Automatic Distributed Deep Learning Using Resource-Constrained Edge Devices

Alberto Gutierrez-Torre, Kiyana Bahadori, Shuja-ur-Rehman Baig, Waheed Iqbal, Tullio Vardanega, *Member, IEEE*, Josep Lluís Berral, *Member, IEEE*, and David Carrera, *Member, IEEE*

***Abstract*—Processing data generated at high volume and speed from the Internet of Things, smart cities, domotic, intelligent surveillance, and e-healthcare systems require efficient data processing and analytics services at the Edge to reduce the latency and response time of the applications. The fog computing edge infrastructure consists of devices with limited computing, memory, and bandwidth resources, which challenge the construction of predictive analytics solutions that require resource-intensive tasks for training machine learning models. In this work, we focus on the development of predictive analytics for urban traffic. Our solution is based on deep learning techniques localized in the Edge, where computing devices have very limited computational resources. We present an innovative method for efficiently training the gated recurrent-units (GRUs) across available resource-constrained CPU and GPU Edge devices. Our solution employs distributed GRU model learning and dynamically stops the training process to utilize the low-power and resource-constrained Edge devices while ensuring good estimation accuracy effectively. The proposed solution was extensively evaluated using low-powered ARM-based devices, including Raspberry Pi v3 and the low-powered GPU-enabled device NVIDIA Jetson Nano, and also compared them with Single-CPU Intel Xeon machines. For the evaluation experiments, we used real-world Floating Car Data. The experiments show that the proposed solution delivers excellent prediction accuracy and computational performance on the Edge when compared to the baseline methods.**

***Index Terms*—Analytics, big data, cloud computing, edge computing, fog computing, Internet of Things (IoT), resource management.**

I. Introduction

THE Internet of Things (IoT) is attracting significant interest from both academia and industry. The potential benefit of applying IoT paradigms to smart cities and healthcare service scenarios suggests to design new architectures for infrastructure, platforms, and services. The issue with more traditional approaches rises from the inherent limitations in connectivity and computing power of Edge devices and dynamic networks. Those IoT architectures are usually composed of real-time sensor-based monitoring systems and actuators running in different locations, connected to data aggregation applications or data-warehouses through dynamic networks (such as 5G, Wi-Fi, or wired Internet).

The main feature of the Cloud is to provide extremely scalable resources to service applications from a remote datacenter. In contrast, emerging scenarios, such as the IoT, smart cities, domotic, intelligent surveillance, and e-healthcare usually require proximity and quick reaction time while generating massive amounts of data transmitted to the analytics applications. Fog computing is more attractive for such demand [1]. Fog computing takes the computation to the Edge, moving data processing close to the sources, and reducing data to synthesized volumes to be transmitted north-bound to the Cloud, as shown in Fig. 1. Additionally, when the Edge services depend only on local data, the service can be provided without using Cloud services. Several fields can benefit from this kind of architecture, specifically Oil & Gas [2], power grid systems [3], smart cities, smart industries, and IoT applications [4]. In these environments, local analytics are required as they need a low-latency QoS [5]. Moreover, backhaul connectivity might fail [6] as the network might not be as reliable as wanted due to extreme conditions. Given the importance of exploiting the data at the Edge level, considerable research effort was devoted to establish a common framework to cope systematically and effectively with the restrictions proposed by this kind of environment [7].

The compute-intensive nature of the training of machine learning (ML) models has so far caused that all the processing is done in Cloud data centers. This typical strategy, to push the data to the cloud and then training the ML models, has the advantage of using powerful computing machines. However, this strategy has several drawbacks: it adds a cost of additional network dependency, increases latency, and moves the processing away from the data producers. In contrast, using limited computing power available at Fog nodes is interesting for training ML models efficiently. Recent work shows the importance of training an ML model on the Edge infrastructure. For example, Plastiras *et al.* [5] showed the importance of doing the computation for training deep learning models on Fog nodes for computer vision tasks like object detection.

Manuscript received 12 May 2021; revised 25 June 2021; accepted 14 July 2021. Date of publication 21 July 2021; date of current version 8 August 2022. This work was supported in part by the Spanish Government under Contract PID2019-107255GB; in part by the Generalitat de Catalunya under Contract 2014-SGR-1051; in part by the University of Padua; and in part by the Severo Ochoa CoE under Grant SEV-2015-0493-16-5. *(Corresponding author: Alberto Gutierrez-Torre.)*

Alberto Gutierrez-Torre, Josep Lluís Berral, and David Carrera are with the Department of Computer Sciences, Barcelona Supercomputing Center, 08034 Barcelona, Spain, and also with the Computer Architecture Department, BarcelonaTech-Technical University of Catalonia, 08034 Barcelona, Spain (e-mail: alberto.gutierrez@bsc.es; josep.berral@bsc.es; david.carrera@bsc.es).

Kiyana Bahadori and Tullio Vardanega are with the Department of Mathematics, University of Padua, 35122 Padua, Italy (e-mail: bahadorikiana@gmail.com; tullio.vardanega@unipd.it).

Shuja-ur-Rehman Baig and Waheed Iqbal are with the College of Information Technology, University of the Punjab, Lahore 54590, Pakistan (e-mail: shuja@pucit.edu.pk; waheed.iqbal@pucit.edu.pk).

Digital Object Identifier 10.1109/JIOT.2021.3098973

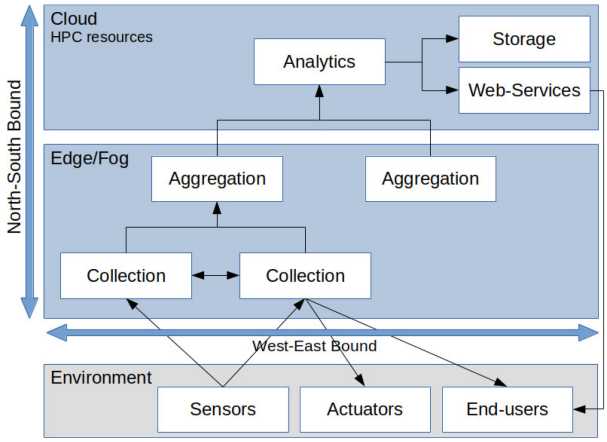

Fig. 1. Edge-cloud aggregation schema, with environment actors (sensors, actuators, and users).

The authors remark the importance of privacy, performance, latency, and power efficiency on this kind of application, which can be a perfect fit for Edge and Fog Computing.

In this work, we investigate the use of Fog devices to train deep learning models by distributing the training on the available devices at the Fog environment intelligently. Our proposed solution takes benefit of already active devices aggregating or collecting data instead of employing additional Cloud resources. This also reduces network communication and protects services from network disruptions by keeping them autonomous on the edge. This concept extends the work by Pérez *et al.* [6], where local models in the Edge level can be independent of global ones. This way, the architecture is resilient against backhaul network interruptions. The model synchronization can be delayed to the moment when network is available.

We present a system to automatically distribute the time-consuming task of training deep learning models on a Fog computing network consisting of low-powered and resource-constrained computing devices. The proposed approach is based on federated learning (FL), which leverages the work of McMahan *et al.* [8] and Bonawitz *et al.* [9]. The proposed solution automates part of the deep learning process for selecting appropriate parameters for the model to reduce the training time while maintaining the model accuracy for the validation data set. We extensively evaluate the proposed system using a road traffic analytics scenario designed for city-wide traffic modeling and prediction running on the Fog computing paradigm. The proposed methodology can make use of any kind of neural network (NN) by distributing the training on Fog devices. In particular, we use gated recurrent unit (GRU) NNs to model the traffic behavior to produce short-term/medium-term traffic predictions following the FL principles. Our evaluation investigates different data aggregation levels, different levels of data processing parallelism, time requirements for achieving suitable accuracy levels for models, and suitability for real-time applications in the Edge. Our evaluation is based on real traffic logs from one week of floating car data (FCD) in Barcelona. The data were provided by one of the largest road-assistance companies in Spain and comprises thousands of vehicles. This approach is tested in a Smart City setting; however, the same approach can be used for other fields such as Oil & Gas, where distributed learning is required or desirable. Moreover, one appealing domain to apply our proposed solution is the healthcare industry, where patient data are collected through IoT devices and required to process locally without sending remote locations for privacy and security concerns.

The experimental results show that predictive analytics requiring complex ML mechanisms like GRUs can be performed cost effectively on the Fog nodes without using expensive Cloud resources. Additionally, compared to prediction methods previously used in other studies, we show that GRUs achieves good accuracy results with constrained training time in comparison using state-of-the-art methods [i.e., conditional restricted Boltzmann machines (CRBMs)]. Even though the modeling process is split to reduce training time, the distributed model shows a stable behavior when modifying training hyperparameters. The research contributions of this article are as follows.

1) A system for distributed modeling for city-wide applications using the Fog computing paradigm for predictive analytics using low-powered and resource-constrained devices.
2) A mechanism to automate run-time decisions for stopping training processes when accuracy levels are reliable for DNNs.
3) Evaluation and comparison of time required to model the DNN using the proposed solution on Fog (low-power and resource-constrained) versus Cloud (high-performance) environments.
4) A comparative analysis of resource usage versus accuracy on training models for real FCD compared with the existing baseline methods.

This article is organized as follows: Section II reviews the background and motivation. Section III illustrates the proposed architecture. Section IV describes our approach methodology. Section V shows the evaluation, and Section VI provides concluding remarks and future challenges.

II. State of the Art

A. Background and Motivation

The Cloud has been widely used to address the emerging challenges of big data analysis in many smart city ecosystems, such as smart houses, smart lighting, and video surveillance [10]–[12]. However, IoT scenarios usually require low latency between sensors/actuators and usually there are scarce computing resources. These restrictions avoiding unnecessary north–south bound communication of data that can be processed on the Edge or intermediate nodes [6]. Location awareness is also a must in several Smart Cities' IoT architectures providing immediate in-place services. As IoT services in Smart Cities are being increasingly used, Cloud services alone can hardly satisfy the mentioned requirements of this ecosystem.

Fog computing, the paradigm combining the Edge and Cloud capabilities, can handle the significant data treatment,

including acquisition, aggregation, analytics, and preprocessing while reducing transportation and storage, even balancing computation power among intermediate nodes [13].

In addition, transforming these data into actionable knowledge and adapting to changing dynamics of modern cities require *intelligent* modeling techniques not only accurate but adaptive. ML techniques enable *smartness* in Smart Cities by modeling, predicting, and extracting useful information from collected data, through advanced statistics and artificial intelligence algorithms. Deep Learning, a ML subfield based on multilayer neuronal networks, is becoming an important tool to city-modeling challenges across many areas, such as forecasting [14], self-driving research [15], image processing [16], [17], or object recognition [18], useful to manage public services, detect hazardous scenarios, or to guide emergency services among others.

However, the increasing amount of data to be processed, along with the computational demands of sufficiently accurate NN algorithms, have led to bigger computational and memory resource requirements. Accelerating NNs training to competitive accuracy within a sufficiently short time is a major challenge that may lead to increase computational demands. Seeking solutions that assure scalable and efficient learning has given rise to the notion of "distributed ML." FL [8] is a promising solution when both data and resources are scattered along in the architecture, with the added challenge of the near impossibility of having all data in the same place, and the cost of constantly offloading computation to the Cloud.

FL aims at distributing the data or, as in the case of Edge computing, keeping the data near where they are produced [8], [9]. This solution can be understood as allowing the Edge devices, the clients, to produce a predictive model with their own local data, and then coordinate with a central node, the server, for model merging. In particular, this is interesting in the contexts where data privacy is an issue as in the work of McMahan *et al.* [8], as the only data exchanged between the data producers and the central server are the weights, i.e., the configuration, of the NN. On the other hand, there have been efforts like in the work of Hu *et al.* [19] that focus on having a model that works properly on both sides, client, and server. Moreover, it has been proved stochastic gradient descent (SGD) converges in this scenario [20], proving the suitability of NNs for this particular task. This approach brings about properties that are desirable for Edge Computing architectures, like the ability to keep on working without network connectivity when the system fails [6].

Even though the methodology per se is already available, there still is a knowledge gap regarding the actual applicability of FL on a Fog architecture using low-powered devices. This work aims to fill this gap, applying the methodologies described in the following sections.

B. Related Work

The exponential growth of the IoT, caused by the opportunity of leveraging smart devices in generalized enterprise settings, motivates the quest for novel approaches to develop a deep learning system that can scale to very large models and large data sets. However, training to competitive accuracy within a sufficiently short time span, for large and complex networks together with huge data sets is especially challenging in Edge/Fog nodes at the present state of the art.

A significant amount of effort and research has been devoted to tackling the challenge of training huge data sets through building large models with more parameters and parallelization or distribution methods based on the Cloud computing infrastructure. For example, Google implemented a distributed framework for training NNs over central processing unit (CPU) based on the DistBelief framework [21], [22], which makes use of both model parallelism and data parallelism. This model has also proved useful for computer vision problems, achieving state-of-the-art performance on a computer vision benchmark with 14 millions of images.

To scale up the training phase of learning, researchers utilize accelerators such as a single or cluster of graphics processing units (GPUs) [23], [24]. Recently, Facebook [25] announced achieving 90% scaling efficiency in training visual recognition model, using data parallelism combined with the use of GPUs.

Hong *et al.* [26] proposed a fog-based opportunistic spatiotemporal event processing system to meet the latency requirement. Their system predicts future query regions for moving consumers, and starts the event processing early to make timely information available when consumers reach the future locations. Yu *et al.* [27] proposed a deep reinforcement learning-based system that is able to share execution of tasks in Edge nodes taking into account the battery, quality of service, and other details.

Works such as Marchisio *et al.* [28] study how to perform ML inference in ultralow-powered devices, and reviewed the usage of NNs with this kind of device. This approach minimizes both power usage and hardware costs. Sudharsan *et al.* [29] proposed a methodology to train a kind of Convolutional neural network (CNN) and then adapt it to run in different microcontroller units (MCUs) to do prediction. Their approach reduces the size of the trained network to the 10% of the original. In the same direction, TinyML [30] enables training an NN with TensorFlow and then converts it to which can be run using TensorFlow Lite on ultralow power MCUs. Neither of these approaches handle training on the device, but other approaches, such as Neuro.ZERO [31], enable training on the device by means of hardware acceleration. However, FL has yet to be covered on this kind of setup with MCUs, so that it enables to train different models and average the model configuration among nodes.

To the best of our knowledge, there currently is no evaluation of this kind of problem with FL using recurrent neural networks (RNNs) with server-class hardware and low-powered devices. Moreover, mechanisms are needed to stop training as soon as a reliable-enough model is obtained. We believe that FL distributed learning can be highly beneficial for data analytics over scenarios such as smart cities.

III. Architecture: Floating Car Data Processing Over Edge

This section presents our proposed architecture for processing FCD using the Edge computing infrastructure. We explain

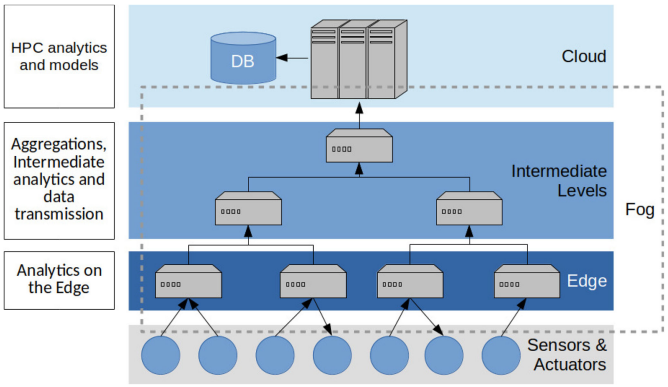

Fig. 2. Schema of the fog infrastructure, from edge to cloud.

the Edge computing network, FCD, and data processing pipeline in the following sections.

A. Edge Computing Networks

Edge computing networks are based on architectures where sensors collect data from nearby cars, users, and equipment and send them to the computing nodes within proximity. Such nodes are low powered with limited resources to perform complex analytics; therefore, the data are pushed to the remote Cloud for processing using sophisticated and powerful hardware. The "Fog" is that part of the architecture embracing Edge nodes receiving data from sensors, Intermediate nodes performing intermediate data aggregation, and Cloud APIs receiving data to be processed and stored, extending the Cloud paradigm [32]. Fig. 2 shows a Fog infrastructure, with near-data nodes on the Edge, intermediate nodes with medium power to preprocess aggregate or localized data, and the Cloud.

Current devices on the Edge are specially designed to consume low power, produce low throughput, and offer low capabilities, such as Raspberry Pi and NVIDIA Jetson. These devices with Edge computing have been recently considered a good solution for smart city image processing challenges [33], showing that industry and public administration are interested in adopting the approach of using low-powered and resource-constrained devices for real scenarios. NVIDIA Jetson [34] is a low-power and small-form factor computer similar to Raspberry Pi (ARM processor). It is a Linux-enabled machine that is equipped with an embedded NVIDIA low-power GPU and the CUDA framework, which can be used to train deep learning calculations. However, a single Jetson device is not sufficient to perform the deep learning training independently for a large data set.

In this article, we propose using the low-power and resource-constrained devices to train the deep learning model at the Edge, close to the users and data, by distributing the training on multiple devices and enabling the Edge efficient analytics. In our system, sensors collect data and transmit to the Edge nodes, and analytics are performed on the Edge nodes instead of offloading the complex analytics tasks to the Cloud.

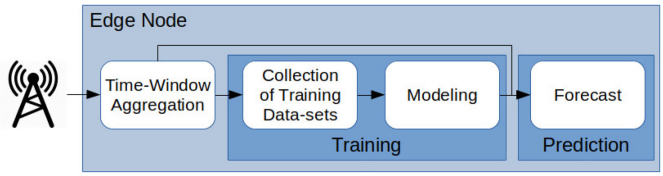

Fig. 3. Pipeline of time-window aggregation, learning, and prediction.

B. Floating Car Data

FCD represents geolocalized timestamped data of moving vehicles, collected and analyzed for various applications, including smart cities, traffic engineering, and traffic management. Typically, FCD is received through antennas deployed in the town representing a large urban zone, a localized neighborhood, a street, or a street segment, depending on which granularity is required for the specific application. These data are provided to the Edge analytics indicating the received timestamp for each vehicle transmission and its speed. Data like vehicle positions, e.g., global positioning system (GPS), are not provided for privacy and security reasons; only the Edge node position is provided.

The FCD arrives asynchronously to our Edge nodes and is aggregated periodically into summaries of traffic information, i.e., the average speed of vehicles surrounding the node (in Km/hour) and vehicles' count, considering that vehicles will be reported once for each aggregation window time. Considering the aggregation of a 1-min interval as lower bound, we aggregate the incoming data into data entries containing latitude, longitude, number of cars, speed average and timestamp. Before performing the analytics, the Edge nodes independently collect and aggregate the FCD into a specific time interval.

The 1-min aggregation data are only the base for larger aggregations, as traffic time series can be aggregated from minutes to hours to days because of its periodic pattern in time. While large aggregations can be easily predicted due to this periodicity, smaller aggregations can be more challenging. For the validation experiments, in Section V, we test different levels of time aggregation varying from 5-min to 1-h intervals for training analytic models.

In this work, we used a week-worth of real FCD from the city of Barcelona, Spain, provided by one of the largest road-assistance companies in the country.

C. Data Analytics Pipelines

Whenever FCD is detected through antenna sensors, it is transmitted to the nearest Edge node. The data aggregation using a specific time interval is performed at the Edge node. For each aggregation, the timestamp is added to the FCD record for building a time-series data set. The FCD time-series data set is used to model the traffic behavior for forecasting and analytics purposes. Fig. 3 shows the FCD collection, modeling, and forecasting pipeline. In our system, we performed distributed model training, explained in Section IV-C, using low-powered Edge nodes.

For a global-scale prediction, the aggregated data and created models on the Edge can be pushed to the Cloud for storage and further analysis. Moreover, the aggregated data from individual Edge nodes can be passed to intermediate nodes for training a generalized model, as depicted in Fig. 2. However, in previous works [6], we observed that local models fit local scenarios better than general models in the Cloud, avoiding intense communication interruption problems.

IV. Methodology

This section presents our proposed methodology for FCD time-series forecasting, automation for the training process, and distributed model learning on the Edge.

A. Traffic Forecasting

We can see the FCD time-series data set as a matrix of size number of time window elements times input features. The forecasting problem targets the prediction of two variables: 1) the number of cars and 2) the average speed of the following time step $(t + 1)$ using the previous d elements from the time window, where d is our *delay* or memory window. As the time window is a whole aggregation period, the goal is to predict the next period of traffic information. We used GRU networks [35] to train the forecasting model. Given the capabilities of the GRUs, it is possible to forecast far from $t + 1$, as GRUs are shown to be capable of medium-term forecasting in many scenarios. GRUs are *generative*, and can generate predictions by using their last prediction and status as input/memory for the next prediction. In our problem, we are predicting from $t + 1$ up to $t + N$, where N is the size of testing data set in the experiments (approximately 1 day in the following experiments).

B. Training Process Automation

ML model training with good accuracy is controlled by a "Training versus Validation" process. In NNs, this process is used to decide when to stop the iterative process. Training data are divided into two batches: 1) "training" and 2) "validation" data sets. The longer NNs trains with the whole training data set (each period is called an epoch), the more fitted the model is expected to be, but only to the training data, which can lead to *overfit*. To mitigate this risk, the validation data set is predicted at each *epoch*, allowing to check how the model behaves with "nontraining" data. While the error in training data decreases at each epoch, error in validation data decreases until the point of *overfitting* and increases from there, as Fig. 5 illustrates. That point is considered the "bouncing point," and data-scientists would manually stop iterating at that point. But for nonstable data, as we face with FCD, validation can differ enough from training data on certain occasions, and those expected behaviors may not be encountered in during training. Hence, the time at which the process should stop iterating must be decided automatically.

To detect the bouncing point to stop the deep networks' training process for minimizing the training time while achieving good accuracy, we propose Algorithm 1. The algorithm shows the technique for fixing p (point of bounce) dynamically, using the error on training and validation, previously smoothing both sequences to facilitate treating *wavy* sequences, using an locally estimated scatterplot smoothing (LOESS) curve fitting method [36]. Among other algorithms with the same objective, LOESS was selected for its simplicity and speed. It is well fitted for low-powered devices and, for our use case, it achieved good results for low computational cost.

Algorithm 1 Detecting GRU Training Cutting-Point (Epoch) p From Training and Validation Error

```
Result: p point of bounce/intersect/convergence/minimum,
        prioritizing error on validation over error on training
smooth_tr, smooth_val ← loess_smooth(error_tr, error_val)
if exists_bounce(smooth_val) then
    return bounce_p(smooth_val)
else
    if exists_intersect(smooth_tr, smooth_val) then
        return intersect_p(smooth_tr, smooth_val)
    else
        if exists_bounce(smooth_tr) then
            return bounce_p(smooth_tr)
        else
            if converges(smooth_tr, min_threshold) then
                p ← converging_p(smooth_tr, min_threshold)
                return min(p, minimum_p)
            else
                return minimum_p
            end
        end
    end
end
```

The process of finding p implies running for a given amount of epochs, to find the trend and detect the bouncing, intersection, or convergence points. This process can be substituted by more sophisticated methods that can be applied online, although the set of rules we have devised can be used to determine p once for a given amount of data, while retaining p for future models.

C. Distributed Model Training

Computing devices at the Edge are low-powered and very limited in terms of the number of cores and storage. The available processing power is mostly used to receive and transmit data from sensors to Cloud; remaining computational resources can perform aggregation and modeling processes.

In our proposed distributed model training solution, we assume the availability of single CPU/GPU processors on each available Edge device. To distribute training across workers (available Edge devices), we partition the training–validation data set and send it over to the available workers. Each worker creates a model from its subset and validates it. At that point, all submodels are joined in the initiator Edge node and merged following the FL principles [8]. The resulting aggregated model can either work better for the dilution of noise among submodels or do worse due to overfitting each submodel to its subset. For this reason and good practice, the

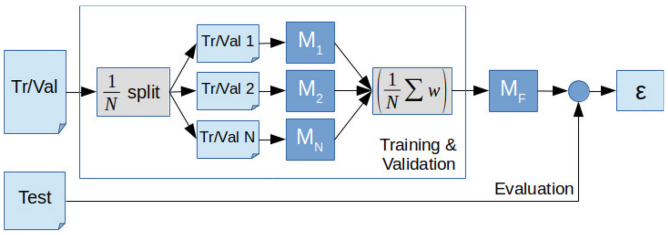

Fig. 4. Distributed modeling technique for training and testing phases. Data split among N workers, creating N models to be merged, creating the final model to be evaluated.

aggregation model is evaluated on the test data set in the initiator Edge node. Fig. 4 shows the process of distributed training, merging, and evaluation. This process is done in a off-line fashion using the whole data set but it also could be done receiving a stream of continuous data, retraining the networks for new batches, and synchronizing after each epoch.

In our test case, we have split the data evenly between nodes using the data coordinates to split regions. Then inside each node, the split of training versus validation is done following a 80%–20% ratio between training versus validation, for every subset. Each worker splits its data to train and validate its model. The test set (a 20% of the total data) is kept for the aggregated model for evaluating the final model.

V. Experimental Evaluation

The proposed approach was evaluated with several experiments designed to test the model learning and accuracy on the previously mentioned real FCD set with one week worth of data. We compare two approaches: a single learning model that learns from all the data versus multiple local models that are synchronized in a FL framework. We compare the effectiveness of the different learning model configurations in low-powered and resource-constrained Edge devices. The implementation and evaluation of the proposed solution were performed using TensorFlow and Keras frameworks over *R*. Note that *R* can run on any Linux-enabled device and that the core of the code is built on top of TensorFlow, which is efficiently implemented in C++. The infrastructure to run and measure training times corresponded to a server-class single thread Xeon processor for comparison experiments among different training configurations and two low-powered devices: a Raspberry Pi 3 (ARM processor), and an NVIDIA Jetson Nano (ARM processor + NVIDIA GPU) to cover both CPU and GPU settings.

Our experiments addressed the following evaluation aspects.

1) The effects of training the GRU with a different number of hidden units and a different number of epochs, and check the usefulness of determining a stop-point p dynamically using the presented set of rules versus fixing a large enough p *a-priori*.
2) The comparison and tradeoff between training epochs versus hidden units versus resulting error versus level of time aggregation.
3) The effects of distributing the training process among N different processors, considering a low range for N matching the dimensions of common low-power devices.

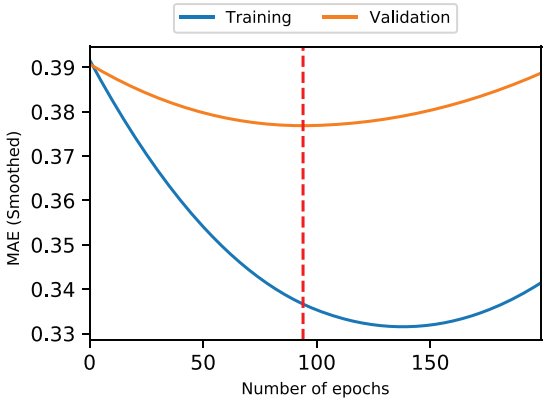

Fig. 5. Zoomed representation of the smoothed MAE as used in Algorithm 1. Observe that at 94 epochs the validation data bounced back, selecting it as a training stop-point.

4) The capability of running the presented methods on low-powered devices, i.e., Raspberry Pi v3 and NVIDIA Jetson Nano.

A. Hyperparameter Identification

We evaluated the capability of deep neural network (DNN) to learn the target time series of *Volume* (*Cars*) and *Average Speed* (*Speed*) of traffic data using the proposed algorithm (Algorithm 1) for identifying appropriate epochs. We also ran a grid search-like strategy for determining the number of hidden units and time aggregation levels, i.e., periods in which data are aggregated into a single value for the proposed solution.

In this experiment, we trained a single model using the entire training data set and our proposed Algorithm 1 to automatically identify the number of epochs with higher accuracy. We evaluated various settings for hidden units and aggregation levels. In the case of two hidden units and 20-min aggregation, our algorithm identified 94 epochs as a bouncing point, as shown in Fig. 5. To compare the proposed solution for identifying epochs, we have performed additional experiments and manually tuned the epochs from 10 to 200. Note that the stop decision is made with the validation data, as doing so with training data could lead to overfitted models.

Fig. 6 shows the error distribution for different training epochs over the *Test* data set and the dynamic stop-training point using Algorithm 1. For the volume of cars, learning seems easy as we observed mean absolute error (MAE) between 1–1.3. Predicting the speed of cars becomes more complex as we yield volatile error between 3–5.5 in km/h (the variability of the traffic speed on those data sets is already known from previous works [6]). While most of the training is done on the first few epochs of the different tested NN configurations, identifying the automatic stop-training point becomes conservative with respect to the best option, but performing almost as good as the optimal.

The experimental results reported in Fig. 6 show the difficulty of establishing a set of rules that match every single training-validation scenario. Selecting the best number of epochs is still an open problem whose solution can be automated with more complex mechanisms. However, for the

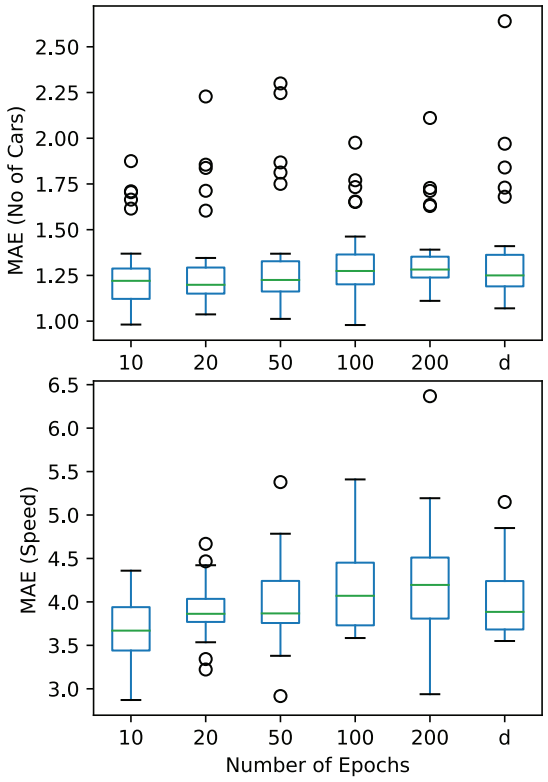

Fig. 6. Comparison of MAE for various static number of epochs with a dynamic number of epochs for estimating the number of cars and average speed. Here, *d* represents a dynamic number of epochs.

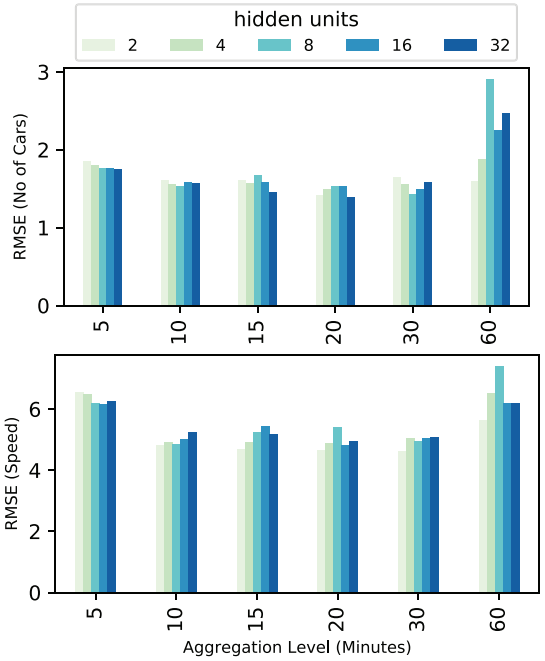

Fig. 7. Error versus hidden units versus time aggregation for number of cars and speed estimation with dynamic epoch value *d*.

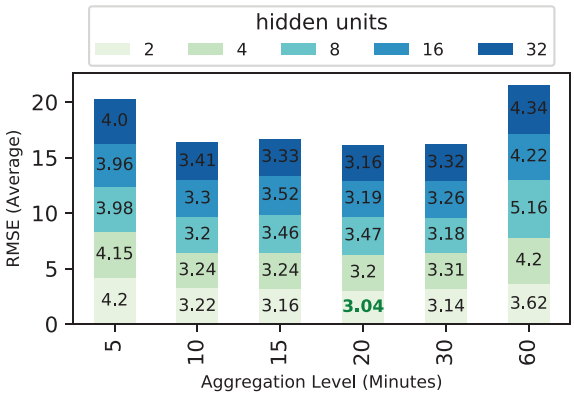

Fig. 8. Comparison of Average RMSE using different number of hidden units with various aggregation levels (*d* epochs).

current scenario, where quick decisions must be made, the presented algorithm becomes an adequate solution. Therefore, from now on, the results shown are the ones using the *d* value for epochs.

Fig. 7 shows the root mean square error (RMSE) for estimating the number of cars on test data for 2, 4, 8, 16, and 32 hidden units with 5, 10, 15, 20, 30, and 60 min aggregation levels. We observed the aggregation yields stable behavior until 30-min aggregations as the RMSE remains under 2. However, at the 60-min aggregation level, we observed a significant increase in the estimation error. This is because, with a higher level of data aggregation, the underlying fine-grained details are hidden, and the model cannot learn from data accurately. We observed the effect of changing the number of hidden units does not have any significant effect on accuracy. The average RMSE of estimating the number of cars remains between 1 and 2 except at the aggregation level of 60 min.

Fig. 7 shows the RMSE for estimating the speed of cars on test data for 2, 4, 8, 16, and 32 hidden units with 5, 10, 15, 20, 30, and 60 min aggregation levels. We observed the high error for aggregation levels 5 and 60; however, it remains similar for other aggregation levels. We do not observe any noticeable accuracy gain for using a different number of hidden units. The average RMSE of estimating cars' speed remains between 4.5 and 5.5 except aggregation levels of 5 and 60 min.

This set of experiments allowed us to determine the appropriate level of aggregation and the hidden units to determine to be used in the final model. We computed the average RMSE of speed and number of cars on test data for (2, 4, 8, 16, 32) hidden units with (5, 10, 15, 20, 30, 60) minutes aggregation levels. Fig. 8 shows the average RMSE for estimating the speed and number of cars. Each aggregation level has its optimal number of hidden units, meaning that there is no optimal configuration able to deal with all levels of aggregation, a desirable state allowing us to decide the precision of the time interval. Determining a time interval where we can trust predictions the most, we observed that the two hidden units with a 20-min aggregation level yield the minimum error compared to the other configurations. Therefore, in the rest of the experiments, we used two hidden units and 20 min of aggregation level.

B. Comparison of Single and Distributed Models

This experiment evaluated the proposed distributed model learning effectiveness and compared it with a single standalone

TABLE I
COMPARISON OF ERROR AND TRAINING TIME WITH FIXED NUMBER OF EPOCHS (FIXED EP) AND PROPOSED NUMBER OF EPOCHS (PROP EP)

Model	RMSE (Cars)		RMSE (Speed)		Time (Sec)	
	Fixed Ep	Prop. Ep	Fixed Ep	Prop. Ep	Fixed Ep	Prop. Ep
N1	1.40	1.40	4.62	4.62	233.71	233.71
N2	3.25	2.05	7.84	5.45	237.79	118.32
N3	6.25	3.29	6.83	4.64	229.53	84.69

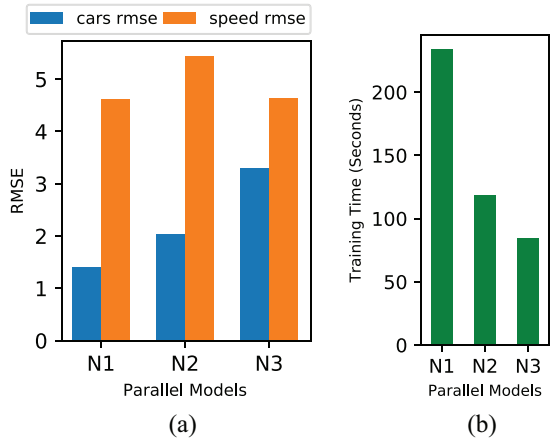

Fig. 9. Comparison of RMSE versus training time for different parallel models (Nx) with number of epochs divide by x, where $x = 1, 2, 3$. (a) RMSE comparison. (b) Time comparison.

learning model ($N1$). We focused on two different scenarios, where the processes can work in multiple CPUs in the same place (e.g., Tensorflow working with multiple CPUs in the same machine), or a scenario where CPUs are disaggregated and become independent of each other (e.g., different Edge nodes cooperating). The best hyperparameter configurations from the previous experiment were used in this experiment, with the addition that when distributing the data to be modeled, we are applying the $1/N$ factor to the number of training epochs as the "proposed epochs."

Table I shows the comparison of RMSE for the number of cars and speed with the fixed and proposed number of epochs for a single model (N1) and distributed models N2 and N3. For training distributed models N2 and N3, we used two and three Edge nodes, respectively. Whereas for N1, we used only one Edge node. We observed that for N2 and N3, using a fixed number of epochs increases RMSE due to overfitting the model. We also observed that training time did not change even when we distribute the input data to be processed by more than one model. This occurred because the number of epochs stays the same for each configuration. However, we observed a significant decrease in training time when the number of epochs is obtained by dividing the optimal number of epochs for $N1$, by the number of parallel models. We observed that RMSE is slightly increased in estimating the number of cars, while it remains almost stable for estimating cars' speed.

Fig. 9 shows the accuracy for estimating the number of cars and speed for using N1, N2, and N3. Fig. 9(a) shows that the accuracy in the estimate of the number of cars slightly decreases with the increase in the number of distributed models used to train the input data. This occurs because models are trained on fewer data and are more specific to a particular input set. This was why we observed this behavior when we combined them in the final prediction model. However, this does not affect the accuracy of predicting the speed of cars, and it somehow remains stable regardless of the number of models used to train the input data set. We also observed losing some accuracy on average, but we are saving more than 50% of training time when we used parallel models, as shown in Fig. 9(b).

TABLE II
COMPARISON OF CHANGE FACTOR IN ERROR AND TRAINING TIME FOR N2 AND N3 WITH N1

Model	+RMSE (Cars)	+RMSE (Speed)	+Time (s)
N2	0.65	0.83	-115.39
N3	1.89	0.02	-149.02

With respect to the speed-up comparison for parallel models (N2, N3) with N1, Table II shows the improvement factor for error and time. We observed that when we distribute the input data set to be processed by two models. There is a decrease of 115.39 s in training time with an increase of 0.65 and 0.83 of RMSE for the number of cars and speed. Similarly, we observed a reduction of training time when using three parallel models and an increase of 1.89 and 0.02 of RMSE for the number of cars and speed estimations.

C. Evaluation on Low-Power Architectures

In this experiment, we compared the proposed solution's effectiveness on low-power and resource-constrained devices designed for the Edge, like the *Raspberry Pi* model 3B and the *NVIDIA Jetson* model Nano. Such devices are built for consuming less than 12 W and embed low CPU and GPU computing resources. Raspberry Pi is used for general purposes while the Jetson integrates a GPU toward AI and NN computing on the Edge and smart devices.

Testing the grid configurations for *Time Aggregation* versus *Hidden Units* on the Raspberry Pi and the Jetson Nano, we observed a noticeable increase in execution time in comparison with the single-CPU Xeon. Still, the training plus validation time is below 30 min for nearly a week worth of data. We tested 4, 32, as 512 batch sizes (BS), i.e., the number of samples used for each training step in the NN to check the Jetson GPU's possible advantages due to data bandwidth. The bigger the BS, the more we profit from the GPU's parallelism up to a certain point. The number of epochs is fixed at 94, to compare the performance of identical training processes, and the steps (iterations) per epoch are proportional to the BS (200 steps/epoch for BS = 4, 25 steps/epoch for BS = 32, 1 step/epoch for BS = 512). The objective was to test the method's performance on low-powered devices with different properties while maintaining the error (that may vary when modifying the batch-size). As a comparison metric, we show the milliseconds per step and the seconds per epoch. When computing the average milliseconds/step, the first epoch was excluded as it carries the overhead on warm-up around $\times 4$ the average epoch. Fig. 10 shows the performance in times per step for the different configurations of the GRU in the different used technologies, for the training time with a common

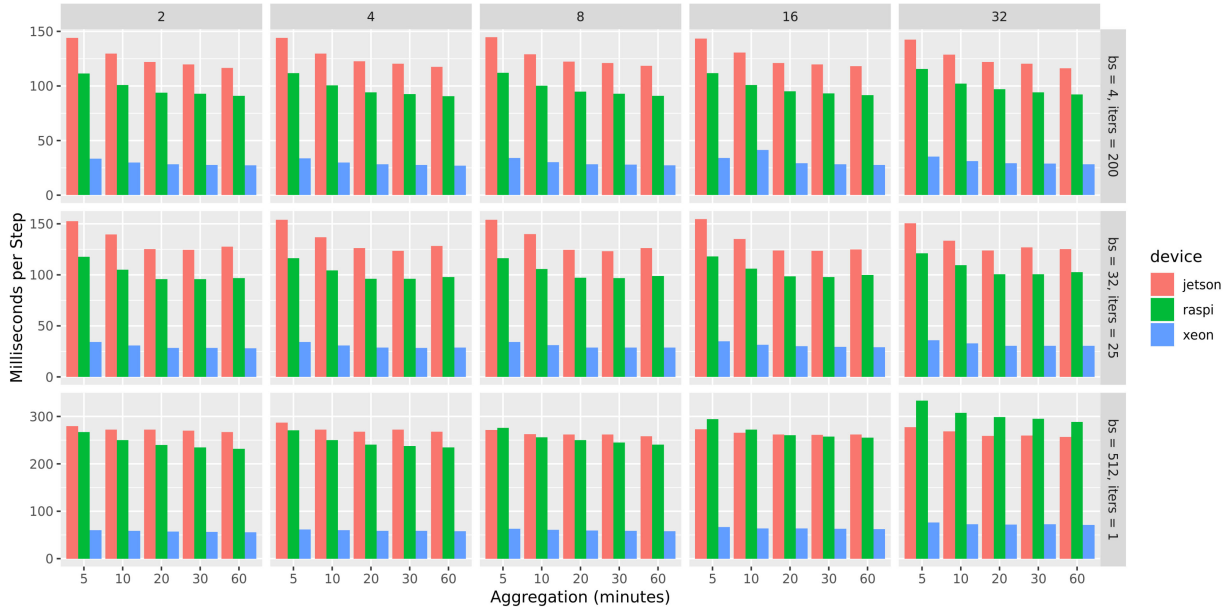

Fig. 10. Time comparison for configurations in low-power devices versus single CPU Xeon ref., for each amount of hidden units.

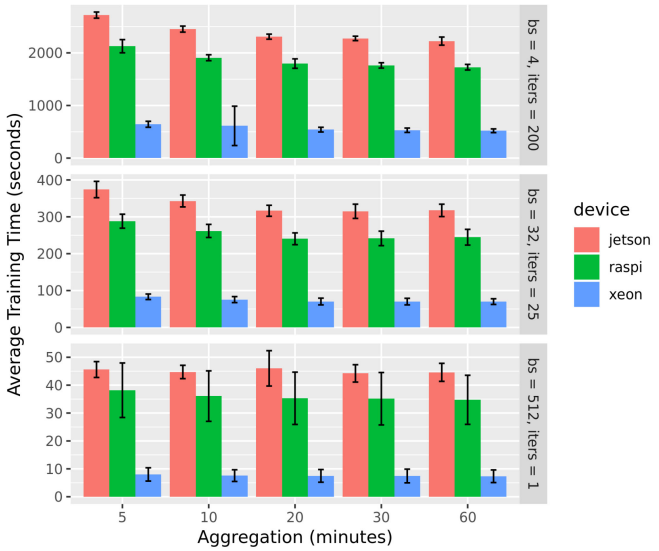

Fig. 11. Average modeling time on different edge devices.

and proper configuration found for the GRU on the single-CPU Xeon, the Raspberry Pi ARM-based CPU, and the ARM-based and GPU enhanced Jetson.

From this experiment, we concluded that our method is fully fit for use on low-power or resource-constrained devices, as training times take at maximum half an hour for a model representing around six days. Moreover, we noted that the GPU at the Jetson Nano does not provide improvement for the kind of data until the BS reaches larger sizes.

Fig. 11 shows the absolute training time for the three devices, where the single Xeon outperforms the low-powered ones but for no more than a factor of 4, and how in scenarios requiring large memory bandwidth (low data aggregation and large BSs), the GPU starts chasing CPU execution times.

TABLE III
COMPARISON WITH BASELINE MODELS VAR AND CRBMS AS $N = 1$ (5 MIN AGGREGATION). NOTE THAT THE RESULTS ARE FOR THE OVERALL BEST CONFIGURATIONS FOUND

Method	RMSE	
	Cars	**Speed**
VAR	4.99	7.44
CRBM	2.03	5.68
GRU	1.76	6.17

D. Comparison With Baseline Methods

To conclude the evaluations, we provide a comparison of the proposed method with previous and other simplistic models used for estimating the traffic data. We compared our solution with vector auto-regression (VAR), a classic time-series analysis method, and CRBMs as used in previous works [6].

As we can see in Table III, the proposed solution based on GRU outperforms VAR and provided comparable performance with CRBM when the granularity is set to 5 min. In Fig. 12, we can observe that GRU is slightly better than CRBM when granularity is finer, as seen in Fig. 7. Both kind of NN perform well in our framework. However, due to our particular interest in finer granularity, GRU is the chosen method for this work. For other experiments with different data sets, both methods should be compared in order to select the final model.

E. Discussion

Computing devices over the Edge are power and resource constrained as compared to the resource available in data centers. Building intelligent solutions requiring training compute-intensive DNN models introduced the challenge of efficiently utilizing the available Edge devices. In this work, we have addressed this challenge and proposed a system that distributes the compute-intensive ML tasks to the available Edge device while obtaining an accuracy comparable to models trained on

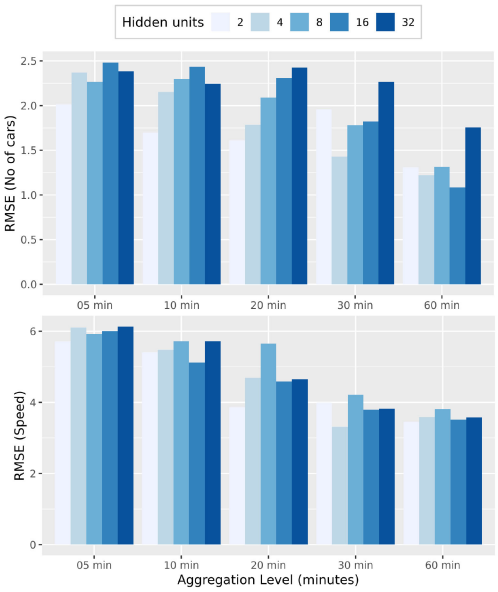

Fig. 12. Error versus hidden units versus Time aggregation for number of cars and speed estimation on CRBMs.

the single machine. Our solution is capable of stopping the model training to achieve acceptable performance dynamically. Our experimental evaluations compared a distributed learning model with a single-model approach and other baseline solutions for traffic forecasting. The results show the potential of the proposed solution for Edge and Fog platforms.

We consider splitting the ML modeling process attractive in scenarios, where we must reduce the training time without losing accuracy and constraint to avoid the task offloading to the cloud data centers. In such a situation, each device can take care of their local data and send only the trained model configuration to coordinate with other devices for building the model for generalizing the estimations. This solution is very useful in Edge computing environments in which we have low-power devices scattered. Our proposed solution will help many scenarios, including smart cities, traffic management and planning, the Internet of Things, and intelligent surveillance.

VI. Conclusion and Future Work

The presented work focused on performing predictive analytics on the Edge, using urban traffic prediction as an essential use case scenario relevant to smart cities applications. Given the amount of data generated on the Edge, not only in volume but also in time, moving modeling and analytics near the data may be a good compromise in front of Cloud models, where data must be massively pushed north bound. Of course, it must be understood that deep learning and other analytic processes are usually designed for high-performance computing environments. In contrast, the Edge front is commonly composed of low-power devices with scarce computing resources.

In this article, we proposed an experimentally evaluated method based on FL to move data analytics processing to Edge. The learning tasks are distributed to multiple Edge nodes, with limited computing resources, in a manner that each node processes its own local data. In doing so, we attempted to balance training time and model accuracy as a function of data distribution. Experiments showed that the tested data sets, provided by a road-service car fleet from Barcelona, can be learned with acceptable accuracy although being unstable on different previous tested techniques. Also, for each configuration of NNs, there exists a multidimensional tradeoff between the time spent on training, the distribution of data and parallelization of the model training process, and the previous aggregation of collected data to be trained, creating an interesting problem on how good we can model traffic against how much available time/resources are on given Edge scenarios.

The presented solution highlights future research and innovation opportunities on smart city applications, capable of providing services near-data and near-users without abusing network hierarchies and Cloud resources. While this work focused on a specific type of NNs, other statistical and ML can be applied, more suitable for particular scenarios far from urban traffic. Also, more complex architectures for distributing ML processes and automation of autonomous learning can be applied, focusing on better decisions when having time/resources for smart management of the device and the data pipeline. Another interesting aspect to cover is the use of MCUs with FL. With this, ultralow-powered devices would be able to collaborate in training and provide models with a wider knowledge on data from their neighboring MCUs.

References

[1] M. Abdullah, W. Iqbal, A. Mahmood, F. Bukhari, and A. Erradi, "Predictive autoscaling of microservices hosted in fog microdata center," *IEEE Syst. J.*, vol. 15, no. 1, pp. 1275–1286, Mar. 2021.

[2] S. Ali *et al.*, "SimpliMote: A wireless sensor network monitoring platform for oil and gas pipelines," *IEEE Syst. J.*, vol. 12, no. 1, pp. 778–789, Mar. 2018.

[3] M. Ghorbanian, S. H. Dolatabadi, and P. Siano, "Big data issues in smart grids: A survey," *IEEE Syst. J.*, vol. 13, no. 4, pp. 4158–4168, Dec. 2019.

[4] G. Premsankar, M. Di Francesco, and T. Taleb, "Edge computing for the Internet of Things: A case study," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 1275–1284, Apr. 2018.

[5] G. Plastiras, M. Terzi, C. Kyrkou, and T. Theocharidcs, "Edge intelligence: Challenges and opportunities of near-sensor machine learning applications," in *Proc. IEEE 29th Int. Conf. Appl. Specific Syst. Archit. Processors (ASAP)*, 2018, pp. 1–7.

[6] J. L. Pérez, A. Gutierrez-Torre, J. L. Berral, and D. Carrera, "A resilient and distributed near real-time traffic forecasting application for fog computing environments," *Future Gener. Comput. Syst.*, vol. 87, pp. 198–212, Oct. 2018.

[7] C. Savaglio and G. Fortino, "A simulation-driven methodology for IoT data mining based on edge computing," *ACM Trans. Internet Technol.*, vol. 21, no. 2, pp. 1–22, 2021.

[8] H. B. McMahan, E. Moore, D. Ramage, and B. A. Y. Arcas, "Federated learning of deep networks using model averaging," 2016. [Online]. Available: arXiv:1602.05629.

[9] K. Bonawitz *et al.*, "Towards federated learning at scale: System design," 2019. [Online]. Available: arXiv:1902.01046.

[10] S. Dey, A. Chakraborty, S. Naskar, and P. Misra, "Smart city surveillance: Leveraging benefits of cloud data stores," in *Proc. 37th Annu. IEEE Conf. Local Comput. Netw. Workshops*, 2012, pp. 868–876.

[11] K. Su, J. Li, and H. Fu, "Smart city and the applications," in *Proc. Int. Conf. Electron. Commun. Control (ICECC)*, 2011, pp. 1028–1031.

[12] M. Castro, A. J. Jara, and A. F. Skarmeta, "Smart lighting solutions for smart cities," in *Proc. 27th Int. Conf. Adv. Inf. Netw. Appl. Workshops*, 2013, pp. 1374–1379.

[13] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "LAVEA: Latency-aware video analytics on edge computing platform," in *Proc. 2nd ACM/IEEE Symp. Edge Comput.*, 2017, pp. 1–13.

[14] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-C. Woo, "Convolutional LSTM network: A machine learning approach for precipitation nowcasting," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 802–810.

[15] A. Sergeev and M. Del Balso, "Horovod: Fast and easy distributed deep learning in TensorFlow," 2018. [Online]. Available: arXiv:1802.05799.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[17] D. Cireşan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," 2012. [Online]. Available: arXiv:1202.2745.

[18] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proc. 14th Int. Conf. Artif. Intell. Stat.*, 2011, pp. 215–223.

[19] B. Hu, Y. Gao, L. Liu, and H. Ma, "Federated region-learning: An edge computing based framework for urban environment sensing," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2018, pp. 1–7.

[20] S. Wang *et al.*, "Adaptive federated learning in resource constrained edge computing systems," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1205–1221, Jun. 2019. 2018.

[21] J. Dean *et al.*, "Large scale distributed deep networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1223–1231.

[22] Q. V. Le *et al.*, "Building high-level features using large scale unsupervised learning," 2011. [Online]. Available: arXiv:1112.6209.

[23] N. Strom, "Scalable distributed DNN training using commodity GPU cloud computing," in *Proc. 16th Annu. Conf. Int. Speech Commun. Assoc.*, 2015, pp. 1488–1492.

[24] A. Coates, B. Huval, T. Wang, D. Wu, B. Catanzaro, and N. Andrew, "Deep learning with COTS HPC systems," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 1337–1345.

[25] P. Goyal *et al.*, "Accurate, large minibatch SGD: Training ImageNet in 1 hour," 2017. [Online]. Available: arXiv:1706.02677.

[26] K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwälder, and B. Koldehofe, "Opportunistic spatio-temporal event processing for mobile situation awareness," in *Proc. 7th ACM Int. Conf. Distrib. Event-Based Syst.*, 2013, pp. 195–206.

[27] L. Yu, Z. Li, J. Liu, and R. Zhou, "Resources sharing in 5G networks: Learning-enabled incentives and coalitional games," *IEEE Syst. J.*, vol. 15, no. 1, pp. 226–237, Mar. 2021.

[28] A. Marchisio *et al.*, "Deep learning for edge computing: Current trends, cross-layer optimizations, and open research challenges," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, 2019, pp. 553–559.

[29] B. Sudharsan, J. G. Breslin, and M. I. Ali, "RCE-NN: A five-stage pipeline to execute neural networks (CNNs) on resource-constrained IoT edge devices," in *Proc. 10th Int. Conf. Internet Things*, 2020, pp. 1–8.

[30] R. Sanchez-Iborra and A. F. Skarmeta, "TinyML-enabled frugal smart objects: Challenges and opportunities," *IEEE Circuits Syst. Mag.*, vol. 20, no. 3, pp. 4–18, 3rd Quart., 2020.

[31] S. Lee and S. Nirjon, "Neuro.ZERO: A zero-energy neural network accelerator for embedded sensing and inference systems," in *Proc. 17th Conf. Embedded Netw. Sens. Syst.*, 2019, pp. 138–152. [Online]. Available: https://doi-org.recursos.biblioteca.upc.edu/10.1145/3356250.3360030

[32] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st Edition MCC Workshop Mobile Cloud Comput.*, 2012, pp. 13–16.

[33] M. Naphade *et al.*, "The NVIDIA AI city challenge," in *Proc. IEEE SmartWorld, Ubiquitous Intell. Comput. Adv. Trusted Comput. Scalable Comput. Commun. Cloud Big Data Comput. Internet People Smart City Innov. (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, 2017, pp. 1–6.

[34] (Jul. 2019). *Nvidia Autonomous Machines: Jetson Nano*. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines

[35] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014. [Online]. Available: arXiv:1412.3555.

[36] F. E. Harrell, Jr., *Regression Modeling Strategies: With Applications to Linear Models, Logistic and Ordinal Regression, and Survival Analysis*. Cham, Switzerland: Springer, 2015, doi: 10.1007/978-3-319-19425-7.

Alberto Gutierrez-Torre received the M.Sc. degree in data science from the Technical University of Catalonia-BarcelonaTech, Barcelona, Spain, in 2017. He is currently pursuing the Ph.D. degree with the Data-Centric Computing Research Group, Barcelona Supercomputing Center, Barcelona.

His research interests are focused on data analysis, statistics, and machine learning with a specific interest on data streams.

Kiyana Bahadori received the M.Sc. degree in computer science from the University of Bangalore, Bengaluru, India, in 2013, and the Ph.D. degree in computer science from the Brain, Mind, and Computer Science Program, University of Padua, Padua, Italy, in 2019.

Her research interests focus on improving performance and resource utilization of services in the cloud.

Shuja-ur-Rehman Baig received the B.Sc. and M.Sc. degrees in computer science from Lahore University of Management Sciences, Lahore, Pakistan, in 2006 and 2009, respectively, and the Ph.D. degree from the Computer Architecture Department, Technical University of Catalonia-BarcelonaTech (UPC), Barcelona, Spain, in 2019.

He holds a Lecturer position with Punjab University College of Information Technology, University of the Punjab, Lahore. His research interests are cloud computing, big data, and machine learning.

Waheed Iqbal received the Ph.D. degree from the Asian Institute of Technology, Khlong Nueng, Thailand, in 2012.

He is an Assistant Professor with Punjab University College of Information Technology, University of the Punjab, Lahore, Pakistan. He worked as a Postdoctoral Researcher with the Department of Computer Science and Engineering, Qatar University, Doha, Qatar, from 2017 to 2018. His research interests lie in cloud computing, distributed systems, machine learning, and large-scale system performance evaluation.

Tullio Vardanega (Member, IEEE) received the M.Sc. degree from the University of Pisa, Pisa, Italy, in 1986, and the Ph.D. degree in computer science from the Technical University of Delft, Delft, The Netherlands, in 1998.

He is an Associate Professor with the Department of Mathematics, University of Padua, Padua, Italy. He first was a Principal Investigator in a consultancy firm in Pisa, Italy, from 1986 to 1991, and then a member of Technical Staff with the European Space Research and Technology Center, Noordwijk, The Netherlands, from 1991 to 2001. His teaching and research interests are in the areas of high-integrity real-time systems, cloud, fog, and edge computing, and software engineering methods and processes.

Josep Lluís Berral (Member, IEEE) received the B.Sc. degree in computer science, the M.Sc. degree in computer architecture, and the Ph.D. degree from BarcelonaTech-UPC, Barcelona, Spain, in 2007, 2008, and 2013, respectively.

He is a Data Scientist, working in applications of data mining and machine learning on data-center and cloud environments at the Barcelona Supercomputing Center, Barcelona, within the "Data-Centric Computing" research line. He has worked with the High Performance Computing Group, Computer Architecture Department, UPC, and also with the Relational Algorithms, Complexity and Learning Group, Computer Science Department, UPC.

Dr. Berral received in 2017 a Juan de la Cierva Research Fellowship by the Spanish Ministry of Economy.

David Carrera (Member, IEEE) received the M.S. and Ph.D. degrees from BarcelonaTech-UPC, Barcelona, Spain, in 2002 and 2008, respectively.

He is an Associate Professor with the Computer Architecture Department, UPC. He is also an Associate Researcher at the Barcelona Supercomputing Center, Barcelona, within the "Data-Centric Computing" research line. He has been involved in several EU and industrial research projects. His research interests are focused on the performance management of data center workloads.

Dr. Carrera received an IBM Faculty Award in 2010. He was awarded an ERC Starting Grant for the Project HiEST, in 2015.