the difficulty of training a good model. Therefore, selecting a model to forecast 30 seconds ahead with a sampling step of 5 seconds can yield better results.

Moreover, the embedded subsystem proves its usefulness for data collection and for real-time speed forecasting, regardless of the low accuracy in some trajectories. As stated above, the vehicle’s speed is hard to forecast, also, the used algorithm does not take into consideration the behavior of the driver, which affects tremendously the speed profile. As a matter of fact, aggressive behavior might result in a dataset that has a lot of speed spikes, as the driver tends to accelerate or decelerate aggressively. This could lead to an incoherent dataset.

VI. CONCLUSION AND PERSPECTIVES

In this study, a prototype of the HELECAR platform for data collection and real-time speed forecasting has been deployed. Several experiments have been conducted to test
its efficiency in developing related in-vehicle services by mainly focusing on speed prediction. The platform was first used for data collection, since no Moroccan historical data, related to EV driving cycles, are available. A driving dataset was established on Rabat-Salé-Kénitra urban roads by our Twizzy EV. The LSTM algorithm was developed and evaluated against well-known algorithms, ARIMA, CNN and ConvLSTM.

From the obtained results, the LSTM algorithm proved its capability to forecast the vehicle’s speed in comparison to the statistical-based (ARIMA) and neural network algorithms (CNN and ConvLSTM). However, despite the effectiveness of the proposed methods (WT and ST) for training LSTM’s model, enhancing its accuracy is highly required. Our ongoing work focuses on improving the accuracy of the LSTM while increasing the forecast horizon. Furthermore, the NVIDIA Jetson nano platform might be a good choice for being used for training and real-time speed forecasting. In fact, despite the obtained results being promising; there is still room for improvement. In particular, in some trajectories in which the speed pattern changes frequently, the accuracy of the proposed speed forecasting algorithm drops for long-term forecast horizons. Therefore, new forecasting strategies, using other algorithms, will be investigated. Also, the in-vehicle embedded sub-system might be optimized for being used to efficiently train the models without relying on a distant infrastructure. Furthermore, more related applications, such as the integration of charging stations, will be included.

**REFERENCES**

[1] H. Jung, R. Silva, and M. Han, “Scaling trends of electric vehicle performance: Driving range, fuel economy, peak power output, and temperature effect,” *World Electr. Vehicle J.*, vol. 9, no. 4, p. 46, Nov. 2018.

[2] M. Safari, “Battery electric vehicles: Looking behind to move forward,” *Energy Policy*, vol. 115, pp. 54–65, Apr. 2018.

[3] A. Lahlu, F. Ossart, E. Boudard, F. Roy, and M. Bakhouya, “A real-time approach for thermal comfort management in electric vehicles,” *Energies*, vol. 13, no. 15, p. 4006, Aug. 2020.

[4] N. Maaroufi, M. Najib, and M. Bakhouya, “Predicting the future is like completing a painting: Towards a novel method for time-series forecasting,” *IEEE Access*, vol. 9, pp. 119918–119938, 2021.

[5] K. Dar, M. Bakhouya, J. Gaber, M. Wack, and P. Lorenz, “Wireless communication technologies for its applications (topics in automotive networking),” *IEEE Commun. Mag.*, vol. 48, no. 5, pp. 156–162, May 2010.

[6] A. Lahlu, F. Ossart, E. Boudard, F. Roy, and M. Bakhouya, “Optimal management of thermal comfort and driving range in electric vehicles,” *Energies*, vol. 13, no. 17, p. 4471, Aug. 2020.

[7] C. Fiori, K. Ahn, and H. A. Rakha, “Power-based electric vehicle energy consumption model: Model development and validation,” *Appl. Energy*, vol. 168, pp. 257–268, Apr. 2016.

[8] M. Zhou, Z. Gao, and H. Zhang, “Research on regenerative braking control strategy of hybrid electric vehicle,” in *Proc. 6th Int. Forum Strategic Technol.*, vol. 1, 2011, pp. 300–303.

[9] A. Jalali, R. Mallipeddi, and M. Lee, “Sensitive deep convolutional neural network for face recognition at large standoffs with small dataset,” *Expert Syst. Appl.*, vol. 87, pp. 304–315, Nov. 2017.

[10] T. Zhang, L. Sun, L. Yao, and J. Rong, “Impact analysis of land use on traffic congestion using real-time traffic and POI,” *J. Adv. Transp.*, vol. 2017, pp. 1–8, Jan. 2017.

[11] H. Rahimi-Eichi, P. B. Jeon, M.-Y. Chow, and T.-J. Yeo, “Incorporating big data analysis in speed profile classification for range estimation,” in *Proc. IEEE 13th Int. Conf. Ind. Informat. (INDIN)*, Jul. 2015, pp. 1290–1295.
Y. NAITEMALEK received the M.Sc. degree in networking and embedded systems from Molay Ismail University, Meknes. He is currently pursuing the Ph.D. degree with the Ecole Nationale Supérieure d’Informatique et d’Analyse des Systèmes (ENSIAS). He joined the International University of Rabat, in 2017, where he worked on the Self-Serv project as an Intern, the project aimed to develop a platform for smarter health organization. He also helped in the development of a monitoring platform used in Micro-Grid (MIGRID) and CASANET projects. He is also developing predictive algorithms for speed forecasting and driving range in electric vehicles using machine learning methods, under the HELECAR Project, which aims to develop approaches for optimal predictive energy management for BEV and PHEV by using Moroccan TIC. His research interests include the IoT and big data technologies, machine learning, and their applications.

M. NAJIB received the Ph.D. degree in computer science from the University of Le Havre, in 2014. He is currently an Assistant Professor of computer science with the International University of Rabat and a TIC Laboratory Member. He has published more than 20 papers in international conferences, journals, and books. He participated to many research projects, such as passage portuaire for the risk management related to containers transportation and the Holistic System (HOLSYS) Project. His research interests include forecasting/prediction, smart building, adaptive systems for workflow management, decision support systems, and simulation and risk management.

M. BAKHOUYA received the HDR degree from the UHA-France, in 2013, and the Ph.D. degree from UTBM-France, in 2005. He is currently a Professor of computer science at the International University of Rabat. He has more than ten years experiences in participating and working in sponsored ICT projects. He has published more than 100 papers in international journals, books, and conferences. His research interests include various aspects related to the design, validation, distributed and adaptive systems, architectures, and protocols. He was the Editor-in-Chief of IJARAS journal and also serves as a Guest Editor for a number of international journals, e.g., ACM Transactions on Autonomous and Adaptive Systems, Product Development Journal, Concurrency and Computation: Practice and Experience, FGCS, and MICRO.

J. GABER (Member, IEEE) received the Ph.D. degree in computer science and computer engineering from the University of Science and Technology of Lille, Lille, France, in 1998. He was a Research Scientist with the Institute of Mathematical Sciences and Informatics, George Mason University, Fairfax, VA, USA. He is currently working the part of the Complex Networks Team (DISCOMNI), FEMTO-ST Institute (UMR CNRS 6174), COMUE University of Bourgogne Franche Comté–University of Technology of Belfort–Montbéliard UTBM, France. His research interests include the analysis and design of emerging devices and systems. He is a member of ACM.

M. ESSAIDI (Senior Member, IEEE) is currently the Chief of the Party of Interactive Digital Center Morocco and a Professor and the Dean with the ENSIAS College, IT Engineering of Mohammed V University, Rabat, Morocco, the Director at the International Cooperation, Ministry of General Affairs and Governance, Morocco, in 2019, and a Faculty Member (Professor and Assistant Professor) with the Faculty of Science, Abdelmalek Essaadi University, Tetuan, Morocco during 1993 to 2011. His research interests include RF and microwave passive and active circuits and antennas for wireless communications and medical systems. He is the IEEE Global Cities Alliance, MEA Chairman during 2021 to 2022, IEEE Humanitarian Activities Committee (HAC) Assessment Committee Member during 2021 to 2022, IEEE EAB Teaching Excellence Editorial Hub, Member during 2021 to 2022. He was extensively involved in the coordination of the World Bank Morocco Country Partnership Framework during 2019 to 2024 and OECD Morocco Country Program during 2019 to 2021 among other international cooperation programs.