SEPARATE: A tightly coupled, seamless IoT infrastructure for deploying AI algorithms in smart agriculture environments

Juan Morales-García[a,*], Andrés Bueno-Crespo[a], Raquel Martínez-España[b], Francisco J. García[c], Sergio Ros[c], Julio Fernández-Pedauyé[d], José M. Cecilia[d]

[a]*Computer Science Department, Catholic University of Murcia (UCAM), Murcia, Spain*
[b]*Information and Communications Engineering Department, University of Murcia (UM), Murcia, Spain*
[c]*Nutricontrol, S.L., Cartagena, Spain*
[d]*Computer and Systems Informatics Department, Polytechnic University of Valencia (UPV), Valencia, Spain*

Abstract

Precision agriculture generates large datasets from IoT infrastructures deployed for continuous crop monitoring. This data requires analysis to usefully transform this data deluge into insights that can deliver value-generating services to farmers in a timely manner. This paper introduces *SEPARATE*; a dynamic interoperable and decentralized infrastructure for executing both, training and inference stages of deep learning (DL) algorithms in smart agriculture scenarios. The presented infrastructure allows the execution of the inference stage at the edge, achieving a highly efficient and responsive local temperature prediction service to take actions based on the predictions generated. Moreover, the training stage is offloaded to the cloud along with the generated historical data, allowing the trained model to be periodically updated at the edge. On the one hand, our results show that the Convolutional Neural Network model together with the Long Short-Term Memory technique (CNNLSTM) obtains the best results in both prediction accuracy and computational time. On the other hand, an analysis has been carried out to determine how often the model must be retrained, obtaining results that indicate that from day 9-10, it would be necessary to retrain the model, although, until day 20, the precision is not greatly reduced. Moreover, the *SEPARATE* infrastruc-

*Principal corresponding author
Email address: jmorales8@ucam.edu (Juan Morales-García)

ture enables the execution of real-time inference from sensor-generated data and seamless model retraining in an operational greenhouse for temperature forecast with satisfactory performance.

Keywords: Publish/Subscribe Infrastructure, Edge Computing, Machine Learning, Deep Learning, Internet of Things, Smart Agriculture

---

1. Introduction

Precision agriculture is increasingly making use of new technologies in order to improve profits and reduce costs [1]. In a globalized world where costs are increasing day by day, it is important to have autonomous systems that allow farmers to better control their crops [2]. Monitoring and anticipation in decision-making can save a lot of production costs and crop losses [3]. Greenhouses, due to their structure, allow crop control, both of growth, pests, and climatic variables that directly influence crop yields [4]. For this control, the devices and sensors provided by the Internet of Things (IoT) provide great versatility and convenience for efficient data capture.

Current operational IoT solutions in the area of smart agriculture in real-world environments are mainly based on centralized IoT systems, where data is sent to a centralized cloud-based architecture, processed, and further analyzed [5]. However, agricultural environments are often located in rural areas where connectivity and power supply are limited [6]. Indeed, a dynamic, interoperable, and decentralized architecture is mandatory in these environments to achieve highly efficient and responsive data-based services. Some solutions have been proposed that bring the execution of machine learning (ML) and even deep learning (DL) algorithms close to the data capture, i.e. at the edge, in agricultural environments [7, 8, 9]. Edge computing [10] is becoming a widely used approach towards decentralization, where preliminary computations on data are carried out in (or close to) the data capture devices, providing a number of advantages, including energy savings, responsive application and service development, highly scalable, reliable and secure system design.

However, computational devices that are available at the edge typically rely on batteries or energy harvesters, leading to ultra-low power designs, but also limiting the workloads to be executed on them. Actually, edge Computing is limited to the inference stage of Machine Learning / Deep Learning (ML/DL) algorithms in what is recently called TinyML [11]. This makes sense because at this level of the

IoT infrastructure, very little computational horsepower is available and thus computationally heavy workloads, such as those found at the training stage of ML/DL algorithm cannot be executed in a reasonable timeframe and with a manageable energy budget. However, training ML/DL algorithm periodically is mandatory to achieve more accurate results in these environments of rapidly changing conditions, so influenced by the abrupt changes brought about by climate change [12]. In this paper, we present *SEPARATE*; a seamless and tightly coupled IoT infrastructure for developing training and inference of ML/DL algorithms in operational smart agriculture environments. *SEPARATE* relies on a publish / subscribe (Pub/Sub) system based on Message Queuing Telemetry Transport (MQTT) protocol to perform data analytics at the edge of an IoT infrastructure deployed in an operational greenhouse located in Murcia (Spain). Particularly, the *SEPARATE* infrastructure receives information through MQTT from the air temperature sensors deployed inside the greenhouse that feed ML/DL models to predict the climatic state of the greenhouse in the following hours (i.e., inference stage). It is also important to study the need to retrain the prediction model to always have the best available precision without affecting response times. Moreover, *SEPARATE* also sends the data to the cloud via MQTT to store the historical data and thus to periodically retrain the model to be updated at the inference stage and thus provide a more accurate forecast. The main contributions of the paper include the following:

1. The *SEPARATE* infrastructure is introduced to provide an interoperable and decentralized dynamic architecture for ML/DL training and inference.
2. *SEPARATE* is designed for a real and operational IoT infrastructure. Our developments and tests have been carried out in an operational greenhouse deployed in Murcia (Spain)
3. Different ML/DL models are considered in terms of execution time and accuracy to establish the best combination in the quality and performance tradeoff.
4. Study and analysis of the days needed to retrain the best model, without significantly losing precision.
5. *SEPARATE* offers an infrastructure not only for monitoring but also for predicting actions in advance, thanks to its prediction models for several hours ahead.

The rest of the paper is organized as follows. Section 2 shows related works within the umbrella of ML/DL as applied to Pub/Sub solutions in forecasting cli-

matic variables in greenhouses. Section 3 describes in detail the different approaches proposed and the artificial intelligence methods used to compare an evaluate results. Section 4 shows the performance results for the Pub/Sub solution and for the artificial intelligence models before discussing them in Section 5. Finally, section 6 presents the main conclusions and discusses future works.

2. Related works

The MQTT communications protocol is increasingly being used in precision agriculture, replacing the HTTP protocol, [13]. Although the HTTP protocol uses a request/response architecture and HTTP can transfer a large number of data in small packets that can cause a large overhead. Therefore, due to this overhead, communication for IoT services via this protocol can cause serious bandwidth problems. Moreover, all HTTP calls are stateless, which leads to authentication every time you connect, as you connect to the IP or URL to make the REST API calls, the session is not saved and thus, after getting the response, the device closes the connection. This is due to possible network overload [14]. In contrast, MQTT (Message Queue Telemetry Transport) is a lightweight protocol that is designed for the IoT ecosystem. It was invented by IBM, is based on the OSI TCP/IP model, and has a very lightweight application layer with a header size of 2 bytes. MQTT follows an asymmetric architecture and operates under the publish and subscribe protocol. It is designed to send a message to one or more devices with low latency but is not recommended for sending large amounts of data. Given its characteristics, it is very useful for use in IoT systems, since among its advantages we find the secure delivery of the message, thus avoiding the loss of data [15, 16]. Given the advantages of the MQTT protocol, it is increasingly used in IoT environments and in the world of precision agriculture. For instance, in [17] the authors propose a precision agriculture system based on wireless sensor networks with the MQTT protocol for monitoring and controlling the environmental conditions of a greenhouse by collecting information on humidity, temperature, light, and nutritional needs of the plants. In [18], the design and implementation of an intelligent irrigation system to automate the irrigation system in agricultural fields are proposed. The system is a near real-time system using the MQTT protocol for communications between the sensor side and the client side. The authors of [2] present an open-source platform covering automated precision agriculture scenarios. For this purpose, the platform is distributed on three levels, Cyber-Physical Systems, Edge computing, and Cloud. To connect the sensors and actuators with the end user, they use IoT technologies and protocols such as MQTT, the Constrained

Application Protocol (CoAP), and FIWARE, among others. The platform is successfully tested in a hydroponic greenhouse. In [19], authors proposed the design of a remote monitoring and control system for greenhouses. The system uses IoT to collect greenhouse parameters such as temperature, relative humidity, and luminosity. After collecting the data and sending it to the server, the IoT polls the data on the internet through the MQTT protocol and compares it with existing parameters in order to send a correction to the greenhouse if necessary. A system for monitoring and controlling microclimatic parameters of a greenhouse is proposed in [20]. In this case, the system collects temperature and humidity values as well as gas values. The data is sent through an Arduino to a raspberry via the MQTT protocol.

Another aspect of agriculture where IoT systems and the MQTT protocol is also very useful is for the prevention and care of bees to avoid Colony Collapse Disorder, a disorder still under study. To obtain data for this study, in [21] the authors propose the design of a near real-time system that collects temperature, humidity, and weight data and sends it via MQTT to a ThingsBoard for analysis, the system called Beemon works continuously and in open-air hives.

As can be seen, the proposed systems are used to monitor and control different agricultural systems, from different points of view. The difference with our proposal is that in our case, in addition to carrying out monitoring tasks, we also carry out advance prediction tasks, in order to take measures before any event occurs, thus achieving a better optimization of resources.

3. Materials and Methods

This section provides the materials and methods of the *SEPARATE* infrastructure. As previously mentioned, it provides an interoperable and decentralized dynamic architecture for ML/DL training and inference in an operational greenhouse. Therefore, we introduce the operational greenhouse where *SEPARATE* is deployed before providing to the reader the main insights of *SEPARATE* infrastructure. The case study here presented aims at forecasting the internal temperature of this greenhouse through different ML/DL methods that are also presented, showing the main parameters used for the evaluation carried out in the next section.

3.1. Operational greenhouse

Figure 1: Targeted operational greenhouse located at Murcia (Spain)

Figure 1 shows the operational greenhouse targeted for this study, namely *ETIFA*. *ETIFA* is an operational greenhouse hosted by NUTRICONTROL; a Spanish leading company in developing automatic fertigation and Climate control technology. *ETIFA* is located in Murcia (South-eastern Spain), a semi-arid region where the average annual temperature is around 25 ºC. This greenhouse has a surface area of 50 m^2 and operates with a system for climate control and fertigation.

ETIFA fertigation and climate control is carried out by NUTRICONTROL's OPTIMUM system; a complete solution for climate management and fertigation of these environments. The NUTRICONTROL's OPTIMUM system is orchestrated by a CPU-based node (OPTIMUM Orchestator, from now on) where all sensors (input/output) are plugged into in a modular way. Among the sensors available in *ETIFA* are temperature, humidity, radiation, and wind speed, just to name a few.

Of particular interest to us is the air temperature inside the greenhouse as it is the target for the forecast carried out in this paper. This variable is measured every 5 minutes in the 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.

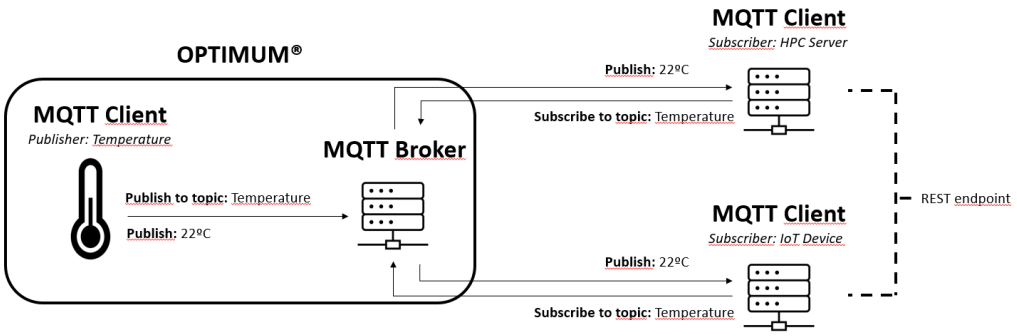


Figure 2: *SEPARATE* main building blocks based on MQTT schema

3.2. SEPARATE infrastructure

The *SEPARATE* infrastructure relies on the MQTT standard protocol [22]. MQTT is a lightweight Pub/Sub messaging transport system where sensors can publish the generated data (e.g. temperature sensor) and several nodes can be subscribed to receive values in NRT. Figure 2 shows the main *SEPARATE* building blocks that are deployed in the greenhouse. First, the MQTT broker or server, deployed in the OPTIMUM orchestrator, is responsible for dispatching messages between the sender (or publisher) and the appropriate receivers. *SEPARATE* is based on Java-based open source HIVEMQ Community Edition (CE) [1].

MQTT clients are subscribed to and publish information on a particular topic. In our case, the MQTT client is developed using the Eclipse Paho MQTT Python client library [23] that enables applications to connect to an MQTT broker to publish messages and to subscribe to topics and receive published messages. *SEPARATE* infrastructure is based on three MQTT clients. The first client is hosted in the OPTIMUM orchestrator and publishes data to the topic "Temperature" from the greenhouse as soon as it is received from the sensors. The second and third clients are subscribed to the same topic but are deployed in different computing nodes, depending on the task associated with each of them.

The second MQTT client is deployed on an Nvidia Jetson Nano in the greenhouse; very close to the capture node, i.e. at the edge. This client is designed to run the ML/DL inference of the model to forecast the indoor temperature of the greenhouse for the next few hours. In this way, the MQTT client periodically receives the data temperature that is stored in a CSV file in the Jetson Nano. Another

[1] https://github.com/hivemq/hivemq-community-edition

background process (i.e. cron job) runs the model inference every set time frame (e.g., every 15 minutes), which collects the information gathered by the MQTT client and generates the prediction. It is important to note that each time frame, the ML/DL inference is executed taking into account the last updated data generated by the greenhouse and thus more accurate predictions should be generated using these new records.

The last MQTT client is deployed in a cloud server hosted in our lab at UPV. This MQTT client is also subscribed to "Temperature" topic and once data is received from this topic, this cloud-based MQTT client stores it in an InfluxDB database where historical data is increasingly generated. On this cloud-based server, data quality controls, sanitization processes, etc. are carried out to prepare the data for an efficient training procedure. It is important to note that the training procedure does not need to be carried out on a sub-daily basis or even a daily basis. This is indeed a configuration parameter that is set according to the computational and accuracy tradeoff.

In preliminary evaluations conducted in the particular context of the ETIFA greenhouse, it has been empirically observed that trained models give predictions at the inference stage within the same quality range as models trained with data from up to a month before. However, important differences in the forecast accuracy start to become relevant if ML/DL models are trained with data of more than a month before the horizon is predicted. Depending on the application, the user may prefer a more conservative scenario and retrain more frequently to obtain the most accurate results, or be more computationally efficient and only train when significant differences in the results are really noticeable. It is worth noting that training the heavier ML/DL ETIFA models can take several computational days, as it will be shown in section 4.2.

In any case, the model will be trained on the cloud server, and once trained, *SEPARATE* has to provide a mechanism to update the model trained on the edge in a transparent way to the user. The chosen way has been to enable a REST endpoint where a process on the edge node can update the model periodically. In our case, a month has been set by default although this can be easily changed in the configuration schema.

3.3. Datasets

For the evaluation of the accuracy of the AI models, two different representative points of the year have been taken into account; i.e. winter and summer. This dichotomy is due to the fact that these are the two points at which the climatological variables (and their behavior) differ the most. Moreover, data time

granularity has been also considered for the evaluation, i.e., the data are grouped into 15-minute, 30-minute, and 60-minute periods. In this way, we can verify that (1) the model fits the data correctly, (2) it is able to predict at any point of the year and (3) that different time granularity can be applied depending on the needs of the problem. Finally, short-term (12 hours) and long-term (24 hours) prediction has also been taken into account.

Datasets	Start date	End Date	# Instances
CLEAN-DS-15-SUMMER	18-12-18	06-06-21	86544
CLEAN-DS-15-WINTER	18-12-18	17-01-21	73104
CLEAN-DS-30-SUMMER	18-12-18	06-06-21	43273
CLEAN-DS-30-WINTER	18-12-18	17-01-21	36553
CLEAN-DS-60-SUMMER	18-12-18	06-06-21	21637
CLEAN-DS-60-WINTER	18-12-18	17-01-21	18277
DIRTY-DS-15-SUMMER	18-12-18	06-06-21	86544
DIRTY-DS-15-WINTER	18-12-18	17-01-21	73104
DIRTY-DS-30-SUMMER	18-12-18	06-06-21	43273
DIRTY-DS-30-WINTER	18-12-18	17-01-21	36553
DIRTY-DS-60-SUMMER	18-12-18	06-06-21	21637
DIRTY-DS-60-WINTER	18-12-18	17-01-21	18277
SMOOTH-DS-15-SUMMER	18-12-18	06-06-21	86544
SMOOTH-DS-15-WINTER	18-12-18	17-01-21	73104
SMOOTH-DS-30-SUMMER	18-12-18	06-06-21	43273
SMOOTH-DS-30-WINTER	18-12-18	17-01-21	36553
SMOOTH-DS-60-SUMMER	18-12-18	06-06-21	21637
SMOOTH-DS-60-WINTER	18-12-18	17-01-21	18277

Table 1: Dataset description

Table 1 summarizes 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 of a greenhouse between the indicated dates, distinguishing whether the dataset ends on a summer or winter day. It is important to clarify that in southeastern Spain, June temperatures are already summer temperatures.

3.4. Artificial Intelligence models

This section introduces four ML/DL models that have been used for the evaluation of *SEPARATE*. We refer the reader to [24] for insights.

- **Artificial Neural Networks (ANN)**: A neural network is a 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 single perceptron (or artificial neuron) can be imagined as a logistic regression. The artificial neural network, or ANN, is a group of multiple perceptrons in each layer forming a multilayer perceptron (MLP). The MLP we use in this work is composed of three layers: input, hidden, and output. The input layer receives the input features, the hidden layer processes the inputs and the output layer produces the output. Essentially, each layer tries to learn certain weights. Artificial neural networks have the ability to learn any complex relationship between input and output because they use an activation function that allows them to learn non-linear properties in the network. [25, 26].

- **Convolutional Neural Network (CNN)**: Convolutional neural network 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. [27].

- **Long Short-Term Memory (LSTM)**: this model of DL is commonly used because in addition to working with time series like recurrent models, but with the advantage that this network allows for long-term memory. LSTM is a type of recurrent neural architecture with a state memory and multilayer cell structure [28]. LSTM unit is composed of a cell, an input gate, an output gate, and a forget gate (Figure 3, only the LSTM layer). The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell. The LSTM differs from a classic recurrent network in that it does not overwrite its content at each time step but is able to decide whether to keep the existing memory through

the introduced doors. If the LSTM unit detects an important characteristic of an input sequence at an early stage, it carries this information over long distances, therefore it detects long-distance dependencies.

- **Convolutional Neural Network + Long Short-Term Memory (CNN-LSTM)**: This model is a combination of CNN and LSTM (also known as ConvLSTM). It presents a convolutional network where the MLP layer fully connected to the final layer has been replaced by an LSTM network. Therefore, the CNN is in charge of automatically extracting the input features and the LSTM is in charge of obtaining the regression results (see Figure 3).

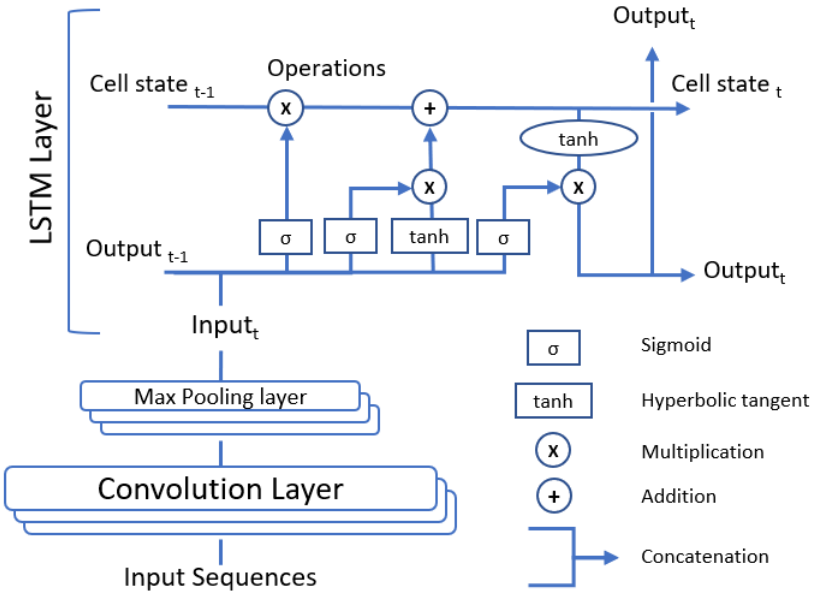


Figure 3: Model CNNLSTM. In the first stage, the convolutional layer is in charge of extracting the feature vector so that finally the LSTM layer performs the forecast.

Table 2 shows all hyperparameters (rows) used for each model (columns). A brief description of the meaning of each hyperparameter is given below:

- **Units**: Number of neurons used in hidden layers.
- **Filters**: Features detector.

- **Kernel size**: Filters matrix used to extract the features from the dataset.
- **Strides**: Number of pixels shifts over the input matrix.
- **Activation function**: Function that decides if a neuron should be (or not) activated.
- **Batch size**: Size of bach used for training/forecasting.
- **Epochs**: Number of epoch used in training.
- **Optimizer**: Function that optimizes the learning of an artificial intelligence model, updating its neurons' weights depending on the error evaluation.
- **Loss function**: Function used to evaluate the error of the model in each epoch.
- **Learning rate**: Percentage change with which weights are updated at each iteration.

HYPERPAREMETER	**MLP**	**CNN**	**LSTM**	**CNNLSTM**
Units	70	–	70	–
Filters	–	64	–	64
Kernel size	–	1	–	2
Strides	–	1	–	4
Activation function	Tanh	Tanh	Tanh	Tanh
Batch size	2880	2880	2880	2880
Epochs (+ *EarlyStopping*)	3000	3000	3000	3000
Optimizer	Adam	Adam	Adam	Adam
Loss function	MSE	MSE	MSE	MSE
Learning rate (+ *ReduceLROnPlateau*)	0.003	0.003	0.003	0.003

Table 2: Hyperparameters used for each model. A dash in a cell indicates that the model does not have that parameter.

4. Results

This section shows the results of the different experiments that have been carried out. They can be divided into two main sets. On one side, an experiment

is proposed to evaluate the effectiveness of the NRT system based on *SEPARATE* (see section 4.1). While, on the other side, an experiment will be proposed to evaluate the accuracy of ML/DL models in an NRT environment (see section 4.2). In addition, a study is carried out on the need to retrain the model from time to time so as not to lose precision in the predictions. These results are broadly discussed in Section 5.

4.1. Pub/Sub solution evaluation

In the following, the execution time on HPC and edge computing platforms for training and inference of ML/DL techniques is shown and analyzed individually. This experiment leads us to argue the computational differences between the two devices and the need for *SEPARATE* to coordinate training and inference in this context.

4.1.1. HPC evaluation

The HPC computing node is evaluated by running the training and inference for the different ML/DL models described in Section 3.4. This node is composed of an AMD EPYC 7282, 64 CPUs (2 sockets x 16 cores per socket x 2 threads per core) at 2.8 MHz for each core. Moreover, it also contains two NVIDIA A100-PCIE-40GB (Tesla architecture) and it is endowed with up to 256 GB and 2 TB SSD.

Tables 3, 4, 5, 6 show the MLP, CNN, LSTM and CNNLSTM execution time on the HPC server targeted for the multicore CPU or a GPU, respectively. Each row of the table represents the performance times for each of the datasets described in Section 3.3. Each column of the table represents the performance times studied and is grouped into training and inference, for each of the two platforms studied (CPU and GPU). For inference, a distinction is made between 12-hour and 24-hour inference.

In particular, table 3 shows that the MLP training time on CPU range from 263.945 to 729.050 seconds, while on GPU they range from 241.099 to 500.245 seconds, obtaining up to 1.5X performance in the best scenario. Inference times are almost identical on both CPU and GPU, being a few milliseconds slower on GPU, and ranging between 0.592 and 5.022 seconds. Regarding the inference between 12 and 24 hours, there is practically no difference, since the 24-hour time is practically twice as long as the 12-hour time, which is consistent since twice as many values are inferred.

Table 4 shows that the CNN training time on CPU range from 2,089.193 to 33,877.128 seconds, while on GPU they range from 275.392 to 755.894 seconds,

Platform	CPU			GPU		
Times	Training	Inference		Training	Inference	
Dataset		12h	24h		12h	24h
CLEAN-DS-15-SUMMER	713.861	2.276	4.472	500.245	2.470	4.862
CLEAN-DS-15-WINTER	628.725	2.246	4.737	445.597	2.428	5.022
CLEAN-DS-30-SUMMER	390.665	1.169	2.266	311.396	1.246	2.419
CLEAN-DS-30-WINTER	358.576	1.165	2.253	293.535	1.252	2.410
CLEAN-DS-60-SUMMER	275.897	0.604	1.151	248.403	0.629	1.194
CLEAN-DS-60-WINTER	264.678	0.596	1.113	241.099	0.637	1.193
DIRTY-DS-15-SUMMER	699.936	2.290	4.514	494.385	2.425	4.900
DIRTY-DS-15-WINTER	645.416	2.374	4.615	440.965	2.472	4.828
DIRTY-DS-30-SUMMER	388.948	1.164	2.243	311.467	1.248	2.406
DIRTY-DS-30-WINTER	354.699	1.159	2.250	294.019	1.226	2.388
DIRTY-DS-60-SUMMER	276.120	0.596	1.146	248.354	0.644	1.197
DIRTY-DS-60-WINTER	264.019	0.591	1.138	242.394	0.648	1.207
SMOOTH-DS-15-SUMMER	729.050	2.490	4.512	491.324	2.639	4.817
SMOOTH-DS-15-WINTER	633.765	2.321	4.570	452.423	2.403	4.809
SMOOTH-DS-30-SUMMER	392.934	1.155	2.594	308.026	1.240	2.685
SMOOTH-DS-30-WINTER	358.637	1.155	2.254	293.757	1.235	2.395
SMOOTH-DS-60-SUMMER	274.794	0.595	1.124	247.762	0.627	1.205
SMOOTH-DS-60-WINTER	263.945	0.600	1.152	242.576	0.642	1.222

Table 3: Execution time in seconds obtained by running the MLP model on CPU and GPU included in the HPC server platform. Execution time of training and inference (12 hours and 24 hours) are provided.

Platform	CPU			GPU		
Times	Training	Inference		Training	Inference	
Dataset		12h	24h		12h	24h
CLEAN-DS-15-SUMMER	33807.923	2.316	4.507	749.809	2.436	4.839
CLEAN-DS-15-WINTER	27835.258	2.348	4.509	660.638	2.463	4.822
CLEAN-DS-30-SUMMER	8460.801	1.192	2.518	404.968	1.236	2.716
CLEAN-DS-30-WINTER	7237.790	1.175	2.223	384.064	1.249	2.384
CLEAN-DS-60-SUMMER	2454.417	0.616	1.122	298.052	0.684	1.232
CLEAN-DS-60-WINTER	2119.112	0.580	1.121	286.328	0.658	1.210
DIRTY-DS-15-SUMMER	33613.773	2.309	4.503	755.894	2.434	4.778
DIRTY-DS-15-WINTER	28009.784	2.405	4.575	667.642	6.271	4.811
DIRTY-DS-30-SUMMER	8423.981	1.189	2.234	408.309	1.241	2.376
DIRTY-DS-30-WINTER	7239.152	1.192	2.246	399.793	6.606	2.426
DIRTY-DS-60-SUMMER	2463.910	0.628	1.145	299.358	0.658	1.221
DIRTY-DS-60-WINTER	2089.193	0.607	1.110	299.714	6.090	1.211
SMOOTH-DS-15-SUMMER	33877.128	2.323	4.477	741.246	2.475	4.776
SMOOTH-DS-15-WINTER	27866.877	2.318	4.943	666.987	2.503	5.166
SMOOTH-DS-30-SUMMER	8456.179	1.165	2.284	400.301	1.265	2.383
SMOOTH-DS-30-WINTER	7228.456	1.454	2.317	377.420	1.669	2.397
SMOOTH-DS-60-SUMMER	2456.905	0.571	1.140	289.699	0.661	1.203

Table 4: Execution time in seconds obtained by running the CNN model on CPU and GPU included in the HPC server platform. Execution time of training and inference (12 hours and 24 hours) are provided.

obtaining up to 10X performance in the best-case scenario. Inference times are almost identical on both CPU and GPU, being a few milliseconds slower on GPU, and ranging between 0.571 and 5.166 seconds.

Platform	CPU			GPU		
Times	Training	Inference		Training	Inference	
Dataset		12h	24h		12h	24h
CLEAN-DS-15-SUMMER	147642.328	3.290	5.605	3362.940	2.966	5.262
CLEAN-DS-15-WINTER	125425.139	3.363	5.973	2931.164	2.974	5.536
CLEAN-DS-30-SUMMER	37469.883	1.751	2.643	1136.169	1.634	2.467
CLEAN-DS-30-WINTER	31228.075	1.723	2.963	1001.663	1.633	2.887
CLEAN-DS-60-SUMMER	8983.771	1.003	1.159	481.747	1.013	1.219
CLEAN-DS-60-WINTER	7821.983	1.015	1.194	446.884	1.015	1.216
DIRTY-DS-15-SUMMER	147086.625	3.582	5.778	3419.389	3.284	5.173
DIRTY-DS-15-WINTER	125981.805	3.363	5.688	2988.742	3.048	5.275
DIRTY-DS-30-SUMMER	37478.132	1.706	2.496	1132.158	1.636	2.488
DIRTY-DS-30-WINTER	31363.002	1.721	2.576	998.420	1.639	2.460
DIRTY-DS-60-SUMMER	8994.831	1.025	1.581	479.596	1.018	1.749
DIRTY-DS-60-WINTER	7892.433	1.412	1.211	446.892	1.431	1.176
SMOOTH-DS-15-SUMMER	148153.696	3.290	5.654	3414.785	3.018	5.227
SMOOTH-DS-15-WINTER	126536.579	3.307	6.020	2991.940	3.291	5.263
SMOOTH-DS-30-SUMMER	37245.633	1.729	2.482	1122.942	1.630	2.467
SMOOTH-DS-30-WINTER	31630.357	1.705	2.617	1003.449	1.633	2.486
SMOOTH-DS-60-SUMMER	8994.010	1.014	1.189	482.858	1.007	1.227
SMOOTH-DS-60-WINTER	7841.688	1.020	1.195	448.608	1.029	1.217

Table 5: Execution time in seconds obtained by running the LSTM model on CPU and GPU included in the HPC server platform. Execution time of training and inference (12 hours and 24 hours) are provided.

Table 5 shows that the LSTM training time on CPU range from 7,821.983 to 148,153.696 seconds, while on GPU they range from 446.884 to 3,419.390 seconds, obtaining a 15X performance. Inference times are almost identical on both CPU and GPU, being a few milliseconds slower on CPU, and ranging between 1.003 and 6.020 seconds.

Finally, table 6 shows that the CNNLSTM training time on CPU range from 1,831.9153 to 31,471.342, while on GPU they range from 323.845 to 890.787, obtaining up to 20X performance. Inference times are almost identical on both CPU and GPU, being a few milliseconds slower on GPU, and ranging between 0.969 and 5.049.

Platform	CPU			GPU		
Times	Training	Inference		Training	Inference	
Dataset		12h	24h		12h	24h
CLEAN-DS-15-SUMMER	31455.494	2.759	4.599	871.329	2.872	4.850
CLEAN-DS-15-WINTER	26227.227	3.148	4.683	775.974	3.189	4.841
CLEAN-DS-30-SUMMER	7883.942	1.588	2.300	487.813	1.647	2.420
CLEAN-DS-30-WINTER	6752.098	1.589	2.707	447.680	1.656	2.877
CLEAN-DS-60-SUMMER	2222.535	1.000	1.616	350.630	1.048	1.777
CLEAN-DS-60-WINTER	1920.816	1.009	1.125	334.212	1.039	1.212
DIRTY-DS-15-SUMMER	31329.182	2.778	4.611	868.439	2.871	4.855
DIRTY-DS-15-WINTER	26049.698	2.797	4.798	890.787	6.681	5.049
DIRTY-DS-30-SUMMER	7960.443	1.555	2.249	484.462	1.635	2.418
DIRTY-DS-30-WINTER	6788.469	1.606	2.245	467.469	6.823	2.415
DIRTY-DS-60-SUMMER	2221.137	0.969	1.060	351.492	1.041	1.203
DIRTY-DS-60-WINTER	1929.761	0.973	1.133	346.623	6.078	1.209
SMOOTH-DS-15-SUMMER	31471.342	2.731	4.541	852.136	2.853	4.848
SMOOTH-DS-15-WINTER	26336.821	2.713	4.564	767.090	2.843	4.803
SMOOTH-DS-30-SUMMER	7924.806	1.596	2.251	476.830	1.656	2.404
SMOOTH-DS-30-WINTER	6765.339	1.584	2.259	441.828	1.651	2.420
SMOOTH-DS-60-SUMMER	2222.474	1.011	1.147	343.871	1.030	1.199
SMOOTH-DS-60-WINTER	1831.915	0.999	1.142	323.847	1.042	1.202

Table 6: Execution time in seconds obtained by running the CNNLSTM model on CPU and GPU included in the HPC server platform. Execution time of training and inference (12 hours and 24 hours) are provided.

In conclusion for the 4 models run on the server, we have that the GPU execution for training is much faster between 1.5 and 20x at best. The speed difference between CPU and GPU increases when the model is more complex, the complexity hierarchy being MLP, CNN, LSTM, and CNNLSTM models. In the case of inference, the same situation occurs for all four models, i.e., execution on the CPU is slightly faster than on the GPU. This has a logical explanation since in the case of inference, no large computational power is needed.

4.1.2. Edge evaluation

The edge computing platform is evaluated by only running the inference for the different ML/DL models described in Section 3.4. In *SEPARATE*, only ML/DL inference is executed at this level of the infrastructure, as training is computationally forbidden. It is worth highlighting that the edge computing node is an Nvidia Jetson Nano that has a 5-core ARM Cortex-A57 MPCore CPU, 2MB L2, 128-core Maxwell GPU, and 4GB 64-Bit LPDDR4 running at 25.6 GB/sec. Therefore, the CPU and GPU runs are analyzed to find out whether the inclusion of GPUs brings better performance results to inference in this scenario. Moreover, *SEPARATE* will send the ML/DL model into the edge device to proceed with the inference with the more updated model. Therefore, the execution time of loading an ML/DL model in the edge device once it is received is also reported below.

Tables 7, 8, 9, 10 show the MLP, CNN, LSTM and CNNLSTM execution time at the edge of both, the Load and Inference stages of the *SEPARATE* pipeline, respectively. Each row of the table shows the performance times for each of the datasets described in Section 3.3. Each column shows the execution times for these stages, for each of the two processors studied; i.e. CPU and GPU.

In particular, table 7 shows that the MLP load time is almost identical on both CPU and GPU, being a few seconds slower on GPU, and ranging between 0.151 and 5.731 seconds. Inference time reported is also almost identical on both CPU and GPU, being a few seconds slower on GPU, and ranging between 2.351 and 20.356 seconds.

Table 8 shows that the CNN load time is almost identical on both CPU and GPU, being a few seconds slower on GPU, and ranging between 0.273 and 17.968 seconds. Inference time is almost identical on both CPU and GPU, being a few seconds slower on GPU, and ranging between 2.320 and 21.107 seconds.

Table 9 shows that the LSTM load time is almost identical on both CPU and GPU, being a few seconds slower on GPU, and ranging between 1.457 and 8.081 seconds. Inference time is almost identical on both CPU and GPU, being a few seconds slower on GPU, and ranging between 3.793 and 25.917 seconds.

Platform	CPU			GPU		
Times	Load	Inference		Load	Inference	
Dataset		12h	24h		12h	24h
CLEAN-DS-15-SUMMER	0.154	9.364	19.212	0.199	9.912	20.136
CLEAN-DS-15-WINTER	0.155	9.192	18.792	0.275	10.069	20.088
CLEAN-DS-30-SUMMER	0.175	5.343	8.938	0.204	5.549	9.851
CLEAN-DS-30-WINTER	0.161	4.851	9.566	0.203	5.208	10.657
CLEAN-DS-60-SUMMER	0.192	2.817	4.787	0.251	3.124	4.839
CLEAN-DS-60-WINTER	0.167	2.326	4.525	0.270	2.684	4.902
DIRTY-DS-15-SUMMER	0.158	9.546	18.579	0.229	9.904	20.016
DIRTY-DS-15-WINTER	0.346	9.774	18.604	5.731	12.104	19.646
DIRTY-DS-30-SUMMER	0.151	4.655	9.667	0.234	4.383	10.310
DIRTY-DS-30-WINTER	0.153	4.469	8.719	0.244	5.389	9.983
DIRTY-DS-60-SUMMER	0.171	2.351	4.687	0.244	2.570	4.972
DIRTY-DS-60-WINTER	0.158	2.752	4.458	0.295	3.159	4.573
SMOOTH-DS-15-SUMMER	0.154	8.975	18.901	0.235	10.259	19.969
SMOOTH-DS-15-WINTER	0.156	8.774	18.566	0.211	9.967	20.356
SMOOTH-DS-30-SUMMER	0.153	4.478	9.508	0.242	5.056	10.472
SMOOTH-DS-30-WINTER	0.154	4.924	9.586	0.490	5.513	9.406
SMOOTH-DS-60-SUMMER	0.180	2.388	5.244	0.238	2.604	5.339
SMOOTH-DS-60-WINTER	0.165	2.357	4.788	0.360	2.757	4.674

Table 7: Execution time in seconds obtained by running the MLP model on CPU and GPU included in the edge computing platform. Times for a load of the ML model (Load) and execute the inference for the next 12 and 24 hours (Inference) are provided.

Platform	CPU			GPU		
Times	Load	Inference		Load	Inference	
Dataset		12h	24h		12h	24h
CLEAN-DS-15-SUMMER	0.325	8.800	18.192	0.478	10.054	20.270
CLEAN-DS-15-WINTER	0.342	9.286	17.946	0.755	11.147	21.107
CLEAN-DS-30-SUMMER	0.273	4.806	9.560	0.405	5.257	10.004
CLEAN-DS-30-WINTER	0.278	4.509	8.893	0.417	5.002	9.567
CLEAN-DS-60-SUMMER	0.315	2.490	4.335	0.477	2.765	4.895
CLEAN-DS-60-WINTER	0.360	2.320	4.559	0.605	2.762	4.682
DIRTY-DS-15-SUMMER	0.403	9.511	18.579	0.569	10.436	20.527
DIRTY-DS-15-WINTER	0.739	9.745	18.280	9.830	28.747	20.656
DIRTY-DS-30-SUMMER	0.291	4.667	8.889	0.449	5.021	9.796
DIRTY-DS-30-WINTER	0.298	4.996	8.801	0.403	6.086	10.305
DIRTY-DS-60-SUMMER	0.269	2.843	4.683	0.574	3.293	4.727
DIRTY-DS-60-WINTER	0.364	2.555	4.509	0.656	3.234	4.935
SMOOTH-DS-15-SUMMER	0.325	9.843	17.758	17.968	10.806	20.851
SMOOTH-DS-15-WINTER	0.329	9.231	18.090	0.501	10.118	20.274
SMOOTH-DS-30-SUMMER	0.280	4.710	8.680	0.415	4.844	9.917
SMOOTH-DS-30-WINTER	0.320	5.105	8.919	0.400	5.556	9.913
SMOOTH-DS-60-SUMMER	0.367	2.439	4.605	0.424	2.740	4.653
SMOOTH-DS-60-WINTER	0.328	2.397	4.514	0.459	2.710	4.869

Table 8: Execution time in seconds obtained by running the CNN model on CPU and GPU included in the edge computing platform. Times for load of the DL model (Load) and execute the inference for the next 12 and 24 hours (Inference) are provided.

Platform	CPU			GPU		
Times	Load time	Forecast time		Load time	Forecast time	
Dataset		12h	24h		12h	24h
CLEAN-DS-15-SUMMER	2.053	13.656	25.046	2.073	12.615	22.895
CLEAN-DS-15-WINTER	1.480	14.284	25.917	1.633	13.007	22.943
CLEAN-DS-30-SUMMER	2.294	6.581	10.545	2.283	7.006	11.355
CLEAN-DS-30-WINTER	1.504	6.802	11.056	1.569	7.157	10.739
CLEAN-DS-60-SUMMER	1.618	4.872	4.599	1.570	4.966	5.222
CLEAN-DS-60-WINTER	2.454	3.793	4.880	1.613	5.176	5.203
DIRTY-DS-15-SUMMER	1.502	14.437	24.876	1.534	13.426	21.957
DIRTY-DS-15-WINTER	2.070	14.355	24.823	8.081	25.461	22.661
DIRTY-DS-30-SUMMER	1.506	7.482	10.003	1.573	8.090	11.177
DIRTY-DS-30-WINTER	2.077	7.024	10.135	1.515	7.820	10.758
DIRTY-DS-60-SUMMER	1.477	4.200	4.978	1.540	4.272	5.305
DIRTY-DS-60-WINTER	1.457	3.919	4.540	1.566	4.518	5.583
SMOOTH-DS-15-SUMMER	1.559	13.906	24.823	1.550	13.544	21.932
SMOOTH-DS-15-WINTER	1.555	14.355	24.377	1.637	13.624	22.311
SMOOTH-DS-30-SUMMER	1.507	6.663	10.268	1.558	7.152	10.994
SMOOTH-DS-30-WINTER	1.548	7.959	10.392	1.550	7.270	12.191
SMOOTH-DS-60-SUMMER	1.534	4.005	5.018	1.528	4.376	5.483
SMOOTH-DS-60-WINTER	1.512	3.995	5.954	1.549	4.337	5.452

Table 9: Execution time in seconds obtained by running the LSTM model on CPU and GPU included in the edge computing platform. Times for load of the DL model (Load) and execute the inference for the next 12 and 24 hours (Inference) are provided.

Platform	CPU			GPU		
Times	Load time	Forecast time		Load time	Forecast time	
Dataset		12h	24h		12h	24h
CLEAN-DS-15-SUMMER	1.751	11.511	19.149	2.138	13.424	22.707
CLEAN-DS-15-WINTER	1.914	12.009	19.610	3.977	13.725	22.218
CLEAN-DS-30-SUMMER	1.669	6.858	9.166	1.964	7.403	11.114
CLEAN-DS-30-WINTER	1.648	6.579	9.989	1.895	7.105	11.608
CLEAN-DS-60-SUMMER	1.816	4.299	4.529	1.829	4.519	5.147
CLEAN-DS-60-WINTER	1.648	4.243	4.678	1.839	4.459	5.443
DIRTY-DS-15-SUMMER	1.912	12.158	19.096	2.009	13.861	22.447
DIRTY-DS-15-WINTER	2.661	11.648	19.100	7.049	29.010	22.319
DIRTY-DS-30-SUMMER	2.500	6.701	9.464	2.621	7.314	10.942
DIRTY-DS-30-WINTER	1.828	6.606	9.504	1.849	7.761	10.723
DIRTY-DS-60-SUMMER	1.783	3.828	3.936	1.859	4.566	5.200
DIRTY-DS-60-WINTER	1.706	4.142	4.628	1.901	5.065	5.188
SMOOTH-DS-15-SUMMER	1.732	11.807	19.284	1.970	14.001	22.177
SMOOTH-DS-15-WINTER	1.722	12.143	19.633	2.066	13.849	22.789
SMOOTH-DS-30-SUMMER	1.690	6.553	10.389	1.922	7.433	11.659
SMOOTH-DS-30-WINTER	1.781	7.347	9.390	1.889	8.221	11.167
SMOOTH-DS-60-SUMMER	1.665	5.113	4.631	1.835	5.823	5.372
SMOOTH-DS-60-WINTER	1.720	5.250	4.600	1.877	5.509	5.426

Table 10: Execution time in seconds obtained by running the CNNLSTM model on CPU and GPU included in the edge computing platform. Times for a load of the DL model (Load) and execute the inference for the next 12 and 24 hours (Inference) are provided.

Finally, table 10 shows that the CNNLSTM load time is almost identical on both CPU and GPU, being a few seconds slower on GPU, and ranging between 1.648 and 7.049 seconds. Inference time is almost identical on both CPU and GPU, being a few seconds slower on GPU, and ranging between 3.828 and 22.789 seconds.

In conclusion, execution on the edge is slightly faster on the CPU than on the GPU for all 4 models tested. This result is consistent with the results shown above for the execution on the server. As inference is an inexpensive process, the computational power offered by the GPU is not necessary. Evidently, inference on the Edge is somewhat slower than on the server, but with completely acceptable times.

4.2. ML/DL models quality assessment

This section shows the results obtained when all ML/DL techniques are used to forecast the internal temperature of the greenhouse. Several metrics are provided to analyze the goodness of fit of the models. They are the Root Mean Squared Error (RMSE) and the Mean Absolute Error (MAE) which shows the gap between the forecast and actual values. MAE and RMSE are calculated by averaging the absolute difference between the predicted and actual values. Moreover, we also provide the R^2 that shows the variance of the forecast variable that is predictable from the actual variable. These metrics are shown for all ML/DL models, temporal granularity, and different evaluation periods (i.e. SUMMER and WINTER). It is important to note that these values were obtained using the Python-based Scikit-Learn library.

Table 11 shows the main quality figures achieved by the MLP model. This technique obtains the best result in MAE and RMSE with the CLEAN-DS-15-WINTER dataset for both 12-hour and 24-hour forecasts. The lowest MAE and RMSE is obtained with 12-hour prediction with 1.276°C and 1.538°C respectively. The difference with the 24-hour prediction is very small for all the datasets evaluated in general.

Table 12 reports the quality figures for the CNN technique. As with MLP, the results for the 12-hour and 24-hour forecasts are quite similar. However, the best result is obtained with the DIRTY-DS-30-WINTER dataset with a MAE and RMSE of 1.252 ºC and 1.492ºC. Contrary to the MLP technique, although with really little difference, the best result is obtained with a dataset without preprocessing. This makes sense as deep learning techniques are better able to remove possible noise in the data.

Forecasting period	12h			24h		
Datasets	R^2_{sd}	$RMSE_{sd}$	MAE_{sd}	R^2_{sd}	$RMSE_{sd}$	MAE_{sd}
CLEAN-DS-15-SUMMER	$0.588_{0.120}$	$4.044_{0.287}$	$3.710_{0.267}$	$0.806_{0.053}$	$3.366_{0.191}$	$2.932_{0.201}$
CLEAN-DS-15-WINTER	$0.838_{0.038}$	$1.538_{0.081}$	$1.276_{0.093}$	$0.886_{0.018}$	$1.643_{0.088}$	$1.279_{0.079}$
CLEAN-DS-30-SUMMER	$0.626_{0.119}$	$3.768_{0.222}$	$3.434_{0.260}$	$0.821_{0.032}$	$3.242_{0.232}$	$2.809_{0.244}$
CLEAN-DS-30-WINTER	$0.853_{0.043}$	$1.634_{0.141}$	$1.403_{0.135}$	$0.891_{0.027}$	$1.706_{0.082}$	$1.348_{0.062}$
CLEAN-DS-60-SUMMER	$0.835_{0.100}$	$3.242_{0.286}$	$2.927_{0.289}$	$0.878_{0.043}$	$3.327_{0.133}$	$2.962_{0.151}$
CLEAN-DS-60-WINTER	$0.830_{0.024}$	$1.708_{0.064}$	$1.506_{0.066}$	$0.899_{0.008}$	$1.654_{0.027}$	$1.327_{0.028}$
DIRTY-DS-15-SUMMER	$0.565_{0.147}$	$3.906_{0.413}$	$3.528_{0.405}$	$0.774_{0.093}$	$3.301_{0.291}$	$2.832_{0.270}$
DIRTY-DS-15-WINTER	$0.816_{0.037}$	$1.519_{0.117}$	$1.278_{0.127}$	$0.876_{0.012}$	$1.578_{0.104}$	$1.262_{0.094}$
DIRTY-DS-30-SUMMER	$0.613_{0.081}$	$3.858_{0.255}$	$3.529_{0.264}$	$0.821_{0.034}$	$3.147_{0.132}$	$2.701_{0.113}$
DIRTY-DS-30-WINTER	$0.787_{0.052}$	$1.728_{0.133}$	$1.488_{0.117}$	$0.870_{0.029}$	$1.677_{0.102}$	$1.357_{0.081}$
DIRTY-DS-60-SUMMER	$0.819_{0.075}$	$3.136_{0.333}$	$2.833_{0.354}$	$0.877_{0.041}$	$3.154_{0.355}$	$2.832_{0.343}$
DIRTY-DS-60-WINTER	$0.828_{0.018}$	$1.737_{0.073}$	$1.533_{0.069}$	$0.900_{0.008}$	$1.635_{0.031}$	$1.322_{0.038}$
SMOOTH-DS-15-SUMMER	$0.685_{0.149}$	$3.781_{0.405}$	$3.494_{0.374}$	$0.849_{0.056}$	$3.387_{0.196}$	$2.995_{0.217}$
SMOOTH-DS-15-WINTER	$0.790_{0.065}$	$1.599_{0.209}$	$1.297_{0.232}$	$0.867_{0.031}$	$1.741_{0.125}$	$1.338_{0.137}$
SMOOTH-DS-30-SUMMER	$0.705_{0.125}$	$3.775_{0.358}$	$3.451_{0.372}$	$0.855_{0.039}$	$3.395_{0.165}$	$3.000_{0.139}$
SMOOTH-DS-30-WINTER	$0.784_{0.043}$	$1.763_{0.182}$	$1.464_{0.197}$	$0.875_{0.021}$	$1.877_{0.100}$	$1.456_{0.087}$
SMOOTH-DS-60-SUMMER	$0.748_{0.077}$	$3.457_{0.160}$	$3.171_{0.143}$	$0.895_{0.027}$	$3.662_{0.227}$	$3.261_{0.214}$
SMOOTH-DS-60-WINTER	$0.722_{0.040}$	$2.012_{0.129}$	$1.733_{0.158}$	$0.854_{0.020}$	$2.127_{0.065}$	$1.718_{0.071}$

Table 11: Results of the MLP technique, values in sub-index indicate the standard deviation obtained after the repetition of each experiment. R^2 (coefficient of determination) RMSE (root mean square error) MAE (mean absolute error). RMSE and MAE are measured in degrees Celsius (°C).

Forecasting period	12h			24h		
Datasets	R^2_{sd}	$RMSE_{sd}$	MAE_{sd}	R^2_{sd}	$RMSE_{sd}$	MAE_{sd}
CLEAN-DS-15-SUMMER	$0.739_{0.189}$	$3.605_{0.286}$	$3.279_{0.208}$	$0.867_{0.044}$	$3.086_{0.137}$	$2.682_{0.158}$
CLEAN-DS-15-WINTER	$0.789_{0.224}$	$1.782_{0.831}$	$1.545_{0.855}$	$0.808_{0.283}$	$1.787_{0.753}$	$1.463_{0.753}$
CLEAN-DS-30-SUMMER	$0.815_{0.174}$	$3.499_{0.255}$	$3.234_{0.194}$	$0.897_{0.049}$	$3.267_{0.052}$	$2.906_{0.100}$
CLEAN-DS-30-WINTER	$0.778_{0.022}$	$1.549_{0.126}$	$1.259_{0.095}$	$0.876_{0.010}$	$1.613_{0.041}$	$1.257_{0.038}$
CLEAN-DS-60-SUMMER	$0.841_{0.022}$	$3.475_{0.070}$	$3.200_{0.096}$	$0.859_{0.008}$	$3.360_{0.056}$	$3.020_{0.059}$
CLEAN-DS-60-WINTER	$0.829_{0.013}$	$1.723_{0.022}$	$1.503_{0.025}$	$0.899_{0.004}$	$1.627_{0.012}$	$1.311_{0.014}$
DIRTY-DS-15-SUMMER	$0.799_{0.102}$	$3.347_{0.106}$	$3.014_{0.153}$	$0.862_{0.025}$	$3.034_{0.174}$	$2.642_{0.162}$
DIRTY-DS-15-WINTER	$0.795_{0.208}$	$1.834_{0.782}$	$1.597_{0.805}$	$0.808_{0.282}$	$1.859_{0.721}$	$1.510_{0.727}$
DIRTY-DS-30-SUMMER	$0.870_{0.074}$	$3.431_{0.187}$	$3.197_{0.165}$	$0.898_{0.040}$	$3.160_{0.073}$	$2.822_{0.049}$
DIRTY-DS-30-WINTER	$0.814_{0.025}$	$1.492_{0.201}$	$1.252_{0.210}$	$0.886_{0.008}$	$1.539_{0.102}$	$1.222_{0.098}$
DIRTY-DS-60-SUMMER	$0.789_{0.029}$	$3.553_{0.075}$	$3.299_{0.088}$	$0.853_{0.008}$	$3.249_{0.046}$	$2.890_{0.056}$
DIRTY-DS-60-WINTER	$0.831_{0.013}$	$1.666_{0.027}$	$1.442_{0.034}$	$0.898_{0.005}$	$1.600_{0.020}$	$1.281_{0.016}$
SMOOTH-DS-15-SUMMER	$0.745_{0.160}$	$3.535_{0.246}$	$3.161_{0.269}$	$0.871_{0.036}$	$3.026_{0.166}$	$2.598_{0.214}$
SMOOTH-DS-15-WINTER	$0.681_{0.310}$	$2.058_{1.008}$	$1.818_{1.044}$	$0.708_{0.373}$	$2.064_{0.955}$	$1.730_{0.958}$
SMOOTH-DS-30-SUMMER	$0.754_{0.156}$	$3.530_{0.182}$	$3.239_{0.161}$	$0.893_{0.053}$	$3.338_{0.095}$	$2.980_{0.122}$
SMOOTH-DS-30-WINTER	$0.756_{0.012}$	$1.745_{0.095}$	$1.443_{0.104}$	$0.863_{0.004}$	$1.793_{0.027}$	$1.401_{0.027}$
SMOOTH-DS-60-SUMMER	$0.846_{0.018}$	$3.482_{0.120}$	$3.249_{0.135}$	$0.933_{0.004}$	$3.617_{0.080}$	$3.249_{0.062}$
SMOOTH-DS-60-WINTER	$0.696_{0.012}$	$2.039_{0.046}$	$1.752_{0.050}$	$0.840_{0.006}$	$2.157_{0.019}$	$1.745_{0.017}$

Table 12: Results of the CNN technique, values in sub-index indicate the standard deviation obtained after the repetition of each experiment. R^2 (coefficient of determination) RMSE (root mean square error) MAE (mean absolute error). RMSE and MAE are measured in degrees Celsius (°C).

Prediction hours	12h			24h		
Datasets	R^2_{sd}	$RMSE_{sd}$	MAE_{sd}	R^2_{sd}	$RMSE_{sd}$	MAE_{sd}
CLEAN-DS-15-SUMMER	$0.605_{0.260}$	$3.602_{0.550}$	$2.501_{0.242}$	$0.521_{0.324}$	$5.819_{1.225}$	$4.773_{0.925}$
CLEAN-DS-15-WINTER	$0.257_{0.057}$	$2.785_{0.043}$	$2.073_{0.056}$	$0.323_{0.058}$	$3.985_{0.167}$	$2.967_{0.064}$
CLEAN-DS-30-SUMMER	$0.836_{0.052}$	$2.658_{0.209}$	$2.209_{0.231}$	$0.912_{0.035}$	$2.813_{0.268}$	$2.451_{0.229}$
CLEAN-DS-30-WINTER	$0.779_{0.041}$	$1.724_{0.196}$	$1.400_{0.185}$	$0.875_{0.029}$	$1.719_{0.124}$	$1.367_{0.107}$
CLEAN-DS-60-SUMMER	$0.827_{0.026}$	$2.756_{0.203}$	$2.401_{0.270}$	$0.930_{0.012}$	$3.000_{0.192}$	$2.646_{0.173}$
CLEAN-DS-60-WINTER	$0.796_{0.022}$	$1.655_{0.154}$	$1.389_{0.153}$	$0.893_{0.010}$	$1.609_{0.088}$	$1.258_{0.079}$
DIRTY-DS-15-SUMMER	$0.481_{0.234}$	$4.324_{1.610}$	$3.151_{1.026}$	$0.530_{0.234}$	$6.649_{2.812}$	$5.518_{2.373}$
DIRTY-DS-15-WINTER	$0.288_{0.109}$	$3.088_{0.256}$	$2.611_{0.353}$	$0.318_{0.136}$	$3.516_{0.408}$	$2.905_{0.191}$
DIRTY-DS-30-SUMMER	$0.805_{0.051}$	$2.688_{0.199}$	$2.275_{0.214}$	$0.909_{0.024}$	$2.803_{0.216}$	$2.460_{0.189}$
DIRTY-DS-30-WINTER	$0.751_{0.077}$	$1.885_{0.375}$	$1.540_{0.340}$	$0.857_{0.054}$	$1.735_{0.248}$	$1.366_{0.217}$
DIRTY-DS-60-SUMMER	$0.802_{0.030}$	$3.000_{0.199}$	$2.691_{0.175}$	$0.927_{0.009}$	$2.808_{0.107}$	$2.466_{0.136}$
DIRTY-DS-60-WINTER	$0.785_{0.057}$	$1.554_{0.209}$	$1.294_{0.171}$	$0.881_{0.041}$	$1.656_{0.102}$	$1.317_{0.096}$
SMOOTH-DS-15-SUMMER	$0.550_{0.269}$	$4.771_{2.427}$	$3.614_{1.750}$	$0.562_{0.171}$	$6.450_{3.994}$	$5.426_{3.485}$
SMOOTH-DS-15-WINTER	$0.401_{0.116}$	$2.936_{0.544}$	$2.555_{0.547}$	$0.385_{0.026}$	$3.236_{0.156}$	$2.788_{0.203}$
SMOOTH-DS-30-SUMMER	$0.711_{0.062}$	$3.215_{0.295}$	$2.689_{0.219}$	$0.803_{0.107}$	$3.968_{0.947}$	$3.348_{0.744}$
SMOOTH-DS-30-WINTER	$0.648_{0.166}$	$2.124_{0.277}$	$1.744_{0.198}$	$0.783_{0.175}$	$2.115_{0.592}$	$1.669_{0.417}$
SMOOTH-DS-60-SUMMER	$0.833_{0.019}$	$2.982_{0.084}$	$2.663_{0.107}$	$0.934_{0.008}$	$3.419_{0.246}$	$3.029_{0.224}$
SMOOTH-DS-60-WINTER	$0.665_{0.027}$	$2.102_{0.115}$	$1.779_{0.111}$	$0.823_{0.016}$	$2.153_{0.032}$	$1.731_{0.039}$

Table 13: Results of the LSTM technique, values in sub-index indicate the standard deviation obtained after the repetition of each experiment. R^2 (coefficient of determination) RMSE (root mean square error) MAE (mean absolute error). RMSE and MAE are measured in degrees Celsius (°C).

Table 13 shows the results of the LSTM technique. This technique obtains a similar behavior to the previous deep learning techniques. In the overall tuning, there is no difference in the prediction at 12 or 24 hours. Moreover, although the results are similar, this technique obtains slightly higher RMSE and MAE values than the other deep learning techniques discussed. The best results are obtained with the DIRTY-DS-60-WINTER dataset, where its MAE and RMSE are 1.294°C and 1.554°C in 12-hour prediction respectively.

Prediction hours	12h			24h		
Datasets	R^2_{sd}	$RMSE_{sd}$	MAE_{sd}	R^2_{sd}	$RMSE_{sd}$	MAE_{sd}
CLEAN-DS-15-SUMMER	$0.786_{0.088}$	$3.344_{0.179}$	$3.014_{0.203}$	$0.876_{0.023}$	$2.947_{0.321}$	$2.553_{0.368}$
CLEAN-DS-15-WINTER	$0.875_{0.023}$	$1.527_{0.185}$	$1.303_{0.193}$	$0.923_{0.008}$	$1.479_{0.150}$	$1.193_{0.110}$
CLEAN-DS-30-SUMMER	$0.794_{0.106}$	$3.483_{0.251}$	$3.210_{0.225}$	$0.860_{0.051}$	$3.223_{0.067}$	$2.843_{0.079}$
CLEAN-DS-30-WINTER	$0.874_{0.017}$	$1.599_{0.087}$	$1.401_{0.091}$	$0.921_{0.007}$	$1.595_{0.035}$	$1.280_{0.032}$
CLEAN-DS-60-SUMMER	$0.935_{0.005}$	$3.102_{0.039}$	$2.867_{0.058}$	$0.939_{0.010}$	$3.291_{0.063}$	$2.956_{0.051}$
CLEAN-DS-60-WINTER	$0.876_{0.003}$	$1.441_{0.023}$	$1.281_{0.022}$	$0.930_{0.001}$	$1.504_{0.040}$	$1.220_{0.018}$
DIRTY-DS-15-SUMMER	$0.827_{0.045}$	$3.381_{0.222}$	$3.092_{0.244}$	$0.863_{0.024}$	$2.975_{0.194}$	$2.595_{0.196}$
DIRTY-DS-15-WINTER	$0.881_{0.025}$	$1.527_{0.111}$	$1.295_{0.124}$	$0.927_{0.009}$	$1.594_{0.120}$	$1.258_{0.102}$
DIRTY-DS-30-SUMMER	$0.868_{0.097}$	$3.359_{0.172}$	$2.887_{0.298}$	$0.857_{0.029}$	$3.575_{0.364}$	$3.142_{0.296}$
DIRTY-DS-30-WINTER	$0.877_{0.009}$	$1.575_{0.052}$	$1.370_{0.051}$	$0.926_{0.003}$	$1.593_{0.057}$	$1.267_{0.040}$
DIRTY-DS-60-SUMMER	$0.937_{0.004}$	$3.093_{0.038}$	$2.866_{0.061}$	$0.942_{0.004}$	$3.278_{0.058}$	$2.952_{0.053}$
DIRTY-DS-60-WINTER	$0.882_{0.006}$	$1.357_{0.043}$	$1.200_{0.048}$	$0.935_{0.003}$	$1.404_{0.018}$	$1.142_{0.022}$
SMOOTH-DS-15-SUMMER	$0.714_{0.079}$	$3.358_{0.315}$	$2.994_{0.366}$	$0.844_{0.104}$	$2.904_{0.346}$	$2.457_{0.335}$
SMOOTH-DS-15-WINTER	$0.848_{0.038}$	$1.630_{0.199}$	$1.407_{0.195}$	$0.904_{0.016}$	$1.596_{0.134}$	$1.283_{0.107}$
SMOOTH-DS-30-SUMMER	$0.776_{0.143}$	$3.528_{0.271}$	$3.253_{0.242}$	$0.893_{0.051}$	$3.281_{0.062}$	$2.903_{0.072}$
SMOOTH-DS-30-WINTER	$0.787_{0.040}$	$1.787_{0.093}$	$1.506_{0.119}$	$0.888_{0.016}$	$1.832_{0.062}$	$1.435_{0.032}$
SMOOTH-DS-60-SUMMER	$0.906_{0.012}$	$3.369_{0.050}$	$3.133_{0.119}$	$0.947_{0.008}$	$3.763_{0.094}$	$3.375_{0.067}$
SMOOTH-DS-60-WINTER	$0.715_{0.015}$	$2.010_{0.051}$	$1.732_{0.055}$	$0.854_{0.008}$	$2.110_{0.060}$	$1.724_{0.033}$

Table 14: Results of the CNNLSTM technique, values in sub-index indicate the standard deviation obtained after the repetition of each experiment. R^2 (coefficient of determination) RMSE (root mean square error) MAE (mean absolute error). RMSE and MAE are measured in degrees Celsius (°C).

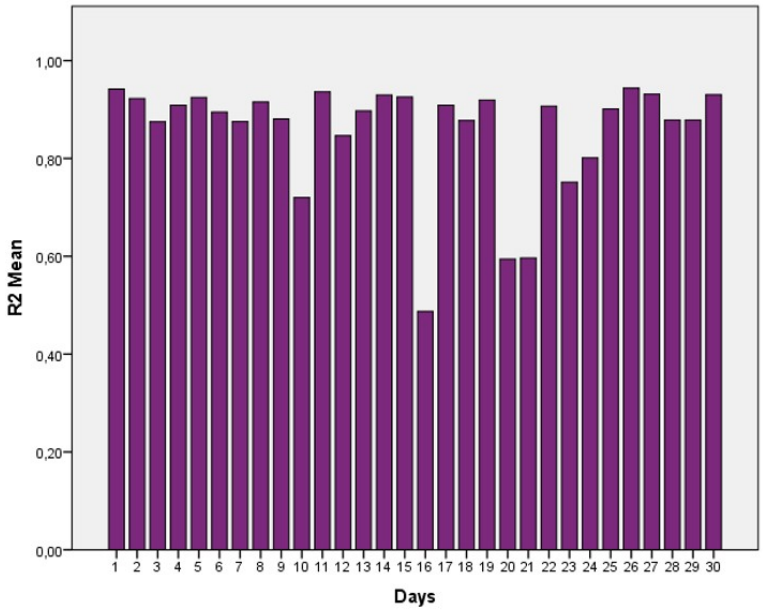
Finally, table 14 shows the quality figures of the CNNLSTM technique, which reports very similar results to the CNN technique. It follows the trend of obtaining very similar results for 12-hour and 24-hour forecasts. Moreover, the best result is obtained with the DIRTY-DS-60-WINTER dataset. In this case, although also with little difference, the best result is obtained with the data without preprocessing. Specifically, the MAE and RMSE results are 1.200 ºC and 1.357ºC respectively.

4.3. Model retraining evaluation

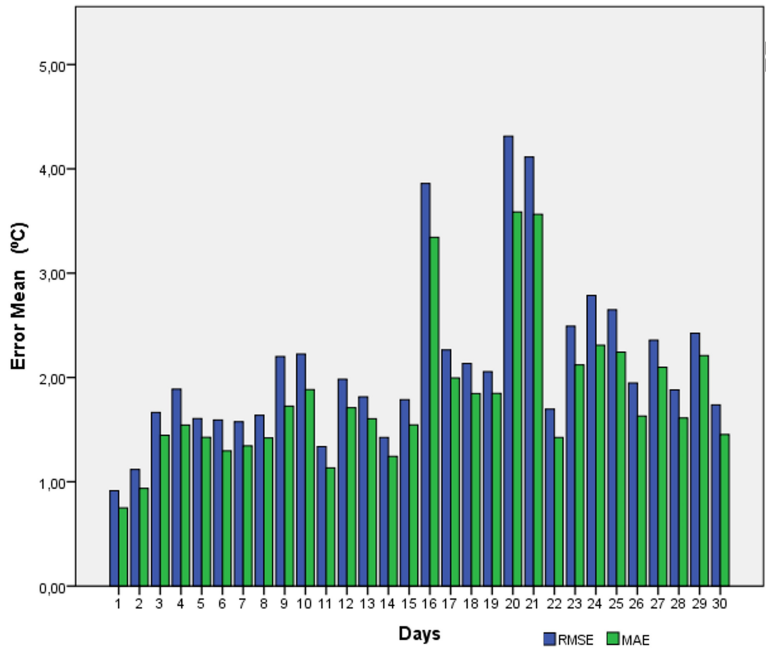
ML models require periodic retraining to improve the quality of predictions as new data is generated. However, these retrainings are quite computationally

expensive, and that is why SEPARATE allows asynchronous communication with an HPC server that retrains the model and, once retrained, can be updated at the edge. The question here was how often it is necessary to perform these trainings. To analyze this point, the following tests have been carried out. First, the model has been trained using all the data from each dataset under study, except for the last 30 days. Subsequently, the model was tested with day 1, then day 2, then day 3, and so on until day 30. Independent tests were performed on each day, in order to visualize and analyze the impact of error and model fit as the days passed. The results of these tests indicate the maximum time that an ML/DL time series model can predict without retraining or loss of prediction quality. Although all techniques have similar performance, the one with the lowest error is the CNNLTSM, so this test has been performed with this technique.

Figures 4a and 4b show R^2 and the mean of RMSE and MAE metrics for all targeted datasets. Particularly, Figure 4a reports significant sharp differences testing 10, 16, 20, and 21 days respectively using R^2 metric. Figure 4b focuses on RMSE and MAE metrics and it can be seen that from days 9 and 10 onwards, there is a significant upward trend in the error. The same occurs on days 16, 20, and 21, the latter two being the days with the greatest error with respect to the rest of the days.



(a) Mean of all datasets in table 1 for the metric R^2. The Y-axis shows the correlation between the real and the forecasted values expressed in a range between 0 and 1.



(b) Mean of all datasets in table 1 for the metric RMSE y MAE. The Y-axis shows the mean error expressed in degrees Celsius.

Figure 4: Mean of all datasets in table 1 for all the metrics

After showing the test results evaluating 30 days separately and analyzing that days 9 and 10, 16, 20, and 21 seem to be days with significant key differences, we now apply non-parametric statistical tests to statistically contract from which day the CNNLSTM model should be retrained due to the significant increase in error and the drop in fit. We perform a non-parametric statistical test, in particular, the Kruskal-Wallis test [29], since the data do not follow a normal distribution. To make this statement, we proceeded to perform a Kolmogorov–Smirnov test [30] for normality of the data, obtaining a p-value of 0.000, which indicates that the RMSE, MAE, and R^2 data do not follow a normal distribution. Kruskal-Wallis tests have been also performed using all datasets, without distinguishing by data pre-processing used, the season of the year, as well as granularity. For a greater amount of information, tests have been performed individually for each of the metrics evaluated in the accuracy, which have been MSE, RMSE, and R^2. The statistical results, with a 95% confidence level, after performing the Kruskal-Wallis test, show the adjusted p-values indicated in tables 15, 16 and 17.

The table 15 shows for the first 10 days, the p-values with values higher than 0.05; i.e., confidence level of 95%. Thus, it is important to note that the first two days have significant differences with a 95% confidence level, with days 9, 10, 12, 16, 17, 18, 19, 20, and 21. Days 3 to 10, however, have significant differences from the models for days 20 and 21. These results lead us to conclude that taking into account the MAE metric, from day 9 onwards, the model starts to lose accuracy, however, when the accuracy drops drastically is from day 19 onwards.

Days	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
9	0.002	0.054								
10	0.002	0.051								
12	0.011									
16	0.000	0.000								
17	0.000	0.000								
18	0.000	0.007								
19	0.000	0.011								
20	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.080	0.007
21	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.012	0.011

Table 15: P-values of the statistical analysis for all datasets, considering the MAE metric of each of the tested days. Only p-values $<$ 0.06 are shown.

Table 16 shows the p-values, with a confidence level of 95%, for the first 10 days for the RMSE metric. As can be seen, the results are very similar to the MAE metric, with the only exception that day 9 has no significant differences

with days 20 and 21. Despite this difference, the conclusion is the same as that obtained for the MAE metric, from day 9 onwards, accuracy decreases but drops drastically from day 19 onwards. Therefore, the two error metrics indicate the same conclusion.

days	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**10**
9	0.000	0.002							
10	0.002	0.043							
12	0.012								
16	0.000	0.000							
17	0.000	0.000							
18	0.000	0.004							
19	0.001	0.002							
20	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.007
21	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.011

Table 16: P-values of the statistical analysis for all datasets, considering the RMSE metric of each of the tested days. Only p-values $<$ 0.06 are shown

The 17 shows the p-values, at 95% confidence level, for the values of the R2 metric. It shows significant differences in change with respect to errors. It should be considered that the R2 metric measures the overall model fit, so it is possible that it does not represent the errors as much as the overall fit. This table also shows how the days are reduced, and there are significant differences in the R2 metric of days 1 and 2 with the models of days 10,12,16,20,21. While for days 4, 5, 7, and 8 there are only significant differences with the models for days 20 and 21. These results lead us to the same conclusion as indicated by the previous metrics. From day 9 onwards, the accuracy and fit of the models drop, on the tenth-day accuracy is lost. However, when the accuracy drops drastically from day 20 onwards.

Days	**1**	**2**	**4**	**5**	**7**	**8**
10	0.000	0.000	0.000	0.000	0.000	0.000
12	0.001					
16	0.005					
20	0.000	0.000	0.025	0.000		0.002
21	0.000	0.000	0.024	0.000		0.002

Table 17: P-values of the statistical analysis for all datasets, considering the R^2 metric of each of the tested days. Only p-values $<$ 0.06 are shown

Thus, we can conclude that days 1 and 2 are the most accurate, but there is really no significant loss of accuracy until day 9. Days 20 and 21 have significant

differences with the first 10 days. This already indicates that from then on, the model decreases in accuracy, so from day 19, if day 9 has not been carried out, the model should be retrained.

The problem of retraining can be solved by using incremental learning. Although there is no common framework for dealing with this problem in deep learning techniques, there are some publications that address it. Incremental learning can be classified into three scenarios [31]: Task-incremental learning, Domain-incremental learning and Class-incremental learning. Task-incremental learning scenario is about learning sequentially to solve a series of tasks in the same context. Domain-incremental learning scenario focuses on solving the same problem but in different contexts. In contrast, Class-incremental learning focuses on learning to discriminate between incrementally observed classes. In the case we are concerned with in this study, we will be able to perform incremental learning to change context, for example, to change the greenhouse, or to perform incremental tasks, which would be to continue predicting the temperature in the same greenhouse. Some publications perform incremental learning using Deep Learning techniques. Thus in [32] the authors present an incremental learning method using an LSTM network for attitude estimation of an object in space using a gyroscope, accelerometer, and magnetometer sensors. It is initially trained with sensor data that is dynamically updated at runtime by the LSTM network. The incremental learning is done in a rough way, starting from the initial weights of the trained network and only training with the new information available. Another incremental learning technique based on an LSTM is presented in [33], in this case, it is not focused on a specific problem, but the technique is evaluated with a synthetic data set and good results are obtained. Another study that also uses Deep learning techniques adapted to computational learning is presented in [34]. The authors present an incremental learning paradigm called Deep Model Consolidation to learn labels when the original training data is not available. The idea of the work is to first train a separate model for new labels only, and then combine the two individual models trained on data from two different sets of labels (old labels and new labels) through a novel double distillation training objective. Therefore, in view of the results obtained in the literature, the problem of retraining is solved by applying incremental learning, although future work may study in depth specific techniques for the problem addressed in this study.

5. Discussion

This section first analyses the proposed Pub/Sub solution (see subsection 3.2) using all the metrics obtained in the subsection 4.1.1 and 4.1.2. Moreover, the precision of the ML/DL models used (see subsection 3.4) is analyzed using metrics obtained in subsection 4.2. Finally, the necessary retraining time is discussed following the metrics shown in the subsection 4.3.

Regarding the Pub/Sub solution, two aspects should be considered: (1) the execution time spent on training and inference with the different ML/DL models, executed on CPU or GPU in the HPC computing platform; and (2) the execution time spent on loading and forecasting with the different ML/DL models, executed on CPU and GPU of the edge computing platform.

HPC platforms are needed for training heavy models, like those ones used in this investigation. Analyzing the obtained results can be concluded that using GPU is much faster than using only a CPU obtaining, on average, a 1.25X performance. However, in terms of inference, both CPU and GPU are very evenly matched, with the CPU being slightly faster than the GPU (on average, 0.94X of performance). This makes sense in our case because, although the models are heavy for training as they use large datasets to do this task, the inference is reduced to the univariate prediction of the greenhouse internal temperature. Furthermore, a maximum prediction horizon of 24 hours is considered, as a larger time horizon is not operational in the greenhouse. However, this implies that at the smallest temporal granularity; i.e. 15 minutes, no more than 96 values are predicted, which implies a light forecast for this type of deep learning model.

Edge computing is needed to generate a forecast as soon as data is generated or received. Analyzing the obtained results can be concluded that using CPU is much faster than using GPU obtaining, on average, a 2.44X performance as far as the model load is concerned. However, in terms of prediction, both CPU and GPU are evenly matched, with the CPU being slightly faster than the GPU (on average, 0.91X of performance). As with HPC platforms, this makes sense in our case because, although the stored models are heavy, once loaded into memory it is much faster to perform the inference on CPU rather than having to take it to GPU since the inference is still reduced to the univariate prediction of the internal greenhouse temperature as in the previous case. Although the model file is heavy, the model loading and univariate prediction tasks are very light tasks, so performing them on GPU means the appearance of bottlenecks, especially in memory access tasks, not taking full advantage of the GPU's potential.

Comparing the model training/loading and inference times on HPC and edge

platforms, it can be concluded that loading the model in an edge platform is much faster than training it in an HPC platform (on average, 1,057.71X performance). Moreover, due to the low computational capabilities of edge platforms, the forecasting time in edge platforms is much slower than in HPC platforms (on average, 0.24X performance).

Regarding the accuracy of ML/DL models, it can be concluded that it depends mainly on the characteristics of the data. For example, when there is a lot of data with a low temporality (such as 15-minute datasets), models such as CNN and CNNLSTM perform very well. However, when the data has a higher temporal granularity (such as 60-minute datasets), models such as LSTM show the best performance. It should also be noted that there are simple (in terms of architecture) models such as MLP that have reported good results. In addition, it is important to note that smooth pre-processing is not effective for any of the techniques, as it always provides lower performance. Moreover, for DL techniques, the best results are usually obtained with the raw datasets, as they are able to remove possible noise from the data.

Although there are no major differences with the other techniques, the most stable technique is CNNLSTM because its accuracy performance is somewhat better than the other techniques, at both 12 and 24 hours, regardless of the temporal granularity of the data. However, in terms of computing time, it is one of the slowest techniques for inference, although the times are still manageable, so given the importance of greenhouse temperature and the fact that the time difference is only 1 or 2 seconds, it can be selected as the best technique to implement.

Finally, we discuss the analysis of the days needed to perform model retraining. Considering the results shown, in the first 9 days, the model remains stable in all metrics. From day 9 onwards, the model loses accuracy. However, the results have shown that when there is really a significant loss of accuracy is from day 20 onwards, as all previous days have significant differences with the metrics obtained from day 20. This shows the need for retraining on the ninth day of the model run. If there are any problems in the system, there would be a margin of 11 days to retrain the model (i.e., until day 20), where the model update should be mandatory. Thus, we can conclude that from day 19 the model should collect all the data and retrain the new model without delay, although the best day not to lose accuracy would be day 9. It is important to consider that from day 9 the model must be trained, if we are looking for optimal accuracy, for any kind of temporal granularity, the season of the year, as well as data pre-processing.

6. Conclusions and future work

New technologies are helping to improve quality, efficiency, and profitability in many areas. Farming in recent years is gaining a great deal of performance when new technologies bring their advantages. Proof of this is the automation that is taking place in greenhouses. IoT and new communications protocols help to connect sensors, with information-gathering and decision-making systems quickly and effectively, achieving fast and efficient monitoring and execution of actions. Thus, in this research, an infrastructure applied to an operational greenhouse has been proposed, consisting of a Pub/Sub-based infrastructure that offers an interoperable and decentralized dynamic architecture for ML/DL training and inference. The infrastructure is evaluated to forecast the internal temperature of an operational greenhouse. This allows the farmer to act in advance to have the best climatic conditions for his crops, with a reliable and fast-running model on the edge. After this research, it can be concluded that Pub/Sub systems are nowadays mandatory for IoT applications that require taking actions in almost real-time. When a real-time data publisher is considered to monitoring an ecosystem, and decisions need to be taken based on analytics and forecast, it is mandatory to have a tightly coupled, seamless IoT infrastructure that covers the whole data cycle from the capture to the processing pipeline to analyze them and make reliable forecasting in the shortest possible time in order to make decisions in NRT. Finally, it should be noted that the best prediction model found for the SEPARATE infrastructure was the CNNLTSM model. Moreover, it has been analyzed and from the 9th day of prediction at any granularity and/or season of the year, the model should be retrained in order not to lose accuracy.

The future work of this research consists of optimizing the developed architecture, following different strategies that could improve the overall performance such as analyzing other possible Pub/Sub architectures that improve the response times of the proposal. Moreover, in order to optimize the models by which this architecture is developed, the following strategies could be studied: incorporating other artificial intelligence models different from the existing ones; increasing the number of variables to be predicted using the developed architecture; and changing the training strategy (batch training) for one that allows optimizing the training time, such as online/incremental training.

Declarations

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".

Authors' contributions

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

Consent to publish

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

References

[1] J. Lowenberg-DeBoer, The economics of precision agriculture, in: Precision agriculture for sustainability, Burleigh Dodds Science Publishing, 2019, pp. 481–502.

[2] M. A. Zamora-Izquierdo, J. Santa, J. A. Martínez, V. Martínez, A. F. Skarmeta, Smart farming iot platform based on edge and cloud computing, Biosystems engineering 177 (2019) 4–17.

[3] M. C. Garrido, J. M. Cadenas, A. Bueno-Crespo, R. Martínez-España, J. G. Giménez, J. M. Cecilia, Evaporation forecasting through interpretable data analysis techniques, Electronics 11 (4) (2022) 536.

[4] D. A. Howard, Z. Ma, C. Veje, A. Clausen, J. M. Aaslyng, B. N. Jørgensen, Greenhouse industry 4.0–digital twin technology for commercial greenhouses, Energy Informatics 4 (2) (2021) 1–13.

[5] J. Yang, A. Sharma, R. Kumar, Iot-based framework for smart agriculture, International Journal of Agricultural and Environmental Information Systems (IJAEIS) 12 (2) (2021) 1–14.

[6] E. Yaacoub, M.-S. Alouini, A key 6g challenge and opportunity—connecting the base of the pyramid: A survey on rural connectivity, Proceedings of the IEEE 108 (4) (2020) 533–582.

[7] K. G. Liakos, P. Busato, D. Moshou, S. Pearson, D. Bochtis, Machine learning in agriculture: A review, Sensors 18 (8) (2018) 2674.

[8] A. Kamilaris, F. X. Prenafeta-Boldú, Deep learning in agriculture: A survey, Computers and electronics in agriculture 147 (2018) 70–90.

[9] D. Garg, M. Alam, Deep learning and iot for agricultural applications, in: Internet of Things (IoT), Springer, 2020, pp. 273–284.

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

[11] P. Warden, D. Situnayake, TinyML, O'Reilly Media, Incorporated, 2019.

[12] Y. Wu, E. Dobriban, S. Davidson, Deltagrad: Rapid retraining of machine learning models, in: International Conference on Machine Learning, PMLR, 2020, pp. 10355–10366.

[13] B. Mishra, A. Kertesz, The use of mqtt in m2m and iot systems: A survey, IEEE Access 8 (2020) 201071–201086.

[14] B. Wukkadada, K. Wankhede, R. Nambiar, A. Nair, Comparison with http and mqtt in internet of things (iot), in: 2018 International Conference on Inventive Research in Computing Applications (ICIRCA), IEEE, 2018, pp. 249–253.

[15] S.-M. Kim, H.-S. Choi, W.-S. Rhee, Iot home gateway for auto-configuration and management of mqtt devices, in: 2015 IEEE Conference on Wireless Sensors (ICWiSe), IEEE, 2015, pp. 12–17.

[16] N. Tantitharanukul, K. Osathanunkul, K. Hantrakul, P. Pramokchon, P. Khoenkaw, Mqtt-topics management system for sharing of open data, in: 2017 International Conference on Digital Arts, Media and Technology (ICDAMT), IEEE, 2017, pp. 62–65.

[17] Y. Syafarinda, F. Akhadin, Z. Fitri, B. Widiawan, E. Rosdiana, et al., The precision agriculture based on wireless sensor network with mqtt protocol,

in: IOP Conference Series: Earth and Environmental Science, Vol. 207, IOP Publishing, 2018, p. 012059.

[18] A. A. Ahmed, S. Al Omari, R. Awal, A. Fares, M. Chouikha, A distributed system for supporting smart irrigation using internet of things technology, Engineering Reports 3 (7) (2021).

[19] F. M. Taha, A. A. Osman, S. D. Awadalkareem, M. S. Omer, R. S. Saadaldeen, A design of a remote greenhouse monitoring and controlling system based on internet of things, in: 2018 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE), IEEE, 2018, pp. 1–6.

[20] T. A. Singh, J. Chandra, Iot based green house monitoring system., J. Comput. Sci. 14 (5) (2018) 639–644.

[21] R. Tashakkori, A. S. Hamza, M. B. Crawford, Beemon: An iot-based beehive monitoring system, Computers and Electronics in Agriculture 190 (2021) 106427.

[22] U. Hunkeler, H. L. Truong, A. Stanford-Clark, Mqtt-s—a publish/subscribe protocol for wireless sensor networks, in: 2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE'08), IEEE, 2008, pp. 791–798.

[23] M. Bender, E. Kirdan, M.-O. Pahl, G. Carle, Open-source mqtt evaluation, in: 2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC), IEEE, 2021, pp. 1–4.

[24] J. Brownlee, Deep learning for time series forecasting: predict the future with MLPs, CNNs and LSTMs in Python, Machine Learning Mastery, 2018.

[25] C. M. Bishop, Pattern recognition and machine learning, Springer, 2006.

[26] S. Haykin, R. Lippmann, Neural Networks: A Comprehensive Foundation, 2nd Edition, Prentice Hall PTR, 1998.

[27] Z. Li, F. Liu, W. Yang, S. Peng, J. Zhou, A survey of convolutional neural networks: analysis, applications, and prospects, IEEE Transactions on Neural Networks and Learning Systems (2021).

[28] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (8) (1997) 1735–1780.

[29] T. V. Hecke, Power study of anova versus kruskal-wallis test, Journal of Statistics and Management Systems 15 (2-3) (2012) 241–247.

[30] G. Marsaglia, W. W. Tsang, J. Wang, Evaluating kolmogorov's distribution, Journal of statistical software 8 (2003) 1–4.

[31] G. M. van de Ven, T. Tuytelaars, A. S. Tolias, Three types of incremental learning, Nature Machine Intelligence (2022) 1–13.

[32] P. Narkhede, R. Walambe, S. Poddar, K. Kotecha, Incremental learning of lstm framework for sensor fusion in attitude estimation, PeerJ Computer Science 7 (2021) e662.

[33] Á. C. Lemos Neto, R. A. Coelho, C. L. d. Castro, An incremental learning approach using long short-term memory neural networks, Journal of Control, Automation and Electrical Systems 33 (5) (2022) 1457–1465.

[34] J. Zhang, J. Zhang, S. Ghosh, D. Li, S. Tasci, L. Heck, H. Zhang, C.-C. J. Kuo, Class-incremental learning via deep model consolidation, in: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2020, pp. 1131–1140.