Received: 13 December 2019 | Revised: 30 September 2020 | Accepted: 30 September 2020

DOI: 10.1002/spe.2914

RESEARCH ARTICLE

WILEY

Evaluating system architectures for driving range estimation and charge planning for electric vehicles

Adam Thor Thorgeirsson[1,2] | **Moritz Vaillant**[1] | **Stefan Scheubner**[1,2] | **Frank Gauterin**[2]

[1]Dr. Ing. h.c. F. Porsche AG, Weissach, Germany

[2]Institute of Vehicle Technology, Karlsruhe Institute of Technology, Karlsruhe, Germany

Correspondence
Adam T. Thorgeirsson, Dr. Ing. h.c. F. Porsche AG, Porschestr. 911, 71287 Weissach, Germany.
Email: Adam_Thor.Thorgeirsson1@porsche.de

Abstract

Due to sparse charging infrastructure and short driving ranges, drivers of battery electric vehicles (BEVs) can experience range anxiety, which is the fear of stranding with an empty battery. To help eliminate range anxiety and make BEVs more attractive for customers, accurate range estimation methods need to be developed. In recent years, many publications have suggested machine learning algorithms as a fitting method to achieve accurate range estimations. However, these algorithms use a large amount of data and have high computational requirements. A traditional placement of the software within a vehicle's electronic control unit could lead to high latencies and thus detrimental to user experience. But since modern vehicles are connected to a backend, where software modules can be implemented, high latencies can be prevented with intelligent distribution of the algorithm parts. On the other hand, communication between vehicle and backend can be slow or expensive. In this article, an intelligent deployment of a range estimation software based on ML is analyzed. We model hardware and software to enable performance evaluation in early stages of the development process. Based on simulations, different system architectures and module placements are then analyzed in terms of latency, network usage, energy usage, and cost. We show that a distributed system with cloud-based module placement reduces the end-to-end latency significantly, when compared with a traditional vehicle-based placement. Furthermore, we show that network usage is significantly reduced. This intelligent system enables the application of complex, but accurate range estimation with low latencies, resulting in an improved user experience, which enhances the practicality and acceptance of BEVs.

KEYWORDS

connected vehicles, distributed computing, electric vehicles, machine learning, modeling and simulation, range anxiety, range estimation, system architecture

1 | INTRODUCTION

Battery electric vehicles (BEVs) have the potential to solve future problems regarding greenhouse gas emissions from passenger and commercial vehicles and establish independence from depleting fossil fuel resources. To increase driving range of BEVs, choosing a larger battery capacity is one option. However, a large battery corresponds to high vehicle weight, costs, and demand for rare minerals. Therefore, a trade-off solution, where driving range can be fully utilized, needs to be found. With poor charging infrastructure and inaccurate range estimation algorithms, drivers of BEVs experience range anxiety, which is the fear of stranding with an empty battery.[1] As a consequence, drivers reserve battery energy, so that the vehicle's total driving range is not fully utilized.[2] To counteract this problem, accurate range estimation algorithms are required. These algorithms estimate the remaining driving range by predicting the energy consumed on a given, planned route. If the distance to destination is greater than the remaining driving range, charge planning is needed. Charge planning suggests suitable charging stations along the route, considering battery capacity and required energy for the planned trip.

Driving range estimation is a problem that can be solved partially or completely with machine learning (ML). It relies on data distributed between vehicles and cloud infrastructures. In the vehicle, information such as vehicle speed and energy consumption are relevant. Information obtained from external infrastructure, such as live traffic and weather forecasts, is also relevant for the range estimation. Unifying and integrating these sources is an issue that currently hinders further development of range estimation and charge planning software.[3] By applying ML algorithms, many different features can be included in the range estimation, without the need of exact mathematical or physical modeling of the influences. Furthermore, an ML algorithm automatically adapts to changes in system behavior, resulting in a robust model. Not all devices and system architectures are ready for using ML software. This is a problem addressed in the *Machine Learning and Systems (MLSys)* whitepaper,[4] where the authors discuss problems regarding the widespread use of ML systems in commercial applications, such as:

- *How can ML algorithms and systems be designed for device constraints such as power, latency, and memory limits?*
- *How should distributed systems be designed to support ML training and serving?*

These and similar problems have been analyzed generally in the context of the Internet of Things (IoT) paradigm and how IoT with its cloud, edge, and fog computing can support the increasing amount of data that is transferred and used, for example, for ML-based systems.[5] An important aspect of system design is the user experience (UX) and how applications can be deployed to ensure high Quality of Experience (QoE).[6] Distributed computing in cloud, edge, and fog systems is highly dependent on communication and exhibits high system complexity.[7] Even with the emergence of 5G, lean communications are still essential and resources need to be used efficiently.[8] Despite these challenges, connected vehicles can achieve significant improvements in efficiency, performance, and QoE with ML and IoT applications. Siegel et al.[9] studied the feasibility of different connected vehicle applications and even identified driving range estimation as an application that could be deployed in connected vehicles. In recent years, fog- and cloud-based ML algorithms have received increased attention. Due to the diversity of ML algorithms and their applications, corresponding systems are analyzed individually. Tuli et al. analyzed a system for a fog–cloud-based object detection with deep learning,[10] as well as a deep learning based smart health-care system.[11] Lin et al.[12] proposed and analyzed a deep learning framework for smart manufacturing inspection systems based on fog computing. In current literature about range estimation algorithms, system architecture and the practicability of the proposed methods is rarely investigated. Researchers frequently propose ML algorithms that rely on big data and distributed computing, yet the analysis from a systems aspect is neglected. In this work, we analyze driving range estimation software and evaluate its performance. Our approach models and simulates the software in a distributed computing setting and enables system architecture evaluation in early stages of the development. Our main contributions are:

- We show that performance and QoE of range estimation and charge planning software is highly dependent on system architecture. Therefore, software concepts should be evaluated with respect to system architecture.
- We propose estimating resource requirements based on route length, algorithm time complexity, and the number of instructions in the compiled code. In that way, the software can be tested for different use cases to ensure a sufficient QoE for routes of varying length.

- We analyze system-architectures corresponding to modern commercial vehicles, as well as to concepts proposed in related works. Furthermore, we propose promising alternatives to existing concepts.
- We consider both learning and inference of ML-based driving range estimation. The system analysis is divided into inference and learning. Inference is the task of performing prediction or estimation with a learned model. Most research focuses on the learning procedure, whereas in terms of user experience, inference is just as important.[13] Thereby, the focus of our work is on model inference and the performance of the system as a whole.

The article is organized as follows. Section 2 introduces the state of the art of driving range estimation and connected vehicles and gives an overview of different techniques for evaluating the performance of distributed systems. In Section 3, the necessary hardware and software is described and modeled to enable system simulation. In Section 4, the performance of possible systems is evaluated through simulations. Section 5 concludes the article and provides an outlook.

2 | RELATED WORK

2.1 | Range estimation, routing, and charge planning

A driving range estimation algorithm is used to determine the distance-to-empty on a given route. Moreover, range estimation is essential in the planning of long-distance trips, where the driving range is shorter than the distance to the destination and charging stops are necessary. The principle of most range estimation algorithms is to first estimate the energy required for the given route. By comparing the required energy with the battery's estimated state of energy (SoE), the driving range and destination attainability can be determined. Figure 1 shows the process of routing, range estimation, and charge planning as a flowchart.

Using street maps and the corresponding graphs, routing algorithms calculate the fastest route from the starting point to the destination. A myriad of routing algorithms exist, the most famous one being Dijkstra's algorithm.[14] Commercial navigation software suppliers do not publish the exact details of their proprietary methods. A relatively fast routing

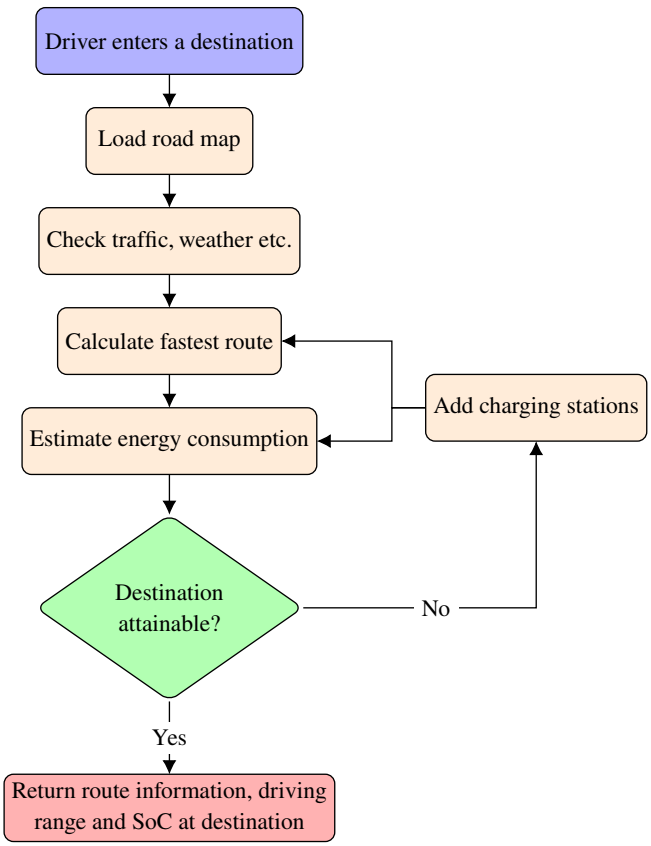


FIGURE 1 A simplified flowchart for routing, range estimation, and charge planning [Color figure can be viewed at wileyonlinelibrary.com]

algorithm is the A-star algorithm.[15] The A-star algorithm can be implemented with different heuristics that speed up the algorithm, such as highway hierarchies, where highway connections are preferred.[16] For BEVs, it is also common to include energy efficiency in the route calculation (eco-routing). In this case, edge weights representing energy consumption could be negative, since BEVs are able to regenerate energy when decelerating or driving downhill. The Bellman–Ford algorithm[17,18] can handle negative edge weights and has been used in eco-routing.[19,20]

The estimation of future energy consumption is frequently performed using ML algorithms.[21-23] Thereby, information from the vehicle, such as mass or current tractive energy consumption, is used together with predictive information about the selected route. This information comprises the map data on each link of the route (e.g., legal speed limit, curvature, slope) as well as live traffic data from other connected vehicles. The ML algorithm is trained to find the relation between the available predictive information and the resulting tractive energy consumption. Thus, a range estimation algorithm uses both data from the vehicle and from other connected vehicles via the cloud. Depending on the ML algorithm and the amount of data, the range estimation algorithm can demand a lot of resources. The data origin and the computing latencies require a smart system architecture. Future energy consumption can moreover be estimated using mechanistic models based on physical principles and equations.[24-26] In addition, a hybrid model combining a mechanistic model and ML can as well be applied.[19,27] Traditionally, automotive software is implemented on electronic control units within the vehicle and uses mostly data from the vehicle itself. This is also true for range estimation and charge planning. Some research articles, however, have proposed distributed systems with software placed partially or completely in the cloud. Table 1 shows a summary of these related works. The articles are classified by the method used for range estimation and by the system architecture which corresponds to the proposed concepts. We differentiate between ML models, mechanistic models, and hybrid models combining the two. Regarding system architecture, we observed whether the software modules are placed mainly in the cloud, in the vehicle or divided uniformly between the two in a hybrid manner.

In the articles, the system architectures are only vaguely described and the feasibility of the proposed concepts regarding system performance is not investigated. In further research articles, proposed algorithms rely on live-information such as traffic or weather, which is normally not available in the vehicle without some sort of connectivity.[21,23,32-35] However, system architecture and performance are not investigated.

2.2 | Connected vehicles and vehicular networks

In connected vehicles, communication with other vehicles (V2V) and with infrastructure (V2I) is possible. Mobile internet (4G, 5G) and direct short-range communication (DSRC) are two competing communication technologies used in vehicle-to-everything (V2X) settings, which can be seen in a survey conducted by Abboud et al.[36] Different computing principles are possible in connected vehicles, such as cloud, edge, and fog computing.[37] In the vehicles, resources such as computing performance and storage are limited. In turn, cloud resources are ample and a synergistic distribution in a system with vehicles and clouds is reasonable. Deploying software in such systems is a challenge, as the module placement has a significant impact on their performance and latency. Brogi et al.[38] give an overview of existing methodologies for optimally solving module placement problems in fog infrastructures. The vehicle communication platform Cloud-Think was developed to securely enable software functionality divided between vehicle and cloud. Its system architecture consists of vehicle ECUs as well as data and gateway servers that communicate over wireless connections.[39] Cloud-based

TABLE 1 Summary of related works on driving range estimation in distributed systems

Work	Estimation method	System architecture
Thibault et al.[19]	Hybrid model	Hybrid
Fukushima et al.[22]	ML model	Cloud-based
Yi and Bauer[24]	Mechanistic model	Vehicle-based
Grubwinkler et al.[25,28]	Mechanistic model	Cloud-based
Jayakumar et al.[26]	Mechanistic model	Hybrid
Scheubner et al.[27]	Hybrid model	Vehicle-based
Ferreira et al.[29]	ML model	Vehicle-based
Lee et al.[30,31]	ML model	Cloud-based

vehicle functions have been classified into four different models: *only cloud*, *fall-back method*, *duplicate function*, and *elastic application*. In this article, we focus on the class *elastic application*, where the software consists of several modules distributed between the vehicle and cloud.[40] One of the identified use cases includes predictive functions relying on predictive cloud information, where the range estimation operates precisely in such a fashion. Examples for elastic applications are functions using computation offloading. In such applications, computational and/or data intensive tasks, such as image processing and ML, are offloaded to the cloud. This offloading can either be permanent or adaptive, but in both cases the module placement is intelligently performed to minimize parameters such as latency, energy, and cost.[41]

There are many examples in the literature where cloud or fog computing is used in vehicular technology. Siegel et al.[42] give an overview of the state of the art of connected vehicles and their applications. Lee et al.[31] use ML to analyze driving behavior in the cloud using data from connected BEVs. Wu et al.[43] use cloud computing in electric vehicle charging control and dispatch optimization. Ozatay et al.[44] implemented a velocity profile optimization with dynamic programming where computationally intensive calculations were performed in the cloud. Saini et al.[45] propose a middleware for vehicular infotainment systems where computation tasks are carried out in the cloud and only relevant content is forwarded to the vehicle. Yaseen et al.[46] perform cloud-based video analytics using convolutional neural networks.

Connectivity is necessary for accurate routing, range estimation, and charge planning. Live route and traffic information are already available in many production vehicles. Traditionally, this is transmitted over the traffic message channel (TMC). Another standard, OpenLR, allows more flexibility and a higher resolution. An overview of live traffic related data formats and protocols is given by Henrickson et al.[47] With improved connectivity, application programming interfaces (APIs) with even greater flexibility are available, where additional information such as weather forecasts can be included. In this work, all external information is requested and delivered through APIs.

2.3 | Evaluation and simulation frameworks for distributed systems

For optimal system design and module placement, performance evaluation is important. In the early stages of development, measurements are rarely possible, but simulations can be performed to approximately evaluate performance of different system architectures and module placements. However, simulating the performance of distributed systems is a challenging task. Cloud, edge, and fog computing concepts introduce increased complexity with high diversity of devices and high numbers of possible system architecture variants. An overview of simulation scenarios in fog and edge computing is given by Svorobej et al. In addition, their review includes a brief comparison of simulation tools.[48] *Cloudsim* is a widely used toolkit for the modeling and simulation of cloud computing environments.[49] In an analysis of cloud and fog simulation tools, three of six tools were extensions of *Cloudsim*.[50] The most used tool according to a citation count is *iFogSim*, which enables the simulation of cloud, edge, and fog computing settings to evaluate the impact of different module placements and resource management techniques on different QoE and QoS (Quality of Service) metrics.[51] Ghosal et al.[52] defined and proposed metrics for the evaluation of system architectures divided into *nonfunctional requirements*, *degree to accommodate changes*, *customer requirements*, and *compatibility to legacy designs*. *iFogSim* fulfills all our requirements, allows for variable modeling of resource requirements, evaluates the correct metrics, and is well established in the scientific community. Therefore, we choose *iFogSim* for our work.

3 | MODELING OF SOFTWARE AND HARDWARE SYSTEMS

To investigate performance through simulations, the range estimation software and the corresponding hardware must be modeled. In the following sections, the modeling of the software and hardware is discussed.

3.1 | Hardware and connectivity

In hardware modeling, memory, storage, and bandwidth properties are trivial to determine and model. Modeling the computational performance is not as trivial and several different approaches and metrics exist. One option is to measure processor capacity and load or the number of cores or virtual cores.[53] Alternatively, generic and traditional metrics such as FLOPS (FLoating-point Operations Per Second) and MIPS (Million Instructions Per Second) can be used.[54] Several

variants and derivative metrics have been defined and a good overview is given by Wang et al.[55] *iFogSim* defines hardware units with the following indicators:

- processor performance (MIPS)
- RAM (e.g., GB)
- cost rate per MIPS used (e.g., $)
- busy and idle power (W)

Each hardware unit is thus modeled with these parameters. MIPS is not a perfect performance indicator, its main defect is that it is architecture-dependent. However, for reduced instruction set computing (RISC), it is an acceptable performance indicator.[56]

For the range estimation and charge planning, several dedicated hardware units are needed. Figure 2 shows an overview of the hardware units and their topology. The vehicle's central ECU is the main processing unit, whereas a connectivity module establishes a connection with the backend and handles data transfer over wireless or mobile internet. In the cloud, an OEM (Original Equipment Manufacturer) backend is used for all processing except the route calculation, which is performed in a third-party backend from the navigation provider. Table 2 shows these devices and their specified memory size, processor speed in MIPS, power usage, and instruction set of the processor. In contrast to a dedicated cloud server, an ECU needs to perform multiple tasks simultaneously, that is, a single function does not generally have full access to the processor's performance and memory. Therefore, an available processor speed in the magnitude of 10^3 MIPS is assumed for the ECUs.[56] The backend units should be significantly more powerful than the ECUs; therefore an estimated processor speed of 10^5 MIPS is assumed.[49] One difference between processors of a cloud and a vehicle ECU is the architecture and instruction set. Usually, RISC architecture chips are used in ECUs, whereas CISC (complex instruction set computing) architecture processors are more commonly used in servers and backend units. This means that different compilers are required, which can affect the number of instructions of a software module and is therefore considered in the module placement analysis.[57] The power usage of different hardware units is estimated based on

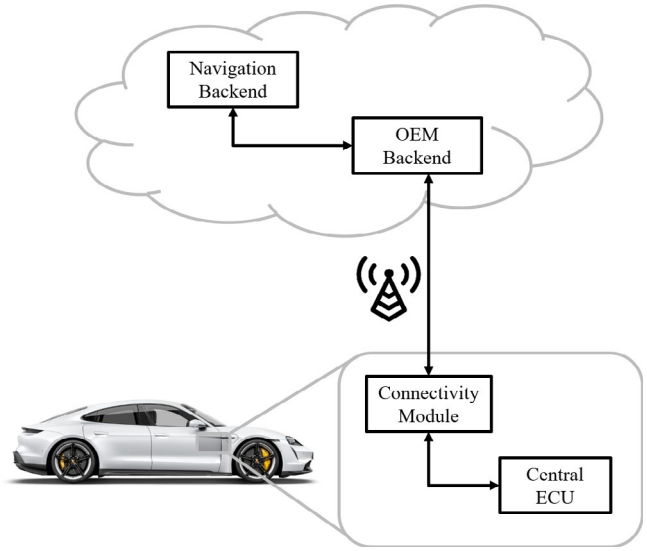


FIGURE 2 Cloud-vehicle hardware topology [Color figure can be viewed at wileyonlinelibrary.com]

TABLE 2 Specifications of the hardware units considered

Device	Location	RAM (GB)	CPU speed (MIPS)	Power A//I (W)	Instruction set
OEM backend	Cloud	200	$1 \cdot 10^5$	$1 \cdot 10^4$ // $8 \cdot 10^3$	CISC
Navigation backend	Cloud	400	$2 \cdot 10^5$	$2 \cdot 10^4$ // $1.6 \cdot 10^4$	CISC
Connectivity module	Vehicle	0.2	$1 \cdot 10^3$	45 // 20	RISC
Central ECU	Vehicle	0.5	$5 \cdot 10^3$	90 // 80	RISC

commercial device information.[58,59] The cost of using cloud instances is estimated to be 1$ per instance per hour, based on commercial cloud services.[49,59]

Wired and wireless connections between hardware units in *iFogSim* are defined by bandwidth and average network communication latency. The vehicle is connected to the cloud via 4G mobile internet. The vehicle ECUs can be connected internally via CAN, ethernet, or similar.[60] It is assumed that the cloud units have a high-speed and low-latency wired connection. In Table 3, the estimated connection speeds and communication latencies are shown. We assume that up to 100 vehicles will simultaneously use the service.

3.2 | Software modules and resource requirements

In this section, the modules of the proposed range estimation and charge planning software are introduced. Figure 3 shows a software block diagram, where modules and their connections are visualized. The software consists of a user-interface, route calculation, energy consumption estimation, charge planning, as well as a vehicle configuration module. In the following, each step of the range estimation is described.

In *iFogSim*, software modules are modeled by the number of processor instructions required for the computational tasks, measured in million instructions (MI). The running time or latency for a certain task is then calculated using the processor speed in MIPS. The time required for a processor instruction is dependent on the type of instruction, that is, performing some instruction may take a shorter or longer time than another instruction. The instructions are dependent on the processor architecture, such as RISC or CISC and the number of instructions for a software module is therefore dependent on the processor. Simple instructions running in a single clock cycle is a typical characteristic of RISC architecture, whereas many instructions in CISC architectures run over several clock cycles. To determine the number of instructions, *iFogSim* or *CloudSim* give no specific directions. We suggest counting the instructions directly in the compiled assembly code. Another possibility is to measure execution time on a certain processor with a known speed in MIPS and then determine the number of instructions. As some modules are dependent on route length, we use the algorithm's time complexity (big-O notation) to describe the number of instructions as a function of route length. In *iFogSim*, the modules are defined as described, with some definitions being functions of route length or graph size.

TABLE 3 Data transfer rate from (row) to (column) in (Mbit/s) and communication latency in (ms)

Device	OEM backend	Navigation backend	Connectivity module	Central ECU
OEM backend	–	100 Mbit/s // 50 ms	15Mbit/s // 100 ms	N/A
Navigation backend	100 Mbit/s // 50 ms	–	N/A	N/A
Connectivity module	10 Mbit/s // 100 ms	N/A	–	10 Mbit/s // 1 ms
Central ECU	N/A	N/A	10 Mbit/s // 1 ms	–

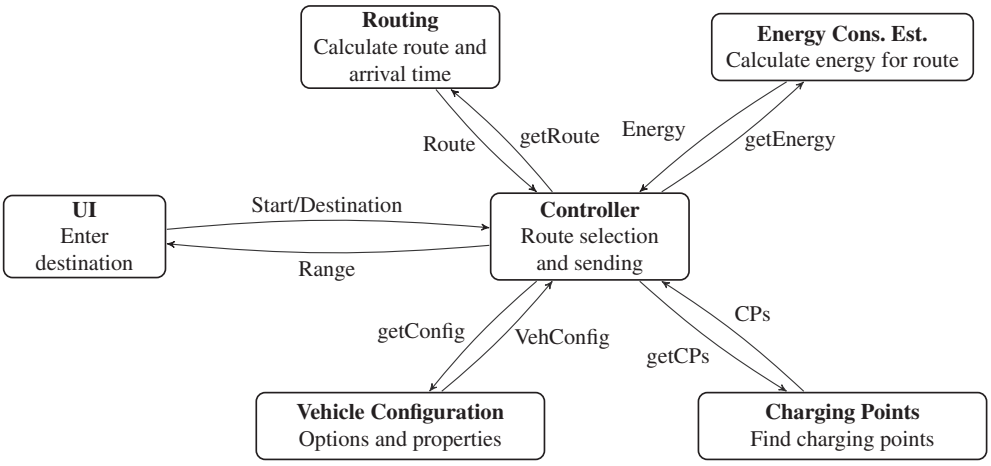


FIGURE 3 Software block diagram

3.2.1 | User interface

In this module, the driver chooses the destination in the user interface (UI) and the destination is sent to the route calculation module. The drivers input is a string with the destination address or name. Before the communication is initiated, the vehicle must be authenticated through a TLS (transport layer security) handshake. The vehicle and backend exchange messages to establish a secure connection. The duration of the TLS handshake comprises time for cryptoprocessing, network latency and other delays due to message parsing.[61]

3.2.2 | Routing

After receiving starting position, destination, and routing options, this module calculates the fastest possible route, with regard to actual traffic information. The performance of a routing algorithm is strongly dependent on the graph that represents the road network. The complexity is dependent on the number of edges $|E|$ and vertices $|V|$ in the graph, as well as its sparsity and the branching factor. For the A-star algorithm, the time complexity is between $\mathcal{O}(|V|)$ and $\mathcal{O}(|E| + |V| \log |V|)$, depending on heuristic and graph type.[62] In our implementation, we estimate the time complexity to be $\mathcal{O}(|E| + |V|)$. Thereby, the number of instructions needed for a route calculation in a subgraph with $|E|$ edges and $|V|$ vertices is linearly dependent on the sum $|E|+|V|$. Based on real graphs for USA[63] and Germany,[64] we assume that for a unit length route and a given heuristic, a subgraph with $V(l) = 1600$ vertices and $E(l) = 4200$ edges is required. In our implementation, the number of instructions for one iteration of the A-star algorithm is 345 I. In addition, a baseline computation of 100 MI is assumed. The total number of instructions is thus:

$$\begin{aligned} \mathrm{CPU}_{req}^{\mathrm{Routing}}(l) &= 100\mathrm{MI} + 345\mathrm{I}(E(l) + V(l)) \\ &= 100\ \mathrm{MI} + 345\frac{\mathrm{I}}{\mathrm{iter.}} \cdot (4200 + 1600)\frac{\mathrm{iter.}}{\mathrm{km}} \cdot l\mathrm{km} \\ &= (100 + 2 \cdot l)\mathrm{MI}, \end{aligned} \tag{1}$$

where l is the route length in kilometers. For a calculated route, its attributes for each route segment are sent to the energy consumption estimation. On average, the length of a segment is approximately 200 m, that is, there are $N = 5$ segments per $l = 1$ km.[65] For each segment, $2 \cdot k$ attributes are stored as doubles (8 Bytes). Additionally, a constant 2000B is used to account for HTTP header and other data and overhead not dependent on route length. The total response size for a route is therefore

$$\begin{aligned} S^{\mathrm{Routing}}(l) &= 2000\ \mathrm{B} + 2k \cdot N(l) \cdot 8\ \mathrm{B} \\ &= 2000\ \mathrm{B} + 2k \cdot 5l \cdot 8\ \mathrm{B} \\ &= (2000 + 80k \cdot l)\mathrm{B}. \end{aligned} \tag{2}$$

3.2.3 | Energy consumption estimation

For the calculated route, the route specific energy consumption (EC) is estimated with the ML algorithm using attributes for the planned route as input parameters. In this regression problem, we use a neural network (NN). The learning of the NN is a non-event-based process which runs in the background and is not considered in the control loop, but rather analyzed separately in Section 4.3. The time complexity of the inference is $\mathcal{O}(|N|)$, where N is the number of segments in the route. In our implementation, inference with NNs can be efficiently performed with 200I.

$$\begin{aligned} \mathrm{CPU}_{req}^{\mathrm{EC}}(l) &= 100\ \mathrm{MI} + 200\frac{I}{seg.} \cdot N(l) \\ &= 100\ \mathrm{MI} + 200\frac{I}{seg.} \cdot 5\frac{seg.}{\mathrm{km}} \cdot l\mathrm{km} \\ &= (100 + 10^{-3} \cdot l)\mathrm{MI}. \end{aligned} \tag{3}$$

By comparing the required energy with the battery's SoE, the destination attainability can be determined. If the current SoE is sufficient to reach the destination, route and energy information can be sent back to the driver. If the destination is not attainable, a charging point search and a charging plan calculation are triggered. The response includes the energy consumption for each route segment, represented by $2 \cdot N$ variables. Thereby, the response size is

$$\begin{aligned} S^{\text{EC}}(l) &= 2000\ \text{B} + 2 \cdot N(l) \cdot 8\ \text{B} \\ &= 2000\ \text{B} + 2 \cdot l \cdot 5 \cdot 8\ \text{B} \\ &= (2000 + 80 \cdot l)\text{B}, \end{aligned} \tag{4}$$

where the 2000 B represent HTTP header and other overhead.

3.2.4 | Charging point search and planner

When triggered, the search finds applicable charging points (CPs) in a geographical corridor along the route. These charging points are then sent to the routing module. With an analysis of the charging infrastructure in western Europe,[66] we established that the mean distance between fast chargers ($P \geq 100\text{k W}$) along major routes is approximately 50 km, that is, the number of charging points along an l km long route is

$$n_{\text{CPs}}(l) = 0.02 \cdot l. \tag{5}$$

The complexity of the charging point search is dependent on the number of possible charging points, that is, it is dependent on route length. When implemented with linear search, the time complexity of therefore is $\mathcal{O}(l)$ and the resource requirements are

$$\text{CPU}_{req}^{\text{CPs}}(l) = (100 + 10^{-5} \cdot l)\text{MI}. \tag{6}$$

The charging points GPS coordinates are given by two variables and the response size is therefore

$$\begin{aligned} S^{\text{CPs}}(l) &= 2000\ \text{B} + 2\frac{vars}{\text{CP}} \cdot 0.02\frac{CPs}{\text{km}} \cdot l\ \text{km} \cdot 8\ \text{B} \\ &= (2000 + 0.32 \cdot l)\text{B}, \end{aligned} \tag{7}$$

where the 2000 B represents HTTP header and other overhead.

Through all reasonable combinations of these charging points, the fastest route is calculated. Depending on battery SoE and maximum driving range, the number of reasonable charging point combinations for a route of length l can be up to

$$n_{\text{CP-Comb.}}(l) = 2^{n_{\text{CPs}}(l)} = 2^{0.02 \cdot l}. \tag{8}$$

Subsequently, the energy for each subroute between the charging points is estimated and charging times at each charger are determined. Thereby, the total travel time is calculated. The charging planner sends multiple requests to the routing and energy consumption estimation, dependent on the number of possible and reasonable routes. Ideally, the routes and waypoints overlap to a certain extent, which means that a previously calculated route and energy consumption can be partially used for another route. In the worst case, all possible routes are different, which means that each route and its energy consumption is calculated individually. The number of possible routes is given by Equation (8). The resulting worst-case resource requirements are

$$\begin{aligned} \text{CPU}_{req}^{\text{Routing, Mult.}}(l) &= n_{\text{CP-Comb.}}(l) \cdot \text{CPU}_{req}^{\text{Routing}}(l) \\ &= (100 + 2^{0.02 \cdot l + 1} \cdot l)\text{MI} \end{aligned} \tag{9}$$

and

$$\begin{aligned} \mathrm{CPU}_{req}^{\mathrm{EC,\,Mult.}}(l) &= n_{\text{CP-Comb.}}(l) \cdot \mathrm{CPU}_{req}^{\mathrm{EC}}(l) \\ &= (100 + 2^{0.02 \cdot l} \cdot 10^{-3} \cdot l)\mathrm{MI}. \end{aligned} \tag{10}$$

3.2.5 | Route selection and sending

The final step is to choose the fastest route and charging plan and to deliver this information to the user interface. Finding the fastest route means searching in the list of calculated routes for the shortest travel time and can be achieved through linear search. The time complexity of linear search is $\mathcal{O}(n)$ where n is the number of routes. Each comparison in the linear search requires approximately 10 I, so the resource requirements are

$$\begin{aligned} \mathrm{CPU}_{req}^{\mathrm{Rate}}(l) &= 100\mathrm{MI} + 10 \cdot 10^{-6} \cdot n_{\text{CP-Comb.}}(l)\mathrm{MI} \\ &= 100\mathrm{MI} + 10 \cdot 10^{-6} \cdot 2^{0.02 \cdot l}\mathrm{MI} \\ &= (100 + 10^{-5} \cdot 2^{0.02 \cdot l})\mathrm{MI} \end{aligned} \tag{11}$$

and the response size is assumed to be 1 kB. Sending the final response, that is, the route, range estimation, and charging plan, to the user interface requires 50 MI. The route is represented by one 8 B variable for each route segment, so the response size is

$$\begin{aligned} S^{\mathrm{Resp.}}(l) &= 2000\mathrm{B} + N(l) \cdot 8\mathrm{B} \\ &= 2000\mathrm{B} + 5 \cdot l \cdot 8\mathrm{B} \\ &= (2000 + 40 \cdot l)\mathrm{B}, \end{aligned} \tag{12}$$

where the 2000 B represent HTTP header and the display values for range estimation and charge plan, which are independent on route length.

In Table 4, we summarize resource requirements of each module. The complete process of the range estimation and charge planning is shown in Figure 4. The figure shows a sequence diagram describing the process beginning from when the driver enters a destination ending with the display of the route, driving range, and charge plan.

In Figure 5, we further visualize the resource requirement of the software modules for different route lengths. Figure 5(A) shows the response size and Figure 5(B) shows the number of instructions of the route-length-dependent software modules as a function of route length. According to Figure 5, it is clear that the routing algorithm and the energy consumption estimation output the most data and also require the most instructions. This gives a hint that the data output from these modules should not be transmitted over mobile internet connection and that placing these in the cloud would result in lower end-to-end latencies.

TABLE 4 Resource requirements of driving range estimation modules for a single route

Component	Complexity	Number of instructions (MI)	Response size (kB)
enterDest()	$\mathcal{O}(1)$	11	1.4
getConfig()	$\mathcal{O}(1)$	20	1
getRoute()	$\mathcal{O}(l)$	$100 + 2 \cdot l$	$2 + 80k \cdot l \cdot 10^{-3}$
getEnergy()	$\mathcal{O}(l)$	$100 + 10^{-3} \cdot l$	$1 + 80 \cdot l \cdot 10^{-3}$
checkAttainability()	$\mathcal{O}(1)$	100	1
getCPs()	$\mathcal{O}(l)$	$100 + l \cdot 10^{-5}$	$1 + 0.32 \cdot 10^{-3} \cdot l$
rateRoutes()	$\mathcal{O}(l)$	$100 + 10^{-5} \cdot 2^{0.02 \cdot l}$	1
writeResults()	$\mathcal{O}(1)$	50	$2 + 40 \cdot 10^{-3} \cdot l$

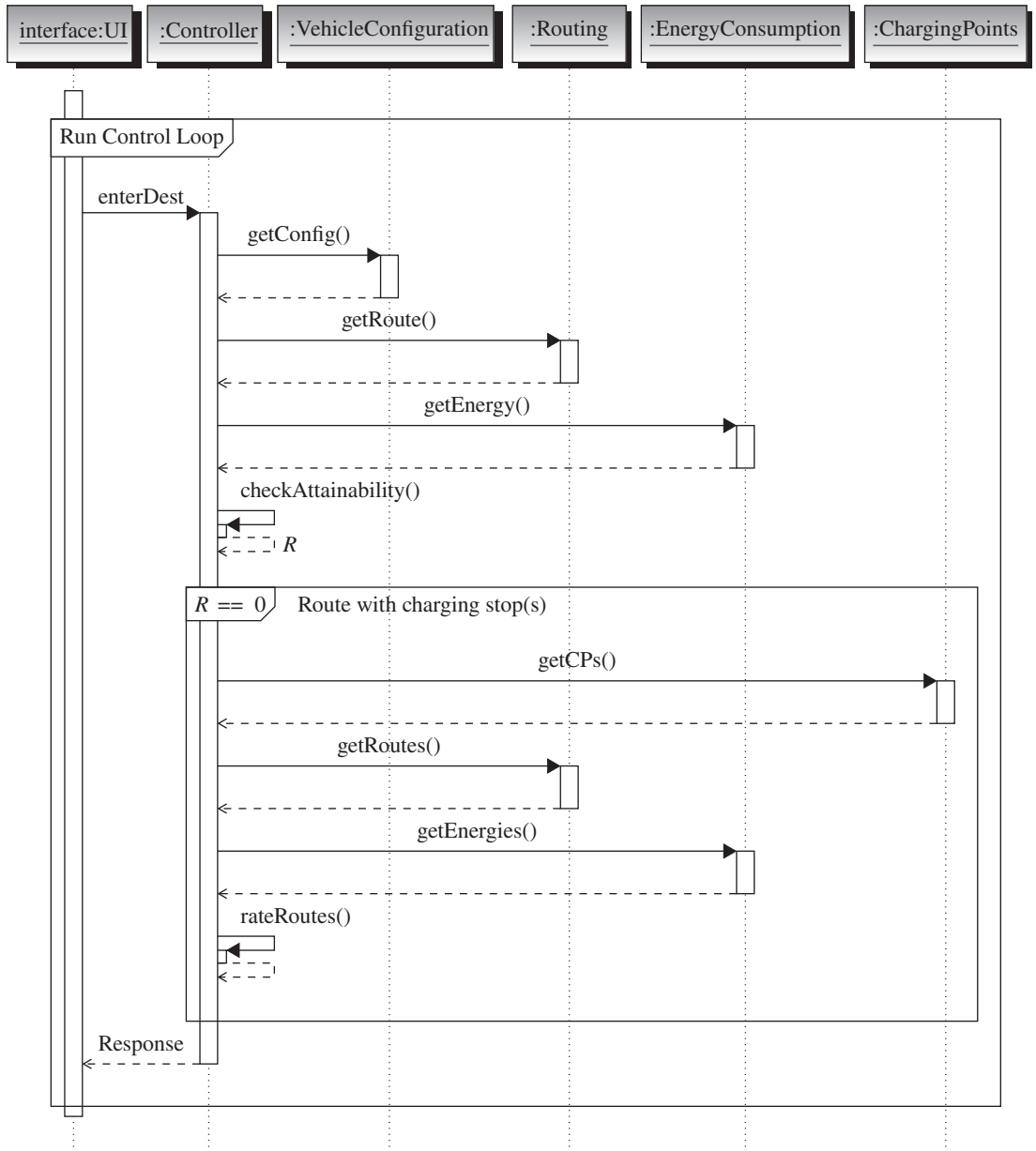
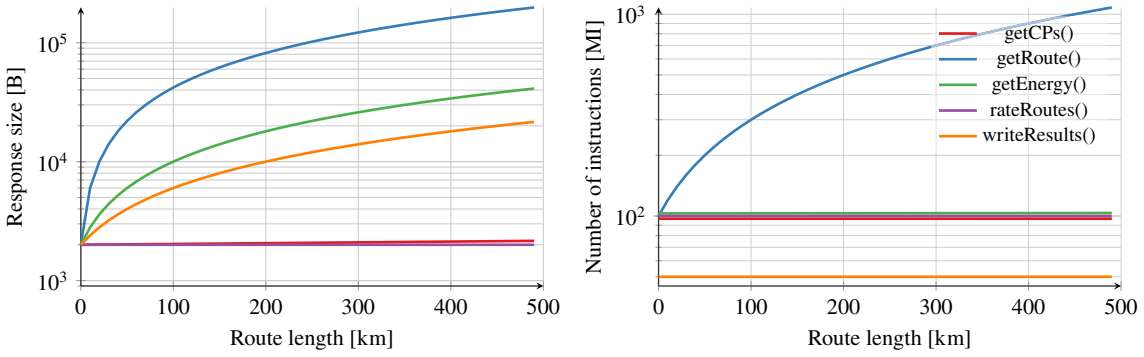


FIGURE 4 Sequence diagram for route calculation, driving range estimation, and optional charging planner



(A) Amount of data generated by the software modules depending on route length.

(B) Number of instructions of the software modules depending on route length.

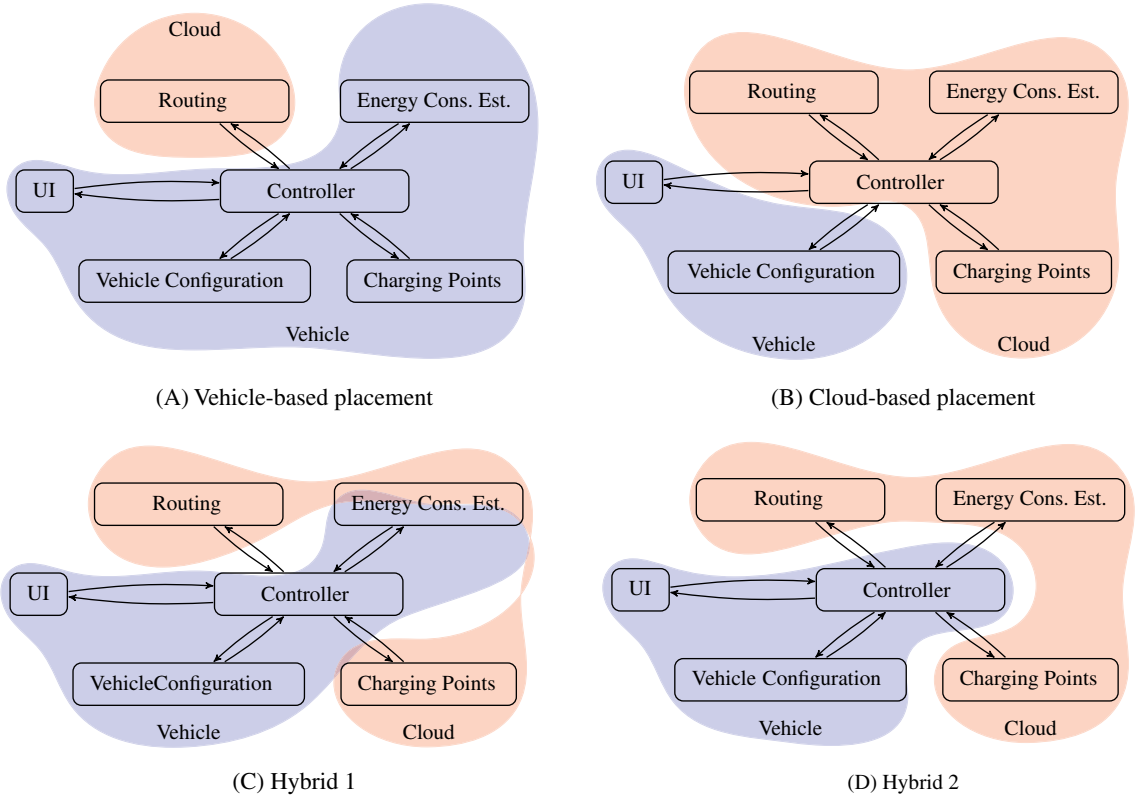
FIGURE 5 Resource requirements of route planning, range estimation, and charge planning [Color figure can be viewed at wileyonlinelibrary.com]

4 | PERFORMANCE EVALUATION

To evaluate the performance of the systems described in Section 3, we simulate these with *iFogSim*. The metrics of greatest importance are: timing, network usage, reliability, availability, flexibility, scalability, expandability, security, energy efficiency, and cost. Timing, energy efficiency, network usage, and cost of cloud execution are all measurable with simulations in *iFogSim* and monetary cost of network communication can be derived from network usage, if the network provider's conditions are known. The other metrics must be assessed subjectively. The control loop shown in Figure 4 is analyzed with different route lengths and module placements. For the control loop, we measure following performance indicators:

- Latency
- Network usage
- Cost of cloud execution
- Energy usage in cloud and vehicle

An optimal module placement maximizes all performance indicators for all route lengths. In this work, the number of possible and reasonable combinations is low. Therefore, all of these combinations can be analyzed to determine the optimal module placement. For each module placement variation, the performance indicators are calculated. Figure 6 shows four module placement variants between the vehicle and cloud. The placement in Figure 6(A) is the classical vehicle-based placement used as a baseline for the evaluation. The vehicle-based placement is commonly applied in current day BEVs. Furthermore, it was suggested by Yi and Bauer,[24] Scheubner et al.,[27] and Ferreira et al.[29] In that placement, all software modules are placed in the vehicle except for the routing algorithm, which is based in the cloud. The routing algorithm relies on real-time information on traffic and road conditions and can be seen as an external service. In the cloud-based placement shown in Figure 6(B), all but the UI and vehicle configuration module are placed in the cloud. In terms of



(A) Vehicle-based placement

(B) Cloud-based placement

(C) Hybrid 1

(D) Hybrid 2

FIGURE 6 Software component diagram showing the four module placement variants [Color figure can be viewed at wileyonlinelibrary.com]

inference, this placement corresponds to the systems suggested by Fukushima et al.,[22] Grubwinkler et al.,[25,28] and Lee et al.[31] The placement shown in Figure 6(C) is a hybrid between the cloud- and vehicle-based placements, where the routing algorithm and the charging point search are implemented in the cloud and the energy consumption estimation module is placed dynamically both in the cloud and in the vehicle. The second hybrid placement shown in Figure 6(D) is the same as Hybrid 1, except the energy consumption estimation module is only implemented in the cloud. This placement corresponds to the systems suggested by Thibault et al.[19] and Jayakumar et al.[26] In the following, the simulations and their results are discussed. In Section 4.1, the setup of our experiments in *iFogSim* is presented. In Section 4.2, inference with the range estimation algorithm is analyzed. In Section 4.3, the learning of the range estimation model is examined.

4.1 | Experiment setup

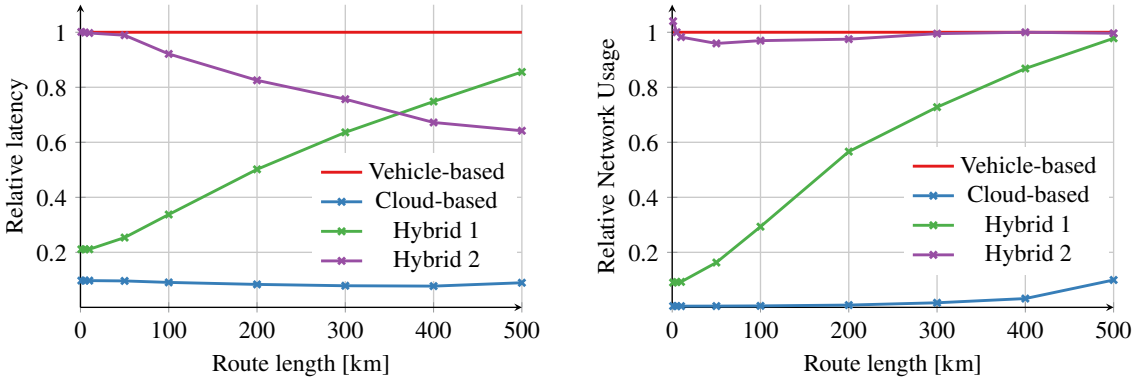
To analyze the driving range estimation and charge planning software, *iFogSim* is configured for the simulation of the systems presented in Section 3. In the following, we describe which classes of *iFogSim* we use to set up our experiments. A *FogDevice* is created for each of the hardware units in Table 2 with the given specifications. According to the topology shown in Figure 2, a direct hierarchy of *FogDevices* is defined. The parent–child pair communication in the hierarchy is configured according to the specification in Table 3. In addition, appropriate *Sensors* are configured in the vehicle to measure velocity, energy consumption, and so on. Finally, one *sensor* and an *actuator* are configured to represent the UI. The latency between a *sensor*/*actuator* and the vehicle's central ECU is estimated to be 5 ms. An *AppModule* is created for each of the modules in the driving range estimation and charge planning software shown in Figure 3. For each of the edges between the software blocks in Figure 3, an *AppEdge* is created. Each edge carries a *tuple* that defines the function of the edge. The processing requirements of the edges' tuples are specified according to Table 4, where the variable l is used to specify the length of the route. The edges are event-based and their function is triggered by an incoming tuple from a source software module. To monitor and measure the end-to-end latency of the control loop shown in Figure 4, an *AppLoop* is specified according to the sequence diagram. With the class *ModulePlacement*, the mapping of the *AppModules* on the *FogDevices* is defined. Thereby, the module placement variants shown in Figure 6 can be configured for the simulations. The results of the simulations include the end-to-end latency of the control loop, as well as the network usage. Furthermore, each *FogDevice* measures the energy used during the simulation and the cost of using cloud instances is calculated. In the following section, we present an analysis of these results.

4.2 | Inference

For the driving range estimation and charge planning software, we simulated four different module placement variants, each for different route lengths. The route lengths simulated are {10, 50,100, 200,300, 400,500}km. The mean results based on the simulations are shown in Table 5. The table shows the performance indicators control loop latency, cost of cloud execution, energy usage in the cloud, energy usage in the vehicle, and total network usage, in proportion to the baseline vehicle-based placement. As the simulation includes uncertainty, we show the results in proportion to the baseline and not the absolute values. In that way, the absolute system specifications are of lower importance compared with the ratio of the specifications of different system architectures. For all performance indicators, the cloud-based placement is the best. The control loop latency, cost of execution in cloud, and total network usage are significantly lower in the cloud-based placement than in the other three placements. The energy usage, both in the cloud and in the vehicle, is similar for all four module placements. The first hybrid placement shows significantly better results than the vehicle-based placement,

TABLE 5 Performance indicators of module placements in proportion to the baseline, vehicle-based, placement

Placement	Latency (–)	Cost (–)	Energy cloud (–)	Energy ECU (–)	Network usage (–)
Vehicle-based	1	1	1	1	1
Cloud-based	0.09	0.03	0.92	0.99	0.03
Hybrid 1	0.56	0.68	0.98	1.00	0.60
Hybrid 2	0.80	1.07	1.00	1.00	0.98



(A) Latency of the control loop in different module placements for different route lengths in proportion to the latency of the vehicle-based placement.

(B) Network usage of the module placements for different route lengths in proportion to the network usage of the vehicle-based placement.

FIGURE 7 Relative latency and network usage of range estimation of a route with charging stops [Color figure can be viewed at wileyonlinelibrary.com]

but still fails to attain the performance of the cloud based placement, which achieves more than 10-fold improvement in latency, cost, and network usage.

Figure 7 shows the control loop latency and network usage of different module placements in proportion to the vehicle-based placement for different route lengths. For the driver, these performance indicators are most important in terms of user experience. Figure 7(A) shows the latency of the control loop and Figure 7(B) shows the network usage. For both latency and network usage, the cloud-based placement is consistently better than the vehicle-based placement. The improvement in latency is approximately 10-fold for all route lengths. The network usage of the cloud-based placement is approximately 10 times lower than that of the vehicle-based placement for a route length of 500 km and even lower for shorter route lengths. The Hybrid 1 placement's performance is similar to the cloud-based placement for shorter routes, but with increasing route length the performance worsens and becomes similar to that of the vehicle-based placement. The second hybrid placement's performance is similar to that of the vehicle-based placement, apart from an improvement in relative latency with increasing route length. The cloud-based placement achieves clearly the highest over-all performance.

If we analyze the causes for the differences, the most important factor is the time used down- or uploading data needed for the range estimation and charge planning. In the cloud-based placement, the range estimation uses the route, road, and traffic information directly within the cloud and only transmits the final display values to the vehicle, which are considerably smaller in size. Furthermore, computation in the cloud is faster, which also improves the control loop's latency. Of the four module placement variants analyzed, cloud-based inference is clearly superior. Our proposed system architecture can enable driving range estimation concepts such as those of Fukushima et al.,[22] Grubwinkler et al.[25,28] and Lee et al.[30,31] to perform efficiently.

4.3 | Learning

In the previous section, we observed that inference can be done efficiently when the software modules are distributed between vehicle and cloud; however, the learning of models has yet to be considered. In this section, we compare latency and network usage of two different strategies: vehicle-based (on-device) learning and cloud-based learning, utilizing either batch or online learning algorithms.[67] In related works on cloud-based-driving range estimation, cloud-based learning has to date been the preferred choice.[22,30,31]

The training data in the range estimation are streaming data that arrive sequentially with a frequency of up to 10 Hz and can be non-IID.[68] For streaming training data, online learning algorithms are a good choice.[69] With this, the learning can be done with a single pass of the observed data, which is then discarded. Optionally, the data may be saved, which lead to a traditional training data set, enabling the application of batch learning algorithms. A significant benefit of online learning algorithms is that the range estimation model is maintained live and can improve the estimation on-the-go. This can also be seen as a requirement for the software, which makes other learning algorithms undesirable.

As discussed in Section 2.1, some of the input parameters for the range estimation algorithms are traffic, navigation, and weather information. This information is also necessary in the learning of the model. However, only a description of the current situation is needed, which can be observed in the vehicle directly. For example, the temperature can be measured with the appropriate sensor and the traffic can be observed with on-board cameras and other sensors.[27]

Stochastic gradient descent (SGD) and its variants are widely used learning algorithms.[70] SGD and variance-reduced versions such as stochastic variance reduced gradient (SVRG) and STRSAGA, can be implemented for the online, single-pass setting.[71] Furthermore, it has been shown that the learning can be implemented in linear time.[72] For the performance of the software, the placement of the learning module must be decided. We compare the options of vehicle-based learning and cloud-based learning.

Consider the number of iterations $\mathcal{I}$ needed for an algorithm, such as SGD, to converge to an acceptable level. The time needed for each iteration is $\mathcal{T}_\mathcal{I}$ and therefore the total time for the learning of an estimation model in a nondistributed setting is

$$T = \mathcal{I} \cdot \mathcal{T}_\mathcal{I}. \tag{13}$$

In a distributed setting, the communication between vehicle and backend is also important. If c is the latency of the communication, the total time needed for a distributed learning of the model is

$$T = \mathcal{I} \cdot (c + \mathcal{T}_\mathcal{I}), \tag{14}$$

if communication in each iteration is assumed. Since algorithms, such as SGD, require many fast iterations, even fast communications result in poor performance, as $c \gg \mathcal{T}_\mathcal{I}$.[73] The learning algorithm can be deployed in a vehicle ECU as well as in the cloud. We estimate the communication latency c based on the data size of the training data for cloud-based learning, compared with the model size for vehicle-based learning. In Figure 8, we show our estimation of the communication cost. In the figure, the size of the 10-Hz training data stream (TD) is shown dependent on driving time in minutes. Additionally, the estimated model size of neural networks with five and 10 hidden layers (NN5 and NN10) is visualized.

The preferred placement of the learning module is dependent on the required frequency of the recalculation of the range estimation. If the required frequency is high, the communication cost of the 1-Hz cloud-based setting is lower than that of the vehicle-based setting. For the 10-Hz cloud-based setting, the communication cost is higher than in the vehicle-based setting if the time interval between recalculations is greater than 20 or 36 s for the five and 10 hidden layer networks, respectively.

To minimize total network usage and communication cost c, the recalculation frequency should be as low as possible and as high as necessary for the optimal user experience. From Figure 8 it can be seen that placing the learning algorithm in the vehicle will reduce the communication cost, as the model size is effectively smaller than the size of the training data. In terms of communication and computation cost, vehicle-based learning can be significantly better than cloud-based learning. In this respect, the vehicle-based ML concepts for driving range estimation by Scheubner et al.[27] and Ferreira et al.[29] can therefore be implemented with our proposed system architecture in an efficient way. Related works such as by Fukushima et al.[22] and Lee et al.,[30,31] which have favored cloud-based learning, could benefit from placing the learning modules directly in the vehicles.

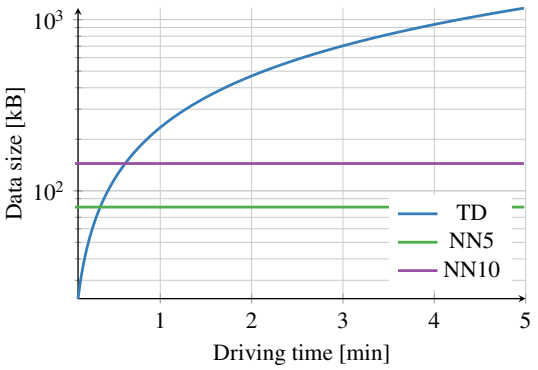


FIGURE 8 Size of the 10-Hz training data stream (TD) and estimated model size of neural networks with five and 10 hidden layers (NN5 and NN10) in kB, dependent on driving time in minutes [Color figure can be viewed at wileyonlinelibrary.com]

5 | CONCLUSIONS AND FUTURE WORK

In this work, we evaluated the performance of an electric vehicle driving range estimation software in different system architectures. Our evaluation is intended for the early stages of the development and enables the comparison of different module placements regarding latency, network usage, cost, and energy usage. By modeling the software and the hardware, simulations with *iFogSim* could be performed. The results show that for inference, a cloud-based module placement is superior to other investigated placements. The cloud-based module placement is significantly better than the current day baseline, the vehicle-based placement. We estimate that the end-to-end latency from the input of the destination to the display of the route and driving range can be improved by a factor of 10. Furthermore, network usage is drastically reduced. Additionally, we analyzed different settings for the learning of the model and found a vehicle-based learning setting to be a more feasible choice than cloud-based learning, which was favored in related works. The result obtained with our analysis can be used in early stages of the development to rate possible solutions, identify which are promising and which are not practicable. To confirm the conclusions drawn from this work, future work will concentrate on further experiments and empirical evaluations. In doing so, the performance of different system architectures shall be tested using a fleet of connected vehicles. In addition, a strategy for deploying the proposed system in large scale shall be devised. Here, frameworks such as *FogBus*,[74] *iGateLink*,[75] or *Aneka*[76] can be applied to ensure a secure, scalable, and cost efficient deployment.

The criteria reliability, availability, flexibility, scalability, expandability, and security cannot be assessed from our simulation results. Our subjective point of view is that the proposed cloud-based module placement offers more flexibility, scalability, and expandability than a traditional vehicle-based placement, as software updates and changes are simpler in the cloud than in the vehicle. As with all connected vehicle functions, the proposed system is dependent on mobile connectivity, which may have negative impact on availability and thus, reliability. With the extension of mobile networks and the development of 5G connectivity, availability, and reliability continues to improve.

Our research shows that the right system architecture for driving range estimation can improve the user experience significantly, by reducing latency and network usage. Hopefully, more accurate range estimation with improved user experience will drive the acceptance of battery electric vehicles.

ACKNOWLEDGEMENT

Open access funding enabled and organized by Projekt DEAL.

ORCID

Adam Thor Thorgeirsson https://orcid.org/0000-0002-7306-8693

REFERENCES

1. Eisel M, Nastjuk I, Kolbe LM. Understanding the influence of in-vehicle information systems on range stress–Insights from an electric vehicle field experiment. *Transp Res F Traffic Psychol Behav*. 2016;43:199-211.
2. Franke T, Neumann I, Bühler F, Cocron P, Krems JF. Experiencing range in an electric vehicle: understanding psychological barriers. *Appl Psychol*. 2012;61(3):368-391.
3. Smuts M, Scholtz B, Wesson J. Issues in implementing a data integration platform for electric vehicles using the Internet of Things. *IFIP Advances in Information and Communication Technology*. New York, NY: Springer International Publishing; 2019:160-177.
4. Ratner A, Alistarh D, Alonso G, et al. MLSys: the new frontier of machine learning systems. *CoRR*. 2019;abs/1904.03257. http://arxiv.org/abs/1904.03257.
5. Chalapathi GSS, Chamola V, Vaish A, Buyya R. Industrial Internet of Things (IIoT) applications of edge and fog computing: a review and future directions. *CoRR*. 2019.abs/1912.00595 1–15. http://arxiv.org/abs/1912.00595.
6. Mahmud R, Srirama SN, Ramamohanarao K, Buyya R. Quality of Experience (QoE)-aware placement of applications in Fog computing environments. *J Parall Distrib Comput*. 2019;132:190-203. https://doi.org/10.1016/j.jpdc.2018.03.004.
7. Buckl C, Camek A, Kainz G, et al. The software car: building ICT architectures for future electric vehicles. Paper presented at: Proceedings of the 2012 IEEE International Electric Vehicle Conference. Institute of Electrical and Electronics Engineers (IEEE), Greenville, SC; 2012:1-8.
8. Dizdarević J, Carpio F, Jukan A, Masip-Bruin X. A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration. *ACM Comput Surv*. 2019;51(6):116.
9. Siegel J, Erb D, Algorithms SS. Architectures: a case study in when, where and how to connect vehicles. *IEEE Intell Transp Syst Mag*. 2018;10(1):74-87. https://doi.org/10.1109/MITS.2017.2776142.

10. Tuli S, Basumatary N, Buyya REdgeLens: deep learning based object detection in integrated iot, fog and cloud computing environments. Paper presented at: Proceedings of the 2019 4th International Conference on Information Systems and Computer Networks (ISCON), Mathura, India; 2019:496-502.
11. Tuli S, Basumatary N, Gill SS, et al. HealthFog: an ensemble deep learning based smart healthcare system for automatic diagnosis of heart diseases in integrated IoT and fog computing environments. *Futur Gener Comput Syst*. 2020;104:187-200. https://doi.org/10.1016/j.future.2019.10.043.
12. Lin SY, Du Y, Ko PC, et al. Fog computing based hybrid deep learning framework in effective inspection system for smart manufacturing. *Computer Communications*. Amsterdam, Netherlands: Elsevier BV; 2020;160. https://doi.org/10.1016/j.comcom.2020.05.044.
13. Crankshaw D, Wang X, Zhou G, Franklin MJ, Gonzalez JE, Stoica I. Clipper: a low-latency online prediction serving system. Paper presented at: Proceedings of the 14th USENIX Conference on Networked Systems Design and ImplementationNSDI'17. USENIX Association; 2017:613-627; Berkeley, CA.
14. Dijkstra EW. A note on two problems in connexion with graphs. *Numer Math*. 1959;1(1):269-271.
15. Hart P, Nilsson N, Raphael B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Tran Syst Sci Cybern*. 1968;4(2):100-107. https://doi.org/10.1109/tssc.1968.300136.
16. Delling D, Sanders P, Schultes D, Wagner D. Highway hierarchies star. *The Shortest Path Problem*. Providence, Rhode Island: American Mathematical Society; 2006:141-174.
17. Bellman R. On a routing problem. *Q Appl Math*. 1958;16(1):87-90.
18. Ford L, Fulkerson D. *Flows in Networks*. Princeton, NJ: Princeton University Press; 1962.
19. Thibault L, Nunzio GD, Sciarretta A. A Unified approach for electric vehicles range maximization via eco-routing, eco-driving, and energy consumption prediction. *IEEE Trans Intell Veh*. 2018;3(4):463-475. https://doi.org/10.1109/tiv.2018.2873922.
20. Cauwer CD, Verbeke W, Mierlo JV, Coosemans TA. Model for range estimation and energy-efficient routing of electric vehicles in real-world conditions. *IEEE Trans Intell Transp Syst*. 2019;21(7):1-14. https://doi.org/10.1109/tits.2019.2918019.
21. Sun S, Zhang J, Bi J, Wang Y. A machine learning method for predicting driving range of battery electric vehicles. *J Adv Transp*. 2019;2019:1-14. https://doi.org/10.1155/2019/4109148.
22. Fukushima A, Yano T, Imahara S, Aisu H, Shimokawa Y, Shibata Y. Prediction of energy consumption for new electric vehicle models by machine learning. *IET Intell Transp Syst*. 2018;12(9):1174-1180. https://doi.org/10.1049/iet-its.2018.5169.
23. Cauwer CD, Verbeke W, Coosemans T, Faid S, Mierlo JVA. Data-driven method for energy consumption prediction and energy-efficient routing of electric vehicles in real-world conditions. *Energies*. 2017;10(5):608. https://doi.org/10.3390/en10050608.
24. Yi Z, Bauer PH. Adaptive multiresolution energy consumption prediction for electric vehicles. *IEEE Trans Veh Technol*. 2017;66(11):10515-10525.
25. Grubwinkler S, Kugler M, Lienkamp M. A system for cloud-based deviation prediction of propulsion energy consumption for EVs. Paper presented at: Proceedings of 2013 IEEE International Conference on Vehicular Electronics and Safety. Institute of Electrical and Electronics Engineers (IEEE), Dongguan, China; 2013:99-104.
26. Jayakumar A, Ingrosso F, Rizzoni G, Meyer J, Doering J. Crowd sourced energy estimation in connected vehicles. Paper presented at: Proceedings of the 2014 IEEE International Electric Vehicle Conference (IEVC). Institute of Electrical and Electronics Engineers (IEEE), Florence, Italy; 2014:1-8.
27. Scheubner S, Thorgeirsson AT, Vaillant M, Gauterin F. A stochastic range estimation algorithm for electric vehicles using traffic phase classification. *IEEE Trans Veh Technol*. 2019;68(7):6414-6428. https://doi.org/10.1109/TVT.2019.2918544.
28. Grubwinkler S, Hirschvogel M, Lienkamp M. Driver- and situation-specific impact factors for the energy prediction of EVs based on crowd-sourced speed profiles. Paper presented at: Proceedings of the 2014 IEEE Intelligent Vehicles Symposium Proceedings, Dearborn, MI; 2014:1069-1076.
29. Ferreira JC, Monteiro VDF, Afonso JL. Data mining approach for range prediction of electric vehicle. Paper presented at: Proceedings of the Conference on Future Automotive Technology-Focus Electromobility; 2012:1-15; Springer, New York, NY.
30. Lee CH, Wu CH. A novel big data modeling method for improving driving range estimation of EVs. *IEEE Access*. 2015;3:1980-1993. https://doi.org/10.1109/access.2015.2492923.
31. Lee CH, Wu CH, Chou CC, Chung XH, Zeng PW, Lin YH. A Framework for a connected electric vehicle cloud to learn drivers' behaviors. Paper presented at: Proceedings of the 2017 25th International Conference on Systems Engineering (ICSEng). Institute of Electrical and Electronics Engineers (IEEE), Las Vegas, NV; 2017:405-411.
32. Zhang Z, Huang M, Chen Y, Gao D. Big-Data Based Online State of Charge Estimation and Energy Consumption Prediction for Electric Vehicles. *SAE 2016 World Congress and Exhibition*. SAE Technical Paper Series. Warrendale, PA: SAE International; 2016.
33. Tseng CM, Chau CK, Dsouza S, Wilhelm E. A participatory sensing approach for personalized distance-to-empty prediction and green telematics. Paper presented at: Proceedings of the 2015 ACM 6th International Conference on Future Energy Systems - e-Energy; vol. 15; 2015; Bangalore, India: ACM Press.
34. Tseng CM, Chau CK. Personalized prediction of vehicle energy consumption based on participatory sensing. *IEEE Trans Intell Transp Syst*. 2017;18(11):3103-3113. https://doi.org/10.1109/tits.2017.2672880.
35. Qi X, Wu G, Boriboonsomsin K, Barth MJ. Data-driven decomposition analysis and estimation of link-level electric vehicle energy consumption under real-world traffic conditions. *Transp Res Part D: Transp Environ*. 2018;64:36-52.
36. Abboud K, Omar HA, Zhuang W. Interworking of DSRC and cellular network technologies for V2X communications: a survey. *IEEE Trans Veh Technol*. 2016;65(12):9457-9470. https://doi.org/10.1109/TVT.2016.2591558.

37. Yousefpour A, Fung C, Nguyen T, et al. All one needs to know about fog computing and related edge computing paradigms: a complete survey. *J Syst Archit*. 2019;98:289–330.
38. Brogi A, Forti S, Guerrero C, Lera I. How to place your apps in the fog: State of the art and open challenges. *Softw Pract Exper*. 2020;50(5):719-740. https://doi.org/10.1002/spe.2766.
39. Wilhelm E, Siegel J, Mayer S, et al. Cloudthink: a scalable secure platform for mirroring transportation systems in the cloud. *Transport*. 2015;30(3):320-329.
40. Milani F, Beidl C. Cloud-based vehicle functions: motivation, use-cases and classification. Paper presented at: Proceedings of the 2018 IEEE Vehicular Networking Conference (VNC). Institute of Electrical and Electronics Engineers (IEEE); 2018:1-4; Taipei, Taiwan: IEEE.
41. Ashok A, Steenkiste P, Bai F. Enabling vehicular applications using cloud services through adaptive computation offloading. Paper presented at: Proceedings of the 6th International Workshop on Mobile Cloud Computing and Services.MCS '15; 2015:1-7; New York, NY, ACM.
42. Siegel JE, Erb DC, Sarma SE. A survey of the connected vehicle landscape—architectures, enabling technologies, applications, and development areas. *IEEE Trans Intell Transp Syst*. 2018;19(8):2391-2406. https://doi.org/10.1109/TITS.2017.2749459.
43. Wu D, Liu B, Chen Z, et al. Cloud computing in electric vehicles charging control and dispatch optimization. Paper presented at: Proceedings of the 2014 IEEE International Conference on Progress in Informatics and Computing. Institute of Electrical and Electronics Engineers (IEEE), Shanghai, China; 2014:597-600.
44. Ozatay E, Onori S, Wollaeger J, et al. Cloud-based velocity profile optimization for everyday driving: a dynamic-programming-based solution. *IEEE Trans Intell Transp Syst*. 2014;15(6):2491-2505. https://doi.org/10.1109/tits.2014.2319812.
45. Saini M, Alam KM, Guo H, Alelaiwi A, Saddik AE. InCloud: a cloud-based middleware for vehicular infotainment systems. *Multimed Tools Appl*. 2016;76(9):11621-11649. https://doi.org/10.1007/s11042-015-3158-4.
46. Yaseen MU, Anjum A, Farid M, Antonopoulos N. Cloud-based video analytics using convolutional neural networks. *Softw Pract Exper*. 2018;49(4):565-583. https://doi.org/10.1002/spe.2636.
47. Henrickson K, Rodrigues F, Pereira FC. Data Preparation. *Mobility Patterns, Big Data and Transport Analytics*. Amsterdam, Netherlands: Elsevier; 2019:73-106.
48. Svorobej S, Endo PT, Bendechache M, et al. Simulating fog and edge computing scenarios: an overview and research challenges. *Future Internet*. 2019;11(3):55. https://doi.org/10.3390/fi11030055.
49. Calheiros RN, Ranjan R, Beloglazov A, Rose CAFD, Buyya R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw Pract Exper*. 2010;41(1):23-50. https://doi.org/10.1002/spe.995.
50. Abreu DP, Velasquez K, Curado M, Monteiro E. A comparative analysis of simulators for the cloud to fog continuum. *Simul Model Pract Theory*. 2020;101:102029. https://doi.org/10.1016/j.simpat.2019.102029.
51. Gupta H, Dastjerdi AV, Ghosh SK, Buyya R. iFogSim: a toolkit for modeling and simulation of resource management techniques in the Internet of Things, edge and fog computing environments. *Softw Pract Exper*. 2017;47(9):1275-1296. https://doi.org/10.1002/spe.2509.
52. Ghosal A, Giusto P, Sinha P, Osella M, D'Ambrosio J, Zeng H. Metrics for Evaluating Electronic Control System Architecture Alternatives. *SAE 2010 World Congress & Exhibition*. SAE Technical Paper Series. Warrendale, PA: SAE International; 2010.
53. Hasenburg J, Grambow M, Grünewald E, Huk S, Bermbach D. MockFog: emulating fog computing infrastructure in the cloud. Paper presented at: Proceedings of the 1st IEEE International Conference on Fog Computing. Institute of Electrical and Electronics Engineers (IEEE), Prague, Czech Republic; 2019.
54. Jain R. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Hoboken, NJ: John Wiley & Sons; 1990.
55. Wang L, Zhan J, Gao W, et al. BOPS, Not FLOPS! a new metric, measuring tool, and roofline performance model for datacenter computing. *CoRR*. 2018;abs/1801.09212.abs/1801.09212. http://arxiv.org/abs/1801.09212.
56. Kapsalis A, Kasnesis P, Venieris IS, Kaklamani DI, Patrikakis CZA. Cooperative fog approach for effective workload balancing. *IEEE Cloud Comput*. 2017;4(2):36-45. https://doi.org/10.1109/MCC.2017.25.
57. O'Loughlin J, Gillam L. Good performance metrics for cloud service brokers. Paper presented at: Proceedings of the 5th International Conference on Cloud Computing, GRIDs, and Virtualization; Citeseer, Venice, Italy; 2014:64-69.
58. Pesovic U, Jovanovic Z, Randjic S, Markovic D. Benchmarking performance and energy efficiency of microprocessors for wireless sensor network applications. Paper presented at: Proceedings of the 2012 Proceedings of the 35th International Convention MIPRO. Institute of Electrical and Electronics Engineers (IEEE), Opatija, Croatia; 2012:743-747.
59. Brebner P, Liu A. Modeling cloud cost and performance. Paper presented at: Proceedings of the International Conference on Cloud Computing & Virtualization 2010 CCV 2010. Global Science and Technology Forum, Singapore; 2010
60. Lukasiewycz M, Steinhorst S, Andalam S, et al. System architecture and software design for electric vehicles. Paper presented at: Proceedings of the 2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC). Institute of Electrical and Electronics Engineers (IEEE); 2013:1-6.
61. Peng C, Zhang Q, Tang C. Improved TLS handshake protocols using identity-based cryptography. Paper presented at: Proceedings of the 2009 International Symposium on Information Engineering and Electronic Commerce. Institute of Electrical and Electronics Engineers (IEEE), Ternopil, Ukraine; 2009:135-139.
62. Sedgewick R, Vitter JS. Shortest paths in euclidean graphs. *Algorithmica*. 1986;1(1-4):31-48.
63. 9th DIMACS implementation challenge: shortest paths; 2006. Full USA road graph files. http://users.diag.uniroma1.it/challenge9/download.shtml. Accessed September 25, 2019.

64. KIT - ITI Algorithmik I - PTV-DIMACS-Europe-Graph; 2014. https://i11www.iti.kit.edu/resources/roadgraphs.php. Accessed September 25, 2019.
65. HERE Global B.V Routing API developer's guide. http://developer.here.com/documentation/routing/dev_guide/topics/resources.html. Accessed August 25, 2019.
66. GoingElectric Stromtankstellenverzeichnis. https://www.goingelectric.de/stromtankstellen/. Accessed September 25, 2019.
67. Bottou L, Cun YL. Large scale online learning. Paper presented at: Proceedings of the 16th International Conference on Neural Information Processing Systems.NIPS'03; 2003:217-224; Cambridge, MA, MIT Press.
68. McMahan HB, Moore E, Ramage D, Arcas YBA. Federated learning of deep networks using model averaging. *CoRR*. 2016;abs/1602.05629.
69. Benczúr AA, Kocsis L, Pálovics R. Online machine learning in big data streams. *CoRR*. 2018;abs/1802.05872.
70. Bottou L. Stochastic gradient descent tricks. *Neural Networks: Tricks of the Trade*. New York, NY: Springer; 2012:421-436.
71. Jothimurugesan E, Tahmasbi A, Gibbons P, Tirthapura S. Variance-reduced stochastic gradient descent on streaming data. In: Bengio S, Wallach H, Larochelle H, Grauman K, Cesa-Bianchi N, Garnett R, eds. *Advances in Neural Information Processing Systems*. Vol 31. Red Hook, NY: Curran Associates, Inc; 2018:9906-9915.
72. Frostig R, Ge R, Kakade SM, Sidford A. Competing with the empirical risk minimizer in a single pass. In: Grünwald P, Hazan E, Kale S, eds. *Proceedings of The 28th Conference on Learning Theory. 40 of Proceedings of Machine Learning Research*. Paris, France: PMLR; 2015:728-763.
73. Konecný J. Stochastic, distributed and federated optimization for machine learning. *CoRR*. 2017;abs/1707.01155.
74. Tuli S, Mahmud R, Tuli S, Buyya R. Fogbus: a blockchain-based lightweight framework for edge and fog computing. *J Syst Softw*. 2019;154:22-36.
75. Mancini R, Tuli S, Cucinotta T, Buyya R iGateLink: a gateway library for linking IoT, edge, fog and cloud computing environments; 2019. arXiv preprint arXiv:1911.08413.
76. Calheiros RN, Vecchiola C, Karunamoorthy D, Buyya R. The Aneka platform and QoS-driven resource provisioning for elastic applications on hybrid Clouds. *Futur Gener Comput Syst*. 2012;28(6):861-870. https://doi.org/10.1016/j.future.2011.07.005.

How to cite this article: Thorgeirsson AT, Vaillant M, Scheubner S, Gauterin F. Evaluating system architectures for driving range estimation and charge planning for electric vehicles. *Softw Pract Exper*. 2020;1–19. https://doi.org/10.1002/spe.2914