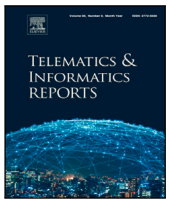
Telematics and Informatics Reports 22 (2026) 100315

Contents lists available at ScienceDirect

Telematics and Informatics Reports

journal homepage: www.elsevier.com/locate/teler



Neural networks-based accurate IoT data reconstruction ☆

Israel N. Matos [a], Gustavo B. Figueiredo [a], Maycon L.M. Peixoto [a], Praveen Kumar Donta [c], Schahram Dustdar [b], Cassio Prazeres [a,*]

[a] Institute of Computing, Federal University of Bahia, Bahia, Brazil
[b] Distributed Systems Group, TU Wien, Vienna 1040, Austria
[c] Department of Computer and Systems Sciences, Stockholm University, 16425 Stockholm, Sweden

ARTICLE INFO

Keywords:
Internet of Things
Time series reconstruction
Edge computing
Lightweight neural networks
Data aggregation
Edge AI

ABSTRACT

IoT gateways commonly aggregate sensor readings to reduce bandwidth and energy consumption, but this aggregation prevents cloud applications from accessing fine-grained historical data. This work investigates whether lightweight perceptron-based neural networks can reconstruct original sensor time series from temporally aggregated values with sufficient accuracy and efficiency. We propose a two-stage approach in which edge-trained perceptron models learn local sensor behavior, while a cloud-based model reconstructs full-resolution signals guided by aggregation averages. Extensive experiments were conducted using the Intel Berkeley Research Lab dataset, exploring multiple aggregation rates, window sizes, and training configurations, and comparing the proposed method against six classical interpolation techniques. Results show that while spline-based methods often achieve the lowest reconstruction error, perceptron models remain statistically competitive in terms of MSE and exhibit different runtime characteristics across deployment settings. These findings indicate that lightweight neural models represent a viable trade-off between accuracy and computational efficiency for IoT scenarios with constrained computational budgets.

1. Introduction

In the age of Big Data, diverse sources such as scientific tools, social networks, and IoT devices generate significant amounts of data [1–4]. The Internet of Things (IoT) already comprises more than 22 billion connected devices worldwide, and projections estimate that by 2025 this number will reach 75.44 billion devices, generating approximately 79.4 zettabytes (ZB) of data globally [5]. In the IoT context, data can be approached from different perspectives. For example, environmental information such as humidity, temperature, pressure, voltage, and luminosity, with high precision [6–8]. Recent studies also highlight that traditional static and cloud-centric AI pipelines struggle to cope with the dynamic, large-scale, and time-critical nature of data generated by IoT systems, motivating a shift towards more adaptive, decentralized, and autonomous intelligence at the edge [9].

IoT devices have limited resources, such as processing power, memory, storage, energy, bandwidth, etc., whereas the data generated by these devices is growing rapidly [10]. Therefore, dealing with this enormous amount of data is a significant challenge [11]. Due to these characteristics, Fog computing has emerged to manage this influx within the IoT, emphasizing edge-based processing through gateways. These gateways aggregate sensor data before transmission, reducing the need for cloud processing for every measurement. Data aggregation is necessary for several reasons [12], including reducing the amount of data transmitted over the network, thereby increasing energy efficiency and efficient bandwidth usage. It effectively addresses storage space, processing capacity, and memory issues. Finally, it eliminates unnecessary information and reveals insights that can improve the performance of data collection and analysis [12–14].

Several studies have proposed solutions to minimize network traffic to the cloud at the edge. Some approaches focus on data compression techniques [15], while others suggest sending only aggregated data to the cloud through strategies such as averaging [16,17]. Additionally, deep learning offers an interesting alternative strategy. For instance, Dridi et al. [18] explore the use of Recurrent Neural Networks (RNNs), particularly Long Short-Term Memory (LSTM) networks, to compress IoT-generated data by transmitting network weights and anomalies rather than raw data, thereby conserving bandwidth. Additionally, Yen et al. [19] address missing values in IoT-related time series data by

☆ This article is part of a Special issue entitled: 'Intelligent Systems and Paradigms' published in Telematics and Informatics Reports.

* Corresponding author.
E-mail addresses: israelnm@ufba.br (I.N. Matos), gustavobf@ufba.br (G.B. Figueiredo), maycon.leone@ufba.br (M.L.M. Peixoto), praveen@dsv.su.se (P.K. Donta), dustdar@dsg.tuwien.ac.at (S. Dustdar), prazeres@ufba.br (C. Prazeres).

https://doi.org/10.1016/j.teler.2026.100315
Received 24 November 2025; Received in revised form 10 March 2026; Accepted 12 March 2026
Available online 14 March 2026

Table 1
Research questions.

Number	Research Question
RQ1	Can perceptron neural networks learn the sensor dataset and perform time series interpolation with gaps?
RQ2	What is the comparative performance between the perceptron neural network and classical interpolation algorithms in recovering IoT data with gaps?
RQ3	If classical algorithms are better than the others, which performs best?
RQ4	Are there data scenarios, such as the employed aggregation rate, where one algorithm significantly outperforms the other?
RQ5	Is the performance difference between the perceptron neural network and classical algorithms substantial enough to influence the adoption of one method over the other?
RQ6	In addition to algorithm performance in terms of effectiveness, how does each method behave regarding interpolation time?
RQ7	Is there any trade-off between interpolation quality and efficiency?

comparing interpolation methods, including linear regression, support vector regression, and LSTM networks, to improve predictions. However, applications requiring original device-generated data require the ability to reconstruct historical data in the cloud.

Further, data compression methods are lossless (retain the original data) and lossy (use approximations to reduce data size) [20], which are another way to minimize data transmission in the network. Lossless techniques enable straightforward data recovery, which is crucial for preserving data integrity [21]. In contrast, lossy methods aim to reduce data size but require procedures to restore it to its original state [17]. The primary concern addressed is reconstructing time-series sensor data in the cloud from aggregated values sent from the edge of the IoT network [22], dealing with recovering detailed data from aggregated data.

In this paper, we propose a novel method for data reconstruction, even in the presence of information loss during aggregation, by leveraging a model that accurately captures the data's temporal behavior. Artificial Neural Networks (ANNs) have demonstrated the ability to generalize from a dataset, capturing the underlying structure of the data [23,24]. Additionally, Hornik et al. [25] mathematically demonstrated that multilayer feedforward networks, of which the perceptron neural network is an example, are universal approximators. The authors claim that their results establish that standard multilayer feedforward networks can approximate any measurable function with any desired degree of precision. They also argue that failures in applications can be attributed to insufficient learning, too few hidden units, or a lack of a deterministic relationship between inputs and targets. We demonstrate that multilayer feedforward networks can learn, consistently estimating the connection strengths that enable the approximations.

The literature indicates that LSTM networks are computationally expensive for time series analysis and forecasting [26,27]. Therefore, this work aims to investigate whether perceptron neural networks, which are relatively simple and computationally less costly, can perform interpolation in time series with gaps in IoT data. Additionally, to compare performance in terms of effectiveness and efficiency with the following classical interpolation algorithms: Linear Interpolation, Moving Average, Linear Regression, Polynomial Regression, Cubic Spline, and Nearest Neighbor. The research questions summarized in Table 1 have been defined to achieve the necessary answers to the objectives of this work.

In summary, this work addresses IoT infrastructure processes, specifically data transmission and retrieval techniques. In particular, the limitations of IoT devices are considered, including processor power, memory, storage, bandwidth, and energy, which may be affected by changes, particularly those requiring more processing power. In this study, we investigate whether perceptron neural networks can effectively learn from sensor datasets and perform time series interpolation in the presence of data gaps. Additionally, it determines, under various IoT scenarios, including different aggregation rates, which method is superior for recovering IoT data with gaps using perceptron neural networks compared to traditional interpolation algorithms. It also evaluates whether perceptron neural networks and classical algorithms differ substantially enough to influence the choice of interpolation method. Furthermore, the study evaluates the trade-offs between interpolation quality and efficiency for each method, focusing on the time required for interpolation.

This paper is organized as follows. Section 2 presents the related works. Section 3 describes the materials and methods used in this research. Section 4 presents results of data reconstruction using a perceptron neural network and a comparison with six interpolation algorithms. Finally, we conclude our work with a future scope in Section 5.

2. Related work

The primary research gap addressed in this work is the need for efficient, accurate methods to reconstruct time-series sensor data from aggregated values, given the resource limitations of IoT devices. While various approaches exist for data aggregation and compression, many are computationally intensive or depend on specialized hardware, which may not be feasible for all IoT scenarios.

IoT-based smart railroad management systems face challenges in storing, processing, and transmitting large volumes of data generated by distributed sensors, particularly in resource-limited edge devices. To address this, Garikipati et al. [28] propose an innovative data compression method leveraging Deep Reinforcement Learning (DRL). The method aims to provide an efficient, real-time data-compression solution tailored to smart railroad management scenarios. Using a Deep Q-Learning algorithm, it eliminates redundant data while preserving essential information, adapting to variable data patterns, and optimizing edge device performance by reducing storage and processing demands. Experimental results demonstrate that the method outperforms traditional approaches, achieving over 18% improvement in compression rates while maintaining critical information. These results highlight its effectiveness and applicability in smart railroad systems with constrained computational resources.

Huang et al. [29] addressed the problem of missing data estimation in time series, particularly in IoT applications where data loss may occur due to sensor failures, communication interruptions, and cost-effective sensing strategies. They propose an approach based on Echo State Networks (ESN) with an enhanced version that incorporates multiple reservoirs to capture temporal, cross-domain, and lag correlations. The authors suggest using an RNN variant, ESN, for missing-data estimation, trained via reservoir computing to capture complex temporal correlations in time series with exogenous variables. They also introduce a pre-filling strategy based on signal direction to improve estimation accuracy. Despite its promise, the ESN approach is computationally intensive, requiring advanced hardware and resources, and thus makes scalability and implementation in resource-constrained environments challenging. IoT devices with limited resources may

struggle to execute complex algorithms locally. Generalizing the models to different contexts and integrating them with existing systems also presents additional challenges.

Mohamed and Cavallaro [30] proposed an energy-efficient FPGA-based architecture for Wireless Sensor Networks (WSNs) to reduce the volume of transmitted data. This design leverages the spatiotemporal characteristics of sensor data and includes a hardware accelerator to optimize calculations. The framework employs data compression, outlier detection, and deep learning networks (DLNN) to explore spatiotemporal correlations and minimize communication between sensor nodes and aggregation points. It enables accurate data reconstruction with fewer than 30% of the original observations, achieving low latency and energy consumption. While effective in reducing data traffic and energy consumption, the FPGA-based solution faces challenges related to limited computational resources in network devices. Additionally, the initial development and setup costs may be high, but the architecture shows promise for broader application across various sensor network scenarios.

Ahmed et al. [31] explored the issue of sensor data scarcity in IoT ecosystems for monitoring elderly individuals in nursing homes. They proposed a missing-data imputation technique based on Bayesian Gaussian Processes (BGaP), which leverages prior knowledge from past observations to fill data gaps while accounting for temporal constraints. They tested the BGaP technique using real data collected over 2 years from sensors in the homes of 10 elderly individuals. The evaluation demonstrated its ability to impute all missing data, enabling the analysis of long-term behavioral changes that would otherwise go unnoticed. However, the BGaP method is computationally intensive, limiting its feasibility in resource-constrained or real-time IoT environments. It requires robust infrastructure, substantial processing power, and increased storage capacity, affecting operational costs and efficiency.

Zhang et al. [32] addressed the challenges of bandwidth and energy consumption in IoT networks arising from the transmission of large volumes of data. They proposed a solution combining compressed sensing and deep learning to develop a sparse-data reconstruction scheme that reduces transmission while maintaining reconstruction accuracy. The proposed method, DCSNet, employs two learning-based mapping stages: the first compresses raw sensor data into a latent domain, and the second reconstructs it in high-dimensional space. The approach was tested on six structured sparse datasets and a real sensor dataset, demonstrating effective reconstruction with low compression rates. However, the method's computational complexity and training time may require specialized hardware, such as GPUs, making it challenging to implement on resource-constrained IoT devices. Additionally, scalability and increased storage and processing demands could pose challenges for broader IoT network implementations.

Yen et al. [19] explored the issue of missing values in IoT time series, which often result from maintenance or sensor failures and can negatively impact the accuracy of predictions and analyses. The study compared interpolation methods, including linear regression, support vector regression, and LSTM networks, to estimate missing values. The researchers conducted a comparative analysis of these methods using a dataset from the Taiwan government to determine the most suitable approaches for various scenarios. The goal was to guide the selection of effective methods for handling missing values in IoT time series data. While the study provides valuable insights into prediction accuracy, it offers limited discussion on computational efficiency. Deep learning models like LSTMs require higher computational resources and longer processing times, posing challenges for resource-constrained IoT devices and systems with minimal infrastructure.

Cao et al. [33] addressed data aggregation in Wireless Sensor Networks (WSNs) to reduce energy consumption while maintaining data fidelity and confidentiality. They proposed a model that integrates rough set theory with an enhanced Convolutional Neural Network (CNN) for efficient data aggregation. The model employs rough set theory in Sink nodes to extract features and reduce data dimensionality, minimizing data transmission. The CNN enhances feature extraction efficiency, thereby reducing energy consumption and improving data aggregation accuracy. However, the method faces limitations in real-time IoT scenarios due to the complexity of feature extraction and the computational demands of CNNs, which require specialized hardware such as GPUs. Practical implementation may also be hindered by the need for advanced infrastructure and higher memory capacity, often unavailable in resource-constrained IoT devices.

Each of these studies offers valuable insights into data aggregation, compression, and missing-data imputation in IoT environments. However, they often involve computationally intensive techniques or specific hardware requirements that may not be suitable for all IoT applications. Our research focuses on the practical application of perceptron neural networks for interpolating time series data with gaps, offering a more computationally efficient solution than more complex models like LSTMs. This study is particularly relevant for applications in smart cities, industrial automation, and environmental monitoring, where accurate and efficient reconstruction of sensor data is crucial. Table 2 summarizes these related works and compares them according to the use of neural networks, their purpose, network type, environment, context, and application type.

3. Materials and methods

The following subsections present the data and tools used in this work, the implementation of the perceptron neural network and the interpolation strategies considered, the experimental setup and procedures, the evaluation metrics adopted to quantify performance, and the statistical analysis used to interpret the results.

3.1. Data and tools

Our experiments use a dataset made available by the Intel Berkeley Research Lab on their Web platform [34]. As shown in Fig. 1, this dataset comprises data from 54 sensors deployed at the Intel Berkeley Research Lab from February 28 to April 5, 2004. The sensors used for the collection are from the Mica2Dot series, each equipped with weather panels. They captured a wide range of information, including topology records with date and time stamps, as well as humidity, temperature, luminosity, and voltage measurements. These measurements were recorded at 31-second intervals, providing a rich, comprehensive overview of environmental conditions.

The data is available on the website in the "data.txt" file, totaling 150.91 MB. This file includes approximately 2.3 million readings collected from the 54 sensors installed in the laboratory. This file has eight columns: date: `yyyy-mm-dd`; time: `hh:mm:ss.xxx`; epoch: `int`; moteid: `int`; temperature: `real`; humidity: `real`; light: `real`; voltage: `real`. Temperature is degrees Celsius, and humidity represents the relative humidity corrected for temperature, ranging from 0 to 100%. Light is in Lux (1 Lux corresponds to moonlight, 400 Lux to bright office light, and 100,000 Lux to sunlight). Voltage is expressed in volts, typically 2–3; in this case, the batteries were lithium-ion cells that maintained a reasonably constant voltage throughout their lifespan.

We selected the Intel Berkeley Research Lab dataset for its widespread adoption in the literature and its suitability for controlled experimental evaluation. The dataset provides multimodal sensor measurements collected under stable environmental conditions, facilitating reproducibility and enabling the isolation of the effects of temporal aggregation on reconstruction performance. However, it represents a relatively controlled sensing environment, with limited noise, drift, and abrupt behavioral changes. As a result, this study presents the experiments as a baseline assessment. Evaluating the proposed approach under noisier, non-stationary conditions and across datasets with different temporal dynamics remains an important direction for future work.

We utilized perceptron neural networks to generate sensor behavior models from IoT data over time and compared them to classical interpolation algorithms. All experiments were conducted on a CPU-only machine (Intel Core i5, 8 GB RAM, SSD) using Python and TensorFlow.

Table 2
Synthesis of related works.

Reference	Did it use a neural network? What type?	How were neural networks used?	Environment	Context	Application type
Garikipati et al. (2024) [28]	Yes/DRL	Real-time data compression solution tailored to smart railroad management scenarios	IoT	Data compression	Smart Railroad
Huang et al. (2023) [29]	Yes/RNN	Missing Data Estimation in IoT Time Series	IoT	Missing Value Estimation	IoT and Time Series
Mohamed e Cavallaro (2022) [30]	Yes/DLNN	Reducing Data Transmitted in WSNs	WSNs	Data compression	Data Reduction with FPGA
Ahmed et al. (2022) [31]	No/Does not apply	Does not apply	IoT	Missing Values Estimation	IoT and Elderly Monitoring
Zhang et al. (2021) [32]	Yes/DLNN	Reducing Bandwidth Consumption in IoT Networks through Deep Learning and Compression	IoT	Sparse Data Reconstruction	IoT network and Reducing Bandwidth Consumption
Yen et al. (2020) [19]	Yes/LSTM	Comparison of Time Series Interpolation Methods IoT	IoT and Time Series	Missing Values Estimation	IoT and Time Series
Cao et al. (2020) [33]	Yes/CNN	Data Aggregation in WSNs with CNNs and Rough Set Theory	WSNs	Data Aggregation with CNNs	IoT
Our Work	Yes/Multilayer Perceptron	Missing Data Estimation in IoT Time and Comparison of Time Series Interpolation Methods IoT Series	IoT and Times Series	Missing Values Estimation	IoT and Time Series

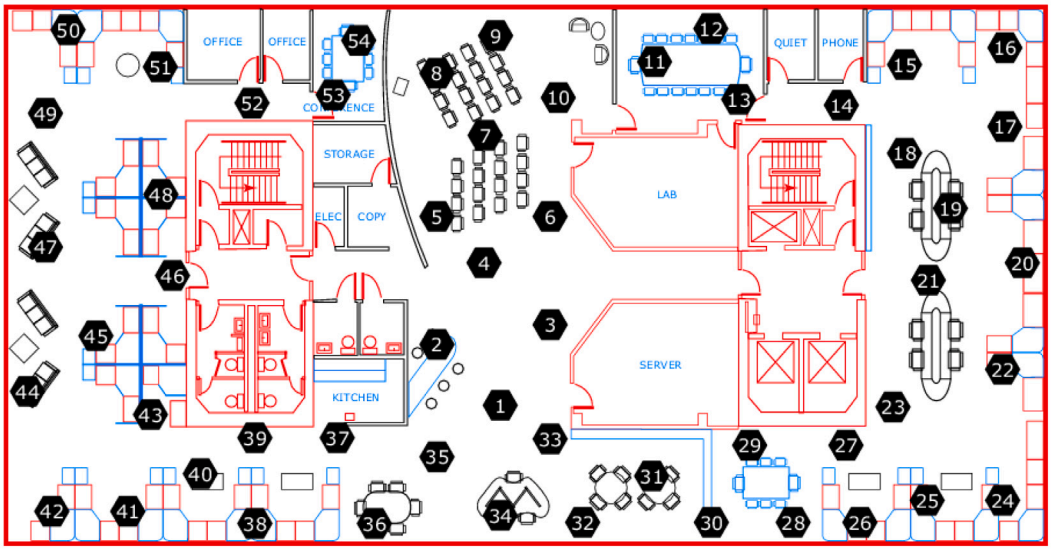


Fig. 1. Intel Lab Data (dataset): Graphical representation of the internal location of sensors in the building.
Source: Obtained from Madden [34].

3.2. Perceptron neural network and interpolation strategies

The perceptron model adopted in this work corresponds to a single-layer feedforward neural network trained for regression tasks. It employs a linear activation function, which is suitable for reconstructing continuous-valued sensor data. The training process minimizes the Mean Squared Error (MSE) using stochastic gradient descent, and the experimental setup defines the learning rate and momentum. This formulation ensures consistency between the mathematical model and the implementation used in the experiments.

We explored a variety of perceptron architectures by iteratively and incrementally developing neural networks. This study examined the learning abilities of these networks by varying the number of neurons in the data input layer, referred to as the "data window", along with other architectural parameters. Based on test results across various scenarios, the models can identify configurations that yield robust, effective results. Before applying the neural network to the interpolation problem, it was necessary to demonstrate that it could learn from the dataset. Initially, we measured the learning rate based on the loss function, then evaluated it based on the network's ability to predict future values. The difference between the network's predicted future values and the actual values in the dataset was measured using MSE (Mean Squared Error). Due to the distinct nature of environmental measurements (temperature, humidity, luminosity, and voltage), we found that each data type required a specialized neural network. Thus, the most suitable

parameters were adjusted for each set, thereby optimizing learning for each data type.

In the proposed work, the neural network consists of two layers: an input layer with n neurons, where n denotes the number of input measurements or "data windows", and an output neuron. The Stochastic Gradient Descent (SGD) algorithm was used with the MSE cost function to calculate error reduction. For this purpose, we set the parameters lr (learning rate) and $momentum$ to $lr = 10^{-6}$ and $momentum = 0.9$, based on prior literature. These values are part of the Keras library's compile function, which is used in this work to implement the neural network. In the proposed work some of the parameters need to be fine-tuned for the network to learn the different types of data: the minimum number of observations required for training, the size of the data window representing the number of values simultaneously considered by the input layer neurons to predict the next data point, the size of the buffer for data shuffling during network training, the batch size of data, and the number of training epochs. Such adjustments directly influenced neural networks' ability to learn the dataset and generate accurate predictions.

After completing the configuration stage, we trained each properly parameterized perceptron neural network, one for each data type. As a result, we generated four neural networks capable of predicting future values for temperature, humidity, luminosity, and voltage. These networks were subjected to various prediction tests to verify, based on MSE, if the accuracy was adequate and if there was overfitting. With this stage completed, it was possible to proceed with the following steps, specifically using these neural networks to interpolate time series data curves with gaps.

This proposed work showed that the perceptron neural network could learn from input data and make predictions for future data; thus, it was possible to continue developing a neural network capable of interpolating curves with data gaps. As a first strategy, we reconstructed the data without using average values; therefore, the neural networks attempted to rebuild the entire data curve from scratch and filled in missing data solely with predicted values. To allow the neural network to start the interpolation processing, only a set of raw data with the size of the data window, which is the quantity the network needs to predict the next value. Thus, the strategy described below details.

We created a vector with the same size as the curve and initialized all its elements to zero. Thus, if the curve contained 1000 values, we created a vector of length 1000 initialized to 0. Next, we filled the first n positions of the vector with n absolute values from the sensor measurements (where n corresponds to the data window size), and the network then attempted to construct the remaining part of the curve. For this, the network received the set of n sequential values to predict the next value: the value at position $n+1$. Then, using the values from positions 2 to $n+1$, the network would try to predict the value $n+2$ until completing the interpolation.

This strategy was ineffective, even after adjusting the network parameters. This naive strategy failed to allow the same neural network, capable of making predictions, to interpolate values across data gaps. Although this result is a failure, it aligns with existing knowledge in neural networks, which suggests that correctly functioning neural networks should be able to abstract datasets without memorizing them completely. Additionally, this result served as a negative control for the interpolation strategy, using the aggregation averages. The second interpolation strategy was employed, this time using the averages of the data sent to the cloud to address the failure of the previous algorithm. In the previous strategy, we initially created a vector of null values with the same size as the dataset to be interpolated. Then, we inserted the average values into the appropriate positions of the vector, following the principle that the ath aggregation value is placed in the pth position of the vector, where $p = a \times ar$ and ar represents the data aggregation rate.

In this study, we adopted temporal averaging as the aggregation mechanism due to its simplicity, broad adoption in IoT systems, and ease of interpretation. However, the proposed reconstruction framework does not inherently restrict aggregation to mean-based methods. From a methodological perspective, we can integrate any aggregation function that produces representative summary values over fixed temporal windows, such as minimum, maximum, median, weighted averages, or exponentially weighted statistics, into the same reconstruction pipeline. In such cases, we place the aggregated values at their corresponding temporal positions and use them as structural anchors to guide the interpolation process. Investigating how alternative aggregation mechanisms affect reconstruction accuracy and stability represents an important direction for future work and would further enhance the practical relevance of the proposed approach.

Algorithm 1 Interpolation using Perceptron Neural Network

```
Initialize an empty vector V with the length equal to the number of observations.
Set all positions in V to null.
Let n be the window size.
Fill the first n positions in V with the first n measured values from the sensors.
Insert the aggregated average values into their respective positions in V.
i ← 0
while V is not fully filled do
    j ← i + n
    Predict value at position j + 1 using the neural network model and the values from V[i] to V[j]
    if V[j + 1] is null then
        V[j + 1] ← predicted value
    end if
    i ← i + 1
end while
```

Algorithm 1 describes our interpolation process using the perceptron neural network. We initially filled the vector with the first n absolute values of the raw data, replacing any null values or averages that would otherwise appear in those positions. These n values guided the neural network at the beginning of the interpolation process, since the perceptron relied on a window of n values to predict the next one. The network received the sequence of n consecutive values and predicted the value at position $n+1$. If position $n+1$ contained a null value, the network replaced it with the predicted value. If the position contained an average value, we preserved it. Next, the network advanced by using the values from positions 2 to $n+1$ – which could include average values or previously predicted values – to predict the value at position $n+2$, and the process continued iteratively. Using this strategy, the network accurately interpolated the entire test dataset.

3.3. Experimental settings and procedures

We initially performed an exhaustive search over the parameter combination space to identify the best parameter set. To do so, we first determined the parameters that most strongly influence interpolation quality. Through empirical analysis, we identified the following parameters as having the most significant impact on the interpolation results: the aggregation rate, the number of observations used to train the neural network, the size of the neural network input window, the data shuffling rate during training, and the number of epochs, which defines how many times the network processes the entire dataset during training.

After identifying the most influential parameters, we defined a discrete set of values for each one. We selected these values based on preliminary experiments to cover a representative range of configurations while remaining computationally feasible. In particular, we chose aggregation rates as multiples of short time intervals, which allowed us to evaluate progressively coarser temporal resolutions. Table 3 summarizes the complete set of evaluated parameter values.

Next, we created a loop to evaluate all possible parameter combinations, training and testing the neural network under each specific configuration. For every parameter combination, we repeated the training and testing process 10 times to determine how many runs of the

Table 3
Simulation parameters and evaluated values.

Parameter	Evaluated values
Aggregation rates (ar)	2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60, 120, 180
Number of observations (O)	1000, 2000, 5000, 10,000
Window size (n)	10, 20, 30, 40, 50
Training epochs (Ep)	100, 500, 1000, 1500
Batch size	32
Shuffle buffer size	1000
Learning rate (lr)	10^{-6}
Momentum	0.9
Iterations per configuration	100

same configuration were necessary to obtain a correctly interpolated curve within a reasonable accuracy margin.

At the end of each simulation iteration, we calculate the MSE to assess how closely the interpolated curve matches the original sensor data curve. We then record these results in a simulation data file. We executed 10,400 iterations to evaluate the performance of different neural network parameter sets for interpolation, corresponding to the product of the number of possible parameter values and the number of repetitions. Through these simulations, we identified the parameter set that achieved the best interpolation performance. Next, we validated the best model for each combination of observation window size and aggregation rate by interpolating 100 randomly selected sensor data curves from the dataset. Finally, we used the validated model with the best performance in the cloud-based *perceptron* neural network and compared its interpolation results with those produced by classical interpolation algorithms.

In the method comparison, for each pair (aggregation rate X observation size), we randomly selected 100 sensor data curves from the dataset. These curves undergo averaging aggregation using the current iteration's aggregation rate. We apply different interpolation algorithms and a perceptron neural network to interpolate the same curve. We then collect the MSE and the interpolation time for each of these curves, interpolated by different algorithms, and compare them with the original curves generated randomly. Finally, we conduct statistical analysis based on the MSE and time values to determine the differences between the algorithms.

During the simulation phase, we adopted a set of assumptions to ensure controlled and reproducible experiments. First, we assumed that sensor data streams remained temporally synchronized and uniformly sampled, with aggregation performed over fixed-size windows. Second, we assumed that missing values resulted exclusively from temporal aggregation, without additional packet loss or transmission errors. Third, we assumed that sensor behavior remained stationary within each aggregation window, which allowed the neural models to learn local temporal patterns. Finally, we performed model training and evaluation offline, assuming sufficient historical data to learn representative patterns. These assumptions allowed us to isolate and systematically evaluate the impact of aggregation on reconstruction quality.

To account for stochastic effects inherent to neural network training, such as random weight initialization and mini-batch updates, we executed each experimental configuration multiple times. We report aggregated statistics across these repeated runs, thereby reducing the influence of outliers and providing a more reliable assessment of reconstruction performance. This procedure also implicitly evaluates the stability of perceptron-based reconstructions across different training initializations.

3.4. Evaluation metrics and statistical analysis

This evaluation assumes that all interpolation methods are applied to the same set of sensor time series under identical aggregation configurations, ensuring fair comparison. We assess reconstruction quality using the Mean Squared Error (MSE) and evaluate computational efficiency based on interpolation time. The statistical analysis assumes independence between samples and comparable group sizes across methods for the application of ANOVA and Tukey tests.

Table 4
Summary of main notations used in the statistical analysis.

Symbol	Description
MSE	Mean Squared Error between original and reconstructed data
F	ANOVA F-ratio statistic
MSB	Mean Square Between Groups
MSW	Mean Square Within Groups
SSB	Sum of Squares Between Groups
SSW	Sum of Squares Within Groups
k	Number of groups being compared
N	Total number of observations across all groups
n_j	Number of observations in group j
X'_j	Mean of group j
X'	Overall mean across all groups
q	Studentized range statistic used in Tukey test
n	Number of observations per group in Tukey test
HSD	Tukey's Honestly Significant Difference statistic

For clarity and consistency, Table 4 summarizes the main symbols and notations used in the statistical analysis presented in this section. The MSE was the metric employed to assess the performance of interpolation methods in accurately reproducing the original data curves. Using MSE, it is possible to quantify the discrepancy between interpolated and non-interpolated curves, providing an objective measure of the effectiveness of each method (technique). This approach enabled a comparative analysis of performances, revealing which interpolation methods yield results closest to the actual values.

We also considered the time factor as an additional metric for evaluating interpolation methods. The speed at which each technique produces its estimates is also a relevant dimension, especially in scenarios where swift interpolation is a significant criterion. In the IoT context, processing time is often closely tied to other crucial device factors, such as energy consumption. Hence, we evaluated the interpolation time of each method to determine its relative efficiency.

Formally, MSE measures the average squared difference between original sensor values and their reconstructed counterparts over a given time series, providing a standard quantitative metric for reconstruction accuracy.

Combining the information provided by MSE and interpolation time enables a holistic assessment of classical interpolation methods and neural networks. The variation in MSE values and their corresponding times offers insights into the effectiveness of interpolation and the agility of each technique. This analysis enables the identification of methods that provide the best trade-off between accuracy and efficiency in the context of this research. In this scenario, statistical analysis is indispensable for interpreting the collected evaluation metrics and grasping the underlying trends in simulation results. Our work analyzed graphs, ANOVA (Analysis of Variance) tests, and Tukey tests.

ANOVA is a crucial statistical tool to compare the means of three or more distinct groups. It helps determine whether there are significant differences among these means. ANOVA applies to multiple groups and determines whether the observed differences between them exceed what we would expect by chance. To conduct an ANOVA test [35], we establish two hypotheses: the Null Hypothesis (H0), which states that no significant difference exists between the group means, and the Alternative Hypothesis (H1), which states that at least one group mean differs significantly from the others. In other words, all groups are equal. Alternative Hypothesis (H1) - States that at least one group differs significantly from at least one of the other groups. At least one group is different from the others. If we reject the null hypothesis, we can conduct additional mean comparison tests, such as the Tukey test, to identify which groups differ.

We compute the F-ratio, or ANOVA coefficient, by comparing two sources of variance to test whether the group means differ significantly. A higher F-ratio suggests that the group means differ more than would be expected by chance alone. It tests the null hypothesis that all group means are equal [35]. The $F - ratio$ is calculated as in Eq. (1).

$$F = \frac{MSB}{MSW} \tag{1}$$

where MSB is Mean Square Between Groups and MSW is Mean Square Within Groups.

MSB is obtained by dividing SSB by the degrees of freedom for the between-groups component as in Eq. (2).

$$MSB = \frac{SSB}{df_b} = \frac{SSB}{k-1} \tag{2}$$

where SSB is the Sum of Squares Between groups, and $df_b = k - 1$ represents the degrees of freedom between groups.

The following Eq. (3) defines the SSB used in the ANOVA to evaluate the variation between the different group means and the overall mean.

$$SSB = \sum_{j=1}^{k} n_j (X'_j - X')^2 \tag{3}$$

where X'_j is the mean of group j, X' is the overall mean, n_j is the number of observations in group j, and k is the number of groups. MSW is obtained by dividing SSW by the degrees of freedom within groups as in Eq. (4).

$$MSW = \frac{SSW}{df_w} = \frac{SSW}{N-k} \tag{4}$$

where $df_w = N - k$, with N being the total number of observations across all groups and k the number of groups.

The following Eq. (5) defines the SSW used in the ANOVA to measure the variation within each group by comparing individual observations to their respective group mean.

$$SSW = \sum_{j=1}^{k} \sum_{i=1}^{n_j} (X_{ij} - X'_j)^2 \tag{5}$$

where X_{ij} is an observation within group j, and n_j is the number of observations in that group.

The Tukey test, also known as Tukey's honestly significant difference (HSD), is a statistical technique used to identify which pairs of groups have significantly different means in a one-way Analysis of Variance (ANOVA) [36]. It is beneficial when the ANOVA reveals significant differences between groups but does not specify which groups differ [36]. The Tukey test addresses this issue by performing paired comparisons between all groups. Following a significant ANOVA, the Tukey test is crucial for identifying which groups have significantly different means. It provides detailed information on the differences between all pairs of groups, helping researchers understand the nuances of mean differences in their data. The following Eq. (6) is used to perform the Tukey test:

$$HSD = q\sqrt{\frac{MSW}{n}} \tag{6}$$

where q = the standardized range statistic, MSW is the mean square for within groups from the ANOVA (see Eq. (4)), and n is the number of subjects in each group (all groups must be of equal size).

3.5. Other datasets usage

The proposed approach extends beyond the Intel Berkeley Research Lab dataset and applies to a broad range of IoT scenarios that rely on temporally aggregated sensor data. For example, in environmental and agricultural IoT applications, we can reconstruct time series for variables such as soil moisture, air quality, or water levels to support continuous monitoring, even under intermittent data collection or constrained communication conditions.

Similarly, smart city deployments, including traffic and urban infrastructure monitoring, often rely on data aggregation to overcome bandwidth limitations across geographically dispersed sensor nodes. In such contexts, we can employ the proposed reconstruction strategy to recover fine-grained temporal behavior from aggregated measurements.

Industrial IoT environments also represent a relevant application domain. Operators often aggregate sensor data to monitor machine performance or environmental conditions, thereby reducing transmission costs. By reconstructing historical sensor data, the proposed method can support downstream tasks such as predictive maintenance and process optimization.

In healthcare IoT scenarios, particularly in patient monitoring and assisted living systems, sensor data gaps may arise due to connectivity issues or device constraints. In these cases, the proposed approach may help preserve the continuity of monitored signals, which is critical for long-term health assessment.

Despite its potential across these application domains, this study does not empirically evaluate the proposed method's performance in noisier, more unpredictable environments characterized by sensor drift, abrupt changes, or strong seasonal effects. Addressing such conditions may require additional preprocessing, robustness mechanisms, or parameter adaptation and constitutes an important direction for future investigation.

4. Results and discussion

In this section, we present and discuss the experimental results on time-series reconstruction under different aggregation rates. The analysis covers (i) the perceptron-based reconstruction strategy, (ii) classical interpolation methods, and (iii) a comparative evaluation in terms of reconstruction error (MSE) and interpolation time.

4.1. Qualitative comparison of interpolation methods

Fig. 2 presents a qualitative comparison of classical interpolation methods applied to the same time series segment with data gaps. Although all methods aim to reconstruct missing values, each exhibits distinct reconstruction patterns that reflect their underlying assumptions and smoothing strategies. These visual differences help contextualize the quantitative error and efficiency analyses presented in the following sections. Unless otherwise stated, we report the results across all aggregation rates and observation sizes defined in Section 3.3.

4.2. Neural networks

Results indicate that perceptron models can learn from sensor streams and provide accurate short-term predictions, enabling their use as reconstruction models under aggregated data gaps.

4.2.1. Neural networks and prediction

Initial configurations revealed unstable training, but after adjusting the parameters, the perceptron models consistently converged and produced accurate forecasts across sensor modalities.

It was possible to identify a set of parameter values that allowed the network to converge through parameter adjustments and to evaluate them empirically by plotting the resulting data curves. Thus, it was possible to demonstrate, even by manually adjusting the parameters, that the network could learn and make accurate predictions of future values that the perceptron neural networks had not yet observed. In addition to the graphs comparing the predicted curve with the actual data, we measured prediction accuracy using the MSE.

Fig. 3 presents the evolution of the loss value over 500 training epochs for the temperature measurement task. The loss decreases as training progresses, which is consistent with the network converging and learning the underlying pattern in the data.

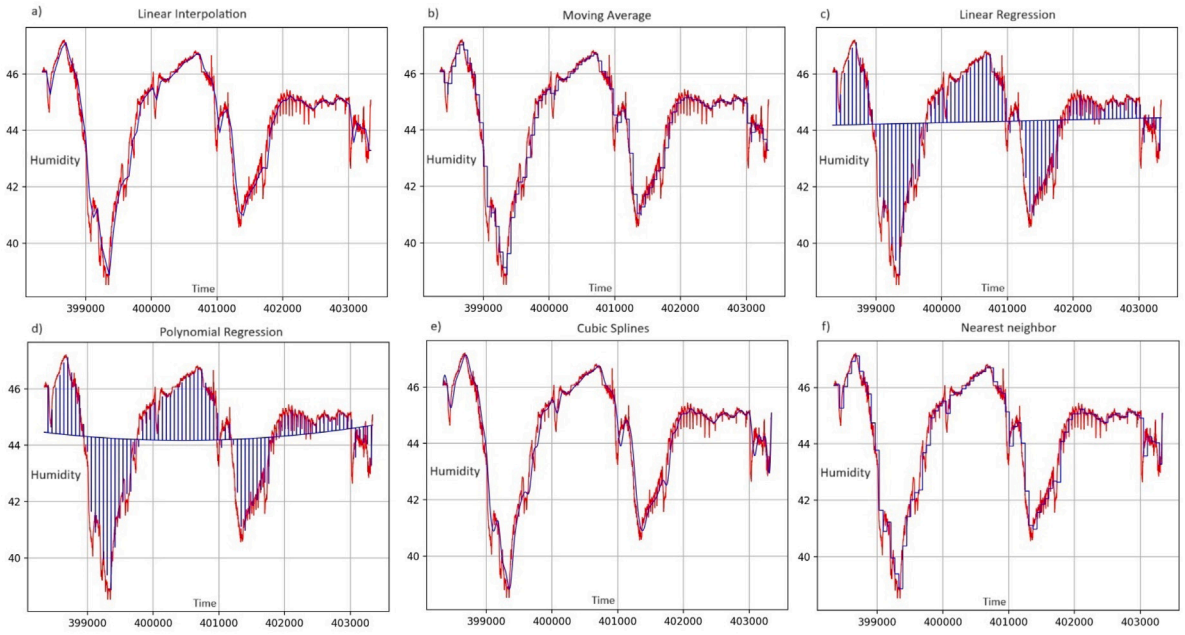


Fig. 2. Curves interpolated by classical interpolation methods.

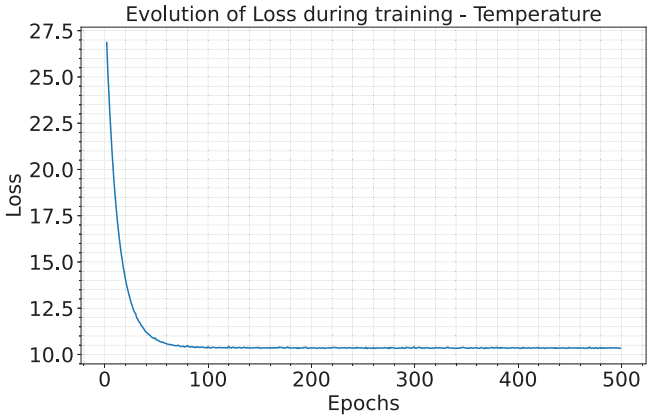


Fig. 3. Evolution of the training loss over 500 epochs for the temperature dataset.

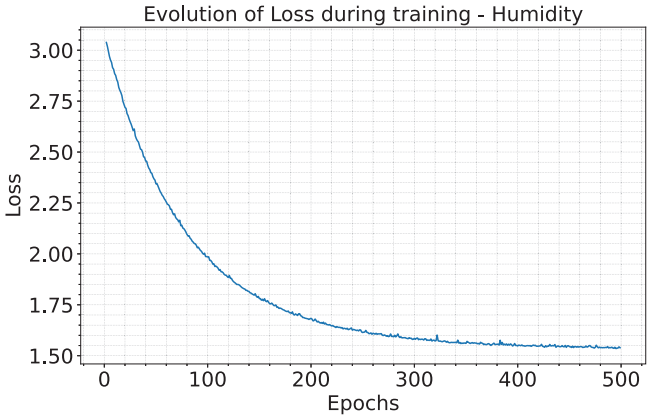


Fig. 5. Evolution of the training loss over 500 epochs for the humidity dataset.

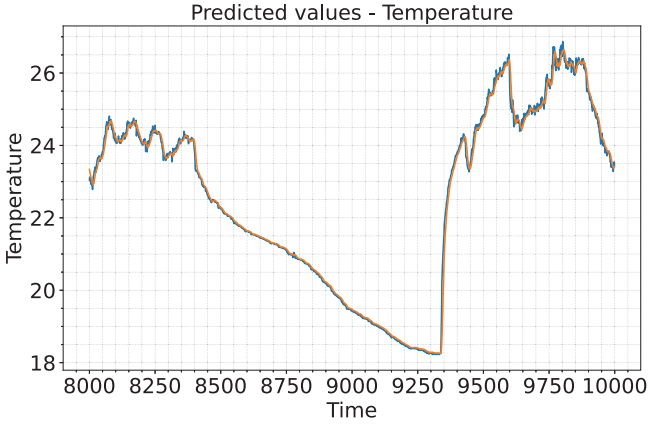


Fig. 4. Ground-truth temperature values and neural network predictions for the 20% held-out segment (samples 8000–10,000).

Fig. 4 compares the ground-truth temperature values (2000 samples corresponding to the 20% held out from training, i.e., indices 8000–10,000) with the predictions produced by the neural network. In this experiment, the model was trained with 8000 measurements (indices 1–8000, i.e., 80% of the 10,000 available samples). The measured MSE was 0.0943, indicating a close match between the predicted and observed curves.

Fig. 5 presents the evolution of the loss value over 500 training epochs for the humidity measurement task. As the number of epochs increases, the loss consistently decreases, indicating effective learning and neural network convergence.

Fig. 6 compares the ground-truth humidity measurements (2000 samples corresponding to the 20% held out from training, i.e., indices 8000–10,000) with the values predicted by the neural network. In this experiment, the model was trained on 8000 samples (indices 1–8000, corresponding to 80% of the 10,000 available measurements). The resulting MSE of 0.1424 indicates good predictive performance and is consistent with the substantial overlap observed between the predicted and measured curves.

Fig. 7 shows the evolution of the loss value over 1000 training epochs for the luminosity measurement task. The loss decreases sharply

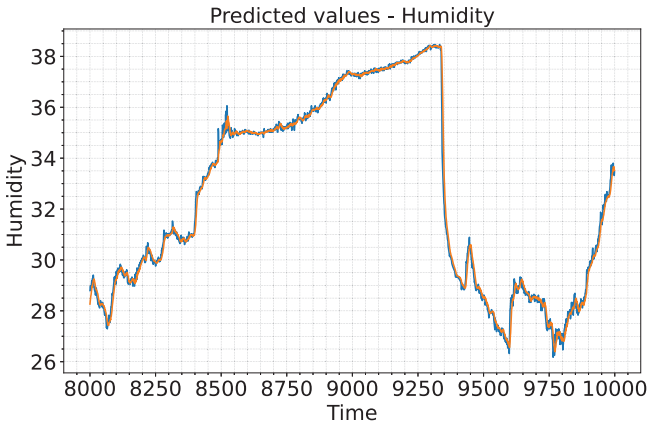


Fig. 6. Ground-truth and predicted humidity values for the 20% held-out segment (samples 8000–10,000).

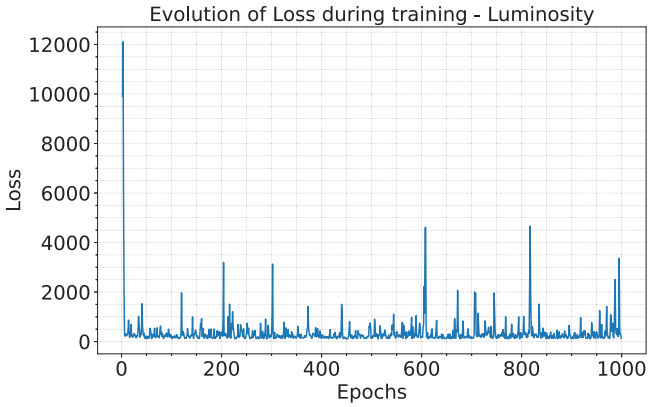


Fig. 7. Evolution of the training loss over 1000 epochs for the luminosity dataset.

during the initial training phase and then exhibits oscillations near zero, indicating convergence with residual variability in the learning process.

Fig. 8 compares the ground-truth luminosity measurements (200 samples corresponding to the 20% held out from training, i.e., indices 800–1000) with the values predicted by the neural network. In this experiment, we trained the model on 800 samples (indices 1–800, corresponding to 80% of the 1000 available data points). The resulting MSE was 0.6289, which, although relatively high, was sufficient to generate predictions that closely follow the overall trend of the observed luminosity curve.

Fig. 9 presents the evolution of the loss value over 1000 training epochs for the voltage measurement task. The loss starts with values close to zero, then decreases, followed by oscillations, indicating a stable learning process with minor fluctuations during convergence.

Fig. 10 compares the ground-truth voltage measurements (2000 samples corresponding to the 20% held out from training, i.e., indices 8000–10,000) with the values predicted by the neural network. In this experiment, the model was trained on 8000 samples (indices 1–8000, corresponding to 80% of the 10,000 available data points). The resulting MSE was 0.0051, indicating high prediction accuracy, with differences between predicted and measured values remaining below 0.1 V, as evidenced by the scale of the y-axis.

4.3. Neural networks and interpolation strategy

By implementing the interpolation strategy using averages and the model sent by the network edge, it was possible to obtain an interpolated curve with values very close to those of the original curve.

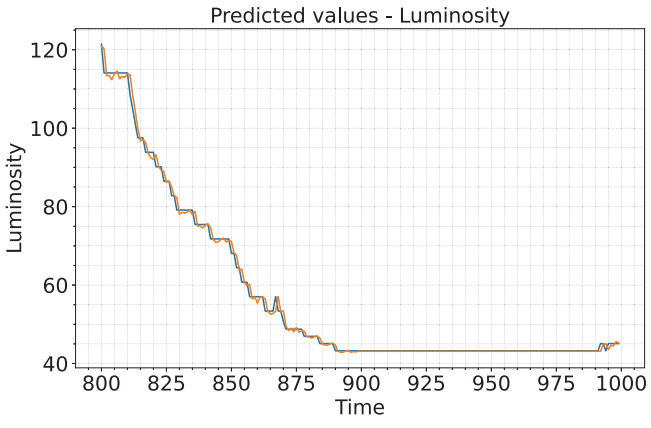


Fig. 8. Ground-truth and predicted luminosity values for the 20% held-out segment (samples 800–1000).

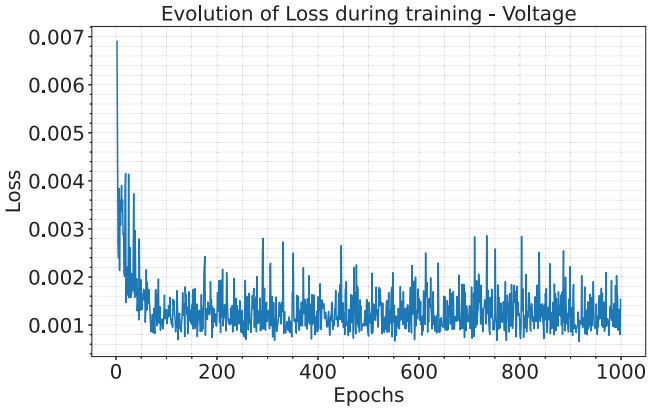


Fig. 9. Evolution of the training loss over 1000 epochs for the voltage dataset.

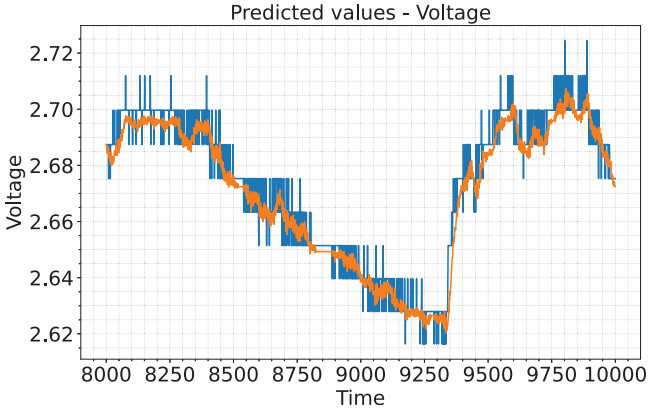


Fig. 10. Ground-truth and predicted voltage values for the 20% held-out segment (samples 8000–10,000).

Fig. 11 shows the original humidity data curve over the time interval 0–1000. We use this curve as the reference signal to evaluate the effectiveness of the proposed interpolation strategy based on aggregated averages.

Fig. 12 presents the curve generated through the proposed interpolation strategy using aggregated averages and the model provided by the network edge. The interpolated curve (blue) is overlaid on the averages used in the interpolation process (red), highlighting how the averages effectively guide the reconstruction of the original signal. In this example, we used a data aggregation rate of 2, yielding an

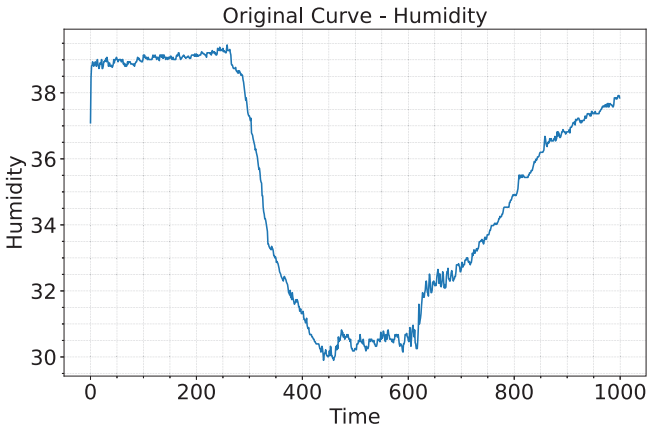


Fig. 11. Original humidity data curve over the time interval 0–1000.

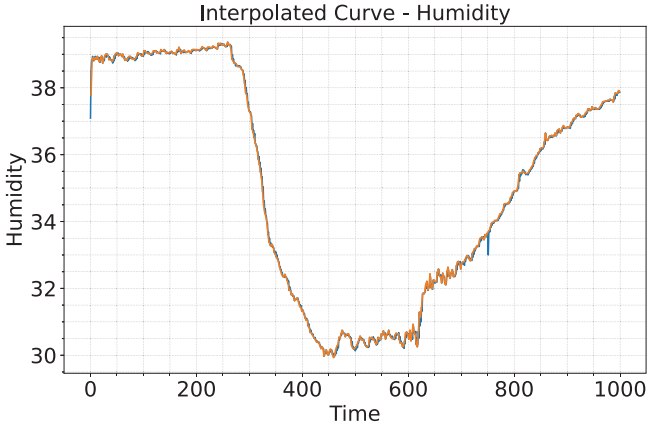


Fig. 12. Interpolated humidity curve (blue) overlaid on the aggregated averages used for interpolation (red), considering a data aggregation rate of 2.

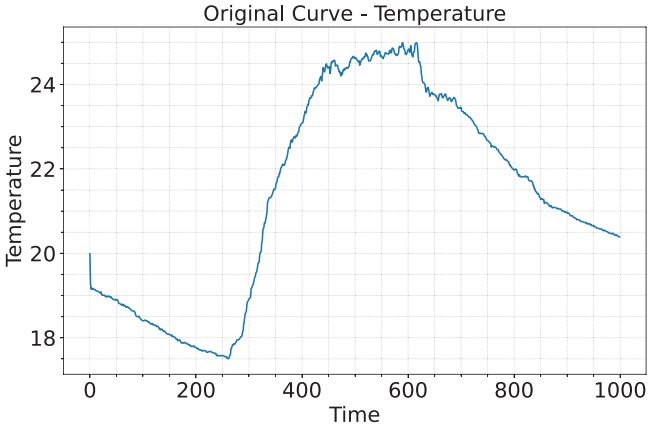


Fig. 13. Original temperature data curve over the time interval 0–1000.

MSE of 0.0617 and indicating a high degree of similarity between the interpolated and original curves.

Fig. 13 presents the original temperature data curve over the time interval 0–1000. This signal is used as a reference to assess the effectiveness of the interpolation strategy based on aggregated averages.

Fig. 14 shows the curve generated using the proposed interpolation strategy with a data aggregation rate of 2. The interpolated curve (blue) is overlaid on the aggregated averages employed in the interpolation

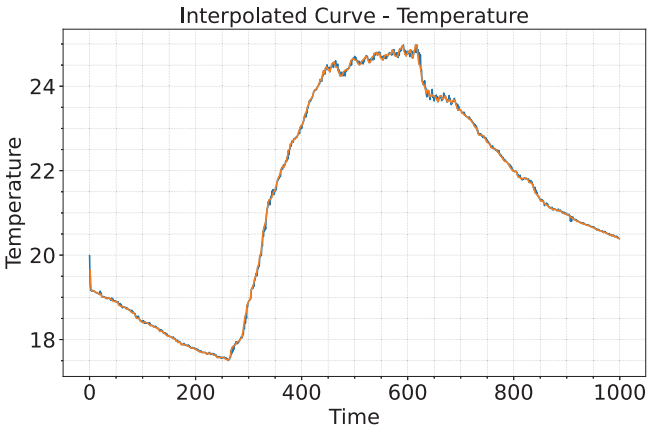


Fig. 14. Interpolated temperature curve (blue) overlaid on the aggregated averages used for interpolation (red), considering a data aggregation rate of 2.

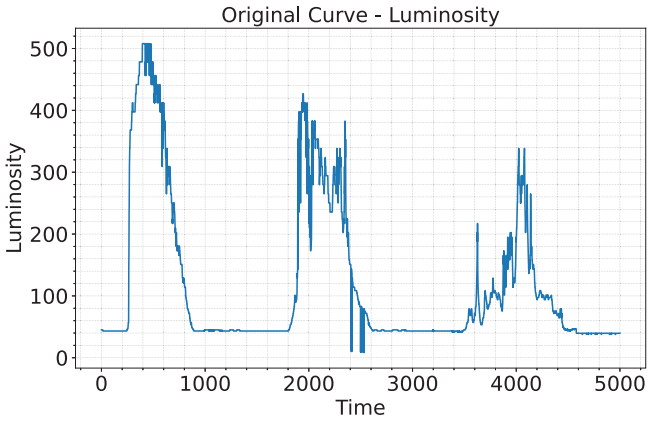


Fig. 15. Original luminosity data curve over the time interval 0–5000.

process (red), illustrating how the averages effectively guide the interpolation algorithm. The resulting MSE was 0.0304, indicating a high degree of similarity between the interpolated and original temperature curves.

Fig. 15 presents the original luminosity data curve over the time interval 0–5000. This signal serves as a reference to evaluate the behavior of the interpolation strategy at a higher data aggregation rate.

Fig. 16 shows the curve generated using the proposed interpolation strategy with a data aggregation rate of 12. The interpolated curve (blue) is overlaid on the aggregated averages used in the interpolation process (red), illustrating how the averages guide the interpolation algorithm even under a higher aggregation level. In this case, the resulting MSE was 10.7202, which, although relatively high, was sufficient to preserve the overall shape and trend of the original luminosity signal.

Fig. 17 presents the original voltage data curve over the time interval 0–2000. This signal serves as the reference for evaluating the effectiveness of the interpolation strategy under a moderate data aggregation rate.

Fig. 18 shows the curve generated using the proposed interpolation strategy with a data aggregation rate of 6. The interpolated curve (blue) is overlaid on the aggregated averages used in the interpolation (red), demonstrating that the averages effectively guide the interpolation. The resulting MSE was 0.0046, indicating a high degree of similarity between the interpolated and original voltage curves.

Although each sensor data type used a distinct perceptron configuration, we treat these configurations as model instantiations rather than fundamentally different network designs. All modalities share

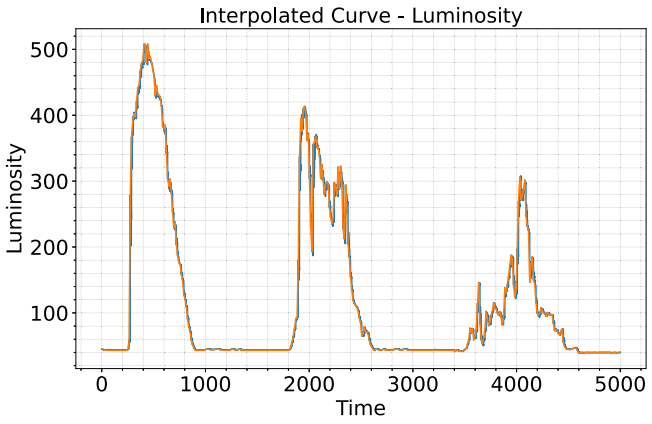


Fig. 16. Interpolated luminosity curve (blue) overlaid on the aggregated averages used for interpolation (red), considering a data aggregation rate of 12.

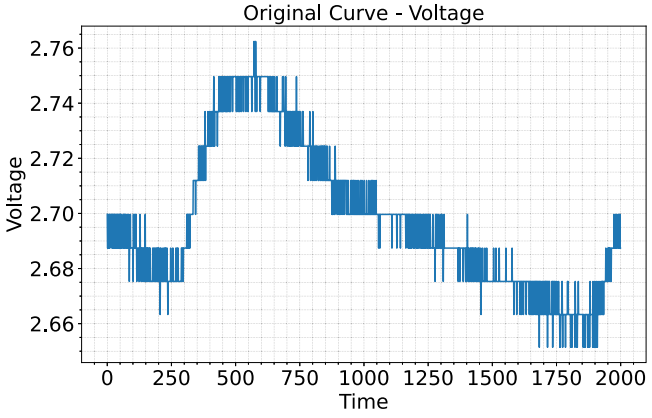


Fig. 17. Original voltage data curve over the time interval 0–2000.

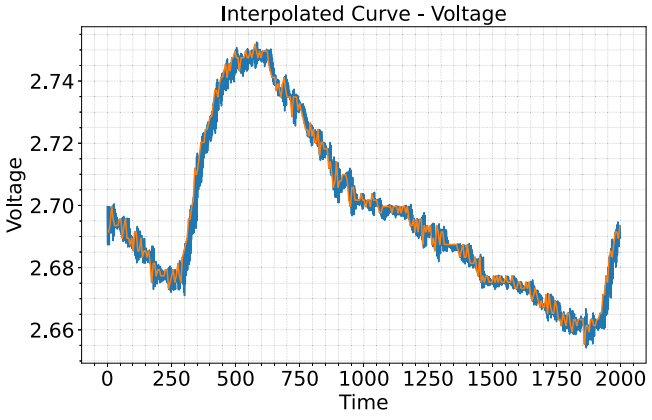


Fig. 18. Interpolated voltage curve (blue) overlaid on the aggregated averages used for interpolation (red), considering a data aggregation rate of 6.

the same underlying architecture and training procedure, differing only in parameter values adjusted to account for data scale and temporal dynamics. As a result, extending the approach to additional sensor modalities does not require redesigning the model; instead, it involves replicating the same lightweight architecture with modality-specific configurations. This design supports scalability in heterogeneous IoT environments, enabling the management of multiple sensor types through standardized training and deployment pipelines.

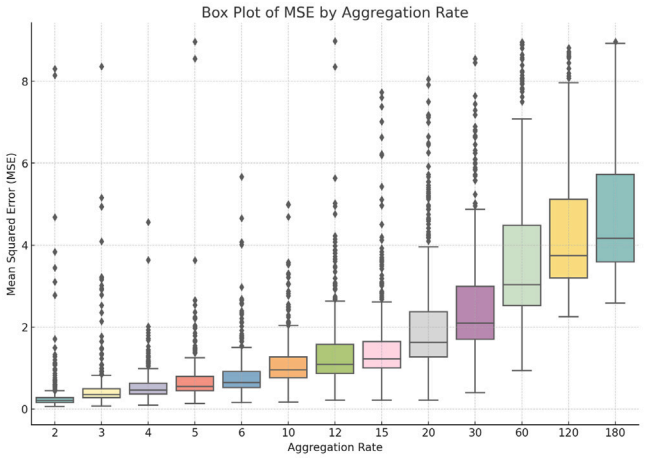


Fig. 19. Box plot showing the relationship between the aggregation rate (A) and the MSE.

As described in Section 3, both in the search stage for the best models and in the validation and comparison stages between algorithms in curve interpolation, hundreds of interpolated curves with different degrees of precision measured by MSE were generated by perceptron neural networks.

In addition to reconstruction accuracy, computational efficiency is evaluated based on interpolation time. All interpolation methods, including classical techniques and perceptron-based models, were executed under identical hardware and software conditions to ensure fair comparison. For neural network-based interpolation, the reported time corresponds exclusively to the inference phase, since model training is performed offline at the edge. Due to its single-layer structure and linear activation function, the perceptron model incurs a constant computational cost per prediction. However, the end-to-end interpolation time observed in our experiments depends on implementation details and runtime overheads, and it is therefore reported empirically in Section 4.4.2.

In this regard, we also present the results of the analyses, using box plots to show the relationship between MSE and the aggregation rate, window size, number of observations, and number of epochs.

Fig. 19 illustrates the relationship between the aggregation rate (A) and the mean squared error (MSE). The aggregation rate ranges from 2 to 180, and considerable variation in MSE values is observed across all configurations. The box plots exhibit different sizes, indicating varying levels of data dispersion, and several outliers are present, particularly at higher aggregation rates. The median MSE shows a slight increasing trend as the aggregation rate increases, suggesting that higher aggregation levels tend to introduce larger prediction errors. In addition, the interquartile range (IQR) widens with increasing aggregation rates, indicating greater variability.

Fig. 20 shows the relationship between the data window size (J) and the MSE. The window size ranges from 10 to 50, and noticeable variability in the MSE values is evident, as reflected in the varying box sizes. Several outliers are present, especially for smaller window sizes. Unlike the aggregation rate, the median MSE does not show a clear trend with increasing window size. Moreover, the IQR remains relatively stable across different window sizes, suggesting that window size has a less pronounced impact on MSE variability than other parameters.

Fig. 21 analyzes the relationship between the number of observations (O) and the mean squared error (MSE). The number of observations ranges from 1000 to 10,000, and noticeable variability in the MSE is observed across all configurations, as indicated by the varying box plots. Outliers are present for all observation levels. The median MSE is lower for 2000 observations, increases for 5000 observations,

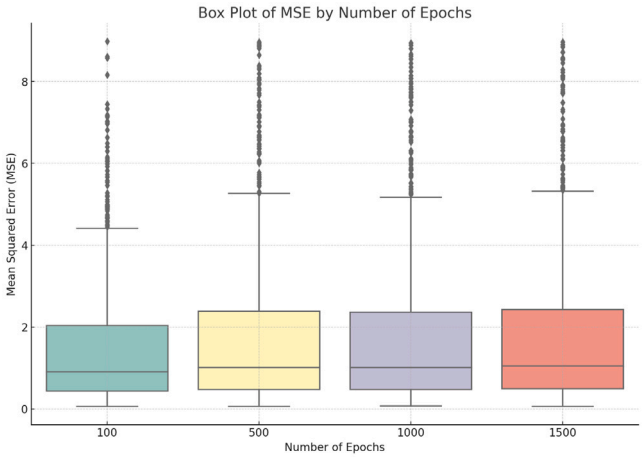


Fig. 20. Box plot showing the relationship between the data window size (J) and the MSE.

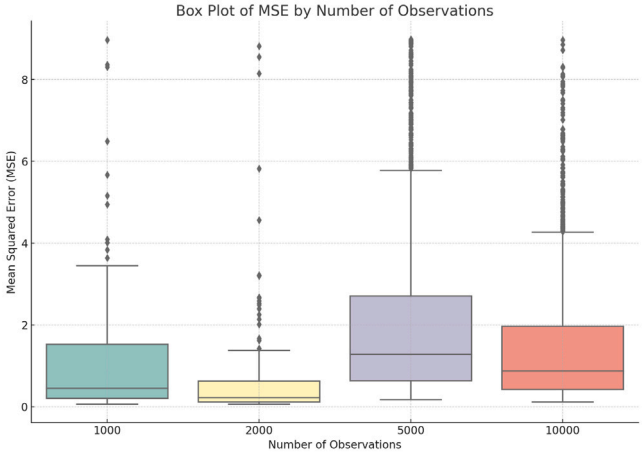


Fig. 21. Box plot showing the relationship between the number of observations (O) and the MSE.

and decreases again for 10,000 observations. In addition, the interquartile range (IQR) is larger for 5000 observations, indicating greater variability in this configuration.

Fig. 22 presents the relationship between the number of training epochs (Ep) and the MSE. The number of epochs ranges from 100 to 1500, and variability in the MSE values is evident, with boxes of different sizes across configurations. We observe outliers across all epoch categories. The median MSE remains relatively stable as the number of epochs increases, suggesting that extending training beyond a certain point does not significantly reduce error. The IQR increases at 100 and 500 epochs, indicating greater dispersion at shorter training durations.

The box plots reveal high variability in the MSE results, even after excluding values greater than 10. The presence of outliers in all figures suggests that several parameter configurations yield errors that are significantly different from those of the rest of the data. Variability is particularly high for specific parameters, such as the aggregation rate and the number of observations. In contrast, the window size and the number of epochs have a less pronounced impact. These results underscore the importance of thorough and meticulous analysis when adjusting model parameters, as even slight changes can lead to substantial variations in performance.

We do not claim that the proposed perceptron-based approach universally outperforms classical interpolation methods across all scenarios. The experimental results show that reconstruction performance

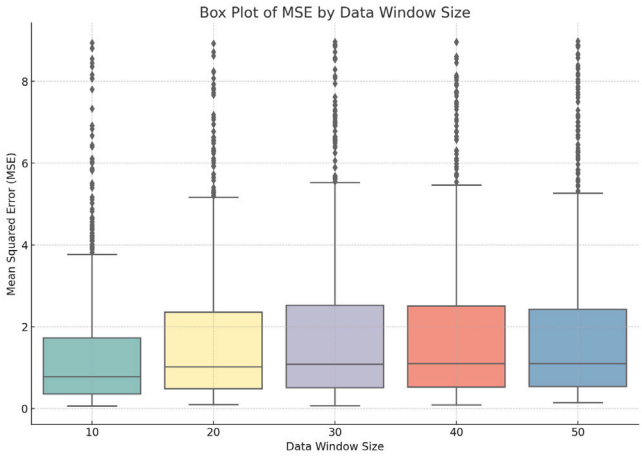


Fig. 22. Box plot showing the relationship between the number of training epochs (Ep) and the MSE.

varies according to factors such as aggregation rate, data modality, and temporal characteristics. Under certain conditions, classical methods achieve lower reconstruction error; however, the perceptron models maintain competitive accuracy while consistently requiring less interpolation time. Therefore, we position the proposed approach as a complementary solution that offers a favorable trade-off between accuracy and efficiency in specific IoT settings, rather than as a strict replacement for traditional interpolation techniques.

Although we conducted an extensive parameter exploration (as described in Section 3.3), the results show that not all parameters influence reconstruction quality equally. The sensitivity analysis derived from the boxplot results indicates that the aggregation rate and the number of observations dominate the reconstruction error. In contrast, the window size and the number of training epochs exert a comparatively minor effect. This finding suggests that, in practical deployments, practitioners can significantly reduce the parameter space by prioritizing aggregation-aware configuration, enabling partial automation, and improving scalability across multiple sensor types.

4.4. Comparison between the methods employed in interpolation

To address the remaining research questions, we developed and implemented a simulation that compares the perceptron neural network with classical interpolation algorithms. This simulation aims to obtain quantitative data on the performance and quality of interpolation among all classical interpolation methods used in this work: linear interpolation, linear regression, polynomial regression, nearest neighbor, moving average, and cubic splines, as well as the perceptron neural network.

We divide the results from each analysis into three blocks, organized by data aggregation rate. The first block describes the results found for low data aggregation rates: A2, A3, A4, and A5. The second block for medium data aggregation rates: A6, A10, A12, and A15. And finally, in the last block, for high data aggregation rates: A20, A30, A60, A120, and A180. The main objective is to verify how the methods behave in terms of interpolation quality and time at different data aggregation rates, and how they evolve with these rates.

In this Section, the simulation data will be presented, along with their interpretation. We divide this discussion into two sections: the first (Section 4.4.1) addresses interpolation quality, and the second (Section 4.4.2) addresses method efficiency.

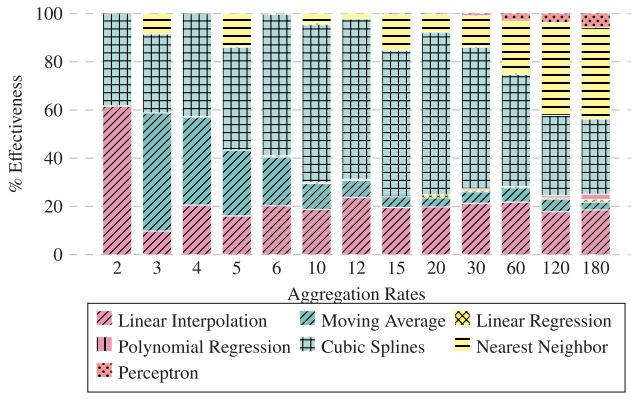


Fig. 23. Interpolation effectiveness across aggregation rates.

4.4.1. Analysis of interpolation quality using the MSE

In this section, we analyze and compare all methods for interpolating curves with data gaps using the MSE to determine the most effective approach. We use the MSE to quantify the difference between the interpolated and original curves. We examine variations in MSE values across interpolation methods and identify their effectiveness under the evaluated scenarios.

Throughout the simulation, we generated overlaid images of the original and interpolated curves to visually assess their similarity over time. However, we emphasize using the MSE as an objective metric to quantify differences between curves in the analysis.

BLOCK 1 (Low aggregation rates):

A2: The linear interpolation method demonstrated the best performance among the algorithms, with the lowest MSE in 61.50% of the interpolated curves, and was followed by cubic splines, with the lowest MSE in 38.25% of the time. This result suggests that these methods perform well in this scenario with a low aggregation rate, with linear interpolation showing a significant advantage. These data can be seen in Fig. 23 in the A2 column.

A3: The moving average interpolation method achieved the best performance in this scenario, with the lowest MSE in 49% of the interpolated curves, followed by cubic splines, with the lowest MSE in 32.75% of the time. In this scenario, linear interpolation showed significant performance degradation, achieving the lowest MSE in only 9.75% of cases, with only a slight increase in the aggregation rate. The nearest neighbor algorithm first appears, achieving the lowest MSE 8.5% of the time. These data can be seen in Fig. 23 in the A3 column.

A4: With the increase in the aggregation rate, cubic splines demonstrated an increase in their performance, with the lowest MSE in 42.75% of the interpolations, surpassing the moving average algorithm, which in this scenario appears with the lowest MSE in 36.50% of the time. Linear interpolation has a slight recovery, with the lowest MSE in 20.50% of the interpolations. It shows this improvement because it was better than the nearest neighbor, which does not have the lowest MSE in any interpolation. In this scenario, linear regression appears discretely, with the lowest MSE (0.25%) of the time. These data can be seen in Fig. 23 in the A4 column.

A5: Once again, cubic splines stood out as the best algorithm, maintaining the lowest MSE in 42.75% of the interpolations. In second place, the moving average with the lowest MSE at 27.25% of the time, experiencing a significant decrease in performance because, in this scenario, the nearest neighbor reappears with the lowest MSE at 14% of the time. Linear interpolation is the third-best algorithm, achieving the lowest MSE 16% of the time and showing a slight decrease in performance compared to the previous aggregation rate. These data can be seen in Fig. 23 in the A5 column.

BLOCK 2 (Medium aggregation rates):

A6: At this aggregation rate, the cubic splines algorithm demonstrated better interpolation effectiveness, increasing to 58.75% of the times it had the lowest MSE. Linear interpolation maintains its position, showing a lower MSE of 20.25% of the time, similar to the previous aggregation rate. As the aggregation rate increases, the moving average begins to lose ground to cubic splines, with the lowest MSE at 20.75% of the time. Linear regression and the nearest neighbor discretely appear, with the lowest MSE at 0.50% and 0.25% of the time, respectively. These data can be seen in Fig. 23 in the A6 column.

A10: With the increase in aggregation rate, cubic splines followed the trend, also increasing the number of times it had the lowest MSE, which occurred in 65% of cases. Linear interpolation is second, with the lowest MSE occurring 19% of the time. Still, the moving average is in a downward trend, and it appears with the lowest MSE 11% of the time. Finally, the nearest neighbor appears 5% of the time, and linear regression discretely with 0.50% of the time. These data can be seen in Fig. 23 in the A10 column.

A12: Still following the upward trend, cubic splines appear again in the first position, with the lowest MSE at 66.50% of the time. The second-best method for this aggregation rate is linear interpolation, which, despite varying, remains in this range 23.75% of the time. The pattern for the moving average continues to be observed; with the increase in aggregation rate, the quality of interpolation continues to degrade, with the method now having the lowest MSE only 7% of the time. The nearest neighbor also appears with 2.75% of the time and for the first time, but discretely, polynomial regression with 0.50% of the time. These data can be seen in Fig. 23 in the A12 column.

A15: With this aggregation rate, cubic splines, despite maintaining the lead, experience a slight decrease in interpolation quality, with the lowest MSE at 60.75% of the time. This decline is justified by the improved performance of the nearest neighbor, which appears with the lowest MSE 15% of the time. Linear interpolation now occupies the second position, with the lowest MSE at 19.50% of the time, whereas the moving average shows continuous degradation in performance as the aggregation rate increases, at only 4.75% of the time. For the first time, the perceptron neural network appears discretely with the lowest MSE (0.25%) of the time. These data can be seen in Fig. 23 in the A15 column.

BLOCK 3 (High aggregation rates):

A20: At this aggregation rate, we have a relatively stable scenario in the positions of the algorithms. Cubic splines maintain their lead, slightly improving their performance over the previous rate and achieving the lowest MSE in 67.50% of the cases. Once again, it is justified by competition with the nearest neighbor, which reduced the number of times it had the lowest MSE to 7.5% of the time. Again, the linear interpolation algorithm appears as the second-place holder in the same value range, with the lowest MSE in 19.75% of the cases. The moving average continues to underperform, this time 3.75% of the time, the second-lowest value recorded by the method. The perceptron and linear regression appear discreetly, with 0.25% and 1.25% of the time, respectively. These data can be seen in Fig. 23 in the A20 column.

A30: At this aggregation rate, cubic splines show a noticeable reduction in dominance, decreasing from 67.5% to 58.75% of the cases with the lowest MSE. This change is driven by improved performance across multiple competing algorithms. Linear interpolation ranks second, achieving the lowest MSE in 21.25% of the cases, followed by nearest neighbor in third place with 13%, and moving average in fourth place with 4.75%. The perceptron shows a slight improvement, reaching 1% of the cases with the lowest MSE. Linear regression and polynomial regression present the lowest frequencies, achieving the best MSE in 0.75% and 0.50% of the cases, respectively. These results are summarized in Fig. 23, column A30.

Table 5
ANOVA result - overall.

Algorithm	df	sumSq	meanSq	F	$PR(>F)$
Linear Interpolation	1.0	1.05e−21	1.05e−21	6.30e−21	1.0
Moving average	1.0	2.46e−23	2.46e−23	1.46e−22	1.0
Linear Regression	1.0	1.09e−23	1.09e−23	6.49e−23	1.0
Polynomial Regression	1.0	3.00e−24	3.00e−24	1.78e−23	1.0
Cubic splines	1.0	6.84e−23	6.84e−23	4.07e−22	1.0
Nearest Neighbor	1.0	6.73e−23	6.73e−23	4.01e−22	1.0
Perceptron	1.0	4.55e−25	4.55e−25	2.71e−24	1.0

A60: Still in the first position, cubic splines have another substantial drop in performance, with the lowest MSE at 46.50% of the time. Continuing to improve performance, the nearest neighbor ranks second, with the lowest MSE at 22.50% of the time. They were followed closely by linear interpolation, in third position, with 21.75% of the time. Next is the 6% moving average. Note a slight improvement in the performance of the perceptron, with 2.75%, followed by polynomial and linear regression, both with 0.25%. These data can be seen in Fig. 23 in the A60 column.

A120: For this aggregation rate, the nearest neighbor method, which has been improving its performance since aggregation A20, reaches its peak here, even surpassing cubic splines. Thus, with the best MSE in 38.50% of the cases, the nearest neighbor takes first place as the best interpolation algorithm. In the second place, cubic splines achieve the lowest MSE 33.25% of the time. In third place is linear interpolation with 17.75%. The perceptron continues to rise, with 3.75% of the time having the lowest MSE. Finally, the moving average was 5.25%, followed by polynomial and linear regression, with 0.50% and 1% of the time, respectively. These data can be seen in Fig. 23 in the A120 column.

A180: Finally, for the last evaluated aggregation rate, the nearest neighbor method continues to lead, although it has slightly reduced its performance, with the lowest MSE in 38% of the cases. Once again, cubic splines rank second, with the lowest MSE in 31% of the cases. Linear interpolation with the lowest MSE in 18.50% of the cases appears in third place. For the first time, the perceptron achieves the lowest MSE in 6% of cases, surpassing even the moving average. The moving average appears with 3%, followed by polynomial and linear regression, with 2% and 1% of the cases, respectively. These data can be seen in Fig. 23 in the A180 column.

The ANOVA test was applied to each aggregation rate (A2, A3, A4, A5, A6, A10, A10, A12, A15, A20, A30, A60, A120, A180), within the aggregation blocks (small, medium, and large), and across the entire dataset. The test revealed no statistically significant differences among the algorithms for all these scenarios. As a result, in terms of MSE, all algorithms exhibit the same effectiveness in interpolation. Table 5 presents an example of these scenarios; in the table, it is possible to see the result for the ANOVA test applied to the entire dataset of MSE, indicating this similarity of effectiveness among the algorithms, based on the F and PR(> F) parameters.

4.4.2. Analysis of interpolation efficiency using time

In this Section, the algorithms will be analyzed by comparing their interpolation speed to determine the most efficient interpolation method, using time as the metric. The algorithms were ranked from fastest to slowest for each aggregation rate evaluated. Since there are seven methods compared, this ranking will have seven positions.

BLOCK 1 (Low aggregation rates):

A2: Linear interpolation was the fastest method, occupying the first position in 100% of the interpolations. The second position was contested by linear regression, which occupied this position 75% of the time, and by cubic splines, which occupied it 25% of the time. The third position was held 75% of the time by cubic splines and 25% by linear regression. The fourth position was occupied in 100% of the interpolations by polynomial regression. The fifth position was shared 50% of the time by moving average and nearest neighbor, which applies to the sixth position. Finally, the perceptron neural network held the seventh position in 100% of the interpolations. This result can be observed in Fig. 24.

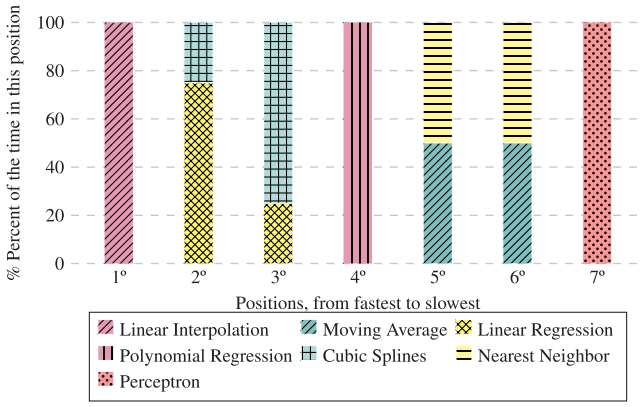


Fig. 24. Comparison of interpolation efficiency among methods for aggregation rate A2 in temperature data, from fastest to slowest.

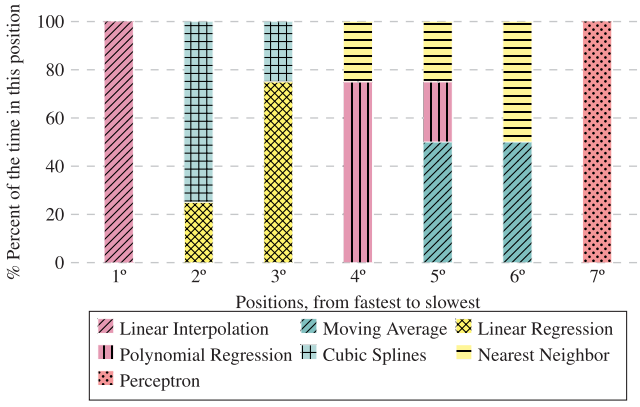


Fig. 25. Comparison of interpolation efficiency among methods for aggregation rate A3 in temperature data, from fastest to slowest.

A3: The first position was occupied by linear interpolation in 100% of the interpolations. The second position was again contested between linear regression and cubic splines in 75% and 25% of the interpolations, respectively. In the third position, the situation reversed, with linear regression in 25% of the interpolations and cubic splines in 75%. The fourth position, also contested, was held 75% of the time by linear regression and 25% by nearest neighbor. In the fifth position, three methods appeared: 50% of the time for moving average, 25% for linear regression, and 25% for nearest neighbor. In the sixth position, 50% of the time for the moving average and 50% for the nearest neighbor. Finally, in the last position, the perceptron neural network appeared in 100% of the interpolations. These results can be observed in Fig. 25.

A4: At this aggregation rate, a clear competition for the first position is observed. Linear interpolation is the fastest algorithm in 75% of the interpolations, while cubic splines lead in the remaining 25%. In the second position, this distribution is reversed, with cubic splines appearing in 75% of the cases and linear interpolation in 25%. Linear regression consistently ranks third across 100% of the interpolations. The fourth position is also consistently held by linear regression across all cases. The fifth and sixth positions are equally shared between the moving average and nearest-neighbor methods, with each appearing 50% of the time in both positions. Finally, the perceptron neural network consistently ranks seventh across all 100% interpolations. These results are illustrated in Fig. 26.

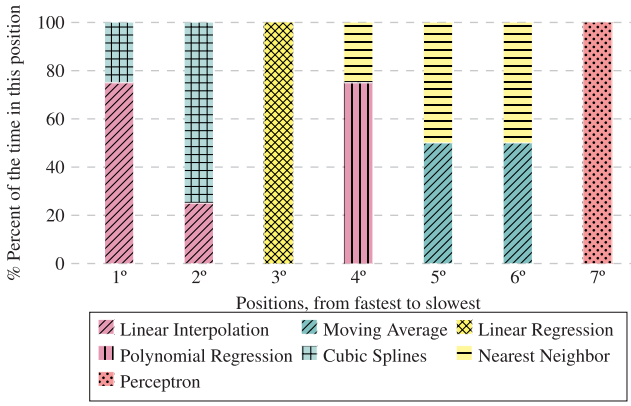


Fig. 26. Comparison of interpolation efficiency among methods for aggregation rate A4 in temperature data, from fastest to slowest.

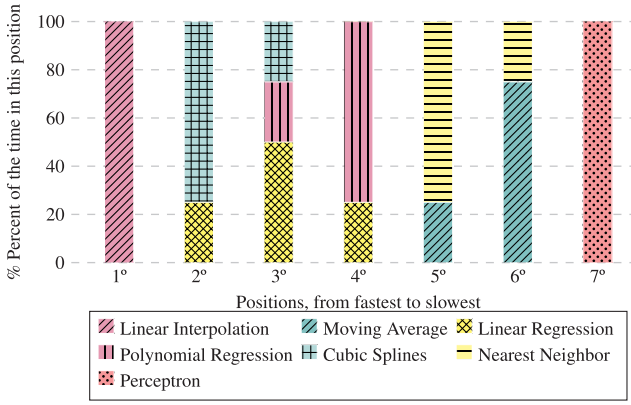


Fig. 27. Comparison of interpolation efficiency among methods for aggregation rate A5 in temperature data, from fastest to slowest.

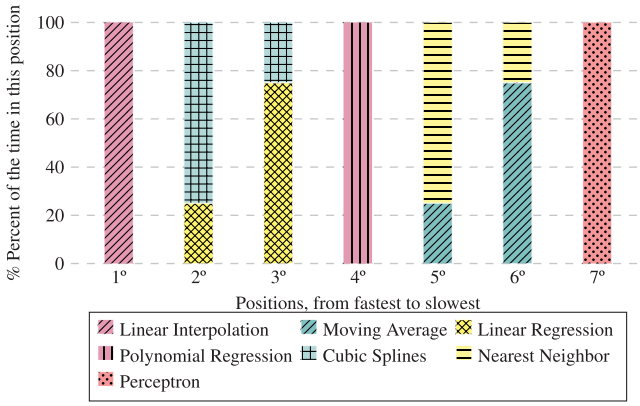


Fig. 28. Comparison of interpolation efficiency among methods for aggregation rate A6 in temperature data, from fastest to slowest.

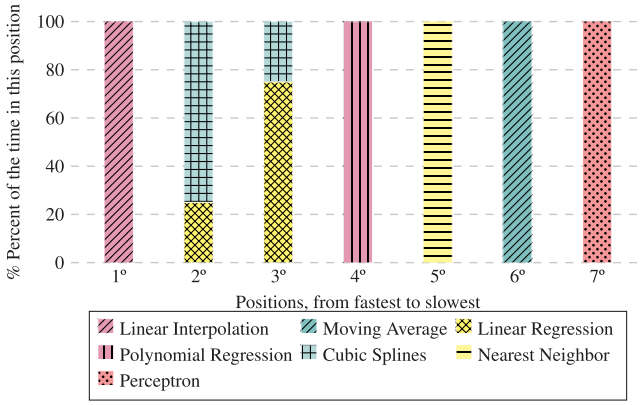


Fig. 29. Comparison of interpolation efficiency among methods for aggregation rate A10, A12 e A15 in temperature data, from fastest to slowest.

A5: The first position was occupied by linear interpolation 100% of the time. The second position was again disputed between linear regression and cubic splines, with 75% and 25% of the time, respectively. In the third position, three methods appear 50% of the time: linear regression, cubic splines, and polynomial regression. The fourth position was disputed between the polynomial regression and moving-average algorithms, with 75% and 25% of the time, respectively. In the sixth position, the situation reversed, with the moving average at 75% and polynomial regression at 25% of the time. The seventh position, once again, by the perceptron neural network 100% of the time. Fig. 27 presents these results.

The ANOVA test was applied separately to all low-aggregation rates (A2, A3, A4, and A5). The ANOVA test revealed statistically significant differences in interpolation times between algorithms but did not indicate within the group which algorithms pairwise performed better. By applying the Tukey test, it was possible to identify that the perceptron exhibited the worst interpolation time, statistically, compared to all other algorithms. Differences among the remaining algorithm pairs were several orders of magnitude smaller, indicating that their interpolation times are statistically similar under low aggregation rates.

BLOCK 2 (Medium aggregation rates):

A6: Linear interpolation was the fastest method, occupying the first position in 100% of the interpolations. The second position was disputed by cubic splines, which held this position 75% of the time and by linear regression 25% of the time. In the third position, the situation reversed, with linear regression appearing 75% of the time and cubic splines at 25%. The fourth position was occupied by polynomial regression 100% of the time. The fifth position is disputed between the nearest neighbor and the moving average, with 75% and 25% of the time, respectively. In the sixth position, the situation reversed, with the moving average at 70% and polynomial regression at 30% of the time. The seventh position, once again, by the perceptron neural network 100% of the time. Fig. 28 presents these results.

A10, A12, and A15: Identical results were observed for these aggregation rates. Linear interpolation was consistently the fastest method, ranking first in 100% of the interpolations. The second position was shared between cubic splines, which appeared in 75% of the cases, and linear regression, which accounted for the remaining 25%. In the third position, this distribution was reversed, with linear regression appearing in 75% of the interpolations and cubic splines in 25%. Polynomial regression consistently ranked fourth, while the nearest-neighbor method ranked fifth in all cases. The moving average method consistently ranked sixth, and the perceptron neural network ranked seventh in 100% of the interpolations. These results are illustrated in Fig. 29.

The ANOVA test was applied individually to all medium aggregation rates (A6, A10, A12, and A15). The ANOVA test revealed statistically significant differences in interpolation times between algorithms but did not indicate within the group which algorithms pairwise performed better. By applying the Tukey test, it was possible to identify that the perceptron exhibited the worst interpolation time, statistically, compared to all other algorithms. Differences among the remaining algorithm pairs were negligible relative to the perceptron gap, indicating statistically similar interpolation times at medium aggregation rates.

BLOCK 3 (High aggregation rates):

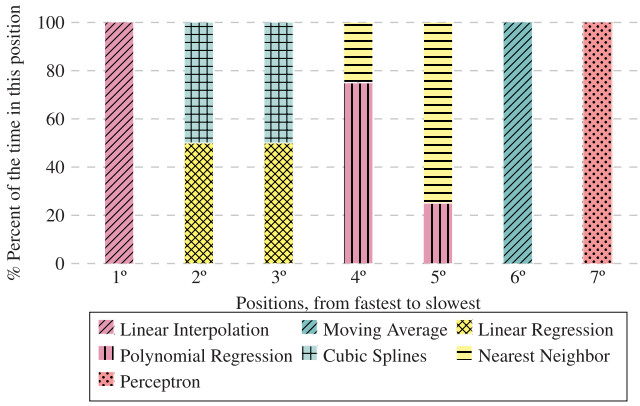


Fig. 30. Comparison of interpolation efficiency among methods for aggregation rate A20 in temperature data, from fastest to slowest.

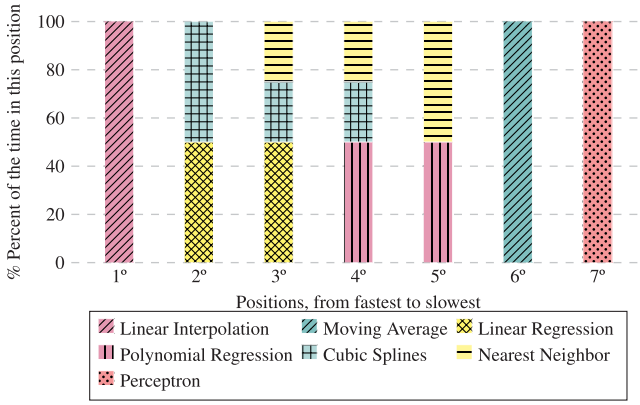


Fig. 31. Comparison of interpolation efficiency among methods for aggregation rate A30 in temperature data, from fastest to slowest.

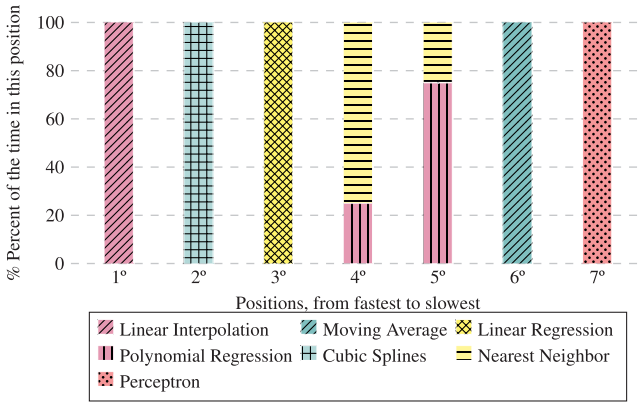


Fig. 32. Comparison of interpolation efficiency among methods for aggregation rate A60 in temperature data, from fastest to slowest.

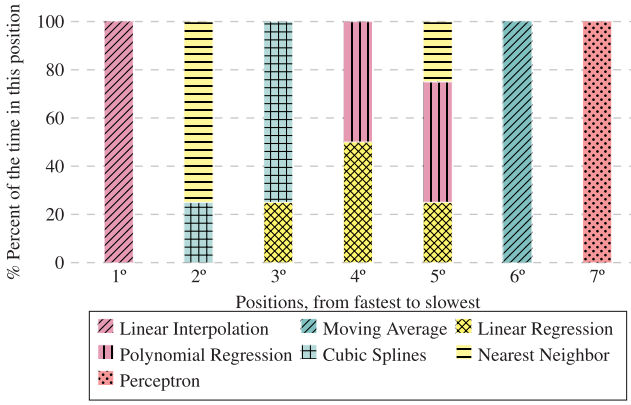


Fig. 33. Comparison of interpolation efficiency among methods for aggregation rate A120 in temperature data, from fastest to slowest.

A20: Linear interpolation appears in the first position in 100% of the interpolations. The second position was occupied by linear regression 50% of the time and by cubic splines 50% of the time, and the same was true for the third position. The fourth position is disputed between polynomial regression and the nearest neighbor, with 75% and 25% of the time, respectively. The fifth position sees a reversal, with the nearest neighbor at 75% and polynomial regression at 25% of the time. The moving average occupies the sixth position 100% of the time. The perceptron neural network occupies the seventh position 100% of the time. These results can be seen in Fig. 30.

A30: Linear interpolation appears in the first position in 100% of the interpolations. The second position was occupied by linear regression and cubic splines, each 50% of the time. In the third position, three algorithms appear: linear regression with 50%, cubic splines, and the nearest neighbor with 25% each. In the fourth position, three algorithms appear: polynomial regression with 50%, cubic splines, and the nearest neighbor with 25% each. The fifth position was occupied by linear regression and cubic splines, each 50% of the time. The moving average occupies the sixth position 100% of the time. The perceptron neural network occupies the seventh position 100% of the time. These results can be seen in Fig. 31.

A60: Linear interpolation appears in the first position in 100% of the interpolations. The second position is occupied by cubic splines 100% of the time. The third position is occupied by linear regression 100% of the time. The fourth position is disputed between the nearest neighbor and polynomial regression, with 75% and 25% of the time, respectively. The fifth position sees a reversal, with polynomial regression at 75% and the nearest neighbor at 25% of the time. The moving average occupies the sixth position 100% of the time. The perceptron neural network occupies the seventh position 100% of the time. These results can be seen in Fig. 32.

A120: Linear interpolation appears in the first position in 100% of the interpolations. The second position is contested between the nearest neighbor and cubic splines with 75% and 25% of the time, respectively. The third position is disputed between cubic splines and linear regression, with 75% and 25% of the time, respectively. The fourth position is occupied by linear and polynomial regression 50% of the time each. In the fifth position, three algorithms appear: polynomial regression with 50%, the nearest neighbor, and linear regression with 25% of the time each. The moving average occupies the sixth position 100% of the time. The perceptron neural network occupies the seventh position 100% of the time. These results can be seen in Fig. 33.

A180: Linear interpolation appears in the first position in 100% of the interpolations. The second position is contested between the nearest neighbor and cubic splines with 75% and 25% of the time, respectively. In the third position, three algorithms appear: cubic splines with 50%, the nearest neighbor, and linear regression with 25% of the time each. In the fourth position, three algorithms also appear: linear regression with 50%, cubic splines, and polynomial regression with 25% of the time each. The fifth position is contested between polynomial and linear regression with 75% and 25% of the time, respectively. The moving average occupies the sixth position 100% of the time. The perceptron neural network occupies the seventh position 100% of the time. These results are presented in Fig. 34.

The ANOVA test was applied individually to all high aggregation rates (A20, A30, A60, A120, and A180) and revealed statistically significant differences in interpolation times between algorithms. However,

Table 6
Research answers.

Number	Research answers
RQ1	Yes, perceptron neural networks can learn the sensor dataset and perform time-series interpolation with gaps with reasonable accuracy.
RQ2	When considering the absolute value of the MSE, classical algorithms perform better, being more effective than perceptron neural networks. However, the differences in MSE values are so minor that they are not statistically significant, so for all practical purposes, all algorithms have the same performance in interpolating curves in all analyzed scenarios.
RQ3	The performance varied depending on the aggregation rate. However, there were no statistically significant differences, so for all practical purposes, all algorithms performed equally.
RQ4	Same as in previous answers. Different algorithms stood out for each aggregation rate scenario, but only when considering the absolute value of MSE. There were no statistically significant differences, so for all practical purposes, all algorithms performed equally.
RQ5	The answer is no. There were no statistically significant differences between the methods, so there is no justification for switching between them.
RQ6	Regarding interpolation time, statistically significant differences were observed among the evaluated methods. The perceptron neural network exhibited higher interpolation times than classical interpolation techniques under the evaluated configurations, indicating differences in computational performance between the approaches.
RQ7	The results indicate a trade-off between interpolation quality and computational efficiency that depends on the application context. While classical methods, such as cubic splines, often achieved lower MSE values, their computational cost may increase with aggregation rate and curve length. In contrast, perceptron-based models showed statistically comparable reconstruction accuracy (MSE), with different runtime characteristics depending on deployment conditions.

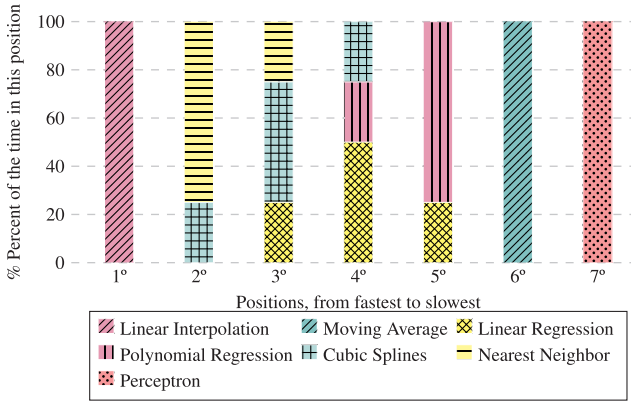


Fig. 34. Comparison of interpolation efficiency among methods for aggregation rate A180 in temperature data, from fastest to slowest.

ANOVA does not indicate which algorithm pairs differ significantly. The Tukey post hoc test showed that the perceptron consistently had the worst interpolation time among all methods. Differences among the remaining algorithm pairs were negligible relative to the perceptron gap, indicating statistically similar interpolation times at high aggregation rates.

In summary, all interpolation methods for reconstructing curves with missing data were analyzed and compared using the MSE metric to assess reconstruction effectiveness. The ANOVA test revealed no statistically significant differences in MSE values among the evaluated algorithms, neither within nor between algorithm groups, indicating statistically equivalent reconstruction accuracy across methods.

In contrast, the analysis of interpolation time revealed statistically significant differences in computational performance among the methods, as identified by the Tukey test. In this context, the perceptron neural network exhibited higher interpolation times than the classical interpolation techniques, highlighting a trade-off between reconstruction accuracy and computational efficiency across the evaluated approaches.

Although classical interpolation methods, particularly cubic splines, often achieve lower reconstruction errors, their computational cost may increase with aggregation rate and curve length. Perceptron-based models, in turn, provide reconstruction accuracy that is statistically comparable in terms of MSE, while their runtime behavior depends on the deployment setting and implementation overhead. Rather than claiming a universal speed advantage, the proposed approach should be understood as a trade-off that may be beneficial in specific IoT deployments (e.g., large-scale data recovery, near-real-time processing, or energy-constrained devices). Conversely, in scenarios where decisions rely on subtle signal variations, classical methods may remain the preferred choice.

The results presented in this section were based on temperature data, but simulations were conducted with the other measurements included in the dataset: Humidity, Luminosity, and Voltage. For these measurements, the results were consistent with what was presented for temperature values and, therefore, were not reproduced in the written work to avoid redundancy. In this sense, based on the temperature data presented and those collected and analyzed for humidity, luminosity, and voltage, it is possible to address the research questions. The answers to the research questions is summarized in Table 6.

5. Conclusion and future works

The rapid evolution of the Internet of Things (IoT) has intensified the use of data aggregation at the edge to reduce bandwidth consumption and improve scalability, at the cost of information loss. To address this challenge, this work investigated the use of neural networks to reconstruct fine-grained sensor data from temporally aggregated data. The results show that lightweight perceptron-based models can achieve reconstruction accuracy statistically comparable to that of classical interpolation methods, while exhibiting different computational performance characteristics.

The adoption of perceptron-based models is a deliberate design choice intended to establish a lower bound on model complexity for data reconstruction under constrained IoT conditions. Despite ongoing advances in IoT hardware, large-scale deployments remain constrained by stringent energy, latency, and management requirements. In this context, simple neural architectures provide an effective trade-off between reconstruction accuracy and computational efficiency, while future work will explore moderately more expressive models as hardware capabilities evolve.

We present the reported results as a baseline assessment. Because the proposed models provide lightweight inference, the approach suits scalable IoT deployments. In future work, we will extend the evaluation to additional datasets and more expressive architectures.

CRediT authorship contribution statement

Israel N. Matos: Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Gustavo B. Figueiredo:** Writing – review & editing, Writing – original draft, Validation. **Maycon L.M. Peixoto:** Writing – review & editing, Writing – original draft, Validation. **Praveen Kumar Donta:** Writing – review & editing, Writing – original draft, Validation. **Schahram Dustdar:** Writing – review & editing, Writing – original draft, Validation. **Cassio Prazeres:** Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology, Investigation, Conceptualization.

Funding

This study was partly supported by FAPESB INCITE, Brazil PIE0002/2022 grant. Additionally, the authors acknowledge the financial support provided by the Pró-Reitoria de Pesquisa e Pós-Graduação (PRPPG) of the Federal University of Bahia, Brazil through the "Programa de Apoio à Publicações Científicas, Edital 11/2026", which covered the article processing charges for this publication. The authors also thank the CNPq, Brazil, CAPES, Brazil, and FAPESB, Brazil organizations for their support of the Graduate Program in Computer Science at the Federal University of Bahia.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

[1] D. Boyd, K. Crawford, Critical questions for big data, Inf. Commun. Soc. 15 (5) (2012) 662–679, http://dx.doi.org/10.1080/1369118X.2012.678878.

[2] P.K. Donta, B. Sedlak, V. Casamayor Pujol, S. Dustdar, Governance and sustainability of distributed continuum systems: a big data approach, J. Big Data 10 (1) (2023) 53, http://dx.doi.org/10.1186/s40537-023-00737-0.

[3] G.P. Pinto, P.K. Donta, S. Dustdar, C. Prazeres, A systematic review on privacy-aware IoT personal data stores, Sensors 24 (7) (2024) http://dx.doi.org/10.3390/s24072197.

[4] G.P. Pinto, C. Prazeres, Data privacy in the internet of things: A perspective of personal data store-based approaches, J. Cybersecur. Priv. 5 (2) (2025) http://dx.doi.org/10.3390/jcp5020025.

[5] IDC, The digitization of the world: From edge to core. IDC white paper, sponsored by seagate, 2018., 2018, Disponível em: https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf. Acesso em: 24 de março 2023.

[6] G. Thirumal, C. Kumar, P.K. Donta, Low discrepancy on non-linear sensor deployment in a time-critical linear iIoT network, Internet Things 26 (2024) 101165, http://dx.doi.org/10.1016/j.iot.2024.101165.

[7] D. Sehrawat, N.S. Gill, Smart sensors: Analysis of different types of IoT sensors, in: 2019 3rd International Conference on Trends in Electronics and Informatics, ICOEI, 2019, pp. 523–528, http://dx.doi.org/10.1109/ICOEI.2019.8862778.

[8] V. Prutyanov, N. Melentev, D. Lopatkin, A. Menshchikov, A. Somov, Developing IoT devices empowered by artificial intelligence: Experimental study, in: 2019 Global IoT Summit (GIoTS), 2019, pp. 1–6, http://dx.doi.org/10.1109/GIOTS.2019.8766355.

[9] S.S. Gill, S.S. Murugesan, K.A. Anurag, P. Verma, H. Kaur, S. Kumar, M. Kumar, Agentic AI: Vision and challenges, 2026, http://dx.doi.org/10.36227/techrxiv.176779782.20689880.v1, TechRxiv Preprint.

[10] C.N. da Silva, C.V.S. Prazeres, Tiny federated learning for constrained sensors: A systematic literature review, IEEE Sensors Rev. 2 (2025) 17–31, http://dx.doi.org/10.1109/SR.2025.3548547.

[11] F. Pereira, R. Correia, P. Pinho, S.I. Lopes, N.B. Carvalho, Challenges in resource-constrained IoT devices: Energy and communication as critical success factors for future IoT deployment, Sensors 20 (22) (2020) http://dx.doi.org/10.3390/s20226420.

[12] B.N. Sudheer, K. Sujatha, A brief survey on data aggregation and data compression models using blockchain model in wireless sensor network, in: 2023 International Conference on Innovative Data Communication Technologies and Application, ICIDCA, 2023, pp. 406–413, http://dx.doi.org/10.1109/ICIDCA56705.2023.10100009.

[13] B.M. Alencar, R.A. Rios, C. Santana, C. Prazeres, FoT-stream: A fog platform for data stream analytics in IoT, Comput. Commun. 164 (2020) 77–87, http://dx.doi.org/10.1016/j.comcom.2020.10.001.

[14] B.M. Alencar, J.P. Canário, R. Lobão Neto, C. Prazeres, A. Bifet, R.A. Rios, Fog-DeepStream: A new approach combining LSTM and concept drift for data stream analytics on fog computing, Internet Things 22 (2023) 100731, http://dx.doi.org/10.1016/j.iot.2023.100731.

[15] S. Hamdan, A. Awaian, S. Almajali, Compression techniques used in iot: A comparitive study, in: 2019 2nd International Conference on New Trends in Computing Sciences, ICTCS, 2019, pp. 1–5, http://dx.doi.org/10.1109/ICTCS.2019.8923112.

[16] R.K. Yadav, M. Gupta, Data aggregation algorithms in IoT: An organized evaluation of the literature, in: 2020 Third International Conference on Smart Systems and Inventive Technology, ICSSIT, 2020, pp. 300–304, http://dx.doi.org/10.1109/ICSSIT48917.2020.9214242.

[17] L. Andrade, C.J.L. De Santana, B.D.M. Alencar, C.J. Silva, C. Prazeres, Data interplay: A model to optimize data usage in the internet of things, Softw.: Pr. Exp. 53 (6) (2023) 1410–1437, http://dx.doi.org/10.1002/spe.3193.

[18] A. Dridi, A. Debar, V. Gauthier, H.I. Khedher, H. Afifi, Deep learning semantic compression: IoT support over LORA use case, in: 2019 2nd IEEE Middle East and North Africa COMMunications Conference, MENACOMM, 2019, pp. 1–6, http://dx.doi.org/10.1109/MENACOMM46666.2019.8988571.

[19] N.Y. Yen, J.-W. Chang, J.-Y. Liao, Y.-M. Yong, Analysis of interpolation algorithms for the missing values in IoT time series: a case of air quality in Taiwan, J. Supercomput. 76 (8) (2020) 6475–6500, http://dx.doi.org/10.1007/s11227-019-02991-7.

[20] A. Hanumanthaiah, A. Gopinath, C. Arun, B. Hariharan, R. Murugan, Comparison of lossless data compression techniques in low-cost low-power (LCLP) IoT systems, in: 2019 9th International Symposium on Embedded Computing and System Design, ISED, 2019, pp. 1–5, http://dx.doi.org/10.1109/ISED48680.2019.9096229.

[21] E. Batista, L. Andrade, R. Dias, A. Andrade, G. Figueiredo, C. Prazeres, Characterization and modeling of IoT data traffic in the fog of things paradigm, in: 2018 IEEE 17th International Symposium on Network Computing and Applications, NCA, 2018, pp. 1–8, http://dx.doi.org/10.1109/NCA.2018.8548340.

[22] J. Fan, Z. Wang, H. Wu, D. Sun, J. Wu, X. Lu, An adversarial time–frequency reconstruction network for unsupervised anomaly detection, Neural Netw. 168 (2023) 44–56, http://dx.doi.org/10.1016/j.neunet.2023.09.018.

[23] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, fourth ed., Pearson Education, Inc., Boston, 2020.

[24] J. Guo, T. Chen, S. Jin, G.Y. Li, X. Wang, X. Hou, Deep learning for joint channel estimation and feedback in massive mimo systems, Digit. Commun. Netw. 10 (1) (2024) 83–93, http://dx.doi.org/10.1016/j.dcan.2023.01.011.

[25] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Netw. 2 (5) (1989) 359–366, http://dx.doi.org/10.1016/0893-6080(89)90020-8.

[26] P. Vedavalli, D. Ch, A deep learning based data recovery approach for missing and erroneous data of IoT nodes, Sensors 23 (1) (2023) http://dx.doi.org/10.3390/s23010170.

[27] A. Graves, N. Jaitly, A.-r. Mohamed, Hybrid speech recognition with deep bidirectional LSTM, in: 2013 IEEE Workshop on Automatic Speech Recognition and Understanding, 2013, pp. 273–278, http://dx.doi.org/10.1109/ASRU.2013.6707742.

[28] K. Garikipati, T. Muppala, A. Vinitha Chowdary, A. Sahay, IoT sensor data stream compression with hybrid compression algorithms, in: 2024 15th International Conference on Computing Communication and Networking Technologies, ICCCNT, 2024, pp. 1–8, http://dx.doi.org/10.1109/ICCCNT61001.2024.10725308.

[29] F. Huang, W. Zheng, W. Guo, Z. Yu, Estimating missing data for sparsely sensed time series with exogenous variables using bidirectional-feedback echo state networks, CCF Trans. Pervasive Comput. Interact. 5 (1) (2023) 45–63, http://dx.doi.org/10.1007/s42486-022-00112-7.

[30] N.A. Mohamed, J.R. Cavallaro, FPGA-based DNN hardware accelerator for sensor network aggregation node, in: 2022 56th Asilomar Conference on Signals, Systems, and Computers, 2022, pp. 322–327, http://dx.doi.org/10.1109/IEEECONF56349.2022.10051920.

[31] H.M. Ahmed, B. Abdulrazak, F.G. Blanchet, H. Aloulou, M. Mokhtari, Long gaps missing IoT sensors time series data imputation: A Bayesian Gaussian approach, IEEE Access 10 (2022) 116107–116119, http://dx.doi.org/10.1109/ACCESS.2022.3218785.

[32] M. Zhang, H. Zhang, D. Yuan, M. Zhang, Learning-based sparse data reconstruction for compressed data aggregation in IoT networks, IEEE Internet Things J. 8 (14) (2021) 11732–11742, http://dx.doi.org/10.1109/JIOT.2021.3059735.

[33] J. Cao, X. Zhang, C. Zhang, J. Feng, Improved convolutional neural network combined with rough set theory for data aggregation algorithm, J. Ambient. Intell. Humaniz. Comput. 11 (2) (2020) 647–654, http://dx.doi.org/10.1007/s12652-018-1068-9.

[34] S. Madden, Intel lab data, 2004, http://db.csail.mit.edu/labdata/labdata.html. (Online; Accessed January 2026).

[35] L.S. ahle, S. Wold, Analysis of variance (ANOVA), Chemometr. Intell. Lab. Syst. 6 (4) (1989) 259–272, http://dx.doi.org/10.1016/0169-7439(89)80095-4.

[36] H. Abdi, L.J. Williams, Newman–keuls test and tukey test, in: The SAGE Encyclopedia of Research Design, second ed., SAGE Publications, Inc., Thousand Oaks, California, 2022, pp. 1058–1063, http://dx.doi.org/10.4135/9781071812082.n386.