Machine Learning for Predictive Diagnostics at the Edge: An IIoT Practical Example

Machine Learning for Predictive Diagnostics at the Edge: an IIoT Practical Example

Paolo Bellavista, Roberto Della Penna, Luca Foschini, Domenico Scotece
Computer Science and Engineering Dept. (DISI) University of Bologna
Viale del Risorgimento, 2 – 40136 Bologna, Italy
{paolo.bellavista, luca.foschini, domenico.scotece}@unibo.it, roberto.dellapenna@studio.unibo.it

Abstract— **Edge Computing is becoming more and more essential for the Industrial Internet of Things (IIoT) for data acquisition from shop floors. The shifting from central (cloud) to distributed (edge nodes) approaches will enhance the capabilities of handling real-time big data from IoT. Furthermore, these paradigms allow moving storage and network resources at the edge of the network closer to IoT devices, thus ensuring low latency, high bandwidth, and location-based awareness. This research aims at developing a reference architecture for data collecting, smart processing, and manufacturing control system in an IIoT environment. In particular, our architecture supports data analytics and Artificial Intelligence (AI) techniques, in particular decentralized and distributed hybrid twins, at the edge of the network. In addition, we claim the possibility to have distributed Machine Learning (ML) by enabling edge devices to learn local ML models and to store them at the edge. Furthermore, edges have the possibility of improving the global model (stored at the cloud) by sending the reinforced local models (stored in different shop floors) towards the cloud. In this paper, we describe our architectural proposal and show a predictive diagnostics case study deployed in an edge-enabled IIoT infrastructure. Reported experimental results show the potential advantages of using the proposed approach for dynamic model reinforcement by using real-time data from IoT instead of using an offline approach at the cloud infrastructure.**

Keywords—Edge Computing; Industrial IoT; Machine Learning; Predictive Diagnostics; Apache Kafka

I. INTRODUCTION

One of the major challenges of the Industrial Internet of Things (IIoT) is to take advantage of the IoT technology in industrial decisions. IoT today generates a myriad of data by the billions of connected devices, including sensors and actuators, that are usually aggregated and stored on cloud platforms [1]. Mainly for manufacturing industries, the interaction and the management of IoT devices become enablers for new service opportunities including predictive maintenance, continuous monitoring of the parts subjected to wear, scheduling and remote running of maintenance interventions, and simulation of operation through digital twin implementation. However, the IIoT is quite different from the general IoT in terms of communication bandwidth needed to handle big data transmission in real-time, with reduced cost, improved latency and robust connectivity, such that real-time decisions that will result to efficiency, safety and stability of large scale IIoT. IIoT, as a promising tool and platform for Industry 4.0, has been widely studied and employed in various scenarios [2]. Essentially, IIoT deploys an integrated infrastructure to collect information from different kinds of sensors, transmit it to the cloud, and update the related parameters in the form of a closed-loop system [3]. For an effective IIoT platform, the need to handle big data timely and efficiently must be satisfied in order to enable automatic services from IoT devices.

The growth of Edge Computing has moved the computing from centralized data centers to the edge of the network. Edge computing aims to address these challenges by moving core capabilities such as networking, computing, storage, and applications closer to the devices. It has enabled intelligent services close to the manufacturing unit to meet the key requirements such as agile connection, data analytics via edge nodes, highly responsive cloud services, and privacy-policy strategy [4]. In this work, we claim that Edge Computing is a possible key enabler for the creation of a distributed digital twin infrastructure. Digital twins are essentially models that accurately represent a system (processes or machines) by using, generally, data generated by IoT. Among the many things these models enable: i) the description of systems; ii) the prediction of systems evolution; iii) the management and maintenance of systems. They are used to detect and diagnose anomalies, to determine an optimal set of actions that maximize key performance metrics, to effectively and efficiently enforce on-line quality management of production processes under latency and reliability constraints, and to provide predictions for strategic planning to help companies to significantly improve their profitability through digitalization, as well as to open up new opportunities for them for the creation of new services and business models.

However, to extract valuable information and consequently producing real-time analytics, Machine Learning (ML) techniques are often applied. The definition of ML is very broad, ranging from simple data summarization with linear regression to multi-class classification and deep neural networks [5]. One key enabler of ML is the ability to train models using a very large amount of data. With the increasing amount of data being generated by IoT devices, we foresee that ML tasks will become a dominant workload in distributed edge-enabled IIoT systems in the future. However, it is challenging to perform distributed ML on resource-constrained Edge Computing systems.

In this paper, we address the problem of enabling digital twins for real-time analytics by leveraging ML tasks in IIoT systems. To ensure this, we propose a three-layer architectural solution that uses AI techniques on each layer of the architecture. The proposed solution has the following primary innovation elements and features. First, we presented the primarily guidelines of our distributed architecture that

enables ML schemes applied to shop floor data retrieved from processes, resources, and products for process optimization, quality inspection, and preventive diagnostics. Second, in our infrastructure raw data is collected and stored at the edge layer without sending the raw data to the cloud, and the ML model is trained from the cloud. Third, edge layer can stimulate an update of the model towards the cloud if the accuracy of prediction is improved. Fourth, the cloud may update all local models (stored at edge layer) with the new improved model. Finally, we present a real use case for predictive diagnostics based on an open-source dataset for Air Pressure System (APS) failures at Scania trucks [6] which is considered a hot topic for IIoT applications. To demonstrate the benefits of using Edge Computing for learning-based algorithms we quantitatively evaluate the advantages of processing predictive diagnostics by tacking local decisions based on knowledge at the edge of the network in place of using off-line optimizations at the cloud. Let us note that the activities presented here are a first significant advancement step accomplished within the context of a large H2020 project called IoTwins for distributed hybrid twins exploiting edge-enabled distributed AI techniques [7].

The remainder of this paper is organized as follows. Section II provides the necessary background material as well as an overview of the most important related literature. Section III provides our proposed approach for distributed intelligence. Section IV provides the description of the use case and the implementation details, while Section V contains the setup of our experiment environment and performance evaluation. Finally, we summarize and conclude our work in Section VI.

II. Background & Related Work

This section provides definitions for the involved technologies and paradigms, and then summarizes the mainstream directions of the literature.

A. Apache Kafka in IIoT environment

Most common IIoT protocols fall into two categories: i) publish-and-subscriber (pub/sub) protocols which connect and publish data to a topic on an intermediary broker; ii) poll-response or client-server protocols in which clients continually connect to the server and make requests to determine if any data has changed. To effectively build a highly scalable solution with a high level of efficiency, in the field of IIoT it is best to adopt a publish-subscribe communication protocol. Rather than connecting applications directly to devices, publish-subscribe protocols decouple devices and allow applications to connect to middleware. MQTT, AMQP, DDS, and XMPP are examples of most used pub-sub protocols.

It is now clear that for a huge and fast amount of data coming from manufacturing machines we need a pub-sub platform to manage all the data. The use of message-oriented middleware (MOM) is mandatory to decouple readers and writers, making our general architecture fitting most of the work machines and the several monitoring and actuation tools present in the different industrial environments.

Apache Kafka is an open source stream processing platform that enables communication between multiple producers and multiple consumers [8]. This tool supports the durable retention of messages and permits to handle a huge amount of data. Very popular use cases for Apache Kafka are:

- Real-time streaming data pipelines used for the data aggregation, processing, and transport;
- Event-reactive streaming applications used for fraud detection, data validation, email sending confirmation;
- Applications for real-time data analytics, stream processing, log aggregation, messaging, audit trail, sync for cooperative nodes.

Apache Kafka provides a publish-subscribe messaging service, where a Producer (publisher) sends messages to a Kafka topic in the Kafka cluster (message brokers), and a Consumer (subscriber) reads messages from the subscribed topic. A topic is a logical category of messages and can have many Producers and many Consumers, and it has a retention period after which messages can be discarded to release space on the machine. Moreover, a topic may be stored in one or more partitions, which are the physical storage of messages in the Kafka cluster. The Kafka cluster consists of several brokers (Kafka servers), and all the partitions of the same topic are distributed among the brokers. Each partition is physically stored on disks as a series of segment files that are written in an append-only manner, and it is replicated across the Kafka broker nodes for fault tolerance. Each partition can be either a leader or a replica for a topic, and only the leader partition handles all reads and writes of messages with producer and consumer, which is performed in a FIFO manner. The fault tolerance policy allows a replica to become the new partition leader in case of old leader fails. Kafka uses partitions to scale a topic across many servers for producers to write messages in parallel, and also to facilitate the parallel reading of consumers.

The distribution of components demands a coordination tool that, for the Apache Kafka platform, is Zookeeper [9]. This orchestrator provides naming and grouping services, coordinating consumers and the broker and identifying the central point for the whole Kafka system to retrieve information on configuration and leadership election. Zookeeper also manages the modification of the cluster topology, catching the presence of new nodes in the system.

B. IoTwins

The original results presented in the following parts of this paper have been achieved within the context of the just started H2020 IoTwins Innovation Action project [7], scientifically coordinated by our research group. IoTwins is a large (3 years, 20.1M€ budget) industry-driven project that puts together 23 partners from 8 countries; it has the ambition to lower the barriers, in particular for SMEs, to building edge-enabled and cloud-assisted intelligent systems and services based on big data for the domains of manufacturing and facility management. To this purpose, IoTwins is working to design a reference architecture for distributed and edge-enabled digital twins and is experimenting its implementation, deployment, integration, and in-the-field evaluation in several industrial testbeds.

In particular, the IoTwins digital twins are essentially models that accurately represent a system (either infrastructure or process or machine) along with its performance. These models enable the description of the system itself and its dynamics (descriptive or interpretative models), the prediction of its evolution (predictive models), and the optimization of its operation, management and maintenance (prescriptive models). They may be hybrid, i.e.,

by exploiting mixed and heterogeneous types of input from in-the-field experimental measurements (online/offline monitoring) and from analytical models as well as simulations/emulations. IoTwins distributed twins are used to detect and diagnose anomalies, to determine an optimal set of actions that maximize key performance metrics, to effectively and efficiently enforce on-line quality management of production processes under latency and reliability constraints, and to provide predictions for strategic planning to help companies, especially SMEs, to significantly improve their profitability through digitalization, as well as to open up new opportunities for them for the creation of new services and business models.

A crucial focus and primary activity of the project will be to deliver twelve industrial testbeds, of significant interest for SMEs, by sharing the same underlying methodology. The IoTwins testbeds are grouped into three classes: (i) testbeds in the manufacturing sector with the goal of optimizing production quality and plant maintenance, (ii) testbeds for the optimization of facility/infrastructure management, and (iii) testbeds for the in-the-field verification of the replicability, scalability, and standardization of the proposed approach, as well as the generation of new business models. In particular, in the manufacturing sector, four industrial pilots are aimed at providing predictive maintenance services that exploit sensors data to forecast the time to failure and produce maintenance plans that optimize maintenance costs; this will permit to reduce the risk of unplanned downtime from 15% to 25%, that is estimated to affect up from 5% to 20% of the overall manufacturing productivity. In the service sector, the three ioTwins testbeds concern facility management, by covering online monitoring and operation optimization in IT facilities and smart grids, as well as intervention planning and infrastructure maintenance/renovation on sport facilities on the basis of data collected by sophisticated and heterogeneous monitoring infrastructures. These three pilots are aimed at improving the environmental footprint of ICT facilities, by increasing the efficiency and resiliency of large critical ICT infrastructures, and at maximizing people safety via online adaptation of evacuation plans (and mobility flows in general) in sport facilities. The five last testbeds have the original goal of showcasing the replicability of the proposed IoTwins methodology in different sectors, the scalability of the adopted solutions, and their capability to help SMEs to generate new business models. For example, some industrial partners are interested to customize and apply the solutions developed in the first set of testbeds in other more articulated deployment environments (larger multi-site production plants in the case of Guala Closures or larger stadium facilities in the case of Barcelona Football Club).

IoTwins claims that IoT, edge computing, and industrial cloud technologies together are the cornerstones for the creation of distributed digital twin infrastructures that, after test-bed experimentation, refinement, and maturity improvements, can be easily adopted by SMEs: (i) industrial cloud, also based on HPC resources, enables the creation of accurate predictive models based on advanced ML for end-to-end deep networks, which require huge computing power for training; (ii) elastic cloud resource availability creates the opportunity to boost model accuracy by fitting and complementing data produced by industrial IoT sensors with data produced by large-scale parallel simulation; (iii) edge computing makes it possible to close the loop between accurate models and optimal decisions by enabling very responsive on-line local management of operational parameters in the targeted plants and filtered/fused reporting to the cloud side of only significant monitoring data (e.g., anomalies and deviations); and (iv) edge computing can leverage and accelerate the adoption of digital twin techniques by exploiting its industry-perceived advantages in terms of increased reliability/autonomy (e.g., independently of continuous connectivity to the global remote cloud) and of improved locality preservation of critical production data that can be maintained and used directly at the plant premises (*data sovereignty*).

C. Related Work

Without any pretense of being exhaustive, in the following, we report main research activities in the three main related areas.

ML is already widely used across a variety of domains to extract useful information from large-scale data. More recently, distributed ML solutions have been employed in Edge Computing infrastructures, including a cloud and edge layer. Many systems support strategies for the allocation of computational resources using deep reinforcement learning in Edge Computing networks [10]. Similarly, Chandakkar et al. in [11] proposed strategies for re-training a deep neural network (DNN) in an Edge Computing infrastructure. Both proposed systems leverage the Edge infrastructure both for the possibility to have computation close to the data source and for the freshness of the data.

Research on Edge Computing supporting IoT is progressing rapidly. In [12], the authors have proposed a novel approach of using the intelligence at the edge of the network by leveraging resource-poor devices such as Raspberry Pi for IoT data analytics. Its use case can be found in intelligent manufacturing. In the same direction, there is the work proposed by Condry et al. [13] that presents a system model for real-time and safer response IoT control operations by using smart edge nodes. Moving to the field of IIoT, Georgakopoulos et al. [14] proposed a roadmap combining cloud edge computing for IoT-based manufacturing. Byers [15] discusses some of the more important architectural requirements for IoT networks in several use cases including real-time manufacturing, and how Edge Computing can help fulfill them. All of them are important references for the deployment of Edge Computing in IoT-based manufacturing. Despite several edge-enabled architectural solutions have been proposed to fulfill the IoT requirements, only a few works aim to take advantage of Edge Computing infrastructure for distributing the knowledge among them and executing real-time data analytics.

Focusing on Edge Computing in Industrial IoT, only recently researchers started to exploit ML or intelligence at the edge of the network for predictive analysis or manufacturing control. In particular, the work by Raileanu [16] proposed an architectural framework for gathering heterogeneous data from the shop floor and aggregating them at the edge of the network. Successively, those data are sent to the cloud control platform that hosts a control system in charge of operation optimization, execution, and monitoring. Despite this, the framework proposed by the authors does not fully exploit the possibility of combining Edge Computing and ML techniques, as they do not propose control operation at the edge based on local knowledge. Another work that leverages Edge Computing in IIoT is presented in [17]. In that work, the authors propose an architecture of edge computing for IoT-

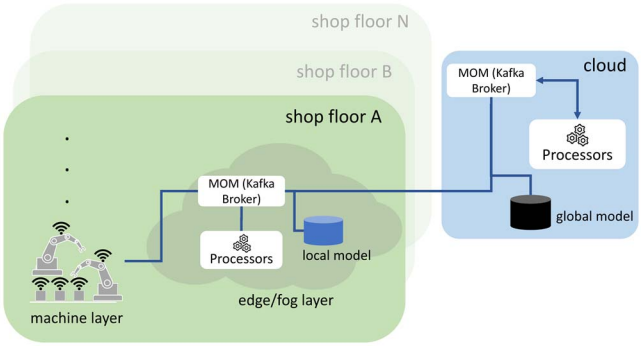

Fig. 1 Proposed Distributed Architecture

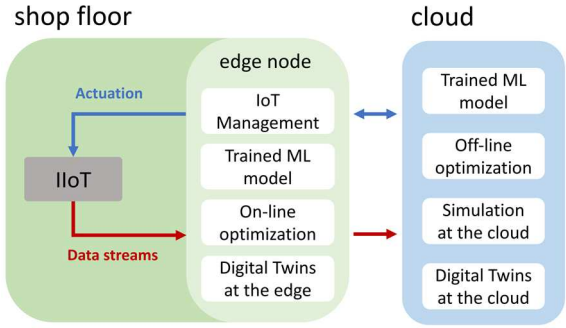

Fig. 2 Architectural Components

based manufacturing. Despite they basically analyze the role of Edge Computing in an IoT-based manufacturing system, the paper explores the idea to run ML at the edge nodes to utilize real-time data to make predictions and to subsequently update the knowledge base. This allows intelligent decision making at the network edge. However, this is not the main aim of that paper while we aim both to support IIoT devices and running ML algorithms at the edge for monitoring them.

III. Distributed Architecture

Our work focuses on the innovation potential of a large set of industrial scenarios that make use of edge-enabled and cloud-based big data services. That is typically the case for predictive maintenance and production optimization in manufacturing and for management optimization of large-scale facilities. Fig. 1 depicts our proposed distributed architecture designed to support and manage both shop floors and industrial devices. The use of MOM facilitates communication between all involved entities; in particular, it facilitates the data gathering from IIoT devices at the edge. Our architecture includes different functionalities at each layer of the architecture; in general, they can work both locally with fresh data and in the cloud, processing also historical data.

Fig. 2 presents our uniform approach for different industrial plants, where the cloud interacts with the edge and IoT devices for implementing our idea of distributed digital twins. In particular, the data streams generated from IoT in a shop floor reach the closest edge node, where they may be processed (e.g., aggregated, filtered, anonymized, etc.) and trigger local actuation actions with very low latency based on ML algorithms. Moreover, data streams may be forwarded to the cloud that is responsible for training ML models, and off-line optimization, among many things. Our proposed architecture consists of five main building modules: i) an IoT management module that facilitates the communication between devices and edge nodes; ii) a set of ML algorithms including decision trees, regression trees, random forests, gradient boosting trees, neural networks, and deep networks for giving back on- off-line predictions; iii) an ML module trained via basic types of learning algorithms including Supervised Learning, Unsupervised Learning, and Reinforcement Learning; iv) a Model that accurately represents a system (infrastructure, process, machine, etc.); v) a Simulation module that sends feedback about models in order to enhance and optimize them.

IoT Management. This module is in charge of managing IoT devices. In particular, the IIoT world encompasses the manufacturing machines, usually equipped with legacy software. Hence, this module contains a set of standards protocols for data gathering from heterogeneous devices.

On- Off-line optimization. This module contains a composition of machine learning algorithms and mathematical models able to take decisions based on knowledge gained from a learning algorithm. In particular, ML algorithms are used, for instance, to detect and diagnose anomalies, to determine an optimal set of actions, to enforce the quality of production processes, and to provide predictions for strategic planning.

Trained ML model. This module is in charge of training the model used by the ML algorithm. Normally, the training phase is a time-consuming task and requires several computational resources. It could be challenging to perform training algorithms on resource-constrained edge nodes. Despite local updates consume computation resources of the edge node, this module allows training the model both at the edge and from the cloud.

Digital Twins. This module contains the knowledge of every single node. The main idea is to host ML models, already trained at the cloud side, at the edge of the network with the possibility to improve the local model via the interaction with underlying devices (e.g., from mobile devices to Industrial IoT devices). Finally, edge nodes return feedback to the cloud manager in terms of model improvements, and update the global model at the cloud, by exploiting collaboration among them.

Cloud optimization. This is a logical component that runs at the cloud and is in charge of optimizing the model. In particular, data gathered at the edge of the network may grow local models which are also forwarded to the cloud for ML models optimization. Therefore, this module selects the more appropriate model and sends it back to selected edge nodes. On the other hand, ML models already trained at the cloud side may be sent back for model optimizations towards edge nodes.

IV. Use Case & Implementation Details

This section presents a real case study in order to illustrate how to leverage distributed digital twins of production plants and facilities, enabled by off-the-shelf edge/cloud technologies for big data processing. In particular, this testbed is aimed at leveraging the intelligence at the edge of an air pressure system by aggregating ML models of single nodes for predictive maintenance. Data are used to detect the health status of the APS, to predict failures, and to plan maintenance operations for reducing unexpected breakdowns. The system

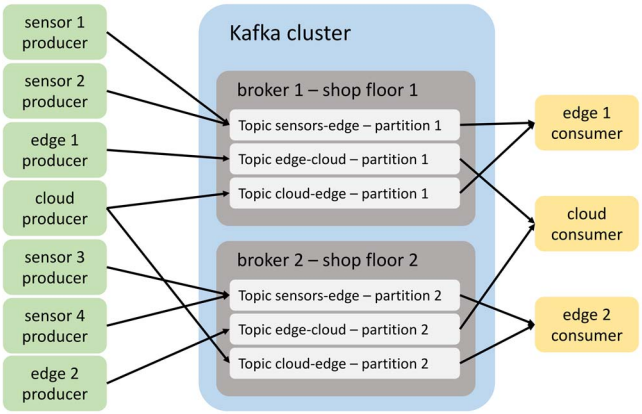

Fig. 3 Our implementation scenario

will be divided into several sub-parts/systems devoted to specific tasks, with specific raw monitoring indicators, such as temperature indicators of each component part.

Following the uniform workflow approach depicted in Fig. 2, each component of the system provides data for the monitoring system that are properly channeled by an edge-computer; these data will be sent to the closest edge to produce local intelligence. ML techniques are trained to produce predictive models on health status and to allow the edge to localized anomaly detection. Trained predictive models are then transferred to the cloud to reinforce the global model which, if it is improved (e.g., in terms of accuracy), will be returned in each edge node. Also, the cloud simulation and off-line optimizations provide maintenance plans and guidelines for future improved design.

In order to simulate this testbed, we used an open-source dataset that consists of data collected from Scania trucks in everyday usage [18]. The system in focus is the APS which generates pressurized air that are utilized in various functions in a truck, such as braking and gear changes. Therefore, failures should be predicted before they occur. Falsely predicting a failure has a cost of 10, while missing a failure has a cost of 500. This makes sense because is simpler to manage the case of false positive rather than the case of a missing a failure. The data contains a training set and a test set. The training set contains 60,000 rows, of which 1,000 belong to the positive class and 171 columns, of which one is the class column and the test set contains 16,000 rows.

A. Implementation Details

As already stated in Section II, we adopted Apache Kafka as a messaging middleware between system entities such as sensors, edge nodes, and the cloud. Our testbed consists of four sensors (two for each edge nodes), two edge nodes, and the cloud. The Kafka cluster is composed of two distinct Kafka Broker respectively one for each edge node that logically represents a shop floor. All system entities communicate with each other via Kafka Topics and for this reason, we provided three distinct Topic: one for the communication between sensors and edge nodes, another for the communication between edge nodes and cloud, and vice-versa. Then, each Topic is divided into two Partitions one for each Broker. Fig. 3 depicts how we implemented the scenario.

The sensors use part of the dataset, in particular, 58,000 over 60,000 instances, as new data. In particular, one sensor retrieves data from the open-source csv file and uses 58K records as newly produced data (each sensor uses 29K records). After this, it creates a Kafka Producer that is used for sending data via the "*sensors-edge*" topic.

Each edge node, one per locality/Kafka Broker, communicates with sensors in the corresponded locality and with the cloud. Its task is to receive data from sensors, classifies the model instances, enhancing the digital twins at the edge, and to send the update models towards the cloud. First, it receives the ML initial model from the cloud and then it takes care of three major tasks including *consume_sensor_data*, *update_model_local*, and *update_model_remote*.

- *consume_sensor_data*. This task is in charge of reading data from sensors and classifying instances by using the initial model. Edge nodes use the topic "sensors-edge" in order to read data.
- *update_model_local*. This task takes care of control of the state of predictions. In particular, it compares the predictions with the supervised data and uses the false-negative instances for creating a new model.
- *update_model_remote*. This task is in charge of updating the local model towards the cloud. In particular, it sends the new model plus the dataset used for its creation. Anyway, the decision to swap the global model and therefore update all local models is up to the cloud.

Finally, the cloud creates the initial model by using only 2,000 of 60,000 instances through a Decision Tree classifier and then calculates the accuracy of the model. Of course, at this moment the accuracy does not have a reasonable value, in fact, the percentage of false-negative is greater than the percentage of false-positive. To trace the quality of the model, we considered also the recall and specificity that represent respectively the percentage of negative instances within the total and the percentage of positive instances within the total.

```
# create the initial model
nb_model = DecisionTreeClassifier()
nb_model.fit(features_train, labels_train)

# calculate the model accuracy
nb_out = nb_model.predict(features_test)
nb_accuracy = accuracy_score(labels_test, nb_out)
nb_recall = recall_score(labels_test, nb_out,
pos_label=1)
nb_specificity = recall_score(labels_test, nb_out,
pos_label=0)
```

```
# calculate the accuracy
out = model.predict(features_test)
accuracy = accuracy_score(labels_test, out)
recall = recall_score(labels_test, out, pos_label=1)
specificity = recall_score(labels_test, out,
pos_label=0)
differences = [accuracy - nb_accuracy, recall -
nb_recall, specificity - nb_specificity]

# if the model is better than the previous one is sent
back to edge nodes
if accuracy > 0.95 and recall > 0.75 and specificity >
0.95:
    ...
    producer.send('cloud-edge', key=b"train",
value=serialized, partition=0)
    producer.send('cloud-edge', key=b"train",
value=serialized, partition=1)
    ...
```

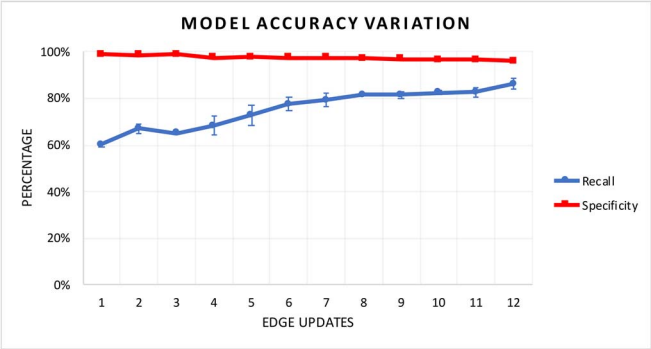

Fig. 4 Model accuracy variation

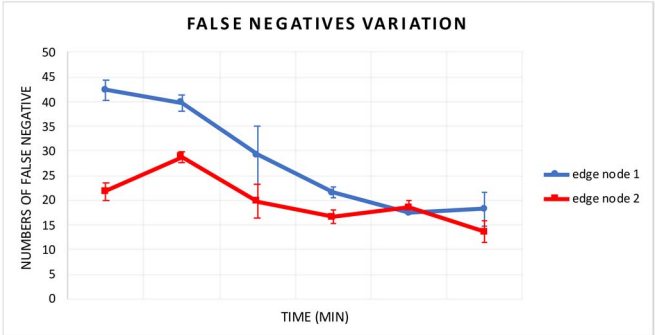

Fig. 5 False-negative instances variation at the edge

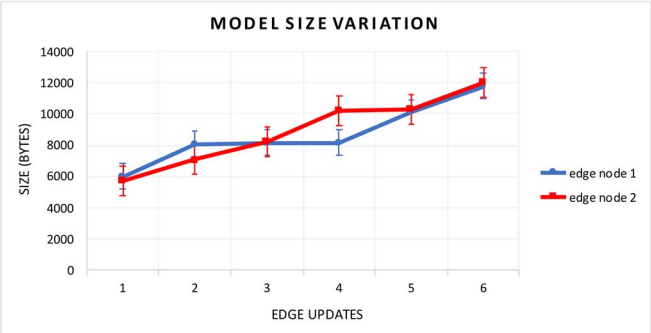

Fig. 6 Model size variation

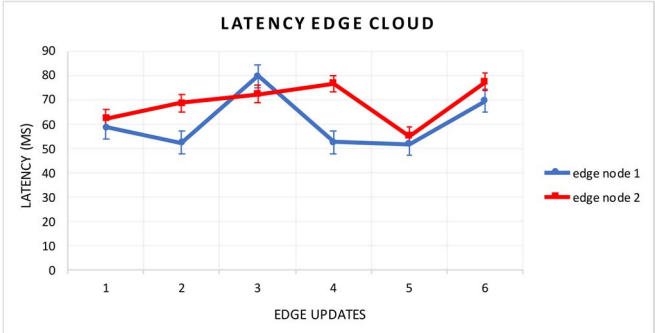

Fig. 7 Latency edge-cloud

Then, each time the cloud receives a new model from an edge node, it calculates the accuracy, recall, and specificity of that model and it sends back only if that model is better. This specific implementation is shown in the above snippet.

V. Experimental Results

We widely assessed and validated the advantages of our solution; this section presents a significant selection of experimental results obtained in our lab deployment scenario. To thoroughly test and evaluate the performance of our proposed architecture, we carried out several different sets of experiments, in terms of quality of prediction model, model size variation, and latency time. The results reported in this section are average values; all presented measurements have exhibited a limited variance (under 5% for 30 runs).

A. Experimental Environment

The prototype of our experimental environment is to set up four sensors, two edge nodes, and the cloud. The edge node is a Raspberry Pi 3 Model B equipped with 64-bit quad-core ARM Cortex-A53 processor, 1 GB of RAM and 16 GB of storage space, Wi-Fi and Bluetooth connection. The sensors are a Raspberry Pi Zero W equipped with a single-core Broadcom BCM 2835 1Ghz, 512 MB of RAM and 8 GB of storage space, Wi-Fi and Bluetooth connection. The cloud is a VM deployed at Amazon Web Service equipped with 8 GB of RAM, and 4 virtual cores Intel. Unless otherwise specified, edge nodes connect with the sensors via IEEE 802.11n connections and their maximum nominal available bandwidth is 40 Mbit/s, while connecting with the cloud through ethernet with 1 Gbit/s bandwidth.

B. Model Accuracy Variation

The first test family that we have done for this testbed regards the model accuracy variation. Please note that the edge nodes send reinforced models, with fresh data, to the cloud. Fig. 4 shows the model accuracy variation considering edge updates. To trace the quality of the model we considered the recall and specificity that represent respectively the percentage of negative instances within the total and the percentage of positive instances within the total. As one can see from Figure 4, the specificity goes down from 0.98 to 0.96 but the recall rising to 0.87 with an increase of 0.3. This because the initial dataset contains a few negative instances compared to the total dataset. Therefore, updating the model with fresh data allows creating a model that the total accuracy is the same but with more accuracy for predicting negative instances. To demonstrate this, Figure 5 shows how the number of false-negative instances recognized at the edge decreases over time increasing the accuracy of each edge node. For both tests, we used the Decision Tree as the classifier algorithm, instead of, for instance, a Naive Bayes classifier. This is because Decision Trees are very flexible, easy to understand, and easy to debug. Furthermore, they work very well with classification problems. Finally, they allow working with low power devices such as Raspberry Pi with very interesting performance results [19].

C. Model size variation and Latency cloud - edge

For the last set of experiments, we compared how the model size can impact the latency time between edge nodes and the cloud. As shown in Fig. 6, for consecutive model updates, the experimental results show that the models stored in each edge nodes grow in a linear fashion. In our specific use case, the model increases its size by approximately 50% (i.e.,

from 6KB to 12KB). Additionally, as shown in Fig. 7, we measured the latency time introduced from these consecutive model updates between edge nodes and the cloud. Actually, in this paper, we do not consider the other side of the latency time between the cloud and edge nodes. This because it's around the same values: 60ms of average latency time. Anyway, in our use case, the latency time introduced does not vary much compared to the model size variation, rather remains the same. Let us note that this delay is completely transparent to the logic of the application. The model updates can be made in a background mode.

VI. Conclusion

In this paper, we proposed an edge-enabled cloud-assisted system for distributed intelligence covering advanced ML algorithms. This work paves the way for ML at the edge by proposing a distributing architecture and discussing a real use case. The cornerstones of our work lie in the ability to have local models stored at the edge and running ML algorithms on it with the possibility to improve the global model through reinforced local models. We present several experimental results that show the advantages of using distributed ML at the edge of the networks in terms of model accuracy and system performances. The major advantage of the edge is to allow the possibility to reinforce local models with fresh data that can be used to enhance the accuracy of ML techniques. Although, results show that the model accuracy increases in a significant way while still retaining a very low latency. Fueled by these significant results, we are working on two main ongoing research directions. On the one hand, we are working extensively on the Cloud Optimization module and on the aggregation of local models (stored at the edge nodes) into the global model (stored at the cloud) and vice versa. On the other hand, we are deploying several testbeds solutions for smart factories where several heterogeneous IoT devices generate data streams. In particular, data streams originated from IoT devices reach edge nodes, where maybe aggregated and trigger local actions. Finally, these data streams may be also forwarded to the global cloud.

References

[1] "Cellular Networks for Massive IoT: Enabling Low Power Wide Area Applications," Ericsson, Stockholm, Sweden, 2016, pp. 1–13.

[2] L. D. Xu, W. He and S. Li, "Internet of Things in Industries: A Survey," in IEEE Transactions on Industrial Informatics, vol. 10, no. 4, pp. 2233-2243, Nov. 2014.

[3] R. Zhu, X. Zhang, X. Liu, W. Shu, T. Mao and B. Jalaian, "ERDT: Energy-Efficient Reliable Decision Transmission for Intelligent Cooperative Spectrum Sensing in Industrial IoT," in IEEE Access, vol. 3, pp. 2366-2378, 2015.

[4] W. Shi and S. Dustdar, "The Promise of Edge Computing," in Computer, vol. 49, no. 5, pp. 78-81, May 2016.

[5] S. Shalev-Shwartz and S. Ben-David, Understanding machine learning: From theory to algorithms. Cambridge university press, 2014.

[6] C. Gondek, D. Hafner, and O. R. Sampson, "Prediction of failures in the air pressure system of scania trucks using a random forest and feature engineering," in International Symposium on Intelligent Data Analysis. Springer, 2016, pp. 398–402.

[7] H2020 Innovation Action project IoTwins, Distributed Digital Twins for industrial SMEs: a big-data platform, ICT-11b call, avalilable at https://cordis.europa.eu/project/rcn/223969/factsheet/it.

[8] Kafka - documentation. Apache Software Foundation. Accessed: Oct. 28, 2019. [Online]. Available: https://kafka.apache.org/documentation/.

[9] Zookeeper - documentation. Apache Software Foundation. Accessed: Oct. 28, 2019. [Online]. Available: https://zookeeper.apache.org/.

[10] T. Yang, Y. Hu, M. C. Gursoy, A. Schmeink and R. Mathar, "Deep Reinforcement Learning based Resource Allocation in Low Latency Edge Computing Networks," 2018 15th International Symposium on Wireless Communication Systems (ISWCS), Lisbon, 2018, pp. 1-5.

[11] P. S. Chandakkar, Y. Li, P. L. K. Ding and B. Li, "Strategies for Re-Training a Pruned Neural Network in an Edge Computing Paradigm," 2017 IEEE International Conference on Edge Computing (EDGE), Honolulu, HI, 2017, pp. 244-247.

[12] P. Patel, M. Intizar Ali and A. Sheth, "On Using the Intelligent Edge for IoT Analytics," in IEEE Intelligent Systems, vol. 32, no. 5, pp. 64-69, September/October 2017.

[13] M. W. Condry and C. B. Nelson, "Using Smart Edge IoT Devices for Safer, Rapid Response with Industry IoT Control Operations," Proc. IEEE, vol. 104, no. 5, 2016, pp. 938–46.

[14] D. Georgakopoulos et al., "Internet of Things and Edge Cloud Computing Roadmap for Manufacturing," IEEE Cloud Computing, vol. 3, no. 4, 2016, pp. 66–73.

[15] C. C. Byers, "Architectural Imperatives for Fog Computing: Use Cases, Requirements, and Architectural Techniques for Fog-Enabled IoT Networks," IEEE Commun. Mag., vol. 55, no. 8, Aug. 2017, pp. 14–20.

[16] S. Raileanu, T. Borangiu, O. Morariu and I. Iacob, "Edge Computing in Indust00rial IoT Framework for Cloud-based Manufacturing Control," 2018 22nd International Conference on System Theory, Control and Computing (ICSTCC), Sinaia, 2018, pp. 261-266.

[17] B. Chen, J. Wan, A. Celesti, D. Li, H. Abbas and Q. Zhang, "Edge Computing in IoT-Based Manufacturing," in IEEE Communications Magazine, vol. 56, no. 9, pp. 103-109, Sept. 2018.

[18] APS Failure at Scania Trucks Data Set. Accessed: Oct. 28, 2019. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/APS+Failure+at+Scania+Trucks.

[19] S. Buschjager, K. Chen, J. Chen and K. Morik, "Realization of Random Forest for Real-Time Evaluation through Tree Framing," 2018 IEEE International Conference on Data Mining (ICDM), Singapore, 2018, pp. 19-28.