Evaluation of low-power devices for smart greenhouse development

Juan Morales-García[1*], Andrés Bueno-Crespo[1], Raquel Martínez-España[2], Juan-Luis Posadas[3], Pietro Manzoni[3] and José M. Cecilia[3]

[1*]Computer Science Department, Catholic University of Murcia (UCAM), Av. de los Jerónimos, 135, Murcia, 30107, Murcia, Spain.
[2]Information and Communications Engineering Department, University of Murcia (UM), C. Campus Universitario, 11, Murcia, 30100, Murcia, Spain.
[3]Computer and Systems Informatics Department, Universitat Politècnica de València (UPV), Camino de Vera, s/n, Valencia, 46022, Valencia, Spain.

*Corresponding author(s). E-mail(s): jmorales8@ucam.edu; Contributing authors: abueno@ucam.edu; raquel.m.e@um.es; jposadas@disca.upv.es; pmanzoni@disca.upv.es; jmcecilia@disca.upv.es;

Abstract

The combination of artificial intelligence and the Internet of Things (AIoT) is enabling the next economic revolution in which data and immediacy are at the key players. Agriculture is one of the sectors that can benefit most from the use of AIoT to optimise resources and reduce its environmental footprint. However, this convergence requires computational resources that enable the execution of AI workloads, and in the context of agriculture, ensuring autonomous operation and low energy consumption. In this work, we evaluate TinyML and edge computing platforms to predict the indoor temperature of an operational greenhouse in situ. In particular, the computational/energy trade-off of these platforms is assessed to analyse whether their use in this context is feasible. Two artificial neural networks (ANNs) are adapted to these platforms to predict the indoor temperature of the greenhouse. Our results show that the microcontroller-based devices can offer a competitive and energy-efficient computational alternative to more traditional edge computing approaches for lightweight ML workloads.

Keywords: Artificial Intelligence, Edge computing, Time-series forecast, TinyML, CPU-GPU Performance, Power consumption

1 Introduction

The Internet of Things (IoT) is a major player in the digital revolution, enabling devices and humans to interact with each other in real time [1]. This interaction generates large amount of data whose analysis can provide insights from uncovering hidden patterns, correlations between variables, etc [2]. However, knowledge is only valid as long as it is generated at the right time to enable the right decisions to be made. Therefore, enabling efficient analysis of this huge amount of data generated by the IoT is crucial to transform this deluge of data into meaningful information.

Machine Learning (ML) algorithms are showing their potential for extracting knowledge from large amounts of data [3]. Traditionally, ML algorithms have been executed in supercomputers, where performance prevails over energy efficiency [4]. However, when performance is not the only concern, other approaches are feasible. For instance, edge/fog computing [5] has been approached towards decentralization, where initial computations on data are carried out in (or close to) the data capture devices. In fact, the edge computing paradigm is providing (1) energy savings by avoiding sending and processing data in the cloud, (2) highly responsive applications and services for mobile environments, (3) highly scalable systems, thanks to the distribution of processing units, (4) guaranteed privacy policies for the IoT and (5) disconnection tolerant systems as transient connection interruptions can be masked.

IoT devices have a limited power budget at this level of the network, as they typically rely on batteries or energy harvesters, leading to ultra-low power approaches. This limited power scenario translates into a major limitation for many components of the architecture, especially energy-intensive components such as wireless transmitters or even processing capabilities [6]. A new trend, called TinyML [7], has recently emerged at the intersection of ML, IoT and computing platforms. This trend aims to leverage microcontroller units (MCUs) that are available in all devices across the IoT ecosystem, from sensor data collection and actuation, to information transfer and reception [8]. In the IoT ecosystem, MCUs had been relegated to information transfer, without taking advantage of the existing computational potential for heavier workloads, which was transferred to the edge, fog or cloud. A major factor restricting the computational use of MCUs is the limited memory size of these devices. In particular, for running ML workloads such as Deep Learning (DL) [9], artificial neural networks (ANNs) [10], or reinforcement learning [11] that require a certain amount of memory space.

In this paper, we evaluate different edge computing and TinyML platforms for the execution of several ANNs for time series forecasting. Performance and power evaluation is provided for up to four edge computing devices, namely Arduino microcontrollers, Raspberry PI and two Nvidia Jetson edge computing platforms. Particularly, two widely used neural network models are analysed; i.e., multi-layer perceptron (MLP) and the convolutional neural network (CNN) for the prediction of the indoor temperature of an operational greenhouse. The development of smart greenhouses is a great benchmarking to assess the intersection between edge computing and artificial intelligence. These environments are often isolated and in very aggressive meteorological conditions (e.g. high temperatures), and where internet access and power supply are not always guaranteed. Therefore, the development of smart greenhouses must be designed with these constraints in mind, guaranteeing the autonomy and continuous operability of the greenhouse. Therefore, the research question of this paper focuses on finding out if it is possible to have neural network-based time series prediction systems in fully autonomous greenhouses. Our initial hypothesis is that with the use of Edge computing/TinyML platforms enable ML in the greenhouse, avoiding cloud-based services and without overly penalising the overall power consumption of the greenhouse.The main contribution of the paper are the following:

1. A comprehensive analysis is provided to assess the limitations of different TinyML/Edge computing platforms to serve a real operating environment such as a greenhouse. Several datasets introducing climate variability typical of semi-arid climates are defined to train and validate the predictive models evaluated.
2. Two different ANNs, namely the multilayer perceptron (MLP) and the convolutional neural network (CNN), are used, evaluated and parameterised to predict the internal temperature of an operating greenhouse. These ANNs are adapted to reduce their memory footprint by testing different neural network architectures, adjusting their hyperparameters, reducing the accuracy of the variables involved and migrating each model to the target low-power devices.
3. The process of migrating these workloads to the targeted edge computing platforms, including an Arduino family microcontroller, is shown.
4. A performance and energy consumption tradeoff of these platforms is under study, showing clear advantages to microcontrollers for these lightweight workloads.

The paper is organised as follows. Section 2 describes a set of related works on prediction of climate variables with different machine learning techniques executed in edge and tiny environments. Section 3 describes the datasets used, as well as the models and methodology followed in the development of the work. Section 4 shows the results and the analysis and discussion of the results. Finally, the conclusions and future work are shown in section 5.

2 Related Works

Weather forecasting has emerged as an important topic of research in the last decades. Particularly the prediction of climatic variables is a topic of special relevance in agriculture. In outdoor agriculture and greenhouses, knowing the parameters of certain climatic variables helps farmers make decisions that optimise their resources. Since time is non-linear and dynamic, the techniques used must also be non-linear [12]. In the case of this work, we have focused on neural networks to predict temperature. Moreover, given that the agricultural world, especially small companies, cannot make large investments in technology, we propose the study to evaluate the performance and consumption of low-cost devices. In this section, we briefly study other works that have made predictions of climatic variables, considering neural networks and their adaptation to run in edge environments.

In [13], a technique is presented for predicting daily minimum, average and maximum temperature using three types of artificial neural networks, namely multilayer perceptron, recurrent neural network, and convolutional neural network. The best temperature prediction result was obtained using a convolutional neural network. Another work that uses convolutional neural networks is presented in [14]. The authors present a hybrid model of convolutional neural networks and recurrent networks to predict temperatures. The model uses daily data from mainland China and obtains an error of less than 1 degree Celsius. In [15] Jung at al propose a comparison of various neural network techniques, with different hourly granularities to predict climate variables, such as humidity and temperature, inside the greenhouse. The best results were obtained with a combination of Recurrent neural networks and Long short-term memory. These works, although related to temperature prediction using neuronal networks, do not take into account the adaptability of these models to be executed at the edge of network.

In [16] the authors present an approach to improve the management of greenhouses and predict their internal variables using recurrent neural networks executed and adapted to the edge computing environment.Among the variables they predict is the air temperature. The proposed model allows them to anticipate the reading of the greenhouse sensors locally. Another work where they also predict air temperature using neural networks and edge computing is presented in [17]. The authors forecast the anticipated temperature to determine when a frost will occur. In the study, the authors make a comparison between running predictions in the cloud and in edge environments. The conclusions indicate that the results in edge environments are satisfactory and that for agricultural environments where connectivity cannot always be ensured, edge environments are efficient. In [18], authors proposed the evaluation of three different types of neural network architecture, using different values of the sliding window associated with the input data run at the edge to predict the indoor temperature of a greenhouse. In [19], authors presented a study analysing advances in work using edge computing, demonstrating its effectiveness in facilitating data analysis, future prediction and decision making at the edge by avoiding the transfer of large volumes of data in classical systems. Also in [20], a study was made for time series data forecasting, where the different factors that influence the energy consumption of smart meters are studied, analysing several issues, including temperature forecasting. This study demonstrates the superiority of LSTMs over other models. From the point of view of efficiency and speed, in [21], the authors used TPU accelerators at the edge, analysing three different types of TPUs. These models allowed for faster and less time-consuming evaluations. In [22], they proposed an adaptive CNN for low power consumption edge computing devices. They proposed a novel adaptive architecture that used an output block predictor to choose the base architecture for inference, providing similar or superior performance to the classical architecture. Another work on Edge computing is presented in [23], where the authors connected and managed IoT devices to analyse information from strawberry crops to detect diseases in these crops. In [24], authors used a model for workload forecasting with adaptive sliding window and use temporal correlation by means of an adaptive sliding window algorithm in order to achieve higher accuracy with lower overhead. In [25], taking into account that the use of cloud applications is becoming more and more popular and specifically through containers, they presented a model for the calculation of container similarity for scenarios presented in the cloud, providing a new solution in workload prediction models for this type of containers.

Regarding TinyML, there are not many studies available about its use to predict climate variables, yet. In [26], authors presented the design of a tiny deep neural network to predict atmospheric pressure, embedded in a microcontroller. The tiny neural network approach presented was compared with the results of a non-tiny neural network, obtaining similar results. Furthermore, experiments demonstrated the performance of the approach in a real environment.

Analysing the related works, the reader can figure out that there is no assessment, analysis and comparison of the performance of climatic variables in both edge and TinyML platforms. This is one of the main novelties presented in this work, together with the discussion and measurement of the different energy consumption.

3 Materials and Methods

3.1 Operational greenhouse

Fig. 1 shows the ETIFA's operational smart greenhouse targeted for this study. ETIFA[1] is part of a group of companies with more than 40 years of experience

[1]https://www.etifa.com/

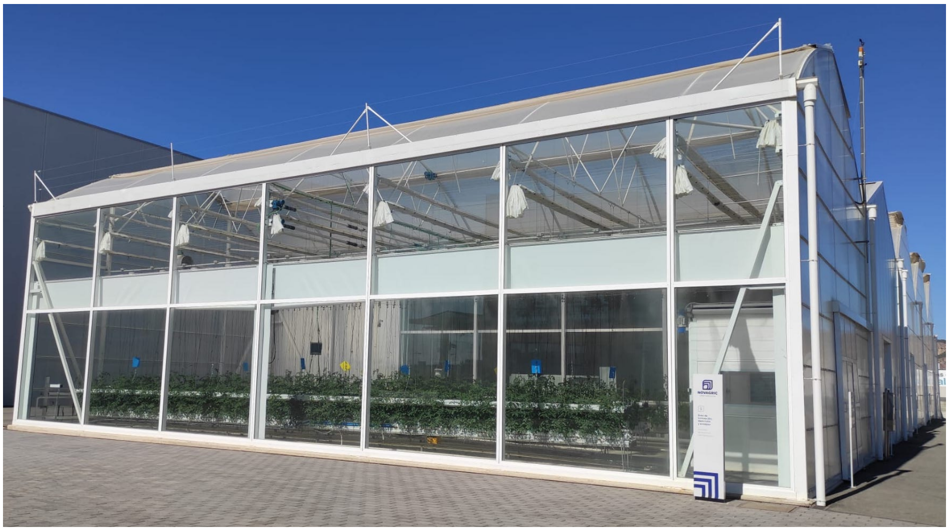


Fig. 1: Targeted operational greenhouse located at Murcia (Spain)

providing agricultural services and technology, manufacturing and innovating to adapt to the continuous demands of the agricultural market. ETIFA's greenhouse is placed in Murcia (South-eastern Spain); a semi-arid region where water is very scarce resource (e.g. the average annual rainfall in the last years is 132 mm) and average annual temperature is around 23 $^{\text{o}}$C. This greenhouse has a surface area of 50 m^2 and operates with the NUTRICONTROL OPTIMUM® system for climate control and fertigation. NUTRICONTROL is a R+D company that offers solutions for the analysis, design, manufacture and marketing of control equipment for irrigation and climate automation in greenhouses and outdoor irrigation installations. The OPTIMUM® system consists of a data logger composed of a server system and several input/output connections to plug-in several sensors and actuators, including temperature, humidity, radiation, wind speed, just to name a few. Of particular interest to us is the temperature of the air inside the greenhouse as this is the most important variable for climate control. This variable is measured every 5 minutes in the ETIFA greenhouse, providing near-real time (NRT) continuous measurements to take actions that can increase/decrease the greenhouse temperature to reach the ideal temperature of the crop being grown.

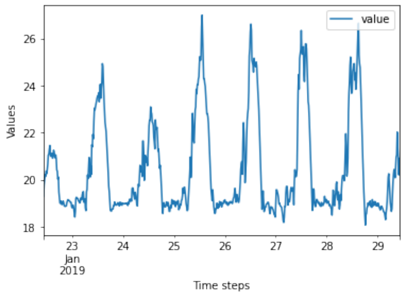
3.2 Dataset

Table 1 summarises the description of the datasets that have been used to carry out the temperature prediction. It shows the start date of the data, the end date and the number of instances contained in each dataset. Each dataset contains the temperature values each 15 o 60 minutes of a greenhouse between the indicated dates.

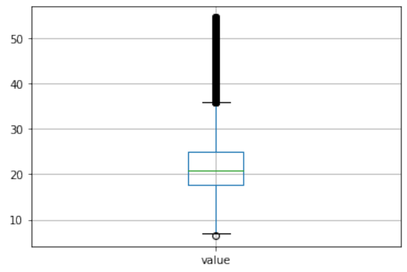
Datasets	Start date	End Date	# Instances
DS-15	11-12-2018	23-03-2021	80018
DS-60	11-12-2018	23-03-2021	20005

Table 1: Dataset description

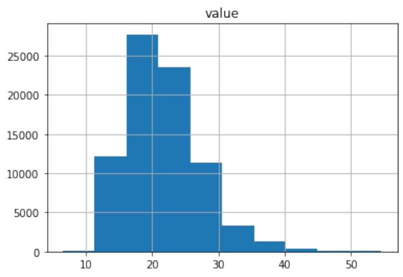
As the sensors return data approximately every 2-5 minutes, to obtain the 15-minute and the 60-minute datasets, an aggregation of the data has been made, in which each value corresponds to the average of the values belonging to that time interval.



(a) 7-Day line plot.



(b) Box plot.

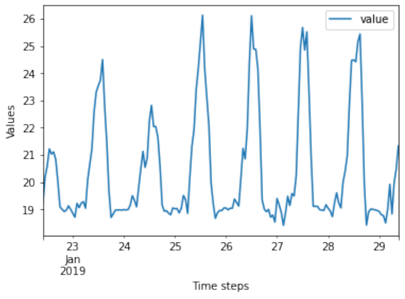


(c) Histogram plot.

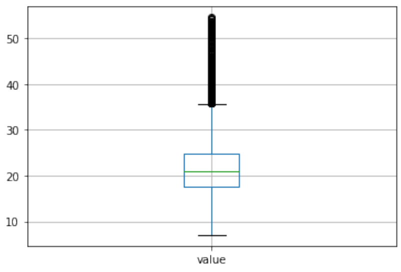
Fig. 2: Exploratory Data Analysis (EDA) for the 15-minutes time serie.

Fig. 2 shows an exploratory data analysis of 15-minutes dataset. This dataset includes 80,0018 different values with an average mean of 21.689°C and standard deviation of 5.726°C. The maximum average value reaches up to 54.620°C. The time series of the internal greenhouse temperature is shown in Fig. 2a which illustrates an example of the 7-day time series. Moreover, the box-and-whisker plot (see Fig. 2b), and the distribution of its values (see Fig. 2c) are shown.

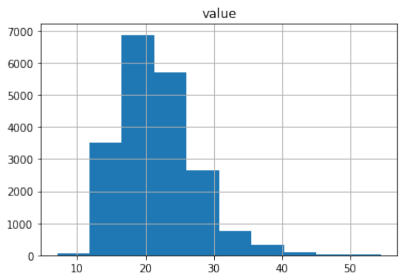
Similarly, Fig. 3 shows the description of the 60-minute dataset. It includes more than 20,000 records with a mean of 21.689°C and a standard deviation of 5.705°C . An example of the 7-day time series is also shown (see Fig. 3a)



(a) 7-Day line plot.



(b) Box plot.



(c) Histogram plot.

Fig. 3: Exploratory Data Analysis (EDA) for the 60-minutes time serie.

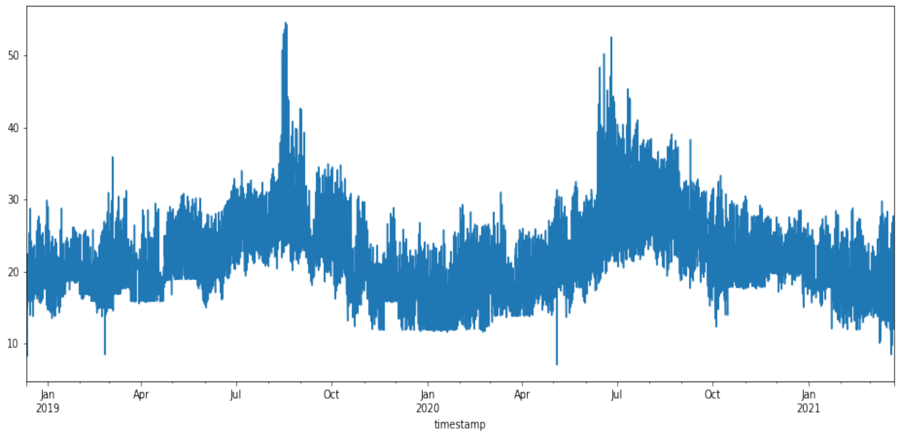
where a smoother data was appreciated. Finally, the box and whisker plot (see Fig. 3b), and the distribution of its values (see Fig. 3c) are shown.

Finally, Fig. 4 shows the complete time series (see Fig. 4a) as well as the result of applying Pettit's homogeneity test [27] (see Fig. 4b), in which it can be seen that the time series is not homogeneous, with a possible cut-off point on 01-05-2020, dividing the time series in two: the first part whose mean is 20.76°C and the second part whose mean is 23.12°C.

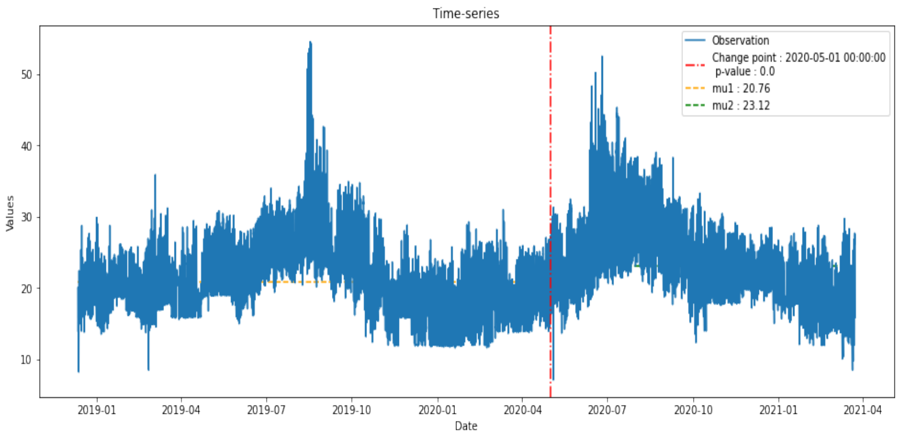
Once the data has been collected as described above, it has to be prepared for training and inference of ANN models. Specifically, we focus on the prediction of a time-series, in this case, the internal greenhouse temperature. Equation 1 shows the input and out layout of our dataset to cast and adapt the data and to be trained by ANNs models.

$$\begin{gathered} input \rightarrow output \\ [t_1, t_2, t_3, ..., t_n] \rightarrow [t_{n+1}, t_{n+2}, t_{n+3}, ..., t_{n+m}] \end{gathered} \quad (1)$$

where n is the number of past elements that the network will have to infer the next value. This parameter is often referred to as look-back. In addition, m defines the number of instances to be predicted by the model. It should be noted that depending on the input dataset (DS-15 or DS-60) each value to be predicted means a different time granularity. For example, if the model predicts the next 3 hours of greenhouse interior temperature, with DS-15 it means



(a) Full time serie.



(b) Results of Pettit's homogeneity test.

Fig. 4: Homogeneity of the time series.

generating 12 values (i.e., $m = 12$). However, if the model has been trained with DS-60, the 3-hours forecast means generating 3 values (i.e., $m = 3$). Fig. 5 shows how the inputs are selected for each of the iterations in the DS-60 case, where each of the three inputs corresponds to one hour. The output obtained is the following three hours. For the DS-15 case, the scheme is the same but instead of having three inputs and three outputs, there are twelve.

Finally, the last data transformation performed on the dataset to adapt it to ANNs is the separation into training data and test data, i.e. data that will be used to make the model learn (training) and data that will be used to validate the accuracy of the model (testing). The percentages associated with

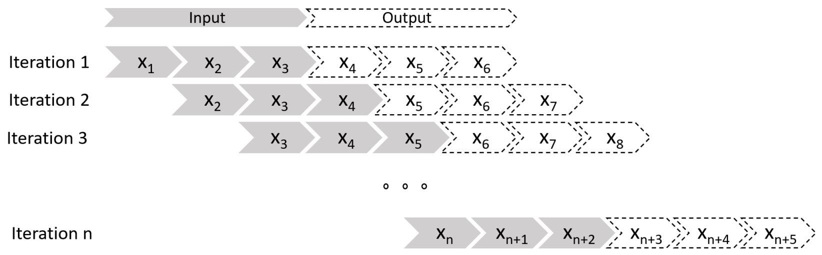

Input
Output
Iteration 1
Iteration 2
Iteration 3
Iteration n
x_1 x_2 x_3 x_4 x_5 x_6
x_2 x_3 x_4 x_5 x_6 x_7
x_3 x_4 x_5 x_6 x_7 x_8
x_n x_{n+1} x_{n+2} x_{n+3} x_{n+4} x_{n+5}

Fig. 5: Inputs and outputs diagram to train ANN models.

each subset of data are 70% for the training data set and 30% for the testing data set.

3.3 The ANN models

This section describes the evaluated ML models. They are ANNs-based models relatively simple as they have to fit in very reduce memory space. The models chosen were the multi-layer perceptron (MLP) and the convolutional neural network (CNN), both very simple models that are supported by the *tflite* conversion that allow the execution on the microcontroller. A neural network is an ML model that mimics the way a set of biological neurons works. Although its use in classification is more widespread, it can also be used in regression models. A perceptron simulates an artificial neuron, so that it represents a simulation of a logistic regression. When a group of perceptrons per layer (MLP) is joined together, it is known as an artificial neural network (ANN). The MLP used in this work is composed of three layers: input (receives the input features), hidden (processes the inputs) and output (produces the output). In the learning process, each layer adjusts its weights to relate inputs to outputs, because they use an activation function that allows them to learn non-linear properties in the network [28, 29]. The second model under study is a Convolutional Neural Network (CNN). The CNN models are used in different applications and domains, where they are most frequently used in imaging for classification. However, they are also used in regression, where they can be used using time series by transforming the data to adapt them to the inputs of the convolutional network. A CNN is made up of blocks of filters, which, through convolution operations, allow the relevant features to be extracted from the input. One of the advantages of CNNs over conventional neural networks (ANNs) is the automatic learning of the filters, so that the necessary and most relevant features are obtained from the input data [30, 31].

ANN models are supervised methods that consist of two main steps. The first is training where a known data set, i.e. where the input and output are known, is used to fit the model with a custom parameters. In our case, we introduce the different temperature tuples prepared to serve as input (i.e.

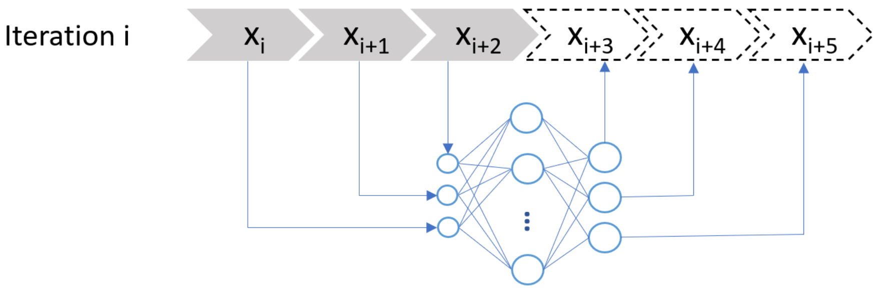


Fig. 6: Model used for the MLP scheme for the case of three-hour inputs The MLP has three layers (input, hidden and output), where the hidden layer learns the model that allows the output to be predicted from the input.

t_1, ..., t_n) to the different networks. The ANNs generate an output which is compared with the previously known output (i.e. t_{n+1}, ..., t_{n+m}).

The ANNs compare the real output and the forecasted output, re-evaluate the error value and update the weights of each neuron in the neural network layers depending on how correct or incorrect the forecast is, so that the neural network is adjusted in the training process, improving the performance of the task it is learning.

The inference process is different from the training process. In inference, the layers of the neural network are not re-evaluated and adjusted. Simply, knowledge from a neural network model that has already been trained is applied and a result is inferred without readjusting the weights of the neural network.

In our case, as said before, we use two different models, MLP and CNN. This models have a custom configuration (a.k.a. hyperparameters) in order to improve the performance of it forecasting.

The most important hyperparameters for a MLP in time-series regression problem are:

- Units: Number of neurons used in one layer of the model.
- Optimizer: It is a function that optimises the learning of an ANN, updating its neurons weights depending on the evaluation error
- Learning rate: It allows the speed of adaptation of the model to the problem to be established.
- Loss function: Function used for evaluate the error of the model in each epoch.

Regarding the MLP, we have used a hidden layer with 100 units, an Adam optimizer, a learning rate equal to 0.000001, and MSE (Mean Squared Error) as a loss function. Also, for training, the model is using a total of 100 epoch for 15-minutes dataset and 200 epochs for 60-minutes dataset. Fig. 6 shows the scheme used for the MLP network for the three-hour case using 60-minutes

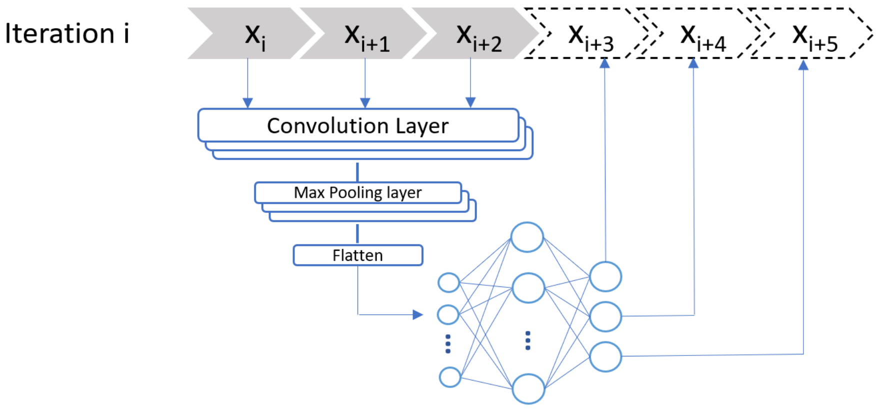


Fig. 7: Model used for the CNN scheme for the three-hour input case, where a new feature vector is obtained due to the convolutional layer. In this way, the hidden layer of the network learns to generate the output from these new features provided by the convolutional layers, unlike the previous case where only an MLP is used with the direct input.

dataset, where the input is passed directly to the MLP network and the prediction of the next three hours is obtained. For the 15-minutes dataset case, the scheme would represent twelve inputs and twelve outputs.

The most important hyperparameters for a CNN in time-series regression problem are:

- Filters: Sets the dimension of the output space.
- Kernel Size: Sets the width and height of the 2D convolution window.
- Padding: Add irrelevant pixels when a window is smaller than standard window.
- Activation: Function to define how to transform the weighted sum of the input into an output for a node or nodes in a network layer.
- Pool size: Window size over which to take the maximum.
- Optimizer: It is a function that optimises the learning of an ANN, updating its neurons weights depending on the evaluation error.
- Learning rate: It allows to control how fast a model adapts to the problem.
- Loss function: Function used for evaluate the error of the model in each epoch.

Regarding the CNN, we have used a convolutional layer with 16 filters, [2, 1] of kernel size, “same” padding, “ReLU” as activation function, [2, 1] of pool size, an Adam optimizer, a learning rate equal to 0.00001, and MSE (Mean Squared Error) as the loss function. Also, for training, the model is using a total of 50 epochs for 15-minutes dataset and 20 epochs for 60-minutes dataset. Fig. 7 shows the scheme used for the CNN network, where the input is passed through a convolutional layer to obtain a feature vector that is used as input to the fully connected neural network. This figure represents the example of three hours and prediction of the next three hours with 60-minutes dataset. For the 15-minutes dataset case, the scheme would represent twelve inputs and twelve outputs.

As a framework we use TensorFlow (developed by Google), which allows us to build machine learning models.

3.4 TensorFlow Platform used to build the ML models

TensorFlow is a powerful open-source deep learning framework that can run machine learning models on various devices, including Raspberry Pi and Nvidia Jetson. Both Raspberry Pi and Nvidia Jetson are popular choices for building intelligent embedded systems and IoT devices due to their low cost, small form factor, and powerful processing capabilities.

When using TensorFlow on a Raspberry Pi, developers can take advantage of the framework's ability to perform complex machine learning operations, such as image classification and object detection, on the device itself. This allows for real-time processing of data, making it possible to build intelligent applications that can respond to their environment in real-time. Additionally, TensorFlow supports a wide range of hardware and software platforms, including Linux and Android, which makes it easy to use on the Raspberry Pi. This can be especially useful for projects that require low-level control of the hardware, such as robotics or home automation systems.

Similarly, Nvidia Jetson boards are also powerful devices that can run TensorFlow models with high performance. Jetson boards are based on the NVIDIA CUDA architecture, which is specifically designed for running deep learning workloads. This makes them ideal for applications such as computer vision, object detection, and image recognition. The Jetson boards also have a powerful GPU and a large amount of memory, which allows them to handle large and complex models.

Using TensorFlow on both Raspberry Pi and Nvidia Jetson can be a great way to take advantage of the framework's powerful capabilities while still keeping the costs and power consumption low. Additionally, TensorFlow allows developers to use pre-trained models and a library of powerful algorithms to train their own models, making the development process faster and more efficient. It also allows developers to deploy the models on the device, which can make the application more efficient, reliable and secure.

TensorFlow Lite is a lightweight version of TensorFlow. It is designed to help developers deploy machine learning models on mobile and IoT devices with limited computational resources, such as Arduino Nano devices. It has been optimised for these types of devices, making it possible to run models on devices with limited memory and processing power.

One of the key features of TensorFlow Lite is its ability to convert pre-trained TensorFlow models into a format that can be run on mobile and IoT devices. This conversion process, known as "model quantization," reduces the size of the model and enables it to run faster on these devices. TensorFlow Lite also includes a number of other performance optimizations, such as support for hardware acceleration, to further improve the performance of models on these devices. Additionally, TensorFlow Lite provides a user-friendly API that makes it easy for developers to integrate machine learning into their mobile and IoT applications.

3.5 Training and deployment of ML models in Edge and Tiny ML platforms

As mentioned above, ANN models are supervised models, so they are run in two different steps: training and inference. Training is the most time-consuming step and is a process that, to date, cannot be carried out on microcontrollers. Therefore, the training process for the models targeted in this paper is carried out on an High-Performance Computing (HPC) platform and it is exactly the same procedure applied to every ML problem, i.e. pre-processing and transformation of the data, development of the ML model and then training with the dataset. Once trained, the model is tested with the test set and if necessary, the model is re-trained with different parameters.

Once the ANN models have been trained, they need to be transferred to the platform where the inference is performed. This paper targets four different computing platforms (see Fig. 8). Three of them can be classified as edge/fog computing platforms; i.e., Raspberry PI Model B and two platforms from Nvidia Jetson family. The Raspberry PI 4 Model B is a single-board computer (SBC) that is extensively used in IoT infrastructure. It has a Quad core Cortex-A72 (ARM v8) 64-bit SoC, running at 1.5GHz and endowed with 8GB LPDDR4-3200 SDRAM. Regarding the Nvidia Jetson edge computing devices, we focus on the Nvidia AGX Jetson Xavier with 8-core ARM v8.2 64-bit CPU, 8MB L2 + 4MB L3, 512-core Volta GPU with Tensor Cores and 16GB 256-Bit LPDDR4x running at 137GB/sec, and Nvidia Jetson Nano that has 5-core ARM Cortex-A57 MPCore CPU, 2MB L2, 128-core Maxwell GPU and 4GB 64-Bit LPDDR4 running at 25.6 GB/sec.

The last device under study is the Arduino Nano 33 BLE Sense that can be categorised as TinyML device since combines small factor, environment sensing and allows ML models to be run using TinyML and TensorFlow Lite. It is build upon the nRF52840 64MHz microcontroller, the memory is 256 KB SRAM, 1MB flash, and runs on ARM Mbd OS. It is important to note that the main way to connect to this Arduino device is via Bluetooth Low Energy although it is also equipped with some sensors to detect audio, colour, humidity, temperature, motion, proximity and more.

As can be seen, the platforms have very different characteristics, with large differences in computational capabilities. Therefore, it is necessary to make a series of modifications to the artificial intelligence models so that they can be adapted to the characteristics of all platforms. In particular, to achieve this objective: (1) different configurations of hidden layers of the models have been studied until we found a configuration that was compatible with all deployment platforms, (2) an optimal hyperparameter configuration has been searched

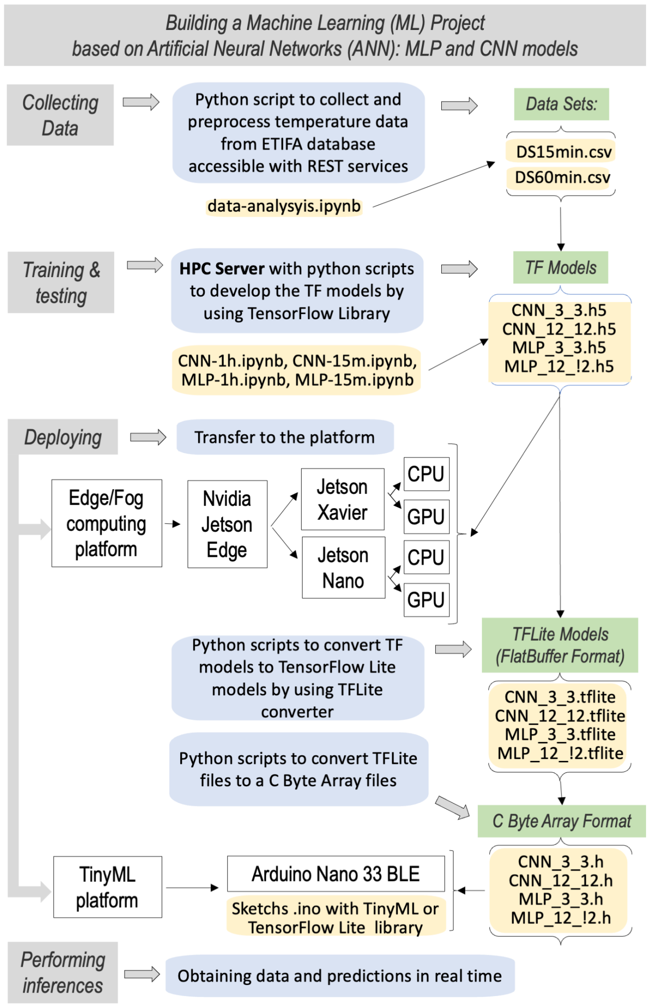

Building a Machine Learning (ML) Project
based on Artificial Neural Networks (ANN): MLP and CNN models
Collecting Data
Python script to collect and preprocess temperature data from ETIFA database accessible with REST services
Data Sets:
DS15min.csv
DS60min.csv
data-analysyis.ipynb
Training & testing
HPC Server with python scripts to develop the TF models by using TensorFlow Library
TF Models
CNN_3_3.h5
CNN_12_12.h5
MLP_3_3.h5
MLP_12_!2.h5
CNN-1h.ipynb, CNN-15m.ipynb, MLP-1h.ipynb, MLP-15m.ipynb
Deploying
Transfer to the platform
Edge/Fog computing platform
Nvidia Jetson Edge
Jetson Xavier
Jetson Nano
CPU
GPU
CPU
GPU
Python scripts to convert TF models to TensorFlow Lite models by using TFLite converter
TFLite Models (FlatBuffer Format)
CNN_3_3.tflite
CNN_12_12.tflite
MLP_3_3.tflite
MLP_12_!2.tflite
Python scripts to convert TFLite files to a C Byte Array files
C Byte Array Format
CNN_3_3.h
CNN_12_12.h
MLP_3_3.h
MLP_12_!2.h
TinyML platform
Arduino Nano 33 BLE
Sketchs .ino with TinyML or TensorFlow Lite library
Performing inferences
Obtaining data and predictions in real time

Fig. 8: Building a Tiny ML project.

to maximize the quality of the results obtained, minimizing the number of trainable parameters of the model and (3) the model has been translated into different programming languages depending on the platform on which it was going to be executed, for example, for Arduino and Raspberry it has been necessary to build the model in C while for the Nvidia Jetsons Python has been used as programming language. In this way, models are obtained with a reduced number of trainable parameters, with a reduced memory space and with low computational requirements that allow them to be executed in each and every one of the platforms that are the object of this study.

To deploy the models on edge computing platforms, it is necessary to first train the models on HPC computers and, using the library's predefined functions, export them to a $.h5$ file. Once the AI model has been exported to a file, it is necessary to move the model file to the edge computing platform on which it will run, load it (also using the library's predefined functions) and finally perform the inference. For the GPU versions of Nvidia's Jetson family, the procedure is similar with the difference of the value of an environment variable, `CUDA_VISIBLE_DEVICES`, whose value was "" to force execution on CPU or "0" to force execution on the GPU available on the system.

Deployment on the TinyML platform is not so straightforward. The trained model must be converted to TensorFlow Lite [32]. TFLite is the framework for deploying ML models on mobile, microcontrollers and other edge devices that has a reduced memory space. FlatBuffers is how a TFLite model is represented, where its extension is .tflite. It allows for a special, portable and efficient format. TFLite has advantages over the buffer model format of the TensorFlow protocol, featuring faster inference because it accesses the data directly without an additional parsing step, as well as a reduced memory size (small code). These features allow TFLite to run efficiently on devices with limited processing and memory resources. A TFLite model can be generated in several ways. First, using an existing TFLite model available in the TFlite SDK [2]. Second, creating a TFLite model, using TFLite Model Maker to create a model with your own custom dataset. It is noteworthy to highlight all models already contain metadata. Third, converting a TF model to a TFLite model by using the TFLite Converter. During conversion, several optimisations such as quantization to reduce model size and latency with little or no loss of accuracy can be applied. Models developed buy this last option no include metadata. In our case, we chose the third option to get the TFlite model to load on the Arduino. It is important to note that not all TF models can be converted to TFLite models that is one of the reason why this work focuses on MLP and CNN models and not in other models such as LSTM. Finally, it is necessary to convert the .tflite file to a byte array in C. in order to run the model on an Arduino.

The last part is represented by the inference process in the device to be used. The edge computing devices receive the data through REST API server, and once they have received the input data, they generate the forecast values. On the other hand, the Arduino receives the information via bluetooh.

4 Evaluation and Discussion

This section shows the performance, energy and quality results obtained on all targeted devices; i.e. Arduino Nano 33 BLE Sense, Raspberry Pi 4, Nvidia Jetson Nano and Nvidia AGX Xavier by executing MLP and CNN models previously presented in Section 3.3. The models developed are named with the following abbreviations: *MLP15min* and *MLP60min* for the MLP model using 15 minutes and 60 minutes data aggregations, and *CNN15min* and *CNN60min*

[2] https://www.tensorflow.org/lite/examples

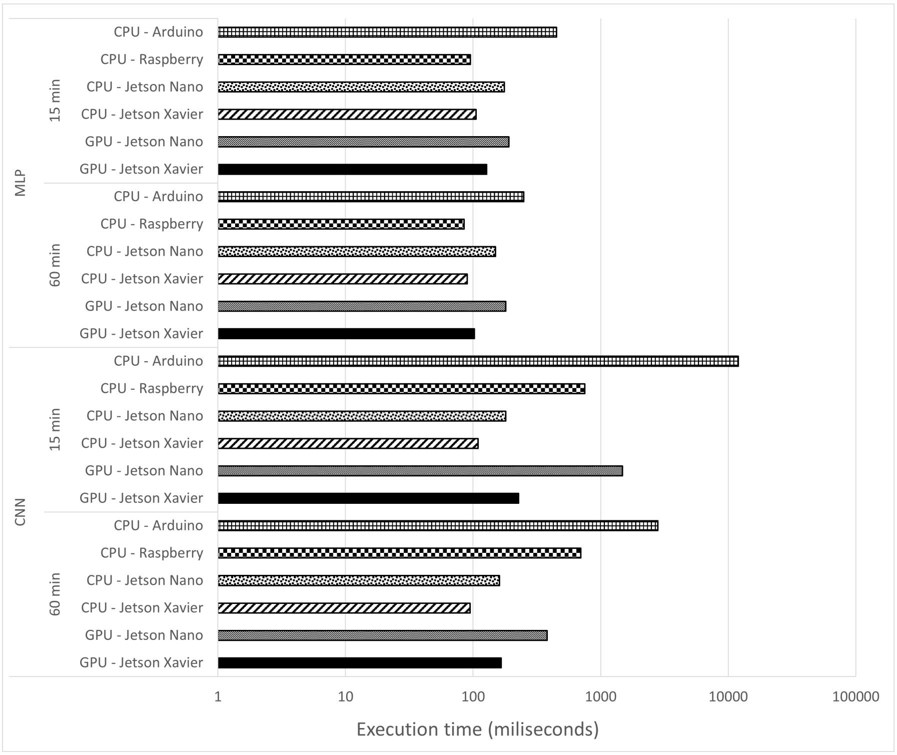


Fig. 9: Execution time (miliseconds) in logarithmic scale of ML models on different devices

for CNN models using the same data aggregations. We refer the reader to Section 3.2 for insights on the dataset construction.

Fig. 9 shows the execution time for inference of MLP and CNN models on all targeted platforms. The Arduino Nano 33 BLE Sense is the worst performance platform as expected. The computational differences of the Arduino against the rest of the platforms are clearly lower in the MLP inference, where a maximum difference of 4.7X speed-up factor is reached, than in the CNN inference, where a speed-up factor of more than two orders of magnitude is reached (110X speed-up factor between Jetson AGX Xavier and Arduino). MLP model inference is computationally lighter than CNN model inference because it has fewer layers and the operations carried out are much simpler than MLP. It is important to note that Fig. 9 is shown in logarithmic scale due to the large difference in performance of the platforms under study. Indeed, it is possible to clearly appreciate, for example, the great difference in time between running the ML workloads on Jetson Xavier CPU or the Arduino one.

Computational differences between Arduino and other platforms are also reduced by the number of elements to be predicted. In the 60-minute dataset, the computational difference is reduced 4 times in terms of speed-up factor for

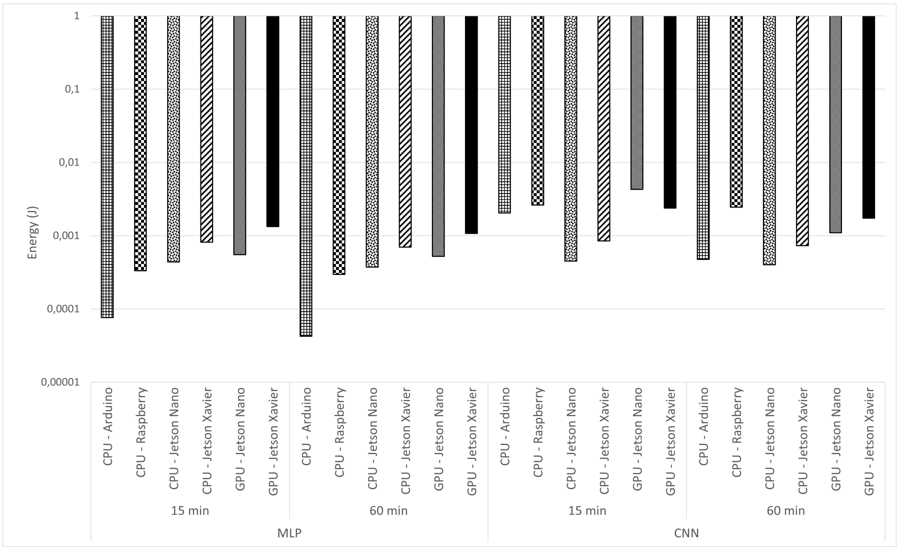


Fig. 10: Energy consumption (Joules) in logarithmic scale of ML models on different devices. Note that below 1, the larger the bar the lower the energy consumption value.

CNN60min compared to CNN15min and 1.5 times for $MLP60min$ compared to $MLP15min$. Overall, the number of elements to be generated in the 60-minute dataset prediction is 4 times smaller than in the 15-minute dataset prediction.

It is also important to note the negative impact of the use of GPUs in this context, reported by the Jetson family. ML models under study are extremely lightweight to be executed on TinyML platforms such as Arduino ones. Therefore, the computational needs of these workloads are not large enough to fill all GPU resources, penalising the execution time of the inference. This penalty is also highlighted in the inference of the CNN model that is clearly affected by memory latency, as CNN has a higher number of memory access. Actually, this memory pressure is also shown by determining the best performing platform, which for MLP inference is the Raspeberry 4 with little difference (5-10%) to the Jetson AGX Xavier which has better technical specifications. However, CNN model inference performs better in the Jetson AGX Xavier by a wide margin since it benefit from the higher memory bandwidth.

Fig. 10 shows the energy consumption in Joules of the different platforms analysed by running the ML model inference. Again, it is important to note that the Y-axis is on a logarithmic scale and therefore below axis 1, the larger the bar, the lower the value of energy consumed. To obtain these figures, the Microchip's PAC1934 power meter is used that provides instantaneous power consumption, energy accumulation, etc., by plugging into USB type-C bus used for power supply [33]. The power consumption is tracked when ML models are running on the targeted devices and thus making predictions. Moreover, Fig.

10 shows the maximum values reached during the execution of each ML model. Fig. 10 is also shown in logarithmic scale given the wide range of values, as was the case in Fig. 9

Regarding the power consumption (Watts) figures, differences are quite large; the Arduino Nano consumes only 0.17 W, defeating Raspberry PI by a wide margin as it consumes 3.5 W. Regarding Nvidia Jetson platforms, there are also some important differences. The idle power consumption of Jetson Nano is in the range of 1.3 W and 1.6 W. The power consumption of CPU-based MLP and CNN models is in range of 2.4 W and 2.6 W. Whenever the GPU is switched on, the power consumption of the Nvidia Jetson Nano increases by a factor of 13.79%, reaching up to 2.9-3.1 W. The power consumption of the Jetson AGX Xavier with GPU enabled reaches up to 10.15 W while with only executing the CPU version consumes 7.77 W. With this in mind, the Arduino is by far the most energy efficient platform (about 1 order of magnitude), followed by the Raspeberry PI, the Jetson in CPU mode and finally, the least energy efficient platform for these workloads is the Jetson family, using the GPU. As discussed above, the small difference in performance between the platforms is not enough to hide the energy consumption of the platforms. The Arduino would have a power consumption of 0.17 Watts. Once the instantaneous consumption of the platforms was calculated, the energy was calculated in a straightforward way, multiplying the execution time in seconds, and thus being able to obtain a metric that shows the best model-device combination in the trade-off between execution time and energy consumption. Actually, according to the energy figures shown in Fig. 10, the most energy efficient model-device combination is the $MLP60min$ and the $CNN60min$, running on the Arduino microcontroller. The Arduino microcontroller consumes less power than the Raspberry although it takes longer to perform the predictions. This result is exactly what was expected due to the hardware difference between the two devices and shows perfectly why TinyML, and more generally bringing ML to microcontrollers, is so important and convenient.

Finally, Fig. 11 shows the accuracy of the models by forecasting the internal temperature of the greenhouse for the next 3 hours. The metrics used in this evaluation are as follows. The Root Mean Squared Error (RMSE) is the square root of the average of squared errors. The Mean Absolute Error (MAE) is the average absolute difference between the observed and the predicted value. These two metrics are measured in Celsius degrees as they would be errors in the temperature prediction. The lower the error, the better the result of the technique obtained. Finally, the coefficient of determination R^2 is used to determine the goodness of the model. This value is estimated at 0 and 1, with values close to 1 being a better fit and values closer to 0 a worse fit. Results in Fig. 11 have been calculated for both the MLP and CNN techniques, dividing into 15-minute and 60-minute datasets, and averaged them for each of the devices used. In addition, the standard deviation obtained for each metric is shown at the top of the graph.

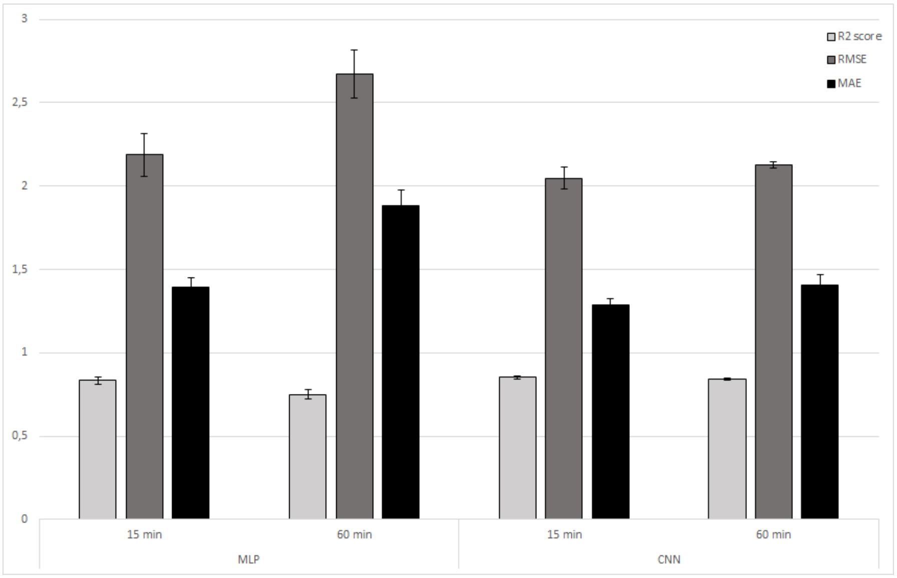


Fig. 11: Metrics (R^2 score, RMSE and MAE) of ML models on different devices. The bars show the mean of the metric in question for the different platforms while the vertical line at the top of the bars shows the standard deviation.

Analysing the results, the best model on the RMSE value is $CNN15m$ as expected. The CNN model is more complex ANN than the straightforward MLP and can theoretically better capture the behaviour of the time series. Moreover, the dataset with fifteen-minute measurements has a larger size, so the models have more data to improve the training. In general, RMSE results show that the mean deviation from the forecast values is between 2°C and 3°C. MAE results, however, show that the mean difference between the forecast and the observed temperature is between 1 °C and 2 °C. Since the RMSE-MAE difference is not large enough, it means that large errors are unlikely to have occurred. As for the R^2 results, $CNN15m$ and $CNN60m$ reached a value of almost 0.85, which means that the predictions of these models are quite accurate, while $MLP15m$ is 0.8 and $MLP60$ has the lowest score of 0.7. The difference in terms of evaluation metrics between the two models is obvious but it does not necessarily mean that MLPs are not a valid choice because, as stated before, they are lighter and their execution is faster. Of the three metrics shown in Fig. 11 and with respect to quality results, $CNN15m$ shows the best performance, followed by $CNN60m$, $MLP15m$ and $MLP60m$.

It is important to highlight the great difference between the 15-minute and 60-minute datasets. As we have already mentioned, the CNN technique obtains the best result compared to MLP, the models obtained for the two datasets have a similar behaviour. Better results are always obtained with the 15-minute dataset than with the 60-minute dataset. The reader may think that the models are over-fitted or under-fitted. But the reality is that the 15-minute dataset aggregates a lower granularity, so the deviation of the 15-minute temperature is smaller than it does for 60-minutes dataset. This leads to a lower prediction error in 15 minutes than in 60 minutes and this also leads to a better system tuning. The 15min-MLP, 60min-MLP models are run with the same parameters optimised to the average case. The same is valid for the models 15min-CNN, 60min-CNN. Thus, we can conclude that when predicting temperature it will always be better to have lower granularity, not only for the improvement of the results, but from an agricultural point of view, to be able to act and control the temperature in short periods of time.

With all of the above in mind and taking into account the performance, energy and quality figures, we are tempted to highlight as the best hardware-software combination the $CNN60m$ model running on the Arduino microcontroller. The $CNN60m$ model offers similar quality numbers to the $CNN15m$ model but the energy used when running that model on the microcontroller is lower. However, there are other factors to consider and these depend on the particular application case where the infrastructure is deployed. In an application such as smart greenhouse management, the energy consumed is probably an important aspect while the accuracy of the model is not so-critical as long as the error is under certain threshold. MLP models executed on the Arduino microcontroller are a better choice as it offers almost an order of magnitude lower energy consumption. Of course, if power consumption is not an issue and power is available, we could scale up to edge solutions such as Raspberry PI in order to optimise the execution time. We discourage platforms with embedded GPUs in such energy-constrained contexts where workloads do not take advantage of all the computational resources provided.

5 Conclusion and Future Work

Edge computing is leading the HPC-AI intersection in the IoT arena. However, the computational gap between the two disciplines is still huge. It requires simpler algorithms and platforms to offer novel solutions to emerging applications to make them viable in this context, i.e., meeting both runtime and power consumption requirements. In this paper, we analyse a wide range of edge computing platforms in the context of TinyML. In particular, we use two lightweight artificial neural networks, namely MLP and CNN, for indoor temperature prediction in greenhouses. We use them as benchmarks to evaluate four different edge computing platforms; an Arduino microcontroller, a Raspberry-Pi, and two platforms of the familiar Nvidia Jetson.

Our results show that a balance between algorithmic complexity, the accuracy of the results obtained, and energy efficiency is essential to obtain robust and operational systems in real-world environments. We claim that the best hardware-software combination for the problem analysed would be the $CNN1h$ model running on the Arduino microcontroller. We also observe the low impact of introducing more computation at this level, as seen in the energy efficiency numbers of Nvidia's Jetson family. Actually, the energy consumption and thus the carbon footprint, is one a critical factor in greenhouses, while the accuracy of the model is not so critical as long as the error is below a certain threshold. That's why MLP models running on the Arduino microcontroller may be a better choice, as they offer almost an order of magnitude lower power consumption. We also conclude that platforms with integrated GPUs in these energy-constrained contexts may not be sufficiently fruitful, unless the model run is sufficiently complex.

We recognise that the problem addressed in this paper is relatively simple. When scaled up to a more complex problem, e.g., multivariate greenhouse modelling or smart irrigation, more complex models will need to be developed to achieve adequate accuracy and require computational power. We envision this algorithmic complexity as a chance to also scale in computational horsepower at the edge.

Declarations

Ethical Approval

Not applicable

Conflict of interest

The authors declare they do not have any conflict of interest.

Authors' contributions

Conceptualization, J.M.C. and R.M.E.; methodology, J.M.C., R.M.E and A.B.C.; software, J.M.G. and J.L.P.; validation, J.M.C., R.M.E., P.M and A.B.C.; formal analysis, A.B.C., J.M.C., P.M. and R.M.E.; investigation, J.M.G. and J.M.C.; data curation, A.B.C., R.M.E. and J.M.G.; writing—original draft preparation, J.M.G., J.L.P., R.M.E., J.M.C. and A.B.; writing—review and editing, P.M. and J.M.C.; visualization, J.L.P., J.M.G. and R.M.E.; supervision, J.M.C. and P.M.; project administration, J.M.C.; funding acquisition, J.M.C.

All authors have read and agreed to the published version of the manuscript.

Funding

This work is derived from R&D projects RTC2019-007159-5, as well as the Ramon y Cajal Grant RYC2018-025580-I, funded by MCIN/AEI/10.13039/501100011033, "FSE invest in your future" and "ERDF A way of making Europe".

Availability of data and materials

All data and materials are available on request from the authors of this paper.

References

[1] Feki, M.A., Kawsar, F., Boussard, M., Trappeniers, L.: The internet of things: the next technological revolution. Computer **46**(2), 24–25 (2013)

[2] Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of things (iot): A vision, architectural elements, and future directions. Future generation computer systems **29**(7), 1645–1660 (2013)

[3] Tahsien, S.M., Karimipour, H., Spachos, P.: Machine learning based solutions for security of internet of things (iot): A survey. Journal of Network and Computer Applications **161**, 102630 (2020)

[4] Papadokostaki, K., Mastorakis, G., Panagiotakis, S., Mavromoustakis, C.X., Dobre, C., Batalla, J.M.: Handling big data in the era of internet of things (IoT). Springer (2017)

[5] Satyanarayanan, M.: The emergence of edge computing. Computer **50**(1), 30–39 (2017)

[6] Capra, M., Peloso, R., Masera, G., Ruo Roch, M., Martina, M.: Edge computing: A survey on the hardware requirements in the internet of things world. Future Internet **11**(4), 100 (2019)

[7] Warden, P., Situnayake, D.: TinyML. O'Reilly Media, Incorporated (2019)

[8] Portilla, J., Mujica, G., Lee, J.-S., Riesgo, T.: The extreme edge at the bottom of the internet of things: A review. IEEE Sensors Journal **19**(9), 3179–3190 (2019)

[9] Deng, L., Yu, D.: Deep learning: methods and applications. Foundations and trends in signal processing **7**(3–4), 197–387 (2014)

[10] Guillén-Navarro, M.Á., Martínez-España, R., Bueno-Crespo, A., Ayuso, B., Moreno, J.L., Cecilia, J.M.: An lstm deep learning scheme for prediction of low temperatures in agriculture, pp. 130–138. IOS Press (2019)

[11] Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)

[12] Abhishek, K., Singh, M., Ghosh, S., Anand, A.: Weather forecasting model using artificial neural network. Procedia Technology **4**, 311–318 (2012)

[13] Lee, S., Lee, Y.-S., Son, Y.: Forecasting daily temperatures with different time interval data using deep neural networks. Applied Sciences **10**, 1609 (2020)

[14] Zhang, Z., Dong, Y.: Temperature forecasting via convolutional recurrent neural networks based on time-series data. Complexity **2020** (2020)

[15] Jung, D.-H., Kim, H.S., Jhin, C., Kim, H.-J., Park, S.H.: Time-serial analysis of deep neural network models for prediction of climatic conditions inside a greenhouse. Computers and Electronics in Agriculture **173**, 105402 (2020)

[16] Codeluppi, G., Cilfone, A., Davoli, L., Ferrari, G.: Ai at the edge: a smart gateway for greenhouse air temperature forecasting. In: 2020 IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor), pp. 348–353 (2020). IEEE

[17] Guillén, M.A., Llanes, A., Imbernón, B., Martínez-España, R., Bueno-Crespo, A., Cano, J.-C., Cecilia, J.M.: Performance evaluation of edge-computing platforms for the prediction of low temperatures in agriculture using deep learning. The Journal of Supercomputing **77**(1), 818–840 (2021)

[18] Codeluppi, G., Davoli, L., Ferrari, G.: Forecasting air temperature on edge devices with embedded ai. Sensors **21**(12), 3973 (2021)

[19] Chang, Z., Liu, S., Xiong, X., Cai, Z., Tu, G.: A survey of recent advances in edge-computing-powered artificial intelligence of things. IEEE Internet of Things Journal (2021)

[20] Dubey, A.K., Kumar, A., García-Díaz, V., Sharma, A.K., Kanhaiya, K.: Study and analysis of sarima and lstm in forecasting time series data. Sustainable Energy Technologies and Assessments **47**, 101474 (2021)

[21] Seshadri, K., Akin, B., Laudon, J., Narayanaswami, R., Yazdanbakhsh, A.: An evaluation of edge tpu accelerators for convolutional neural networks. arXiv preprint arXiv:2102.10423 (2021)

[22] Rashid, N., Demirel, B.U., Al Faruque, M.A.: Ahar: Adaptive cnn for energy-efficient human activity recognition in low-power edge devices. IEEE Internet of Things Journal (2022)

[23] Cruz, M., Mafra, S., Teixeira, E., Figueiredo, F.: Smart strawberry farming using edge computing and iot. Sensors **22**(15), 5866 (2022)

[24] Feng, B., Ding, Z., Jiang, C.: Fast: A forecasting model with adaptive sliding window and time locality integration for dynamic cloud workloads. IEEE Transactions on Services Computing (2022)

[25] Ding, Z., Feng, B., Jiang, C.: Coin: A container workload prediction model focusing on common and individual changes in workloads. IEEE

Transactions on Parallel and Distributed Systems **33**(12), 4738–4751 (2022)

[26] Alongi, F., Ghielmetti, N., Pau, D., Terraneo, F., Fornaciari, W.: Tiny neural networks for environmental predictions: an integrated approach with miosix. In: 2020 IEEE International Conference on Smart Computing (SMARTCOMP), pp. 350–355 (2020). IEEE

[27] Pettit, A.: A non-parametric approach to the change-point problem. Applied statistics **28**(2), 126–135 (1979)

[28] Bishop, C.M., et al.: Neural networks for pattern recognition. Oxford university press (1995)

[29] Tadeusiewicz, R.: Neural networks: A comprehensive foundation: by Simon HAYKIN; Macmillan College Publishing, New York, USA; IEEE Press, New York, USA; IEEE Computer Society Press, Los Alamitos, CA, USA; 1994; 696 pp.; 69–95; ISBN: 0-02-352761-7. Pergamon (1995)

[30] Li, Y., Hao, Z., Lei, H.: Survey of convolutional neural network. Journal of Computer Applications **36**(9), 2508 (2016)

[31] Sahu, M., Dash, R.: A survey on deep learning: convolution neural network (CNN). Springer (2021)

[32] Tensorflow: Tensorflow Lite for Microcontrollers. https://www.tensorflow.org/lite/microcontrollers Accessed 2021-07-06

[33] Inc., M.T.: PAC1934 USB C POWERMETER. https://www.microchip.com/en-us/development-tool/ADM00921