applied sciences

MDPI

Article

Embedded Federated Learning for VANET Environments

Renato Valente [1], Carlos Senna [1], Pedro Rito [1,*] and Susana Sargento [1,2]

[1] Instituto de Telecomunicações, 3810-193 Aveiro, Portugal
[2] The Department of Electronics, Telecommunications and Informatics, University of Aveiro, 3810-193 Aveiro, Portugal
* Correspondence: pedrorito@av.it.pt

Abstract: In the scope of smart cities, the sensors scattered throughout the city generate information that supplies intelligence mechanisms to learn the city's mobility patterns. These patterns are used in machine learning (ML) applications, such as traffic estimation, that allow for improvement in the quality of experience in the city. Owing to the Internet-of-Things (IoT) evolution, the city's monitoring points are always growing, and the transmission of the mass of data generated from edge devices to the cloud, required by centralized ML solutions, brings great challenges in terms of communication, thus negatively impacting the response time and, consequently, compromising the reaction in improving the flow of vehicles. In addition, when moving between the edge and the cloud, data are exposed, compromising privacy. Federated learning (FL) has emerged as an option for these challenges: (1) It has lower latency and communication overhead when performing most of the processing on the edge devices; (2) it improves privacy, as data do not travel over the network; and (3) it facilitates the handling of heterogeneous data sources and expands scalability. To assess how FL can effectively contribute to smart city scenarios, we present an FL framework, for which we built a testbed that integrated the components of the city infrastructure, where edge devices such as NVIDIA Jetson were connected to a cloud server. We deployed our lightweight container-based FL framework in this testbed, and we evaluated the performance of devices, the effectiveness of ML and aggregation algorithms, the impact on the communication between the edge and the server, and the consumption of resources. To carry out the evaluation, we opted for a scenario in which we estimated vehicle mobility inside and outside the city, using real data collected by the Aveiro Tech City Living Lab communication and sensing infrastructure in the city of Aveiro, Portugal.

Keywords: federated learning; distributed forecasting; smart city; FL on edge devices

Citation: Valente, R.; Senna, C.; Rito, P.; Sargento, S. Embedded Federated Learning for VANET Environments. *Appl. Sci.* **2023**, *13*, 2329. https://doi.org/10.3390/app13042329

Academic Editor: Hannu Laaksonen

Received: 31 December 2022
Revised: 3 February 2023
Accepted: 4 February 2023
Published: 11 February 2023

1. Introduction

In the scope of smart cities, the sensors scattered throughout the city generate information that can be used to supply intelligence mechanisms to learn the city's mobility patterns. These patterns are used in applications such as traffic estimation, which allow for improvement in the quality of experience in the city. However, current solutions are based on centralized approaches that generate high latency and increase data transmission. In addition, the mobility patterns based on the information from across the city are not suitable for medium/large cities, where regionalization of habits directly interferes with mobility.

With the increased importance of the Internet of Things (IoT), monitoring points in the cities are always growing, and the transmission of their data from the end-node devices to the cloud brings great challenges, thus increasing the load on the network, which negatively impacts the response time and, consequently, compromises the reaction time to improve the flow of vehicles [1]. Furthermore, traditional cloud-centric solutions suffer from loss of privacy as data (messages, images, videos, and personal information) must be sent to the cloud to be processed.

One of the options to improve the problems mentioned above is the support of distributed architectures that enable the use of machine learning (ML) on edge devices

where learning is achieved. In this way, there is no movement of data, which guarantees privacy. Among the distributed options, the new paradigm based on federated learning (FL) emerges as a great option [2]. Federated learning (FL) is a machine learning method that uses the datasets hosted on edge devices to train machine learning (ML) models locally and thus be prepared to meet regional demand. However, still, it is important to maintain the global scope view, and to do so, federated learning (FL) operates in a federated way; that is, the edge devices send their models to be aggregated globally. Since only model parameters are transacted, federated learning (FL) naturally ensures data privacy. By sending only the parameters, it has the benefits of lower latency and lower network overhead, in addition to being potentially faster than the centralized options.

To evaluate how federated learning (FL) can effectively contribute, we present a federated learning (FL) framework, FedFramework [3], for which we built a testbed where edge devices, such as NVIDIA Jetson, were connected to a cloud server, that is, a typical solution used in communication infrastructures for smart cities. In this testbed, we implemented the FedFramework to estimate the mobility of vehicles inside and outside the city, using real data collected by the communication and sensing infrastructure of the Aveiro Tech City Living Lab in the city of Aveiro [4]. In this testbed, machine learning (ML) algorithms were distributed in the city's 52 road-side units (RSUs)—FL client containers hosted in the edge device units—that capture the traffic that feeds the algorithms. Periodically, the parameters of all these distributed models were sent to a centralized unit (server installed in a virtual machine), where a refinement process was applied through aggregation algorithms and then returned to the federated learning (FL) clients. Using this lab test environment, we evaluated the devices' performance, the effectiveness of machine learning (ML) and aggregation algorithms, the impact on end-to-server communication, and the resource consumption of the devices.

In summary, the main contributions of this work are the following features:

- An exploration of several federated algorithms to decide which one best fits the vehicle data;
- Performance evaluation and resource consumption on edge devices;
- A performance assessment of the network in the presence of multiple federated clients and multiple federated learning predictors;
- Scalability evaluation of the user-friendly federated learning (FL) framework on edge devices used in the infrastructure of smart cities.

The remainder of this paper is organized as follows: In Section 2, the related work associated with FL-based solutions on edge devices is discussed, and in Section 3, we briefly present the FedFramework. Section 4 presents the deployed testbed and shows the details of the evaluated use case. Section 5 presents and discusses the performance of the FL-based solution on edge devices, the communication overhead, the performance of the aggregation algorithms, the performance of the entire federated process, and the scalability of the system components. Finally, Section 6 concludes the paper and presents future work.

2. Related Work

Federated learning (FL) is a recent paradigm in which machine learning (ML) processing is distributed, allowing several decentralized clients to cooperatively train a shared model globally [2]. Despite being recent, it is a topic of great interest, with a significant amount of work on distributed and federated learning [5–9]. Due to the scope of this work, we focused our assessment on the work related to the deployment of distributed/federated learning solutions on edge devices that are somehow used in intelligent transportation system (ITS) solutions within the scope of smart cities.

Chen et al. [1] proposed a vehicle detection algorithm based on the YOLOv3 model trained with a large volume of traffic data. This solution uses images to identify vehicles that generate a dataset used to train the model for flow detection. The model is trained on a centralized server and then migrated to the devices that use NVIDIA Jetson to perform the

discovery work on edge devices. Despite using edge devices to carry out the identification, the entire training process is centralized.

The work in [10] described a testbed assembled with Intel Movidius Neural Compute Stick and Raspberry Pi 3 Model B. The testbed hosts convolutional neural network (CNN) algorithms used to analyze objects in real-time videos for vehicular edge computing. The authors show a small performance evaluation of the algorithms in Raspberry-Pi-based devices.

The authors of [11] proposed a communication-efficient FL framework, the goal of which is to improve FL convergence time. The authors used a probabilistic device selection scheme that aims to choose the most efficient ones in transmitting the model for aggregation. The authors formalized their schemes for probabilistic device selection and network resource management and analyzed their performance through simulation. Therefore, no evaluation was carried out with real devices.

In [12], Rahman et al. presented a lightweight hybrid FL framework that uses blockchain smart contracts to add security to the edge training plane, manage the trust and authentication of the participating federated nodes, and distribute the trained models globally or locally based on the reputation of edge nodes and their loaded datasets or models. To evaluate the proposed solution, they used a hybrid testbed with various types of devices such as Jetson, Raspberry Pi, and others. The focus of this work and evaluation was the security added with the use of the blockchain-based solution, and therefore, they did not deepen their evaluation in other directions.

Vita et al. [13] presented an extension of Stack4Things [14], a cloud platform, adapting its functionalities to edge devices in order to provide a federated learning (FL) platform. To evaluate the new platform and demonstrate the efficiency of the federated approach, a heterogeneous testbed was set up with different types of clients, a laptop, three Raspberry Pi 3s, and an NVIDIA Jetson Nano, distributed on a university campus so that they were not in the same network, thus emulating a distributed deployment. In this testbed, a smart city application was deployed that exploits deep learning techniques to classify the vehicles crossing the intersection for traffic analysis purposes. In this scenario, they measured the loss of the federated model, training time, accuracy, and acceleration; that is, they focused their evaluation on the performance of the federated option and did not evaluate scalability, consumption of edge devices, or network overhead.

Süzen et al. [15] presented a study in which they compared the performance of single-board computers on NVIDIA Jetson Nano, NVIDIA Jetson TX2, and Raspberry Pi 4. In this comparative study, the authors used CNN algorithms applied to the datasets of fashion product images. In the benchmark work, they evaluated basic points such as memory and CPU consumption but did not evaluate scalability, the concurrent use of multiple instances, processing times, or the aspects related to communication.

In [16], the authors compared the most popular deep learning (DL) models for multi-leaf disease detection to assess which model is best suited for actual deployment. They used a real-world large-scale dataset and a Raspberry Pi 3 to evaluate the accuracy, memory usage, number of parameters, model sizes, and training time, among others. It is an interesting work, directly focused on the performance of DL models, but the authors evaluated machine learning models for a specific purpose: multi-leaf diseases. Moreover, they did not address distributed or federated learning.

The work in [17] presented a benchmark of a Pi Spark cluster used to process distributed inference built with TensorFlow. The authors compared the performance between a two-node Pi 4B Spark cluster and other systems, including a single Pi 4B and a mid-range desktop computer. To perform the analysis, they used image classification and face detection, that is, all the experiments related to image processing. Although interesting, the testbed was restricted, since it did not use real data and did not assess scalability. Furthermore, it was very ingrained in Spark-aligned applications. By contrast, our solution is not limited to any type of application and has a native design aimed at distributed/federated processing.

Baghersalimi et al. [18] presented and evaluated a standard federated learning (FL) framework in the context of epileptic seizure detection using a deep-learning-based approach, which operates on a cluster of NVIDIA Jetson Nanomachines. They evaluated the accuracy and performance of the proposed approach with epilepsiae, a seizure detection database. The authors evaluated the performance of the proposed platform in terms of the effectiveness, accuracy, and convergence of the federated learning option, and they also included an assessment of energy consumption. The work is interesting, but it had a specific focus (neurological disorders), and scalability and platform heterogeneity were not assessed.

Zhang et al. [19] presented a solution for an evaluation of federated learning (FL) in IoT devices and a platform for anomaly detection. They also established the FedDetect algorithm for anomaly identification. To evaluate the solution, the authors used a platform with Raspberry PI, the N-BaIoT, and LANDER datasets, and a GPU server. In their evaluation, they used precision and epoch and discussed memory cost and end-to-end training time on constrained IoT devices. However, the proposed solution was specific for anomaly detection, not offering options regarding ML models or aggregation strategy. Moreover, the evaluation made is considered very preliminary, and a more thorough analysis is needed.

In [20], the authors proposed the communication-efficient FedAvg (CE-FedAvg), a modified FedAvg algorithm that incorporates a distributed variant of Adam's optimization. CE-FedAvg's objective is to reduce the total number of rounds required for convergence. To test the real-time convergence of CE-FedAvg over FedAvg, they used an RPi-based testbed to simulate a heterogeneous low-power edge computing scenario. The testbed uses 5 RPi 2Bs, 5 RPi 3Bs, and 1 desktop as a server on a wireless network used to emulate lower bandwidth networks. However, as expected, the evaluation exclusively focused on CE-FedAvg's performance and was, therefore, restricted in relation to other evaluation criteria.

Gao et al. [21] compared the learning performance of federated learning (FL) and split neural networks in terms of model accuracy and confluence speed. They focused on unbalanced, non-independent, and identically distributed data that are best suited for IoT applications. To evaluate the proposed solution, they set up a testbed composed of Raspberry Pi (IoT gateway) as a server to aggregate the models and IoT sensors (Arduino, RFID, etc.) that interacted with the server to perform distributed learning. They used this testbed to evaluate the model overhead, including training time, communication overhead, power consumption, and memory usage.

In [22], Kim et al. developed a federated learning (FL) platform, called KubeFL, based on Kubernetes technology. KubeFL hosts client models in Docker and Kubernetes containers that connect to a server. The authors deployed their platform to NVIDIA Jetson TX2 devices, where they performed several performance tests. The proposed solution seems to have the potential to face challenges related to environments that require distributed ML. However, the testbed was very simple, and so was the evaluation carried out. For example, there was no assessment of scalability, communication overhead, device overhead, or heterogeneity between client devices.

Jiang et al. [23] proposed a federated learning (FL) approach with adaptive and distributed parameter pruning (PruneFL), which adapts the model's size during federated learning (FL) rounds to reduce both communication and computation overhead and minimize the overall training time. To evaluate the performance of their approach, they conducted experiments with real edge computing prototypes (notebook as a server and clients hosted in Raspberry Pi devices) used to simulate multiple clients and a server. In their experiments, computation and communication times were analyzed, and some measurements involving either Raspberry Pi devices were performed. However, their evaluation was restricted to the proposed PruneFL; other works such as [24–26] also used the pruning strategy to minimize communication time and training rounds.

Our work stands out for evaluating solutions based on federated learning (FL) in its different features, namely the performance of edge devices, the communication overhead, and the impact of aggregation algorithms. In addition, by using the data directly collected from a communication and sensing infrastructure in current use, the ATCLL, we provide system designers with field information that can speed up the design of solutions for smart cities.

3. FedFramework: Federated Learning Framework

The main component of our platform is the federated learning (FL) framework, called FedFramework. This framework was first proposed in [3], and this section only provides a very brief overview to understand the evaluation and the obtained results. As shown in Figure 1, FedFramework is composed of two main services: Edge FL Service (EFLS) and Cloud FL Service (CFLS).

The Edge FL Service (EFLS) runs on edge devices where local data are used for the continuous training of the learning model to prepare it to deliver on-demand predictions to users that are within edge device (RSU) coverage. An important part of Edge FL Service (EFLS) is the *Edge Aggregator*, a component that interacts with Cloud FL Service (CFLS) in which local machine learning (ML) models send their parameters to be aggregated in the global service.

The Cloud FL Service (CFLS), commonly hosted in the cloud, manages the aggregation process of edge models. It receives the parameters from machine learning (ML) models installed at the edge, coordinates their aggregation into a global model, and controls the aggregation process of these parameters until the desired accuracy is identified.

In an overview of using the framework, when installed on edge devices, Edge FL Service (EFLS) trains the base models using local data made available by Data Gathering Emulator Service (DGES). With this initial training, they are now available to make predictions through the *prediction interfaces*. When identifying the moment to refine the local models (*aggregation rule*), Cloud FL Service (CFLS) notifies the Edge FL Service (EFLS) so that they can send their parameters. Upon receiving all the edge models, Cloud FL Service (CFLS) applies the aggregation algorithm and sends the new aggregated model to the Edge FL Service (EFLS). They retrain using their local data and check for accuracy. This aggregation cycle, called federated learning round (FLR), runs continuously until the accuracy of the local models reaches the expected result.

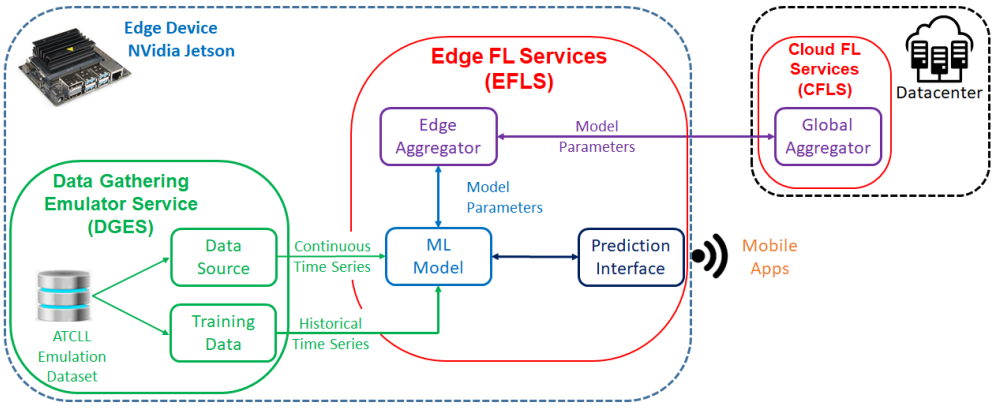


Figure 1. FedFramework architecture.

The Data Gathering Emulator Service (DGES) is an emulator that produces real data and allows offline testing. To use it, we only need to include the desired dataset in the corresponding folder in the container. In our tests, we supplied Data Gathering Emulator Service (DGES) with the data collected directly from ATCLL, thereby ensuring that our algorithms work with the real data of vehicle mobility in the city of Aveiro.

One of our concerns when designing a framework was the "ease of use". With the design and implementation based on Docker, FastAPI (https://fastapi.tiangolo.com/, accessed on 22 November 2022), and Flower tools (https://flower.dev/, accessed on 22 November 2022), it offers a simplified installation process starting with the deployment of a Restful API in all methods for operation and use, and the deployment of two Docker images, one related with Cloud FL Service (CFLS) and another for creating Edge FL Service (EFLS) instances.

4. Distributed Learning Laboratory Testbed

This section presents the laboratory testbed. The testbed included a hardware platform where edge devices are connected to the server over a local area network, and the collected data are used in the evaluation experiments. These elements are presented below.

4.1. Hardware Platform

There are several embedded device options for edge devices that can house AI solutions [27–29]. They can be specific solutions such as Titan RTX (NVIDIA) [30], Cloud TPU (Google) [31], or Xeon D-2100 (Intel) [32], or they can be single-board computers (SBCs), such as Data Box Edge (Microsoft) [33], Movidus Neural Compute Stick (Intel) [34], or Jetson (NVIDIA) [35]. Motivated by the experience gained through ATCLL, we chose NVIDIA Jetson Nano [36] and NVIDIA Jetson Xavier [37] as edge devices to assemble our laboratory testbed. These embedded board units, widely used in IoT installations in smart cities [18], are detailed in Table 1.

Table 1. Equipment characteristics.

	Server	Jetson Nano	Jetson NX Xavier
Processor	Intel(R) Xeon(R) 4215	Arm(R) Cortex(R)-A57	ARMv8 rev 0 (v8l)
Architecture	x86_64	aarch64	aarch64
CPUs	4	4	6
Memory	8 GB	2 GB LPDDR4	8 GB LPDDR4
Storage	64 GB	128 GB	128 GB
Operation System	Ubuntu 18.04.6 LTS	Ubuntu 18.04.5 LTS	Ubuntu 18.04.6 LTS
Linux Kernel	4.15.0-197-generic	4.9.140-tegra	4.9.253-tegra

The testbed was composed of four Jetsons Nano and one Jetson Xavier connected to the data center through a 1 Gbps wired network. The data center contained virtual machines that hosted the servers responsible for aggregating the machine learning (ML) models distributed at the edge (Figure 2).

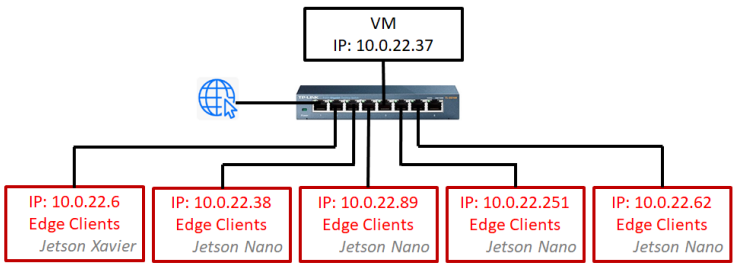


Figure 2. Distributed Learning Laboratory Testbed.

One of the main objectives was to provide AI services for applications within the scope of the ATCLL project [4]. In this way, we developed a framework through which it is possible to develop, test, and generate the most varied machine learning (ML) algorithms, and in addition, to offer an option to work with federated learning (FL) when this option is the most suitable for an application. Our strategy for building the framework was to virtualize it through containers to isolate the different services. This isolation would ensure

that tasks such as updating, shutting down, crashing, and consuming resources from a single service would not affect the others. In addition, we were also able to configure several network options easily.

4.2. ATCLL Overview and Its Datasets

The datasets used in the evaluation were collected from the infrastructure in the city of Aveiro, the ATCLL. The ATCLL comprises a large number of Internet-of-Things (IoT) devices with communication, sensing, and computing capabilities constituting a smart city infrastructure [4]. The 44 communication points of access in the city are connected through fiber and millimeter-wave (mmWave) links and integrate radio terminals (WiFi, ITS-G5, C-V2X, 5G, and LoRa). All these points combine and interconnect a set of sensors, such as mobility sensors (radars, LiDARs, and video cameras) and environmental sensors. The RSUs, indicated on the map as smart lamp posts P1, P30, P33, and P35, combine communication and computing devices such as PCEngines APU2 units (https://www.pcengines.ch/apu2.htm, accessed on 22 November 2022) and NVIDIA Jetsons. APUs use AMD GX-412TC with 4 CPU cores, 4GB of DRAM, 256GB of SSD, and Debian GNU/Linux 11.

As illustrated in the left side of Figure 3, network nodes are also installed in the vehicles and local boats to transmit the mobility and environment data to the data-processing center. In the vehicular network supported by the ATCLL, the vehicles are equipped with on-board units (OBUs) to establish a connection with the RSU and with the other vehicles, over ITS-G5 and 5G. One of the main objectives of the ATCLL is to provide a range of services that can make the city smarter by offering, for example, services to support the management of urban mobility or emergency and safety applications. To achieve this purpose, the ATCLL hosts a base software platform capable of hosting services at three levels of operation: on mobile devices (OBUs), on edge devices (RSUs), and on the central processing (cloud). Among the services hosted on mobile devices, it is possible to highlight the identification of the mobility of each vehicle (position, speed, etc.) and also the collection of sensing information, such as temperature, humidity, etc. In the edge devices, in addition to collecting and processing sensing information and performing data collection from mobile devices, the various types of activity identification stand out, such as the number and type of vehicles that pass through the coverage perimeter, the capture and evaluation of pictures, etc. Most of the information collected by mobile and edge devices is sent to a central unit, where it is analyzed to gain a strategic overview of the city "behavior".

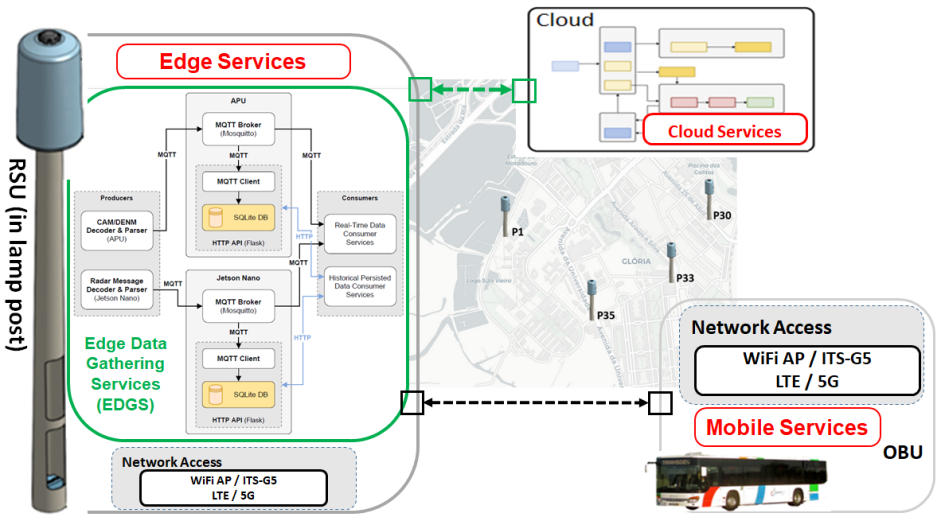


Figure 3. ATCLL infrastructure.

To perform the evaluation of the FedFramework deployed at edge devices, we opted for a simple system to predict the number of vehicles entering the city, and to do so, we

used the time series datasets from the radars. Each dataset contains the number of vehicles that entered the city, identified at the collection points (RSU/Client) installed in strategic locations in the city of Aveiro. More specifically, we selected a period of four weeks, from 22 April 2022 to 22 May 2022. In this period, we considered the vehicles that passed through RSU in P1, P30, P33, and P35, as shown in Figure 4. Vehicle accounting was closed hourly, generating 24 sets of accounts each day. During this period of four weeks, we used three weeks to train the models and used the trained models and compared their predictions with the data from the fourth week to identify the accuracy of the models.

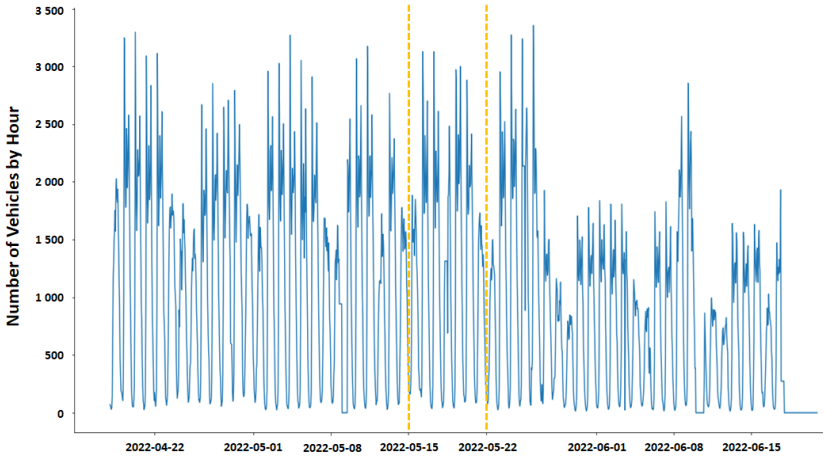


Figure 4. All vehicles that entered Aveiro through the four roads.

5. Results

In the assembled testbed, we performed experiments with the objective to evaluate the performance of FedFramework in real devices. We began by evaluating the performance of the federated learning (FL) solution in a smart city environment. To do so, we measured (1) the time for training on edge devices (Jetsons); (2) the time for sending the edge models to the centralized aggregator; (3) the time for aggregating the models and generating the new aggregated model; (4) the time for sending the aggregated model for the edge devices; and (5) the time involved in the aggregation process as a whole. This detailed assessment not only allowed the illustration of the functionalities and performance of our solution but also served as a parameter for choosing between the federated option and the centralized option in relation to the use of machine learning. In addition, we evaluated the progressive resource consumption of these devices (CPU, RAM), analyzed the scalability of the hardware and software set with 1 to 70 clients, and measured the network overhead.

Our framework can work with the most used machine learning models, such as recurrent neural networks (RNNs), gated recurrent units (GRUs), convolutional neural networks (CNNs), and long 277 short-term memory (LSTM). For the evaluation part of our framework, we selected a vehicular use case and the prediction of the number of vehicles entering a city. In accordance with state-of-the-art research, convolutional neural networks (CNNs) and long 277 short-term memory (LSTM) work well for regression problems with time series predictions [38,39]. Taking this into account, we decided to focus our performance tests by benchmarking convolutional neural networks (CNNs) and long 277 short-term memory (LSTM) in our framework. In this preliminary evaluation, we used two convolution layers, both with eight filters, one kernel size, and rectified linear unit (ReLU) for the activation of the convolutional neural network (CNN) option. For long 277 short-term memory (LSTM), we chose one long 277 short-term memory (LSTM) layer, with eight units and rectified linear unit (ReLU) for the activation setting. These are the best parameters according to [38]. Moreover, we chose three metrics to evaluate the regression model: MSE [40], MAE [41], and R-Squared (R^2) [42].

As shown in Table 2, knowing that 0 in MSE and MAE values means that the model is perfect, and R^2 should have a score close to 1, all edge clients had better results with a convolutional neural network (CNN). Thus, the convolutional neural network (CNN) model was our choice. For all the tests, the convolutional neural network (CNN) model was established with the execution of 10 rounds for aggregation and 5 epochs in the local training of each client (edge device).

Table 2. CNN versus LSTM.

	CNN			LSTM		
Client	**MSE**	**MAE**	R^2	**MSE**	**MAE**	R^2
1	0.004	0.033	0.782	0.006	0.045	0.644
2	0.011	0.058	0.742	0.013	0.062	0.710
3	0.012	0.069	0.862	0.018	0.083	0.793
4	0.013	0.065	0.789	0.018	0.086	0.680

The strategy used in the evaluation was to progressively add more clients to each edge device and then add more devices. During the initial tests, with the increase in the number of clients on the same device, it was noticed that the Jetsons became exponentially slower until they were no longer able to run the containers, thus becoming completely blocked. In Jetson Nano, this was more noticeable, because, with 14 clients installed, it was excessively slow, ending up blocking without recovery. With that in mind, it was assumed that 14 clients would be the maximum number that this type of device can support for operations with FedFramework. The complete sequence of tests was as follows:

- One Jetson Nano with only one client (a total of one client);
- One Jetson NX Xavier with only one client (a total of one client);
- Four Jetson Nanos with one client and one Jetson NX Xavier with one client (a total of five clients);
- Four Jetsons Nano with five clients and one Jetson NX Xavier with five clients (a total of twenty-five clients);
- Four Jetsons Nano with ten clients and one Jetson NX Xavier with ten clients (a total of fifty clients);
- Four Jetsons Nano with fourteen clients and one Jetson NX Xavier with fourteen clients (a total of seventy clients).

Figure 5 illustrates the testbed configuration with all devices and shows the configuration details (IP, ports, etc.), where each federated learning (FL) client was deployed in a container.

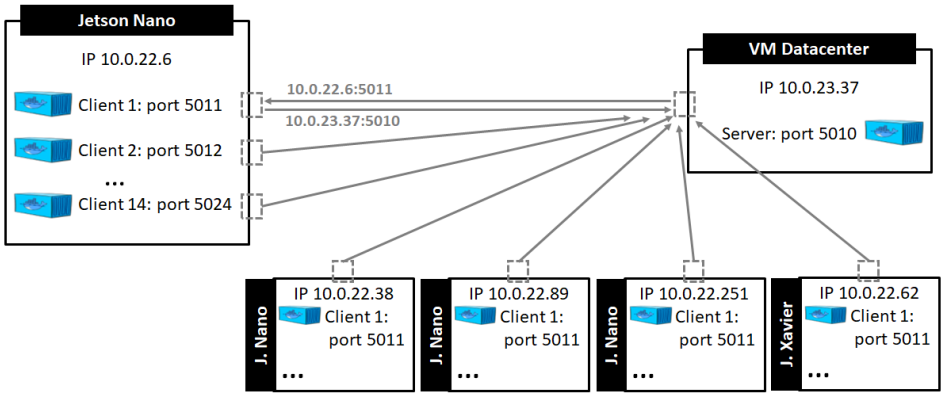


Figure 5. FL client distribution and its connection with server.

Figure 6 depicts the strategic points where measurements were taken. In the figure, signaled by the letters from a to e, all the stages of a federated learning round (FLR) are shown. The following times were evaluated during one round: training the local model (arrow a), sending that model to the server (arrow b), aggregating the local models (arrow c), downloading the aggregated model to the clients (arrow d), and finally, evaluating the global model with the client's local data (arrow e). As mentioned earlier, the data for each test were averaged from the 10 results obtained from 10 repetitions for the same test.

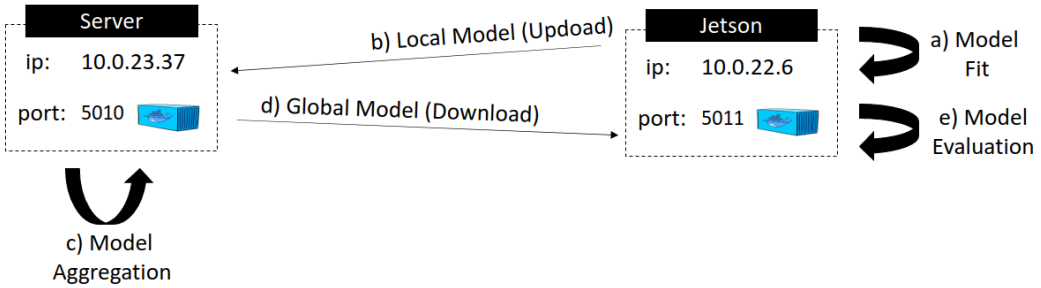


Figure 6. Steps of FLR observed during the tests.

The next sub-sections discuss several tests and results. We start with an overview of the aggregation process, followed by an evaluation of the training time and communication delay of the machine learning (ML) model. After evaluating the performance of the solution based on federated learning (FL), we analyze the direct consumption of resources. In this way, we assess the server and Jetsons resource overloads.

5.1. Overall Performance of the Federated Learning (FL) Application

We start by evaluating the behavior of federated learning (FL) from a broader perspective, that is, how long it takes for the federated process to reach the best accuracy. All the tests were executed 10 times, and the time necessary to connect all clients to the server, the average time for a round, and the total time until the server had the global model after 10 aggregation rounds were considered. Table 3 summarizes the obtained results.

Table 3. Runtimes of the six tests previously described.

Total of Clients	Connection Time	1 Round	Total Time
1 client Nano	12.36 seg.	1.22 seg.	25.81 seg.
1 client Xavier	12.26 seg.	1.40 seg.	26.75 seg.
5 clients	38.02 seg.	1.41 seg.	52.56 seg.
25 clients	69.27 seg.	4.78 seg.	118.02 seg.
50 clients	161.67 seg.	10.32 seg.	265.61 seg.
70 clients	418.19 seg.	23.83 seg.	657.72 seg.

Through the analysis of the times in Table 3, we can observe that the client's connection time to the server increased when there was more than one client. For 70 clients, the connection time increased excessively because, from a certain number of clients, the edge devices had difficulty in creating other clients (containers), delaying the connection process of all clients to the server. Regarding the average time for each round, we can observe that, in the first three tests, the time was the lowest because there was only one client (container) on the same edge device. When we increased the number of clients on the same edge device, the time per round increased because the device had more resource consumption. Consequently, with the increase in the connection time and the time of each round, the total time of the test also increased.

In the following sections, we will take a closer look at the relationship between the increase in the number of clients and resource consumption.

5.2. ML Model Training Time

Figure 7 shows the time for the model aggregation on the server for a specific number of rounds and the number of clients used in the testbed. With the number of clients increasing, it is possible to observe a slight increase in the time it takes to aggregate the models. For 70 clients, the average time was 0.14 s, which is quite fast, but for cases in which we need to scale the federated system to thousands of clients, the aggregation time will already be relevant for the total system time.

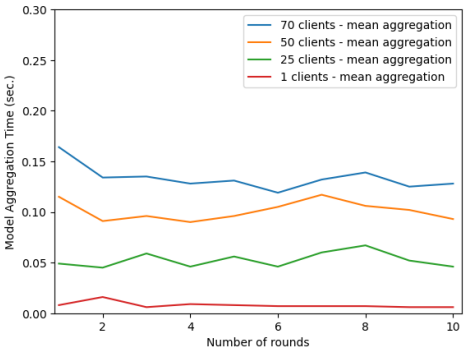


Figure 7. Model aggregation time for each round at the server.

Since we used two different types of devices for the tests on the client side, Figure 8 shows the results for a client on the Jetson Nano and a client on the Jetson NX Xavier. On each device, a client was selected, and the training time was evaluated. With the increase in the number of clients on the same device, the time increased significantly for the selected client. This occurred because the device was overloaded with the processes of several clients. As shown in Figures 8 and 9, the first round was removed due to the fact that the Jetsons had memory problems and yielded false values. When comparing the behavior of both Jetsons (Figure 8), we can observe that the Jetson NX Xavier performed better when the number of customers increased. This is because the Jetson Xavier is a device with a better processor and more CPUs than the Jetson Nano.

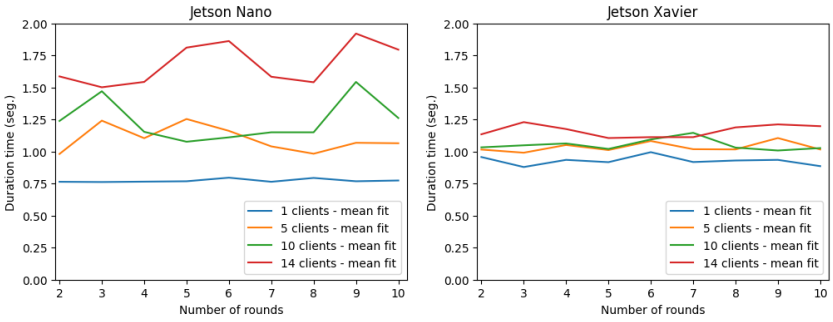


Figure 8. Model training time for each round for the client.

Regarding the time required to evaluate the model on the client side for both devices, the same conclusion as above holds (Figure 9). Therefore, comparing both Jetsons, the Jetson NX Xavier performed better than the Jetson Nano when scaling the number of customers. Later on, we will show the performance of the Jetsons relative to the percentage of Central Processing Unit (CPU) and Random 351 Access Memory (RAM) used by the Jetson during the experiments.

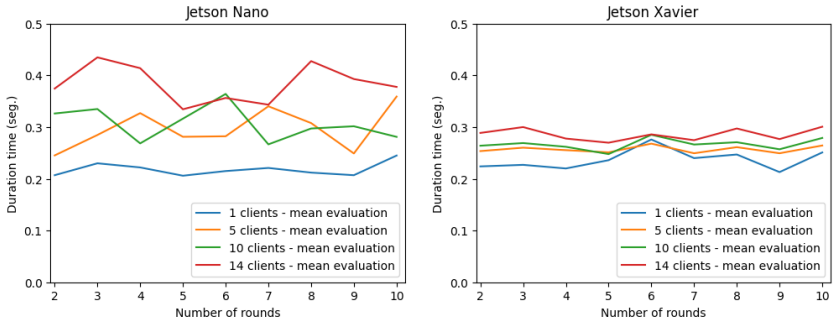


Figure 9. Model evaluation time for each round for the client.

5.3. Communication Delay Evaluation

In the evaluation of the model transfer times between the client and the server, a system with only one server and one client was created. On the left side of Figure 10, we can observe the result for a system with one client on a Jetson Nano and on the right side a system with one client on a Jetson NX Xavier. In this specific case, since the figures only contain two curves, we show the mean and 95% confidence intervals of the ten runs. When comparing the results for the two Jetsons, we can observe that both had a fairly fast upload and download time of only a few milliseconds; we can also observe a slightly better performance on the Jetson Nano.

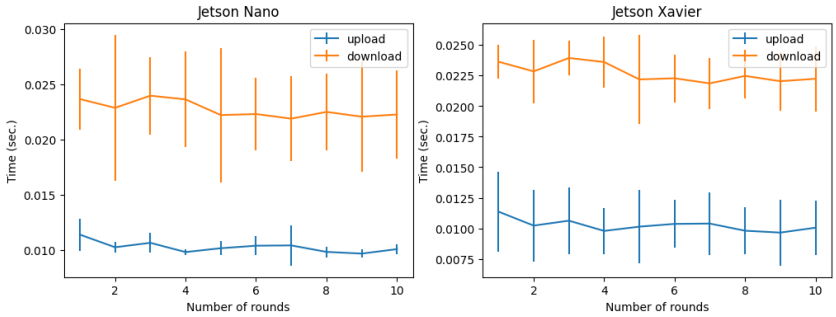


Figure 10. Model upload and download times.

After evaluating the performance of the federated learning (FL) solution itself, we now take a closer look at the resource consumption of the edge and server devices.

5.4. Server Resource Overhead

To better understand how Jetsons and Server performance behaved, we measured each device's Random 351 Access Memory (RAM) and Central Processing Unit (CPU) usage over several tests. To gain more information about the Central Processing Unit (CPU) and Random 351 Access Memory (RAM) used on the server, the Python library '*psutil*' (https://psutil.readthedocs.io/en/latest/, accessed on 22 November 2022) was used. This library provides an API for obtaining information about operating systems, such as Central Processing Unit (CPU) usage, memory usage, processes, etc.

In Figure 11, below, it is possible to observe that the Random 351 Access Memory (RAM) usage values on the server with the addition of more clients to the system do not interfere with the performance of the machine; in the use of one client or 70 clients, the percentage of Random 351 Access Memory (RAM) used is always about 5.5%. The only difference with the addition of clients is the increase in the emulation time. Regarding the analysis of the percentage of Central Processing Unit (CPU) used in the server (Figure 12), we can observe that at the beginning, when the clients are connected to the server, there is a large peak, and then there are only some smaller peaks when the aggregation of the client models happens, these peaks are around 30%. This means that the server has no difficulty

in running the processes, and even with the addition of new clients, it does not interfere with the performance of the server.

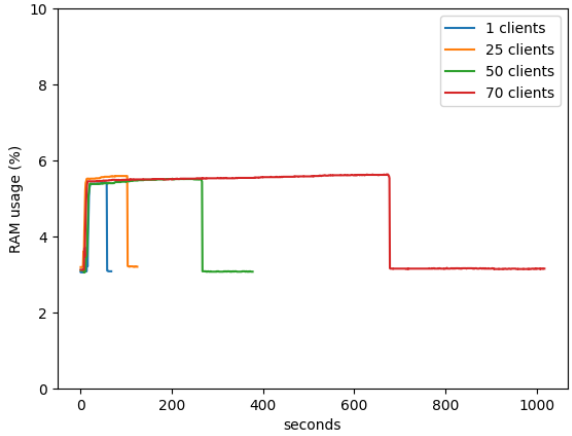


Figure 11. Server RAM consumption.

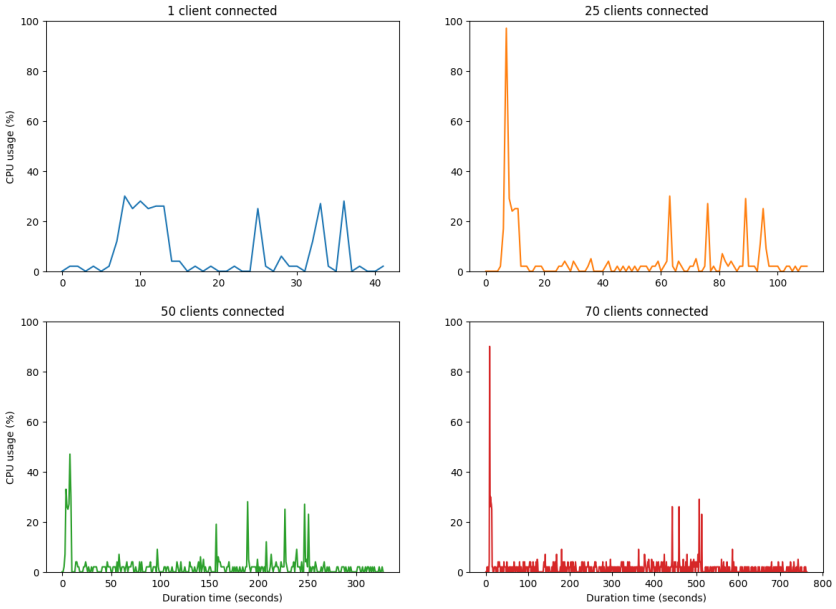


Figure 12. Server CPU consumption.

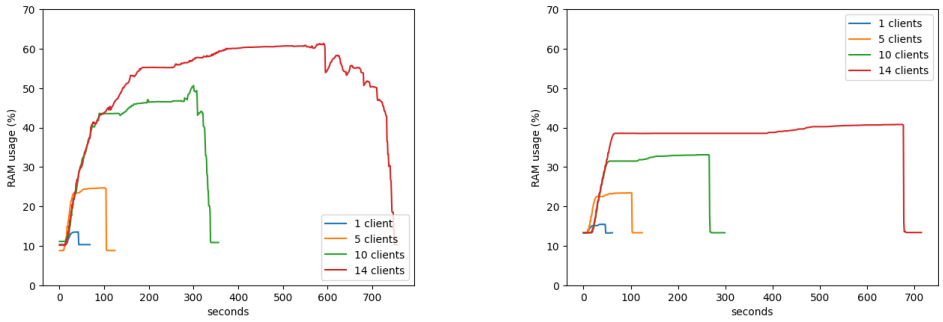
5.5. Jetson's RAM Consumption

We now discuss the performance of the edge devices used in our testbed, NVIDIA Jetson Nano e Xavier. We start by analyzing the Random 351 Access Memory (RAM) consumption. To do so, we used the *'free -t -m'* command line available on the Linux-based operating systems, in our case the Ubuntu distribution.

In Figure 13a, the percentage usage of Random 351 Access Memory (RAM) is shown. The Jetson Nano had difficulty in terminating the connection with the server when it had 14 clients connected because a vertical line normally indicates that all clients are terminated simultaneously, but instead of a vertical line, it shows a gradual decrease in the percentage with many ups and downs. When analyzing the containers during this time, it is also possible to observe that the client containers were being disconnected one at a time over a long time.

Considering the Jetson NX Xavier (Figure 13b), on the other hand, the performance was much better, when compared to the Jetson Nano. The percentage of the Random 351 Access Memory (RAM) used during the tests was 40% instead of 62% for the same number of clients created on the device (14 clients). After the end of the 10 rounds of aggregation,

the clients on the Jetsons closed the server connection. We can observe that the display of the vertical line means that it instantaneously terminated the connection of the 14 clients. It can also be observed that Jetson NX Xavier terminated all client connections to the server before 700 s, while in the Jetson Nano, all connections were only disconnected after 700 s.

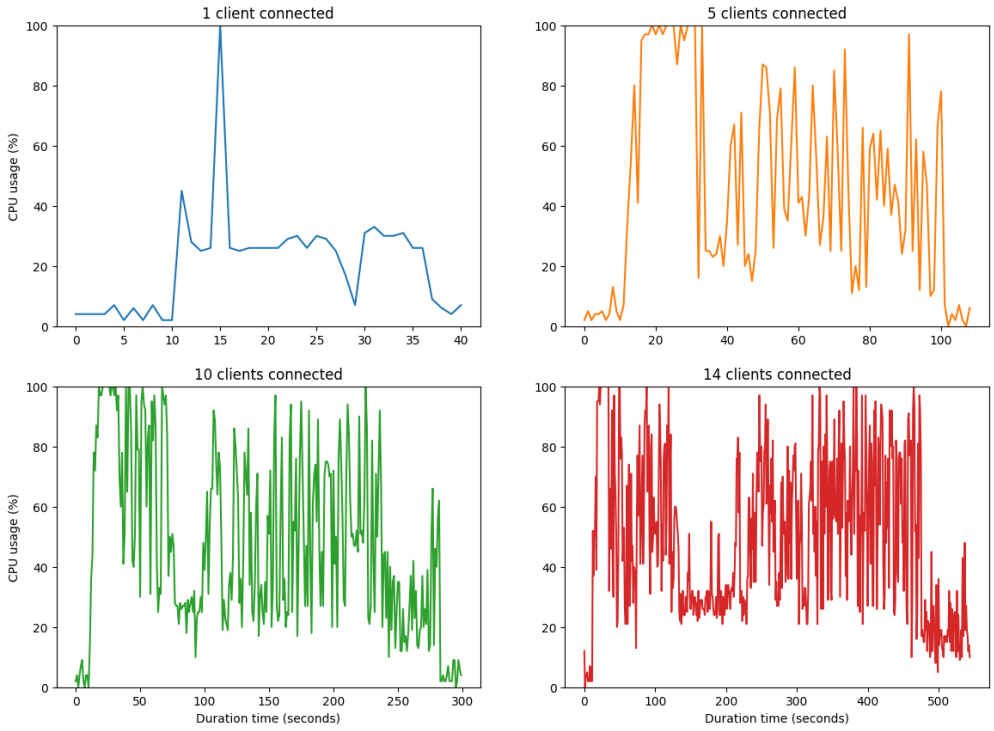


(**a**) Jetson Nano. (**b**) Jetson NX Xavier.

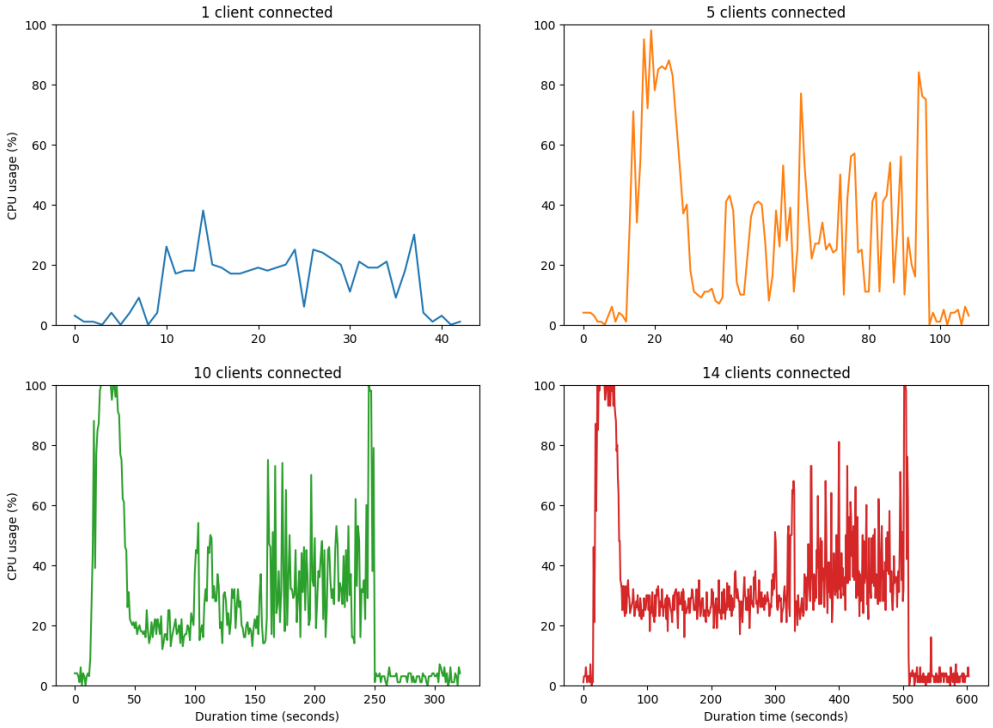
Figure 13. RAM consumption.

5.6. Jetson's CPU Consumption

The results obtained with the Jetson Nano showed that the device's performance became critical, reaching very high values in the percentage of Central Processing Unit (CPU) used, as shown in Figure 14a. To measure the CPU utilization, we used the *'/proc/stat'* script located in the /proc directory of the Linux-based distributions. In the Jetson Nano, the usage percentage of the Central Processing Unit (CPU) often came close to 100%, even staying at 100% for quite some time when various clients were being created on the device. These high values led to a low-performance value of the Jetson Nano, so the maximum number of clients that can be created was 14 clients because, after this number, the Jetson Nano became excessively slow and eventually locked up.



(**a**) Jetson Nano.



(**b**) Jetson NX Xavier.

Figure 14. Jetson's CPU consumption.

Comparing the Central Processing Unit (CPU) performance of the two models of Jetsons, we can observe that it was better and more stable in Jetson NX Xavier (Figure 14b). Our tests confirmed what was expected. Due to its more robust configuration, the Jetson Xavier outperformed the Jetson Nano when scaling the number of clients on the edge device. This reinforced that, in our case study, the client should be seen as a federated learning (FL) service running on the edge device.

6. Conclusions

Federated learning is a promising candidate to address the gap in centralized ML solutions. It has lower latency and communication overhead when performing most of the processing on the edge devices; it improves privacy, as data do not travel over the network; and it facilitates the handling of heterogeneous data sources and expands scalability. To assess how federated learning (FL) can effectively contribute to smart city scenarios, we presented FedFramework and built a testbed that integrates the components of the city infrastructure, where edge devices such as NVIDIA Jetsons were connected to a cloud server. We deployed our lightweight container-based federated learning (FL) framework in this testbed where we evaluated the performance of devices, the effectiveness of machine learning and aggregation algorithms, the impact on the communication between the edge and the server, and the consumption of resources. To carry out the evaluation, we opted for the scenario in which we estimated vehicle mobility inside and outside the city, using the real data collected from the Aveiro Tech City Living Lab infrastructure in the city of Aveiro.

In the context of smart cities, several services are used in edge devices, where great diversity is expected in relation to the consumption of resources by these services. Taking into account the scale of a smart city, the important thing is to properly assess the scope of use in each one of them to have the best cost/benefit ratio, thus reaching an ideal project. In this way, our study contributes to the field by showing the limits of each of these options, facilitating the adequacy of services $\times$ devices. The better performance of Jetson Xavier compared with the Nano has its price; thus, the Nano can meet several situations where services with softer requirements can be used.

In future work, we aim to scale our framework to more edge devices and to mobile nodes such as OBUs.

Author Contributions: All authors designed the solution, analyzed the results, and wrote the paper. R.V. implemented the solution, prepared the evaluation scenarios, and evaluated the results. C.S., P.R., and S.S. supervised the entire research process. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by FEDER, through POR LISBOA 2020 and COMPETE 2020 of the Portugal 2020 Project CityCatalyst POCI-01-0247-FEDER-046119, and by the Urban Innovation Action EU/H2020 Aveiro STEAM City.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Chen, C.; Liu, B.; Wan, S.; Qiao, P.; Pei, Q. An Edge Traffic Flow Detection Scheme Based on Deep Learning in an Intelligent Transportation System. *IEEE Trans. Intell. Transp. Syst.* **2021**, *22*, 1840–1852. [CrossRef]
2. McMahan, H.B.; Moore, E.; Ramage, D.; Hampson, S.; Arcas, B.A. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS), Fort Lauderdale, FL, USA, 20–22 April 2017; pp. 1–10.
3. Valente, R.; Senna, C.; Rito, P.; Sargento, S. Federated Learning Framework to Decentralize Mobility Forecasting in Smart Cities. In Proceedings of the 36th IEEE/IFIP Network Operations and Management Symposium (NOMS 2023), Miami FL, USA, 8–12 May 2023.
4. Rito, P.; Almeida, A.; Figueiredo, A.; Gomes, C.; Teixeira, P.; Rosmaninho, R.; Lopes, R.; Dias, D.; Vítor, G.; Perna, G.; et al. Aveiro Tech City Living Lab: A Communication, Sensing and Computing Platform for City Environments. *arXiv* **2022**, arXiv:2207.12200.
5. Mayer, R.; Jacobsen, H.A. Scalable Deep Learning on Distributed Infrastructures: Challenges, Techniques, and Tools. *ACM Comput. Surv.* **2020**, *53*, 1–37. [CrossRef]
6. Bellavista, P.; Foschini, L.; Mora, A. Decentralised Learning in Federated Deployment Environments: A System-Level Survey. *ACM Comput. Surv.* **2021**, *54*, 3. [CrossRef]
7. Yang, Z.; Chen, M.; Wong, K.K.; Poor, H.V.; Cui, S. Federated Learning for 6G: Applications, Challenges, and Opportunities. *Engineering* **2021**, *8*, 33–41. [CrossRef]
8. Zhang, C.; Xie, Y.; Bai, H.; Yu, B.; Li, W.; Gao, Y. A survey on federated learning. *Knowl.-Based Syst.* **2021**, *216*, 106775. [CrossRef]
9. Ogundokun, R.O.; Misra, S.; Maskeliunas, R.; Damasevicius, R. A Review on Federated Learning and Machine Learning Approaches: Categorization, Application Areas, and Blockchain Technology. *Information* **2022**, *13*, 263. [CrossRef]
10. Hochstetler, J.; Padidela, R.; Chen, Q.; Yang, Q.; Fu, S. Embedded Deep Learning for Vehicular Edge Computing. In Proceedings of the 2018 IEEE/ACM Symposium on Edge Computing (SEC), Seattle, WA, USA, 25–27 October 2018; pp. 341–343. [CrossRef]
11. Chen, M.; Shlezinger, N.; Poor, H.V.; Eldar, Y.C.; Cui, S. Communication-efficient federated learning. *Proc. Natl. Acad. Sci. USA* **2021**, *118*, e2024789118.
12. Rahman, M.A.; Hossain, M.S.; Islam, M.S.; Alrajeh, N.A.; Muhammad, G. Secure and Provenance Enhanced Internet of Health Things Framework: A Blockchain Managed Federated Learning Approach. *IEEE Access* **2020**, *8*, 205071–205087. [CrossRef]
13. De Vita, F.; Bruneo, D. Leveraging Stack4Things for Federated Learning in Intelligent Cyber Physical Systems. *J. Sens. Actuator Netw.* **2020**, *9*, 59. [CrossRef]
14. Longo, F.; Bruneo, D.; Distefano, S.; Merlino, G.; Puliafito, A. Stack4Things: A sensing-and-actuation-as-a-service framework for IoT and cloud integration. *Ann. Telecommun.* **2017**, *72*, 53–70. [CrossRef]
15. Suzen, A.A.; Duman, B.; Şen, B. Benchmark Analysis of Jetson TX2, Jetson Nano and Raspberry PI using Deep-CNN. In Proceedings of the 2020 International Congress on Human–Computer Interaction, Optimization and Robotic Applications (HORA), Ankara, Turkey, 26–27 June 2020; pp. 1–5. [CrossRef]
16. Anh, P.T.; Duc, H.T.M. A Benchmark of Deep Learning Models for Multi-leaf Diseases for Edge Devices. In Proceedings of the 2021 International Conference on Advanced Technologies for Communications (ATC), Ho Chi Minh City, Vietnam, 14–16 October 2021; pp. 318–323. [CrossRef]
17. James, N.; Ong, L.Y.; Leow, M.C. Exploring Distributed Deep Learning Inference Using Raspberry Pi Spark Cluster. *Future Internet* **2022**, *14*, 220. [CrossRef]
18. Baghersalimi, S.; Teijeiro, T.; Atienza, D.; Aminifar, A. Personalized Real-Time Federated Learning for Epileptic Seizure Detection. *IEEE J. Biomed. Health Inform.* **2022**, *26*, 898–909. [CrossRef] [PubMed]
19. Zhang, T.; He, C.; Ma, T.; Gao, L.; Ma, M.; Avestimehr, S. Federated Learning for Internet of Things. In Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems (SenSys '21), Coimbra, Portugal, 15–17 November 2021; Association for Computing Machinery: New York, NY, USA, 2021; pp. 413–419. [CrossRef]
20. Mills, J.; Hu, J.; Min, G. Communication-Efficient Federated Learning for Wireless Edge Intelligence in IoT. *IEEE Internet Things J.* **2020**, *7*, 5986–5994. [CrossRef]

21. Gao, Y.; Kim, M.; Abuadbba, S.; Kim, Y.; Thapa, C.; Kim, K.; Camtep, S.A.; Kim, H.; Nepal, S. End-to-End Evaluation of Federated Learning and Split Learning for Internet of Things. In Proceedings of the 2020 International Symposium on Reliable Distributed Systems (SRDS), Shanghai, China, 21–24 September 2020; pp. 91–100. [CrossRef]
22. Kim, J.; Kim, D.; Lee, J. Design and Implementation of Kubernetes enabled Federated Learning Platform. In Proceedings of the 2021 International Conference on Information and Communication Technology Convergence (ICTC), Jeju Island, Korea, 20–22 October 2021; pp. 410–412. [CrossRef]
23. Jiang, Y.; Wang, S.; Valls, V.; Ko, B.J.; Lee, W.H.; Leung, K.K.; Tassiulas, L. Model Pruning Enables Efficient Federated Learning on Edge Devices. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, 1–13. [CrossRef] [PubMed]
24. Anwar, S.; Hwang, K.; Sung, W. Structured Pruning of Deep Convolutional Neural Networks. *J. Emerg. Technol. Comput. Syst.* **2017**, *13*, 32. [CrossRef]
25. Lin, T.; Stich, S.U.; Barba, L.; Dmitriev, D.; Jaggi, M. Dynamic Model Pruning with Feedback. In Proceedings of the International Conference on Learning Representations, Addis Ababa, Ethiopia, 26–30 April 2020.
26. Lym, S.; Choukse, E.; Zangeneh, S.; Wen, W.; Sanghavi, S.; Erez, M. PruneTrain: Fast Neural Network Training by Dynamic Sparse Model Reconfiguration. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC' 19), St. Louis, MO, USA, 17–22 November 2019; Association for Computing Machinery: New York, NY, USA, 2019. [CrossRef]
27. Wang, X.; Han, Y.; Leung, V.C.M.; Niyato, D.; Yan, X.; Chen, X. Convergence of Edge Computing and Deep Learning: A Comprehensive Survey. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 869–904. [CrossRef]
28. Pomšár, L.; Brecko, A.; Zolotová, I. Brief overview of Edge AI accelerators for energy-constrained edge. In Proceedings of the 2022 IEEE 20th Jubilee World Symposium on Applied Machine Intelligence and Informatics (SAMI), Poprad, Slovakia, 19–22 January 2022; pp. 461–466. [CrossRef]
29. Basford, P.J.; Johnston, S.J.; Perkins, C.S.; Garnock-Jones, T.; Tso, F.P.; Pezaros, D.; Mullins, R.D.; Yoneki, E.; Singer, J.; Cox, S.J. Performance Analysis of Single Board Computer Clusters. *Future Gener. Comput. Syst.* **2020**, *102*, 278–291. [CrossRef]
30. Nvidia. NVIDIA TITAN RTX. 2022. Available online: https://www.nvidia.com/en-us/deep-learning-ai/products/titan-rtx/ (accessed on 22 November 2022).
31. Google. Cloud Tensor Processing Units. 2022. Available online: https://cloud.google.com/tpu/docs/tpus (accessed on 22 November 2022).
32. Intel. Developer Kits with Intel® Xeon® D-2100 Processor Product Family. 2022. Available online: https://www.intel.com/content/www/us/en/developer/topic-technology/edge-5g/hardware/xeon-d-dev-kit.html (accessed on 22 November 2022).
33. Microsoft. Azure Data Box Edge. 2022. Available online: https://azure.microsoft.com/en-us/updates/announcing-azure-data-box-edge/ (accessed on 22 November 2022).
34. Intel. Intel® Movidius™ Neural Compute Stick. 2022. Available online: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-movidius-neural-compute-stick.html (accessed on 22 November 2022).
35. NVIDIA. Jetson Products. 2022. Available online: https://developer.nvidia.com/buy-jetson (accessed on 22 November 2022).
36. NVIDIA. NVIDIA® Jetson Nano™. 2022. Available online: https://developer.nvidia.com/embedded/jetson-nano (accessed on 22 November 2022).
37. NVIDIA. Jetson AGX Xavier Series. 2022. Available online: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier/ (accessed on 22 November 2022).
38. Almeida, A.; Brás, S.; Oliveira, I.; Sargento, S. Vehicular traffic flow prediction using deployed traffic counters in a city. *Future Gener. Comput. Syst.* **2022**, *128*, 429–442. [CrossRef]
39. Benidis, K.; Rangapuram, S.S.; Flunkert, V.; Wang, Y.; Maddix, D.; Turkmen, C.; Gasthaus, J.; Bohlke-Schneider, M.; Salinas, D.; Stella, L.; et al. Deep Learning for Time Series Forecasting: Tutorial and Literature Survey. *ACM Comput. Surv.* **2022**, *55*, 121. [CrossRef]
40. Hyndman, R.J.; Athanasopoulos, G. *Forecasting: Principles and Practice*, 3rd ed.; OTexts: Melbourne, Australia, 2018.
41. Willmott, C.; Matsuura, K. Advantages of the Mean Absolute Error (MAE) over the Root Mean Square Error (RMSE) in Assessing Average Model Performance. *Clim. Res.* **2005**, *30*, 79. [CrossRef]
42. Draper, N.R.; Smith, H. *Applied Regression Analysis*; Wiley-Interscience: Hoboken, NJ, USA, 1998.