Data imputation on IoT gateways using machine learning

Cinthya M. França*, Rodrigo S. Couto* and Pedro B. Velloso *†
* Universidade Federal do Rio de Janeiro - GTA/PEE-COPPE/DEL-Poli - Rio de Janeiro, RJ, Brazil
† LIP6, Sorbonne Université, Paris, France
Email: cfranca@gta.ufrj.br, rodrigo@gta.ufrj.br, velloso@poli.ufrj.br

Abstract—**IoT (Internet of Things) gateways receive data from thousands of sensors and send it to the cloud, which runs intelligent services. However, collected data might have missing or anomalous values due to various reasons, such as network problems, damaged sensors, or security attacks. Missing and noisy data can affect future decision-making, so IoT gateways need to transmit consistent data to the cloud. This work proposes a method to impute missing data on IoT gateways based on neural network regression. We validate this method using six years of weather data from a station located in Rio de Janeiro, considering different percentages of missing data. The results show that the regression models have more than a 0.92 R-squared score and low errors when predicting sensor measurements. Furthermore, we show that the neural network implementation can run on IoT gateways due to its short execution time and low memory utilization. Finally, we show that a single model performs well even when 50% of the data is missing, highlighting the proposed approach's generality.** [1]

I. Introduction

In the last years, there has been an increase in the number of IoT (Internet of Things) devices. These devices might have sensors that monitor, collect, and transmit data to the cloud to provide intelligent services to users in different scenarios, such as smart homes, factories, and hospitals [1]. Each of these applications has its requirements for integrity, correctness, and on-time delivery [2]. However, the growth of devices sending data over short time intervals leads to a massive amount of data traveling through the network. Hence, various challenges arise [2]–[5]:

- IoT systems are highly heterogeneous with various types of devices, shared data, available resources, and communication technologies;
- with a large number of IoT devices, lots of data are generated and transmitted over the network, which can lead to network congestion and thus packet loss;
- sensors usually have a low computational capability and are more vulnerable to failures or attacks such as physical impairment and data interception;
- sensors can be compromised and behave maliciously, sending inconsistent data, for example.

IoT applications may fail to achieve their goals if the challenges mentioned before are not addressed. Our work focuses on the last challenge. Collected data might present missing or anomalous values due to various reasons, such as network problems, damaged sensors, or security attacks. Such inconsistent data introduce errors in data analysis, modifying statistical estimators such as means and variances [6]. Therefore, to ensure high-quality services, it is imperative to identify all missing or anomalous data to minimize their impact on data analysis.

Traditionally, cloud servers perform the task of identifying missing and anomalous data. However, some applications require real-time answers, and waiting for a cloud server response can be unfeasible, given the amount of data, traffic, and network delay. It is thus interesting to consider implementing data storage and processing closer to the network's edge, such as IoT devices or gateways [4]. In this context, this work proposes a methodology based on neural network regression to deal with IoT data in gateways.

Our work addresses data imputation, which is the procedure of replacing missing data with estimated values. A record is a set of attributes related to a specific entity or subject. In IoT, a record contains data collected by sensors, such as temperature and humidity, in a specific timestamp. However, for various reasons, a record may have one or more attributes without value, which is called missing data. Although missing data is a widespread problem in IoT, most data analysis algorithms work only with complete records, with values in all attributes [7]. So, a typical approach is to eliminate incomplete records. However, this elimination may reduce the accuracy and reliability of the analysis. Our methodology overcomes these issues by finding any missing data and replacing it with a new value predicted by a neural network regression model.

Even though several works have addressed data imputation using machine learning [2], [8]–[11], including neural networks [12]–[14], they do not conceive techniques to run in processing-constrained IoT gateways at the edge of the network. As a consequence, most of these works are not concerned with processing time and memory usage. Therefore, we consider these two aspects in our methodology and evaluate their impact on the overall performance. Our results show that the neural network models employed perform well on various percentages of missing data. The predicted values are close to the actual values, presenting more than a 0.92 r-squared (R^2) score. Besides, our models have a short execution time and low memory utilization. Thus, our methodology fits in the IoT environment, respecting computational resources and energy constraints.

The remainder of this paper is organized as follows. Section II presents a brief literature review of missing data imputation methods. Section III describes the proposed methodology and neural network models, while Section IV describes the employed dataset. Section V presents the evaluation of our methodology, comparing the missing data imputation methods. Section VI presents a clustering application case study that uses the discussed methods to impute missing data. Finally, Section VII concludes this work.

II. Related Work

Although missing data is a prevalent problem in IoT, most data analysis algorithms do not perform well when there are missing values in the dataset [7], [11], [15]. A typical approach chosen by developers is to eliminate incomplete records (i.e., records where at least one attribute is missing). However, this approach results in information loss and may lead the application to have incorrect conclusions [14]. Hence, we need to impute missing data with appropriate values.

A straightforward method is mean imputation. It replaces the missing values with the average of all non-missing values in the dataset for the respective attribute. However, this method distorts the probability distribution of the attribute [15]. Hence, it does not provide values close to those expected in some cases [2]. Other simple methods include, for example, replacing the missing value with the median of all non-missing values in the dataset, random values, and zeros.

Many imputation methods employ machine learning techniques, such as kNN (k-Nearest Neighbors) [8], SVM (Support Vector Machine) [9], and K-means [10]. The authors in [10] propose imputing missing data using fuzzy K-means, a clustering method that assigns a record to more than one cluster. More precisely, each record has a membership function that describes the degree that this record belongs to a specific cluster. Therefore, the imputation uses information about membership degrees and cluster centroids' values. This proposal performs better than the mean substitution and basic k-means methods. In [2], the authors first use the K-means algorithm to separate sensors into groups, according to the collected values' similarities. They apply a PMF (Probabilistic Matrix Factorization) algorithm within each cluster, which is a probabilistic Bayesian method for factoring a matrix into two other matrices. Then, they recover the missing values of each cluster using the property of PMF that obtains the original matrix by calculating the product of two matrices, which correspond to the values collected by neighboring sensors. This approach performs better in terms of accuracy, errors, and execution time when compared to SVM and a deep neural network method. The work in [11] discovers the correlated n-sensors, filling the missing value with the measurement of the same hour collected by the sensor with the highest correlation. Although these clustering methods perform better than some well-known methods, such as kNN and SVM, they impute all missing data at once. It is unclear whether the clustering should run again whenever a record with missing data arrives in the database.

The work in [12] focuses on predicting indoor temperature values using as inputs other correlated weather attributes of close geographical locations from previous periods. They implement a time series solution with neural networks, Random Forest, and SVM. The authors adopt an online learning methodology that updates the model in each iteration. This approach is practical in a real-time scenario in which the entire dataset is not available. The work in [13] proposes a hybrid neural network with a genetic algorithm to predict the missing data, where the former is responsible for predicting the value. In its turn, the latter optimizes the weights of the neural network. The models proposed in [14] use neural networks and assume that a record has three attributes. They estimate the value of one attribute using the other two attributes of the same hour. However, they deal with one missing attribute at a time.

The works mentioned above focus on prediction quality, generally disregarding the processing and memory requirements. This concern is crucial to IoT, where the hardware is generally constrained. Only [2] and [14] consider the execution time of the proposed methods. The papers [2], [12] affirm, without experimental results, that the proposed methods can run on an edge computing platform (i.e., an IoT gateway), which has a lower processing capacity than a cloud server. However, none of these proposals analyze their memory footprint. Unlike the studies described above, our work imputes missing data as soon as it arrives at the IoT gateway before forwarding it to the application server in the cloud. An additional difference is that we measure both execution time and the amount of memory used by various methods, given that our approach runs in IoT gateways. Since gateways have less computational resources than cloud servers, imputation methods for IoT must consider processing and memory limitations.

III. Proposed methodology

Our work aims to identify missing data and to impute appropriated values at the IoT gateway before sending it to the application server at the cloud, ensuring a more efficient analysis. The definition of what is an appropriate value depends on previous data from a specific sensor. For example, a temperature of 5°C can be acceptable in one city but unexpected in others. Therefore, we should consider previous measurements from a sensor when choosing a value to impute.

A. Considered Scenario

In our methodology, each gateway trains a machine learning model using data received by sensors, and then this model is used to impute missing data. Fig. 1 shows the scenario where our methodology runs. Each sensor collects data and sends it to a gateway, responsible for gathering all these data into a single record per timestamp. More precisely, each record is a set of attributes collected by sensors.

A record can have missing data (i.e., lack of attributes collected by sensors) at a given timestamp. For example, in Fig. 1, $Sensor$ 1 failed to send the humidity attribute. One approach is to ignore the entire record. A problem with this approach is that a record can have a few missing values due to damaged sensors. Discarding the entire record ignores all other valid measurements collected by other sensors. In terms of reliability, deleting these records can impair the damaged sensor identification, preventing the infrastructure manager from tracing back such sensor. Besides, deleting incomplete records reduces the number of valid samples to feed data-analysis algorithms, as shown next in Section IV. It is thus essential to impute missing data with appropriate values to improve data analysis performance.

In our methodology, before sending a record to the cloud, the gateway imputes an appropriate value and marks it whenever there are any missing attributes. The marking allows the cloud application to identify the values that do

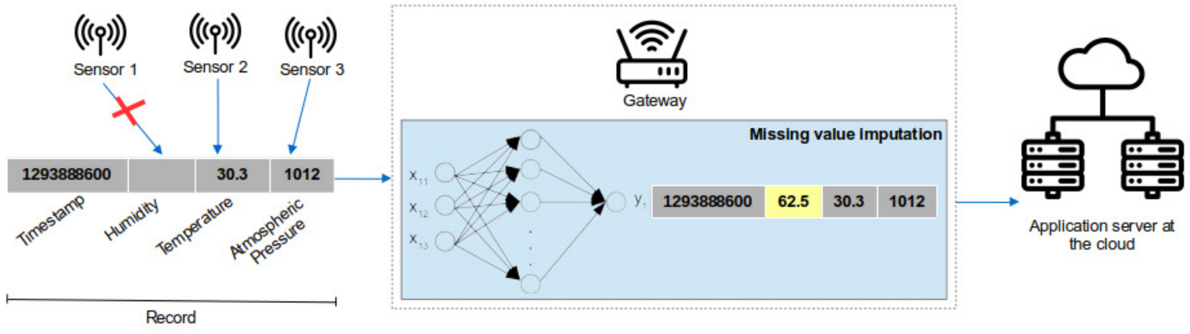

Fig. 1. Proposed methodology.

not correspond to actual measurements. When the gateway does not receive any of the attributes for a specific timestamp, it does not send it to the cloud to prevent increasing network traffic with a completely predicted record worthless for the application. In this case, empty records' predictions are only important to help the gateway improve its training.

B. *Neural Network Models*

The methodology proposed in this work is based on neural network regression to impute missing data received from sensors as soon as they arrive at the gateway. We employ two neural network models, based on MultiLayer Perceptron, to predict the expected measurements of each attribute. These models have one hidden layer each and differ from each other by the number of inputs. Each input consists of previous measurements of the respective attribute. We use a model with only one hidden layer to allow its employment in resource-constrained IoT gateways. Each hidden layer's neuron computes a linear combination of the inputs and the weights and passes through an activation function, before sending the results to the output layer. Subsequently, the output layer computes a linear combination of the activation function's results, evaluating the predicted value.

The first neural network ($NN1$) has as input the measurements of the three hours that immediately precede the current hour, which may represent a local trend. As we show in this work, this model consumes few gateway resources since it needs only the three previous values. The second neural network ($NN2$) has four input values. $NN2$ uses the same three inputs of $NN1$, plus the measurement of the same hour of the previous day, assuming that it may be a good estimate of the measurement of the current hour.

This work uses Scikit-learn 0.24.1 [16], a machine learning library for Python that implements neural networks. We empirically choose the parameters needed by Scikit-learn for creating the models as follows. We compute the Coefficient of Determination (R^2 Score) and the Root Mean Squared Error (RMSE) using the original values from the test dataset and the predicted values. Hence, we choose the combination of parameters with higher R^2 Score values. Both models use the same set of parameters. The solver for weight optimization is Adam [17], and the activation function is ReLU [18], with a constant learning rate. The maximum number of iterations is 500, and the training terminates when the validation score is not improving for a defined number of consecutive iterations. Table I lists the values of each Scikit-learn parameter.

TABLE I
PARAMETERS OF NEURAL NETWORK MODELS.

Parameter	Value	Parameter	Value
activation	ReLU	momentum	0.9
alpha	0.0001	n_iter_no_change	4
batch_size	auto	nesterovs_momentum	True
beta_1	0.9	power_t	0.5
beta_2	999	random_state	1
early_stopping	True	shuffle	True
epsilon	1e-08	solver	Adam
hidden_layer_sizes	(100)	tol	0.0001
learning_rate	constant	validation_fraction	0.1
learning_rate_init	1	verbose	False
max_fun	15000	warm_start	False
max_iter	500		

IV. MonitorAr Dataset

This paper uses the $MonitorAr$ [19] dataset, which keeps data on Rio de Janeiro's air quality, measured per hour at stations located in various city locations. This dataset contains a historical series of air quality and meteorological attributes. We use atmospheric pressure (Pres) [mbar], temperature (Temp) [°C], and relative humidity (RH) [%] attributes in this work.

The experiments conducted in this work use data from 2011 to 2019 of a station located in a neighborhood named *São Cristóvão*. This station has records with missing data and gaps between timestamps (i.e., consecutive hours when no data were collected). Hence, before the analysis, we preprocess the dataset by inserting empty records to fill the timestamps where no data is received. After the preprocessing, each day has 24 records (i.e., one per hour), and then each year has 8,760 or 8,784 records, depending on whether it is a leap year or not.

Table II shows the percentage of missing values for each attribute over the years in *São Cristóvão* station. More specifically, this table shows the number of records without a given attribute over the total number of records expected for each year. The analysis of Table II shows that missing data is a severe problem in the employed dataset, especially in the years from 2017 to 2019. Considering the entire interval from 2011 to 2019, the number of records with at least one missing attribute, not shown in the table, represents 16.8% of the total number of records. If we delete these incomplete records, the sensors' measurements are lost, and then the application will have less information to analyze. Hence, it shows the importance of data imputation.

The analysis consists of varying the percentage of missing data by removing valid measurements to verify our method-

TABLE II
PERCENTAGE OF MISSING DATA OF *São Cristóvão* STATION (%).

Attribute	2011	2012	2013	2014	2015	2016	2017	2018	2019
Pres	0.96	1.42	4.41	5.08	2.40	4.94	0.35	0.76	3.05
Temp	0.74	0.54	3.38	4.59	2.71	5.73	1.67	30.78	52.36
RH	0.78	0.54	3.96	4.54	2.71	5.54	8.96	44.26	75.49

ology's effectiveness. Thus, our analysis only uses data from 2011 to 2016 since the percentage of missing data from 2017 to 2019 is too high to let us introduce more missing data for testing. Table III shows statistics of the dataset considering all the data from 2011 to 2016. The CV column represents the coefficient of variation, which is the standard deviation divided by the mean, enabling a comparison between the attributes. For example, Atmospheric Pressure and Temperature attributes have close standard deviation values. However, as they have a different order of magnitude, we can see that Atmospheric Pressure variability is much smaller. This measure helps to understand the experimental results in Section V-B.

TABLE III
STATISTICS OF MONITORAR DATA (2011 TO 2016).

Attribute	Mean	Min	Max	Std. dev.	CV (%)
Pres [mbar]	1014.94	999.00	1032.33	4.81	0.47
Temp [°C]	27.24	8.07	49.08	4.98	18.28
RH [%]	73.42	11.98	100.00	17.17	23.39

Missing values can occur in isolation or bursts. The burst size is the number of consecutive missing values for each attribute. Fig. 2 shows the cumulative probability for each burst size, considering the entire interval from 2011 to 2016. This figure shows that a significant part of losses has small burst sizes. However, we still have a non-negligible number of large burst sizes. Consequently, a data imputation model must cope with different loss patterns.

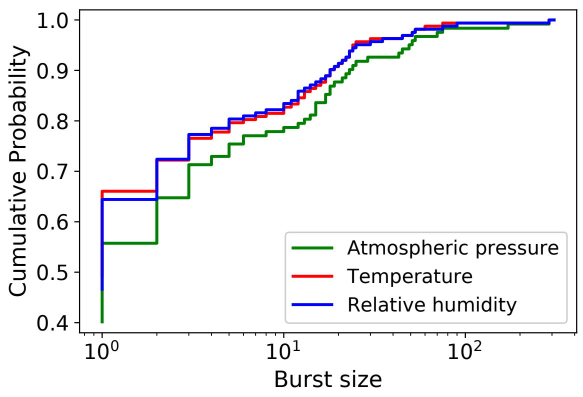

Fig. 2. Cumulative probability of burst size (2011 to 2016).

V. EVALUATION

This section presents the baseline methods and describes our experiments. Next, we present and discuss the results.

A. Baseline methods and experiment description

For imputing missing data in the dataset, we compare the neural network models of Section III-B with the following methods:

- **Repeat the last received value (*LastValue*)** - we replace the missing value with the last received value.
- **Average the values received *(Average)***- we replace the missing value with the average of all previously received values.

We choose to repeat the last value due to its simplicity and processing requirements since it only needs to store one value. Replacing the missing value by the average of the received values is also a simple and widely used imputation method. However, in our experiments, imputing values with the average was the worst method, as already expected [2], so we omit its results.

The experiments use data from 2011 to 2014 to train the neural networks and use data from 2015 to 2016 to test the methods. We also standardize the dataset to have zero mean and unit variance. Our experiment consists of verifying, for each method, how the percentage of missing data on the dataset affects the results. We evaluate each method's performance at different percentages (i.e., 5%, 10%, 25%, and 50%) of missing values. For example, if the complete dataset has 100 records, 10% means that we remove ten values for each attribute.

The original dataset already has missing data, as shown in Section IV. Hence, we first measure how much data is missing for each attribute. This measurement aims at knowing how much missing data we must introduce to achieve the desired percentage. Using a uniform distribution, we randomly select records with values for a given attribute to remove from the dataset. For a given missing data percentage, we perform this procedure 50 times, resulting in 50 samples of training and test sets for each percentage. We express all results with an average of 50 samples with a confidence interval of 95%.

Our evaluation simulates an online scenario. Each method predicts and imputes a missing value as soon as the gateway receives the record, respecting the timestamp order. Therefore, we only use records before the current timestamp to make the predictions.

B. Comparison of imputation methods

Figs. 3, 4, and 5 show the R^2 score for Atmospheric Pressure, Temperature, and Humidity, respectively. Each figure shows the results when varying the percentage of missing data on the test set and the training set simultaneously. The best possible R^2 score is 1, so the higher the score, the better the model fits the samples. It can also be less than zero when the model does not fit the data. We plot all results with 95% confidence intervals, although they are tiny for some figures.

$NN1$, $NN2$, and $LastValue$ have a high R^2 score for atmospheric pressure (Fig. 3). As shown in Table III, the variability (i.e., CV) of the atmospheric pressure is low. So, repeating the last received value can represent a good prediction. However, this characteristic does not apply to the other attributes. Consequently, for temperature (Fig. 4) and relative humidity (Fig. 5), the difference between $LastValue$

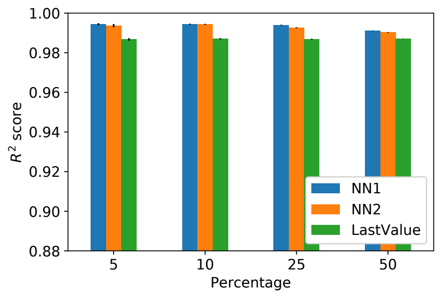

Fig. 3. R^2 Score of Atmospheric Pressure.

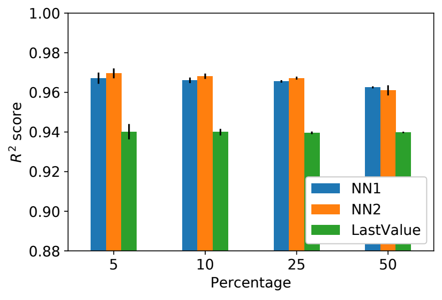

Fig. 4. R^2 Score of Temperature.

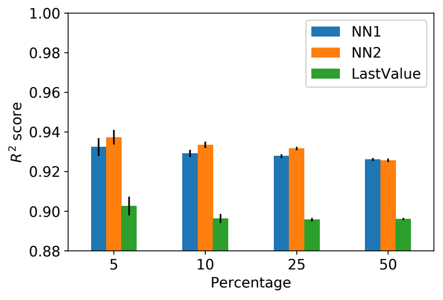

Fig. 5. R^2 Score of Relative Humidity.

and the neural networks is more significant than in the case of atmospheric pressure. Also, these results show that using the measurement of the last day does not help $NN2$ in the prediction. Hence, $NN1$ is simpler than $NN2$, but its performance is statistically close to $NN2$. Besides, we can see that the percentage of missing data has a low impact on the models' performance since the R^2 scores do not decrease as the percentage of missing data increases. We expect this behavior since the missing value events follow a uniform distribution.

Figs. 6, 7, and 8 complement the previous results, presenting the RMSE calculated for each attribute when varying missing data percentage. The best possible RMSE is 0. Therefore, the smaller the RMSE measure, the fewer errors the model has. The RMSE for the $LastValue$ method is higher than $NN1$ and $NN2$ in all attributes. For $NN1$ and $NN2$, the RMSE values are close to each other for all attributes for all tested percentages.

Our methodology aims to work on gateways, which have scarce computing resources. Consequently, it is also necessary to analyze the processing requirements and memory footprint beyond the prediction quality. The training phase is usually done in a cloud server since it consumes more processing requirements. However, it is essential to verify if an IoT gateway can train the models to allow applications that need continuous training. Therefore, we measure the execution time and memory utilization in both training and testing phases. The evaluation employs a Raspberry Pi 4 Model B with a 1.5 GHz 64-bit quad-core Arm Cortex-A72 CPU and 4 GB of RAM, running Raspberry OS, with a Linux kernel 5.10. We use Raspberry Pi since it is a typical choice in the literature of IoT Gateways [20], [21].

Table IV shows the execution time and the total allocated memory during the training and testing phases for all methods. In the training phase, we measure the processing time and memory that each neural network model needs to train all attributes using the training set. In the test phase, we measure the processing time and memory that each method needs to impute all missing data from the test set. Note that this is a worst-case analysis since we impute all missing data at once, while in an online scenario, both execution time and memory to impute one single record are much shorter. Each percentage of missing data consists of average values and 95% confidence intervals for 50 samples. We use the time and tracemalloc Python libraries for the execution time and the allocated memory, respectively.

Table IV shows that, in the training phase, both neural networks take less than 30 seconds and need less than 4,200 KiB of allocated memory. In the testing phase, the neural networks slightly increase the execution time compared to $LastValue$. The methods consume almost the same amount of memory. In an IoT scenario, gateways may have different hardware configurations. Therefore, we verify how slow the neural network models are when compared to the $LastValue$ method. This comparison allows a more general analysis. Fig. 9 shows the relative execution time, which is the execution time of the neural network model divided by the execution time of the $LastValue$ method. We can note that both neural networks are slower than the $LastValue$ method for all percentages. Their worst performance occurs when there are 5% of missing data, in which $NN1$ and $NN2$ are approximately 2.2 times slower than $LastValue$. However, as shown in Table IV, their absolute values are still small.

The results of Table IV and Fig. 9 show that, although the $LastValue$ method is faster than the employed neural networks, the latter is still competitive in terms of computational resources. Hence, the increase in memory and execution time is a small price to pay considering the higher R^2 score and lower RMSE offered by the proposed methodology. Therefore, the results confirm that IoT gateways can run both neural networks.

C. Training Analysis

In Section V-B, we assume that the test set has the same percentage of missing data as the training set. However, in reality, it is not possible to know *a priori* the percentage of missing data experienced by an IoT gateway. It means that, after choosing a training set with a specific missing data percentage, it is not possible to guarantee that the test set will present the exact percentage value. Therefore, in this section, using the same methodology of Section V-A, we train the neural networks using a fixed percentage of missing data and vary the percentage of missing data on the test set. The goal is to evaluate the difference in the percentage of missing data between the training and the test sets.

Fig. 10 indicates the R^2 score when we train the neural networks considering 5% and 50% of missing data. We show only the Relative Humidity attribute for conciseness because it presents worse R^2 scores than the other attributes. Also, we omit the RMSE results for all attributes since they are similar to Figs 6 to 8. Despite this, the behavior for all attributes and RMSE in this analysis leads to the same conclusions. The $LastValue$ method does not need any training, but we show its R^2 scores for comparison purposes.

Fig. 10 shows that the neural networks models have similar R^2 scores, regardless of whether the training set considers 5% or 50% of missing data. This experiment shows that despite training with 5% or 50% of missing data, the R^2

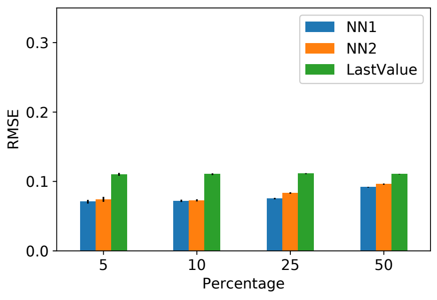

Fig. 6. RMSE of Atmospheric Pressure.

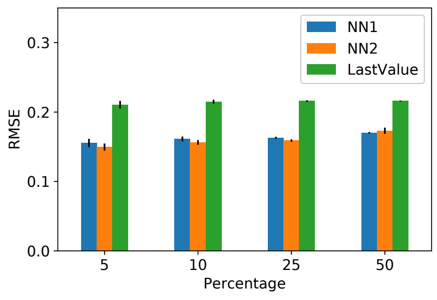

Fig. 7. RMSE of Temperature.

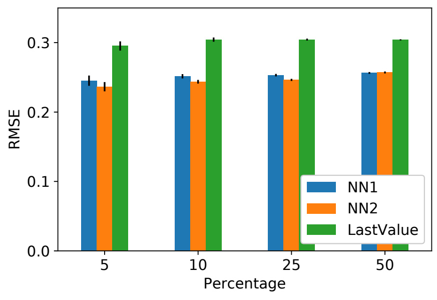

Fig. 8. RMSE of Relative Humidity.

TABLE IV
EXECUTION TIME AND TOTAL ALLOCATED MEMORY OF EACH METHOD, PER PERCENTAGE OF MISSING DATA.

		Training Phase		Test Phase		
Percentage	**Metric**	**NN1**	**NN2**	**NN1**	**NN2**	**LastValue**
5	**Execution Time (s)**	26.53 ± 0.66	28.82 ± 0.74	2.42 ± 0.01	2.47 ± 0.01	1.14 ± 0.00
	AllocMemory (KiB)	3908.46 ± 6.05	4190.00 ± 5.88	140.00 ± 0.90	140.00 ± 0.90	140.45 ± 1.31
10	**Execution Time (s)**	23.11 ± 0.69	24.29 ± 0.69	5.66 ± 0.01	5.72 ± 0.00	2.93 ± 0.00
	AllocMemory (KiB)	3903.48 ± 0.35	4187.76 ± 0.48	139.67 ± 0.24	139.67 ± 0.24	139.50 ± 0.24
25	**Execution Time (s)**	16.27 ± 0.34	12.88 ± 0.43	20.09 ± 0.06	20.16 ± 0.02	12.46 ± 0.01
	AllocMemory (KiB)	3904.10 ± 0.24	4188.50 ± 0.41	139.57 ± 0.11	139.57 ± 0.11	139.39 ± 0.11
50	**Execution Time (s)**	6.35 ± 0.23	5.02 ± 0.14	60.56 ± 0.14	61.12 ± 0.16	44.12 ± 0.14
	AllocMemory (KiB)	3907.08 ± 0.31	4194.15 ± 0.64	139.61 ± 0.16	139.61 ± 0.16	139.44 ± 0.16

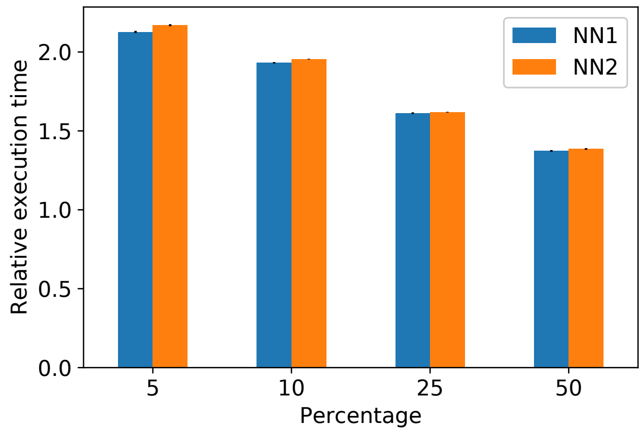

Fig. 9. Relative execution time - Test.

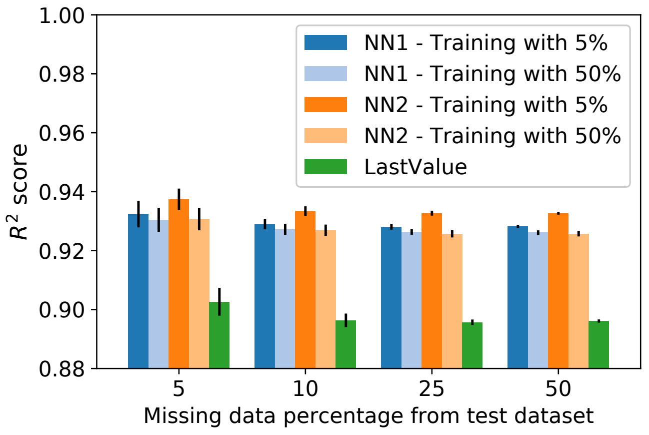

Fig. 10. R^2 Score of Relative Humidity - Training with 5% and 50% of missing data.

scores of both neural network models remain high and better than the $LastValue$ method, regardless of the missing data percentage in the test dataset. Therefore, it is an interesting result since, with only 50% of complete records available for training, the employed neural networks still provide good performance.

VI. CLUSTERING APPLICATION CASE STUDY

DbScan is an algorithm widely used in IoT, which clusters dataset records and detects outliers [22], [23]. However, this algorithm does not work when there are missing values in the dataset since it requires only complete records. Eliminating the incomplete records can reduce the analysis's reliability, justifying the need to impute missing data. This section shows a case study of a clustering application that uses DbScan. Our goal is to compare how each method for imputing missing data affects DbScan clustering and outlier detection.

For this experiment, we use two base datasets. In the first one, named $DropTest$, we delete a record from the dataset if it has missing values. In our case, 767 records have missing values, so we eliminate them. In the second dataset, named $PosInf$, instead of removing records with missing values, we use imputation methods. We then have three datasets: *PosInf-NN1*, *PosInf-NN2*, and *PosInf-LastValue* using, respectively, $NN1$, $NN2$, and $LastValue$ as imputation methods. Finally, we run DbScan on all four datasets.

Dbscan requires two parameters: `eps`, which specifies the maximum distance that two points must be apart from each other to be considered neighbors; `min_points`, which corresponds to the minimum number of neighbors a given point should have to be a core point. So, all neighbors within the `eps` radius of a core point belong to this core point cluster [22], [23]. As the $DropTest$ dataset has fewer records than $PosInf$, we consider it our baseline and use it to choose the `eps` parameter, calculating neighboring records' distances [22]. Consequently, `eps` is 0.556 in all Dbscan executions.

According to [23], the value of `min_points` is usually twice the dataset dimensionality (i.e., number of attributes). In our case, the recommendation is to set this value to six. However, for large datasets that may have much noise, higher `min_points` may improve the clustering results. We first fix the `min_points`, choosing an arbitrarily value of 72.

We want to verify if the imputation methods impair clustering and outlier identification by Dbscan. In other words, we want to see if the imputation makes some known valid records become outliers and if Dbscan classifies as outliers the records with imputed values. Table V shows the results of Dbscan for all datasets when `min_points` is 72. It shows the total records used by Dbscan for each dataset and the number of outliers detected. Besides, the column $\cap DropTest$ compares the $DropTest$ dataset with the others, showing how many outliers they have in common. In this case, it is interesting to note that *PosInf-NN2* has the same outliers as $DropTest$ and that *PosInf-NN1* and *PosInf-LastValue* are very close. Table V also shows the number of new outliers, namely, records that are not outliers in the $DropTest$ dataset but are outliers in the *PosInf* datasets. Our manual analysis of these results indicates that all new outliers are records with imputed data that do not exist on the $DropTest$ dataset. Other values of `min_points`, used in the subsequent analysis, lead to the same conclusions. Hence, imputing missing data does not make valid records become outliers.

TABLE V
COMPARISON OF DBSCAN RESULTS FOR EACH DATATEST, WHEN MIN_POINTS=72.

Dataset	Total	Outliers	∩ DropTest	New out.
DropTest	16777	207	207	-
PosInf-NN1	17544	217	206	11
PosInf-NN2	17544	215	207	8
PosInf- LastValue	17544	222	205	17

As we fix the `eps`, it is interesting to vary the `min_points` to analyze its impact. Therefore, we execute Dbscan with different values of `min_points` (i.e., 24, 72, 120, 240, and 720). As pointed before, the *PosInf* datasets contain 767 records with imputed values. We want to analyze how many records with imputed values Dbscan identifies as outliers in the *PosInf* datasets when we vary `min_points`.

Table VI shows the percentage of outliers in each *PosInf* dataset, using different values for `min_points`. We can observe that the percentage of records defined as outliers by Dbscan is smaller than 2.22%. Besides, the higher the `min_points`, the smaller the percentage of outliers inserted by the methods. The *PosInf-NN2* dataset presents the lowest percentage of outlier insertion for `min_points` up to 240. When `min_points` is 720, *PosInf-NN1* and *PosInf-NN2* present the same result.

TABLE VI
PERCENTAGE OF NEW OUTLIERS INSERTED AFTER IMPUTATION METHODS.

Min_points	PosInf-NN1	PosInf-NN2	PosInf-LastValue
24	2.09	1.30	2.22
72	1.43	1.04	2.22
120	1.04	0.78	1.96
240	0.78	0.65	1.69
720	0.26	0.26	0.65

The results confirm that the imputation methods based on neural networks provide accurate values without distorting the Dbscan's output. The $NN2$ model as an imputation method performs better than the others since it inserts only a few outliers for all `min_points` values. Also, the $NN1$ model is still better than the $LastValue$ method.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we addressed the problem of imputing missing data on IoT gateways. Our goal was to verify if simple neural network models perform well as imputation methods running on IoT gateways, which usually have limited resources. We validated our models using actual weather data from Rio de Janeiro, using different missing data percentages. Our results show that the neural network models present a high R^2 score and a low RMSE compared to another simple method. Also, the neural network implementations present a short execution time and require less than 140 KiB of RAM, suitable for IoT environments. Another interesting result is that we do not need to know *a priori* the expected amount of missing data. It is true since our neural networks perform well for different percentages of missing data on the test set, even when we fix the percentage of missing data on the training set.

We also presented a clustering application case study that confirms that the employed methods do not impact the original data distribution since it inserts a small percentage of outliers. Therefore, an application can use all sensor data using simple imputation methods instead of removing entire records. As future work, it would be interesting to verify if our methodology also identifies anomalous data in the dataset. Another direction is to remove values from the dataset using different probability distributions. This analysis can be interesting to study the performance of the neural networks for different failure patterns.

ACKNOWLEDGEMENT

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. It was also supported by CNPq, FAPERJ Grants E-26/203.211/2017 and E-26/211.144/2019, and FAPESP Grant 15/24494-8. We would like to acknowledge xnimrodx and Freepik from https://www.flaticon.com/ for the icons used in this paper.

REFERENCES

[1] D.-Y. Kim, Y.-S. Jeong, and S. Kim, "Data-filtering system to avoid total data distortion in IoT networking," *Symmetry*, vol. 9, no. 1, p. 16, 2017.

[2] B. Fekade, T. Maksymyuk, M. Kyryk, and M. Jo, "Probabilistic recovery of incomplete sensed data in IoT," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2282–2292, 2017.

[3] J. Canedo and A. Skjellum, "Using machine learning to secure IoT systems," in *IEEE 14th annual conference on privacy, security and trust (PST)*, 2016.

[4] J. Pan and Z. Yang, "Cybersecurity challenges and opportunities in the new "edge computing+IoT" world," in *Proceedings of the 2018 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, 2018.

[5] K. Siddique, Z. Akhtar, H.-g. Lee, W. Kim, and Y. Kim, "Toward bulk synchronous parallel-based machine learning techniques for anomaly detection in high-speed big data networks," *Symmetry*, vol. 9, no. 9, p. 197, 2017.

[6] P. Schmitt, J. Mandel, and M. Guedj, "A comparison of six methods for missing data imputation," *Journal of Biometrics & Biostatistics*, vol. 6, no. 1, p. 1, 2015.

[7] X. Yan, W. Xiong, L. Hu, F. Wang, and K. Zhao, "Missing value imputation based on gaussian mixture model for the internet of things," *Mathematical Problems in Engineering*, vol. 2015, 2015.

[8] O. Troyanskaya, M. Cantor, G. Sherlock, P. Brown, T. Hastie, R. Tibshirani, D. Botstein, and R. B. Altman, "Missing value estimation methods for dna microarrays," *Bioinformatics*, vol. 17, no. 6, pp. 520–525, 2001.

[9] F. Honghai, C. Guoshun, Y. Cheng, Y. Bingru, and C. Yumei, "A SVM regression based approach to filling in missing values," in *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*. Springer, 2005.

[10] D. Li, J. Deogun, W. Spaulding, and B. Shuart, "Towards missing data imputation: a study of fuzzy k-means clustering method," in *International conference on rough sets and current trends in computing*. Springer, 2004.

[11] I. P. S. Mary and L. Arockiam, "Imputing the missing data in IoT based on the spatial and temporal correlation," in *IEEE International Conference on Current Trends in Advanced Computing (ICCTAC)*, 2017.

[12] D. Paul, T. Chakraborty, S. K. Datta, and D. Paul, "IoT and machine learning based prediction of smart building indoor temperature," in *IEEE 4th International Conference on Computer and Information Sciences (ICCOINS)*, 2018.

[13] N. Al-Milli and W. Almobaideen, "Hybrid neural network to impute missing data for IoT applications," in *IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, 2019.

[14] M. Guzel, I. Kok, D. Akay, and S. Ozdemir, "Anfis and deep learning based missing sensor data prediction in IoT," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 2, p. e5400, 2020.

[15] A. Chong, K. P. Lam, W. Xu, O. T. Karaguzel, and Y. Mo, "Imputation of missing values in building sensor data," *Proceedings of SimBuild*, vol. 6, no. 1, 2016.

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[18] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[19] D. I. P. Passos. Dados horários do monitoramento da qualidade do ar - monitorar. [Online]. Available: https://www.data.rio/datasets/dados-hor%C3%A1rios-do-monitoramento-da-qualidade-do-ar-monitorar

[20] S. M. Raza, J. Jeong, M. Kim, B. Kang, and H. Choo, "Empirical performance and energy consumption evaluation of container solutions on resource constrained IoT gateways," *Sensors*, vol. 21, no. 4, p. 1378, 2021.

[21] A. Glória, F. Cercas, and N. Souto, "Design and implementation of an IoT gateway to create smart environments," *Procedia Computer Science*, vol. 109, pp. 568–575, 2017.

[22] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[23] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "Dbscan revisited, revisited: why and how you should (still) use dbscan," *ACM Transactions on Database Systems (TODS)*, vol. 42, no. 3, pp. 1–21, 2017.