Future Generation Computer Systems 160 (2024) 251–263

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Online learning and continuous model upgrading with data streams through the Kafka-ML framework

Alejandro Carnero [a,*], Cristian Martín [a], Gwanggil Jeon [b], Manuel Díaz [a]

[a] ITIS Software, University of Malaga, Malaga, Spain
[b] Department of Embedded Systems Engineering, Incheon National University, Incheon, South Korea

ARTICLE INFO

Keywords:
Artificial Intelligence
Data streams
Kafka-ML
Online machine learning

ABSTRACT

A pipeline of constant data streams is being built by the Internet of Things (IoT) to monitor information about the physical environment. In parallel, Artificial Intelligence (AI) is constantly developing and enhancing industrial, economic, and academic endeavors as well as quality of life thanks to these IoT data. In streaming contexts, Kafka-ML is our open-source framework that enables the management of Machine Learning (ML) and AI pipelines over data streams. Accordingly, it simplifies the deployment of Deep Neural Networks (DNNs) in practical applications. Nonetheless, this framework did not support the possibility of carrying out an Online Learning (OL) process, which is needed when new data are continuously arriving, and the models need to adapt to them on the fly. In this work, we have extended our previous work, the Kafka-ML framework, to enhance the management of ML/AI pipelines with OL features to enable both ML/AI distributed and centralized models to learn indefinitely over time. These models are continuously upgraded thanks to a process where automatic and flexible inference is carried out when improvements in the model performance are achieved. This opens up a large number of new possibilities within different fields of application, development, and work under the premise of incremental learning with ML models such as Electrical Vehicles and Industry 5.0. We have validated these new features by adapting and deploying state-of-the-art DNN models in different online scenarios, for both single and distributed configurations. The results show the capability of Kafka-ML to execute effective online training processes for ML models, improving their performance over time as new data becomes available.

1. Introduction

In Machine Learning (ML), there are two primary learning paradigms: batch or offline learning and incremental or Online Learning (OL). On the one hand, in offline learning settings, an optimization procedure is often carried out to update the neural network's knowledge base in relation to the training data samples. On the other hand, OL makes an effort to gradually update the neural network's knowledge base after each training sample is presented [1].

Traditional batch learning assumes equally dispersed training data and a static view of reality. Following this approach, network parameters are often modified following the presentation of the entire training dataset. As a result, adding new data samples requires retraining the network due to the lack of an internal mechanism that would allow the network to incorporate new information into its knowledge base on the fly. Another option is to apply transfer learning [2] so that the weights of certain layers of the neural network (usually the final ones) can be frozen and only the others are retrained with a new dataset. In this way, the basic notions of knowledge the network has previously acquired are maintained, while the new ones, corresponding to the new problem, are added. This approach works well when the network is addressing a stationary issue environment, and the training data samples are suitably representative of the situation at hand [3]. Before a gradient reasonable weight update is performed, the total error on all training set instances is gathered. When employed with small to medium networks and datasets, offline training can be quick. However, it is relatively ineffective and prone to local minima when applied to big networks and datasets [4].

On its side, the error on every sample can be decreased in OL by updating the network parameters every time a new training sample is shown, which can be supplied with or without repetition. One major point of OL is the ability of the trained network to form an optimized knowledge base for tackling dynamically changing problems [5]. OL, which is necessary for learning and adaptation in a constantly changing environment, consists of a stochastic process since the training example for each update is selected randomly [6].

* Corresponding author.
E-mail addresses: alexcarnero97@uma.es (A. Carnero), cristian@uma.es (C. Martín), gjeon@inu.ac.kr (G. Jeon), mdiaz@uma.es (M. Díaz).

https://doi.org/10.1016/j.future.2024.06.001
Received 22 December 2023; Received in revised form 26 May 2024; Accepted 3 June 2024
Available online 6 June 2024

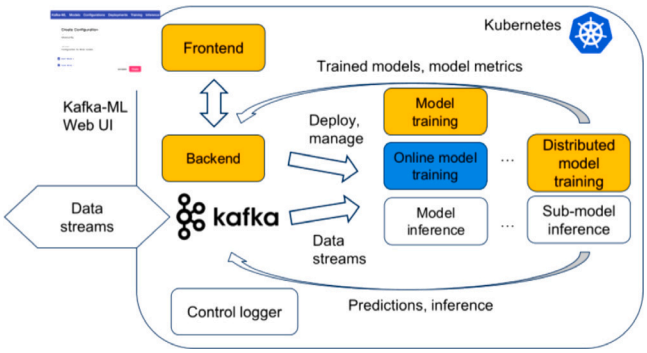


Fig. 1. Overview of the new Kafka-ML architecture and its components deployed and managed in Kubernetes clusters. Yellow component: modified; blue component: added.

Due to the current importance of developing online ML models and the need to efficiently integrate these algorithms and systems with the massive amount of data that is continuously generated by Internet of Things (IoT) devices in the form of data streams, in this work we have addressed this challenge through an extension of our previous work, the data streams ML framework Kafka-ML [7]. Currently, there is a lack of frameworks that can work with data streams and OL. Moreover, within the online paradigm, it is also important that ML models continue their learning indefinitely over time so we can potentially evaluate the model that has the most knowledge of the problem. For this reason, we have enabled the possibility of performing automatic and flexible upgrading of models for further inference. This is achieved through the improvements in the metrics obtained during OL. While these upgrades are performed automatically, the base model continues to be trained in a time-limited or time-unlimited manner, depending on the user's choice. Both options are available when deploying the models and differ in that the time-limited option is run at a user-defined time interval and if at any time no data is received during this period, the training is terminated. On the other hand, the function with unlimited time consists of a continuous training process, i.e., the model is always waiting for new data, and as it obtains improvements in its validation metrics, upgrades of the latest version are made to make future inferences. Fig. 1 shows an overview of the redefined Kafka-ML architecture, highlighting the new components in blue and the modified components in yellow. As before, the control logger manages control messages and transmits them to the back-end. Furthermore, Kubernetes[1] is responsible for managing Apache Kafka as a Docker[2] container. Currently, we use a newer version of Apache Kafka that self-manages the synchronization of brokers, so we no longer need to use Zookeeper. The main contribution of this work is the online learning module to the Kafka-ML framework and the continuous deployment of inference models.

As a result, our redefined Kafka-ML framework enables the flexible management of ML models throughout their entire lifecycle and their efficient integration with data streams, both for training and inference, and both for batch and online learning. There are other well-known works on this topic. For example, in this Github project[3] multiple examples demonstrate how to deploy analytic models to mission-critical, scalable production environments leveraging Apache Kafka and its Streams API. Models are built with Python, H2O, TensorFlow, Keras, DeepLearning4 and other technologies. Despite this, the authors only propose a number of ready-made examples. However, they do not offer a mechanism to deal with the training phase of these models in a flexible way like Kafka-ML. On the other hand, SAMOA [8] provides a collection of distributed streaming algorithms for the most common data mining and ML tasks such as classification, clustering, and regression, as well as programming abstractions to develop new algorithms that run on top of Distributed Stream Processing Engines (DSPE). It features a pluggable architecture that allows it to run on several DSPEs such as Apache Storm, Apache S4, and Apache Samza. Nevertheless, SAMOA does not work with well-known and fairly widespread ML frameworks such as TensorFlow or PyTorch, while Kafka-ML does support them and more are planned to be added in the future. Also, Kubeflow [9] is dedicated to making deployments of ML workflows on Kubernetes simple, portable, and scalable. Their goal is to provide a straightforward way to deploy open-source systems for ML to diverse infrastructures. However, in order to perform training and inference tasks, additional building of different containers are needed with Kubeflow. With Kafka-ML, users simply need to work with its web UI. Finally, the support of Kubleflow for data streams has to be done manually by its developers. In Kafka-ML, the managing of data streams is supported throughout the entire pipeline.

Due to the lack of platforms that integrate all of the needs of ML developers these days and allow the necessary continuous refinement of ML models over time in an automatic manner, this work represents an important step forward in this field. In this way, our Kafka-ML tool is a central meeting point and orchestrator for all these methods, technologies such as IoT, and continuous data streams. Therefore, the main contributions of this work are summarized below:

- Extension of our Kafka-ML framework to enable online learning with continuous data streams.
- Managing and deploying single and distributed models for online training.
- Flexible and automatic deployments of inference models according to predefined metrics.

The rest of the article is organized as follows. Section 2 presents the related projects of this field of research. In Section 3, the complete workflow of online ML models in Kafka-ML is detailed, with special attention to the incremental learning included. Then, in Section 4, the Kafka-ML architecture for OL deployments and their features are presented. Section 5 shows the results of the validation of the framework. Section 6 presents the main challenges and limitations of OL. And finally, Section 7 presents our conclusions and future directions of Kafka-ML.

2. Related work

Numerous works related to OL have been published recently in the field of Artificial Intelligence (AI). This section aims to bring together these works and provide an overview of the topic divided into different approaches.

2.1. Adaptive models

In [10], the authors present a new online deep learning framework that attempts to tackle the challenges in this field by learning adaptable depth Deep Neural Networks (DNN) models from a sequence of training data in an online learning setting. In particular, they propose a novel Hedge Backpropagation (HBP) method for online updating the parameters of DNNs effectively and validate the efficacy of their method on large-scale datasets, including both stationary and concept-drifting scenarios. In [11], a real-time DNN adaptive control architecture is developed for uncertain control-affine nonlinear systems to track a time-varying desired trajectory. The developed method establishes a framework to simultaneously update the weights of multiple layers for a DNN of arbitrary depth in real time. The real-time controller and

[1] https://kubernetes.io/.
[2] https://www.docker.com/.
[3] https://github.com/kaiwaehner/kafka-streams-machine-learning-examples.

weight update enable the system to track a time-varying trajectory while compensating for unknown drift dynamics and parametric DNN uncertainties. In addition, in [12], a selective ensemble-based online adaptive DNN (SEOA) is proposed to address concept drift. First, the adaptive depth unit is constructed by combining shallow features with deep features and adaptively controls the information flow in the neural network according to changes in streaming data at adjacent moments, which improves the convergence of the online deep learning model. Then, the adaptive depth units of different layers are regarded as base classifiers for the ensemble and weighted dynamically according to the loss in each classifier. Although these frameworks allow the network depth to be dynamically adapted, they do not provide the possibility of splitting the models into different parts and deploying them along various nodes in a distributed manner like Kafka-ML. Kafka has single processing, only stateless support, and it does not use threading or parallelism. On the other hand, Kafka Streams [13] is built on top of Kafka clients and it provides a single Kafka stream to consume and produce, perform complex processing, support stateless and stateful operations, and threading and parallelism. Also, stream partitions and tasks are logical units for storing and transporting messages. Nevertheless, it does not support batch processing and interacts only with a single Kafka cluster. Therefore, the choice between one or the other depends on the requirements and characteristics of the system where it is applied. In this case, it was decided to use Apache Kafka because inference is a simple process and does not require the use of Kafka Streams. Currently, we control the load with the number of replicas. Furthermore, Kafka Streams is only supported in Java, and right now most AI languages are in Python. However, Kafka Streams can be a good candidate for data pre-processing.

Moreover, the works carried out are limited in scope. Meanwhile, we propose a complete ML framework in which models can be registered, trained, and deployed for inference. The possibility of integrating their new algorithms with our platform would be very interesting.

2.2. Generative models

In [14], the authors propose a model that is able to perform online data classification and can adapt to data classes never seen by the model before while preserving previously learned information. Their approach does not need to store and reuse previous observations, which is a big advantage for data-streams applications, since the dataset one wants to work with can potentially be of very large size. To make up for the absence of previous data, the proposed model uses a recently developed Generative Adversarial Network (GAN) to drive a Deep Convolutional Network for the main classification task. More specifically, they propagate generative models instead of the data themselves, to be able to regenerate the historical training data that they did not keep. This work makes use of GANs to propagate the information whereas we base our entire system on data streams, which are lighter than ML models. In the same way, none of them have to store the data. Alternately, entity alignment methods adopt self-training to select pairs from predicted alignments, but accurately distinguishing positive and negative pairs is challenging, potentially leading to inappropriate alignments. Additionally, combining pre-aligned pairs during retraining may result in overfitting. To address these problems, authors from [15] propose a self-training entity alignment framework based on variety-aware GAN and OL algorithm named SEAGAN. To select reliable newly-aligned pairs from the predicted alignment, a variety-aware GAN with a metric of match variety that eliminates negative pairs differing significantly from positive pairs is designed. It leverages the distribution of entity pairs to determine the boundary between different types of pairs. Moreover, SEAGAN designs an OL algorithm that combines newly-aligned pairs with their one-hop and two-hop neighbor entities in pre-aligned pairs to update model parameters, which alleviates overfitting problems. However, although the first paper presents a neural network that is able to dynamically adapt to new classes and the second addresses the pair alignment problem, these projects only work with a specific model each (GANs). With Kafka-ML this is not the case—our platform allows the integration of different deep ML models and configurations, offering more flexibility for both traditional and incremental training paradigms. Using their model within our platform and seeing how it behaves might be something to consider in future work.

2.3. Time series models

In [16], the authors investigate online nonlinear regression and introduce novel regression structures based on the Long Short Term Memory (LSTM) networks. They provide efficient and effective online training methods for these structures, including particle filtering-based updates. The results illustrate significant performance improvements achieved by their algorithms compared to conventional methods on several different real-life datasets. Meanwhile, paper [17] introduces a method for online inference of temporal logic properties from data. Specifically, the authors tackle the online supervised learning problem. In this setting, the data are in the form of a set of pairs of signals and labels and become available over time. They propose an approach for efficiently processing the data incrementally. In particular, when a new instance is presented, the proposed method updates a binary tree that is linked with the inferred Signal Temporal Logic (STL) formula. This approach presents several benefits. Primarily, it allows the refinement of the current formula when more data are acquired. Moreover, the incremental construction offers insights into the trade-off between formula complexity and classification accuracy. These works present important advances but do not meet the same needs as our framework in the field of OL. Kafka-ML allows us to register and configure different ML models flexibly, their training and dynamic deployment for inference, together with data stream integration, for both traditional and incremental online settings. In this way, our framework can also manage models that work with time series.

2.4. Reinforcement learning models

In [18], a systematic incremental learning method is presented for Reinforcement Learning (RL) in continuous spaces where the learning environment is dynamic. RL is a subset of ML where an agent interacts with an environment, learning to perform actions that maximize rewards. It involves trial and error, with the agent adjusting its strategies based on feedback. On the other hand, incremental learning or OL involves continuously updating a model knowledge with new data while retaining previous knowledge. It adapts to changing patterns over time, refining its capabilities without starting from scratch. So both methods enhance a model performance: OL refines existing knowledge, while RL trains agents to make optimal decisions in dynamic environments [19]. Then, the goal of the paper is to adjust the previously learned policy in the original environment to a new one incrementally whenever the environment changes. To improve the adaptability to the ever-changing environment, the authors propose a two-step solution incorporated with the incremental learning procedure: policy relaxation and importance weighting. On their side, authors from [20] mine the task patterns from a large volume of historical resource allocation data and propose a RL model termed IRDA to learn the allocation strategy incrementally. They observe that historical allocation data are not identically distributed, so, to improve the learning efficiency, they partition the whole historical allocation big dataset into multi-batch datasets, which forces the agent to continuously explore and learn on the distinct state spaces. IRDA reuses the strategy learned from the previous batch dataset and adapts it to the learning in the next batch dataset, to incrementally learn from multi-batch datasets and improve the allocation strategy. Ezenkwu et al. [21] seek to overcome the dynamic goal or environment limitations of traditional RL through a task-agnostic, self-organizing autonomous agent framework.

The proposed algorithm is a hybrid of TMGWR (temporospatial merge grow when required) for self-adaptive learning of sensorimotor maps and value iteration for goal-directed planning. A new sensorimotor-link update rule is presented in the article to enable the adaptation of the sensorimotor map to new experiences. These works contain significant contributions to RL, but this field has an important challenge which is dealing with learning from limited samples in the real system. Model-based RL seems especially promising to address the issue of sample efficiency, and it can also help address off-policy evaluation, robustness, and explainability [22]. Our Kafka-ML framework covers this need and could help in this particular aspect when working with ML models in the field of RL.

2.5. Models addressing catastrophic forgetting

An important point in OL is that current state-of-the-art incremental learning methods require a long time to train the model whenever new classes are added, and none of them takes into consideration the new observations of old classes. Therefore, in [23], they propose an incremental learning framework that can work in the challenging online learning scenario and handle both new class data and new observations of old classes. They address the catastrophic forgetting problem in online mode by introducing a modified cross-distillation loss together with a two-step learning technique. Also in [24], the authors propose a learning method to optimize algorithms for mitigating catastrophic forgetting. Instead of trying to formulate a new constraint function themselves, they propose to train another neural network to predict parameter update steps that respect the importance of parameters to the previous tasks. In the proposed meta-training scheme, the update predictor is trained to minimize the loss on a combination of current and past tasks. However, these projects do not present an efficient integration between ML models with data streams in an online setting such as Kafka-ML. This would make it possible not to rely on a database in which all training information has to be stored. Currently, our framework does not address the problem of catastrophic forgetting, but we want to provide a possible solution in the future.

Kafka-ML is presented as a novel open-source framework that allows the flexible management and deployment of ML models throughout their entire lifecycle, from implementation to training, and finally inference with data streams. Apart from this, our platform has a number of advantages compared to all these related works, which are: (1) the continuous integration between ML models and data streams in a simple and fast way; (2) the ability to work with different neural network architectures applicable to different domains; (3) the fact of not having any kind of data storage system thanks to the support of Apache Kafka, thus making the platform lighter; (4) support for distributed neural networks and their later deployment on different nodes, as well as their easy configuration and start-up thanks to technologies such as Docker and Kubernetes; and finally, (5) support for incremental learning for both definite and indefinite in time of ML models and their subsequent dynamic and automatic upgrading based on performance improvements obtained in their validation phase.

3. Online training deployment in Kafka-ML

In this section, the process for performing an online deployment for training models incrementally in Kafka-ML is described, as well as its main technical aspects. Before deploying an incremental training job, several steps, part of the Kafka-ML pipeline, are carried out to manage the complete lifecycle of ML models. The first thing to do is to define the ML models in the framework (just by inserting the ML code in the Web UI). Once registered, users can create a configuration, which is used to group a set of models for training with the same data stream. Users can deploy configurations so that they perform the training of the respective models through data streams. When the training is finished, users can visualize and compare the metrics of the models with the help of different graphs. Finally, once the models are trained, users can deploy the ML models for inference and make them available to obtain predictions. Fig. 2 shows the complete pipeline of Kafka-ML.

As explained in previous works [7,25,26], the only thing that needs to be done to perform a training deployment is to configure a series of parameters, such as batch size, training and validation settings, and estimation of GPU memory usage, in the form dedicated to this task. The new online functionality has been divided into two possible scenarios: (1) one that considers performing incremental training of the models but in a time-limited way; and (2) one that aims to perform incremental training indefinitely over time. In the latter case, the function to perform automatic model deployments has been enabled if any user-specified metric has improved a certain defined value in the validation phase. In the meantime, the current deployment continues to receive training data to further improve its performance. This functionality is similar to the concept of EarlyStopping[4] in TensorFlow. In our case, instead of stopping the training when a particular metric does not improve, we keep receiving data and training the model, and if there is an improvement in the specified metric, we deploy the model with the knowledge acquired up to that point for further inference. This way, we always get an improved version, and it allows for a possible continuous refinement of the model performance with new data indefinitely. A possible example of a real-world use case for the time-limited incremental training option could be the development of a mobile monitoring IoT device within an industry where assembly and production times are limited in time, so the model would have a defined period of training time. An example for the unlimited training option over time could be a smart thermostat that automatically sets a temperature given the ambient temperature, relative humidity, time of day, and other measurements, and can learn the user indoor temperature preferences over time.

The time-bound training provides a controlled duration, setting a specific time limit for training and allowing for better planning and resource allocation. By having scheduled intervals, the model can be updated at fixed time points, ensuring it incorporates the most recent data. In turn, time-bound training can be useful when computational resources are limited, as it allows an efficient utilization of those resources. However, this approach can lead to potential underfitting or overfitting. If the time-bound training duration is too short, the model may not have sufficient time to learn complex patterns in the data, resulting in underfitting. On the other hand, if the duration is too long, overfitting to the training data may occur. It can also lead to loss of information because once the training duration ends, no new data arriving after that point can be utilized to update the model. On the contrary, indefinite training enables continual learning, allowing the model to learn from the most recent data continuously. It can adapt to changing patterns and trends over time. Using this technique, the model can avoid becoming stale or outdated since it stays up to date with new information. Because of this, the model may potentially perform better as time progresses. Nevertheless, this approach usually requires dedicated resources as the training process continues indefinitely. There may also be a risk of overfitting. If not carefully monitored, the model could become too specialized for the training data and perform poorly on new unseen data.

Regarding time-limited incremental training, it only adds a new parameter called *stream timeout*, which is the stream timeout while waiting for new data (in milliseconds). *60 000* has been assigned to the stream timeout as a default value. Fig. 3(a) shows the new form fields for the time-limited deployment parameters. In contrast, the scenario that aims to carry out incremental training indefinitely over time requires more parameters, which are: (1) *monitoring metric*, which is the metric to monitor for indefinite training, as we need one metric to perform automatic deployments based on its performance. The metric

[4] https://www.tensorflow.org/api_docs/python/tf/keras/callbacks.

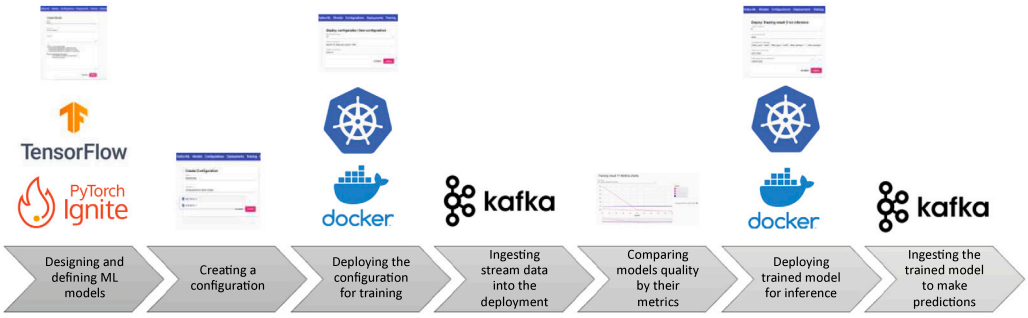


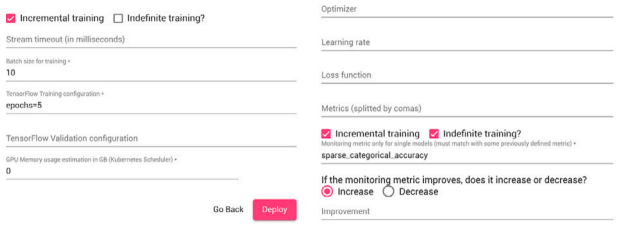
Fig. 2. Complete pipeline of Kafka-ML.

specified in this field must match a metric previously defined in the model by the user. In the case of distributed models (deep learning models distributed and partitioned along the Cloud-to-Things continuum, see [25]), the monitoring metric is considered the loss of the network as a whole for simplicity; (2) *change*, which is the direction in which the monitoring metric improves. Kafka-ML needs to know that because the user could define any specific metric; and finally, (3) *improvement*, which is how much the monitoring metric has to improve to carry out the automatic deployment of the model (*0.01* has been assigned to the improvement parameter as the default value).

On the other hand, users can also consider distributed deployment of deep learning models along the Cloud-to-Things continuum [25]. Partitioning deep neural networks involves splitting the model into multiple parts or sub-networks and distributing them in the continuum. Each node processes a subset of the data and contributes to the final prediction. This has several benefits such as scalability, since distributed deployment enables handling large datasets and models that may not fit on a single embedded device, allowing for scalable training and inference; low-latency processing, because partitioning the neural network allows fast and parallel computation across multiple nodes, leading to faster inference times and early exits; fault tolerance, since distributed setups can tolerate individual node failures, as the workload is distributed among multiple nodes. In this case, the system can still function with the remaining nodes, ensuring reliability; and reduced memory footprint, because partitioning enables distributing the memory requirements across multiple machines, potentially reducing the memory burden on each node. However, the distributed approach can lead to several issues that we have to take into account, which are an increased complexity, a possible communication overhead, network latency, and an imbalance load between the nodes [25].

As a result, the parameters for the deployment of distributed models have been made fully flexible. Previously, these variables were defined as constants in the training component. Now, several fields have been added to the form corresponding to these parameters so that the user can define them freely. The new fields are: *optimizer*, *learning rate*, *loss function*, and *metrics*, and their default values assigned are *adam* [27], *0.001*, *sparse_categorical_crossentropy*, and *sparse_categorical_accuracy*, respectively. Fig. 3(b) shows the new form fields for the indefinite training deployment and distributed model parameters.

As for the deployment of non-online learning tasks, if the user submits the form, a Kubernetes training component will be deployed for each model. However, the difference here is that the training phase will start as soon as the deployed models are loaded in the component because now they do not have to wait for a full data stream to be received (as in the original version of Kafka-ML), but are trained in small batches as the data arrive. In this way, the treatment of data streams in Kafka-ML has changed slightly. Two Kafka topics are still maintained with the same functions, the first being the data topic itself, which only contains training and evaluation data streams for the training and evaluation phases, and the second is the control topic, which is used to allow the deployed ML models to use the data topics. The only difference now is that the control message is sent



(a) Time-limited training (b) Indefinite training and distributed model parameters

Fig. 3. Kafka-ML Web UI deployment fields for incremental learning.

once the model is ready to be trained and not after a complete dataset is received. Therefore, the next step to perform once the models are deployed is to send the data streams (training dataset) incrementally. To facilitate the use of this functionality, we have created a simple Python script in the *Examples* of the Kafka-ML GitHub repository[5] that sends data in small batches over a predefined time to simulate their arrival, and as soon as the data start coming in, the model begins its training.

Finally, once the models start training, users can monitor the results of the defined metrics that are updated as the training progresses from the Training tab of Kafka-ML, for both training and validation. In turn, this tab is where the user will be able to visualize the automatic deployments of the models as they obtain improvements in their validation metrics. In addition, the user can also visualize the model metrics curves in real-time through their representation in graphs, just by selecting the *Chart* button on the web interface corresponding to each model. Fig. 4 shows an example of the *Visualization* tab, where users will be able to observe the behavior of the model metrics during their training and validation.

4. Kafka-ML architecture for online learning

In this work, the open-source Kafka-ML architecture has been redefined to enable the management of online deployments, enabling ML models to train indefinitely over time. The Kafka-ML structure employs a microservice architecture, consisting of multiple components designed with a single responsibility. These components are packaged as Docker containers, enabling enhanced isolation and flexibility. Kubernetes manages the deployment of Kafka-ML and its components, as well as node clusters for distributed and production infrastructures. Kafka-ML is an open-source project with its implementation, configurations, Kubernetes deployment files, and examples available on our GitHub repository referenced before.

On the one hand, the back-end component manages all the Kafka-ML information and deploys the necessary components in Kubernetes

[5] https://github.com/ertis-research/kafka-ml.

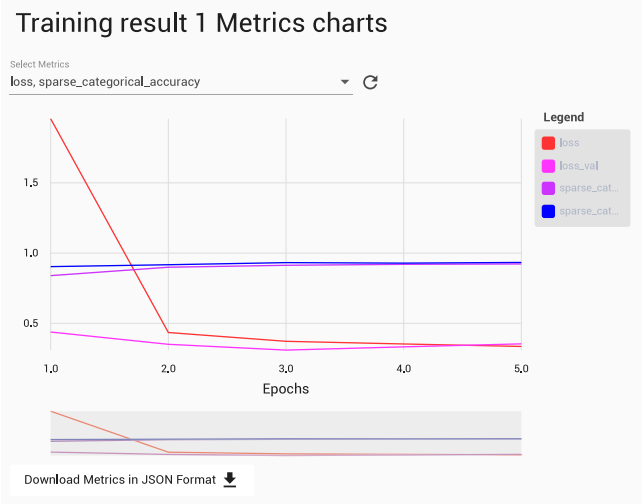


Fig. 4. Metrics display in Kafka-ML.

when requested (e.g., training jobs). The main changes incorporated in the back-end are those aimed at supporting the online deployment of ML models, mostly affecting the definition of models and deployments, as well as the new functions developed to perform the online training phase of the models. Likewise, the front-end component continues to provide the web interface necessary for users to access Kafka-ML and all its functionalities. This component uses the REST API defined in the back-end and the main changes included in this new version are those aimed at completing the dedicated forms in the framework pipeline to handle and support online deployments of ML models.

On the other hand, the inference component remains the same for both single and distributed models, as the online deployment does not affect this phase of the pipeline. This component continues to leverage the work of the replication controller of Kafka-ML to ensure that a given number of replicas are running at the same time, thus allowing load balancing between them and high availability of the component. Kafka-ML provides fine-control data streaming management for training and inference phases. The Kafka topic is the base element for connecting data sources to Kafka-ML tasks. Fault tolerance and load balancing capabilities can be controlled through topic partitions to ensure the availability of information sent through Kafka. The management of topics within Kafka's architecture is handled by a cluster of Kafka brokers. These brokers have the important responsibility of receiving a continuous stream of data from various data sources and then distributing them to subscribing consumers, such as training and inference tasks. The supported data formats are RAW, JSON, and AVRO for data streams sent through Kafka. In the case of online training, instead of sending the control message after the training data stream is loaded and ready for consumption, it is sent just before starting to receive the first data, since with this method, small batches of data will be received continuously. Regardless of this, Kafka's ability for the same online data stream to be used by different training tasks is still available.

4.1. Online training ML models

A refactoring of the entire training component was made, and new functions were implemented to enable the possibility of training the models incrementally over time. This is based on the incremental dataset KafkaBatchIODataset[6] from TensorFlow, which represents a streaming batch dataset from Kafka using consumer groups. It includes some of the parameters mentioned in the previous section, which are related to the incremental deployments of the models. Those are *topics*, which specifies the Kafka topics from which to consume data; *group_id*, which is the ID of the Kafka consumer group; *servers*, a list of bootstrap servers for the connection to Kafka; *stream_timeout*, an optional timeout value (in milliseconds) to wait for the new messages from Kafka to be retrieved by the consumers; and *message_poll_timeout*, an optional timeout duration (in milliseconds) after which the Kafka consumer throws a timeout error while fetching a single message. This value also represents the intervals at which the Kafka topic or topics are polled for new messages while using the stream_timeout; *configuration*, an optional dictionary containing some possible configurations, such as *enable.auto.commit=false*, *heartbeat.interval.ms=2000*, etc.; and *internal*, a Boolean variable referring to whether the dataset is being created from within the named scope. Once a configuration that encompasses a series of models is deployed for subsequent joint training with the same data stream, a Kubernetes job is run for each of those models in its entirety and containerized into a Docker container. Fig. 5 shows the sequence diagram of the online training process without a time limit, so it considers the automatic deployments of the model if improvements are achieved in its performance. Note that some steps, such as management of exceptions and data stream decoding, have not been included for simplicity. The first thing that Kafka-ML does is to deploy the jobs of each of the models included in the configuration (step 1). These jobs, when started, download their corresponding models from the back-end (2–4). Then, the jobs start receiving control messages until they receive the one they are waiting for (6), i.e., it matches the deployment_id received. The control messages no longer contain the exact position of the data corresponding to the deployment, since we will have a continuous data stream indefinitely over time. What it does maintain is the percentage of the data to be used for validation, which is defined by the user. Although the data stream is continuous, in each iteration, a subset of the data is used and divided into training and validation based on that parameter (*validation_rate*).

Algorithm 1 describes the complete training process for the indefinite deployment. Some preprocessing and parameters have been omitted for readability. Once the models are loaded and the control message has been received, the jobs are ready to start training. Therefore, the user can start sending streaming batches of data for training and validation (7). As the model is incrementally trained, metric results are sent to the back-end at the end of each epoch (11). Finally, if improvements are obtained in the monitoring metric specified by the user (greater than the improvement parameter defined), a request is made to the back-end with the information of the current model to carry out an automatic upgrade of a new model for further inference. In this way, we leave the current job receiving new training data indefinitely over time to obtain further improvements in its performance (12–14). It is worth noting that no additional resources are needed to achieve the model update, as the moment the improvement is detected, the current model weights are simply sent to the back-end, and the model is automatically deployed. As a recap, our online model update algorithm checks at the end of each mini-training batch that there has been such an improvement in the validation of the monitoring metric. If there is no such improvement, training is continued. But if there is, apart from continuing the training of the base model, its current weights are sent to the back-end for further deployment as a new model. From here, the user can use that model to predict data, download it for embedding in an IoT device, etc. Fig. 6 shows the flowchart of the developed method for the indefinite training.

On the other hand, in the case of training that is not indefinite in time, steps 1 to 11 of Algorithm 1 remain the same, and only the last part would change. In this way, the model continues to receive data and is trained until the time limit defined by the stream timeout without receiving data is reached. Once this last condition is met, the job associated with the training ends, and the model is deployed for further inference.

[6] https://tensorflow.org/io/api_docs/python/tfio/experimental/streaming/KafkaBatchIODataset.

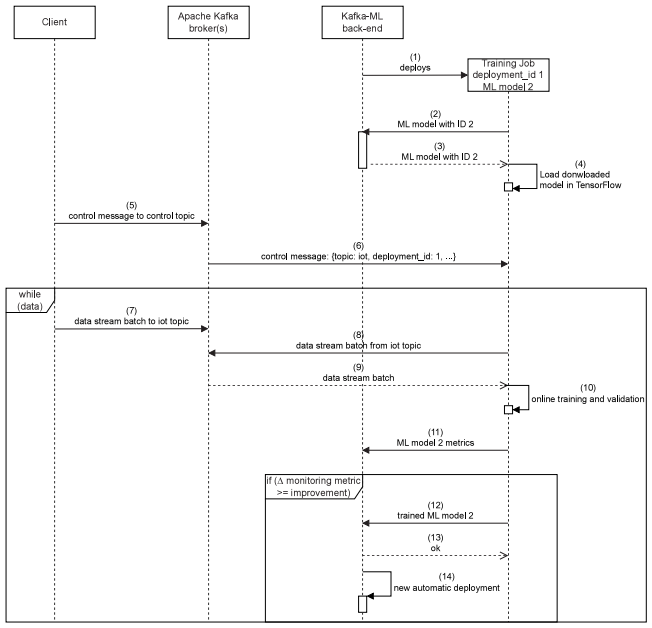


Fig. 5. Sequence diagram of the indefinite online training process in Kafka-ML.

Algorithm 1 Workflow algorithm of indefinite training

```
result_id ⇐ deployTrainingJob(deployment_id, model_id)
sendControlMessage(topic, deployment_id)
if indefinite == True then
    while data do
        dataset ⇐ receiveDataBatch(topic)
        training_data, validation_data ⇐ splitDataset(dataset, validation_rate)
        metrics ⇐ onlineTraining(training_data, validation_data)
        updateEpochMetrics(result_id, metrics)
        if Δ monitoring_metric >= improvement then
            sendModel(result_id, metrics)
            automaticDeployment(metrics)
        end if
    end while
end if
```

5. Evaluation

The Kafka-ML OL capability presented in this paper allows users to deploy and train their machine, distributed, and deep learning incrementally over time. For the evaluation of this feature, we have defined different scenarios intending to compare the results obtained by performing standard training and incremental training deployments. The parameters to be compared are the accuracy of the models in the validation phase and the training times of both approaches.

Moreover, five different scenarios have been considered. The first scenario involves the training of deep learning models with a specific time. In this case, the precision and recall of the models will also be assessed in their validation phase. The Kafka-ML framework is flexible to the use of metrics. The second scenario involves the training of distributed models with a specific time, and the third scenario carries out indefinite incremental training to evaluate the Kafka-ML ability to perform automatic upgrades of models as the monitoring metric improves over time. The fourth involves incremental training of a ML model from a real industry use case. Lastly, the last scenario evaluates incremental and traditional learning paradigms when the models continuously receive the same data distribution over time, comparing both results.

Distributed models consist of a set of neural network models that together, and interconnected with inputs and outputs, form a larger global network. These ML structures are characterized by the possibility of the incorporation of early exits [28] in the connections between the sub-models. In this way, they allow faster predictions to be obtained that are only processed through a part of the network. This is especially suitable for critical systems where predictions need to be made in a short time. The models used in the first three scenarios are the well-known VGG16[7] and AlexNet,[8] both in their individual and distributed architectures. In the latter case, Edge-Fog-Cloud partitioning has been considered [25]. The dataset used to evaluate these models was the CIFAR-10 [29], which has a total of 60 000 images with a size of 32 × 32 in color format and ten classes, using 90% of the data for training and the remaining 10% for validation in RAW format. Currently, Kafka-ML supports RAW format, ideal for single-input data streams that might necessitate reshaping, such as images. Additionally, it supports Apache Avro [30], which is suitable for intricate and multi-input datasets, where a schema dictates the decoding process of the data stream, facilitating the serialization and deserialization of data streams. The training configuration used was 256 as batch size, so we trade off training time, memory usage, regularization, and accuracy; the Adam optimizer [31], thanks to its faster convergence by adapting the learning rate during training; a learning rate of 0.001, the common default value so the optimizer would update the parameters just right to reach the local minima; and a total of 50 epochs, due to the size and complexity of the dataset, and the architecture and capacity of the networks.

The tests have been carried out on our on-premises cluster of 7 state-of-the-art servers. Each machine has an Intel(R) Xeon(R) Gold 6230R CPU with two NVIDIA(R) Tesla(R) V100 GPUs as well as 384 GB of RAM. Each one of the seven machines runs Kubernetes v1.21.6 and Docker 20.10.8 on top of Ubuntu 20.04.3 LTS. A Kubernetes master was deployed in one node, while the remaining six are Kubernetes workers. One of the machines runs a virtual machine with identical software characteristics, and this one is enabled as a Kubernetes master, while the rest of the machines are Kubernetes workers.

The streaming data sent divides the training data set into equivalent parts (ten thousand data each in this case), and they are sent one by one, progressively leaving intervals of 30 s between them to simulate an interrupted but continuous arrival of data in time. Algorithm 2 describes the data-sending process for incremental training. The client that sent the information and where the results were measured was a PC with an Intel(R) Core(TM) i9-10900K CPU and 64 GB of RAM.

Algorithm 2 Data sending process for incremental training

```
dataset ⇐ OnlineRawSink(boostrap_servers, topic, deployment_id, description, validation_rate)
(x_train, y_train) ⇐ load_data()
subSets ⇐ split_Dataset(x_train, y_train, size)
for s ∈ subSets do
    for (x, y) ∈ s do
        dataset.send(data=x, label=y)
    end for
     sleep(time)
end for
dataset.online_close()
```

[7] https://www.kaggle.com/code/blurredmachine/vggnet-16-architecture-a-complete-guide/notebook.

[8] https://www.kaggle.com/code/blurredmachine/alexnet-architecture-a-complete-guide.

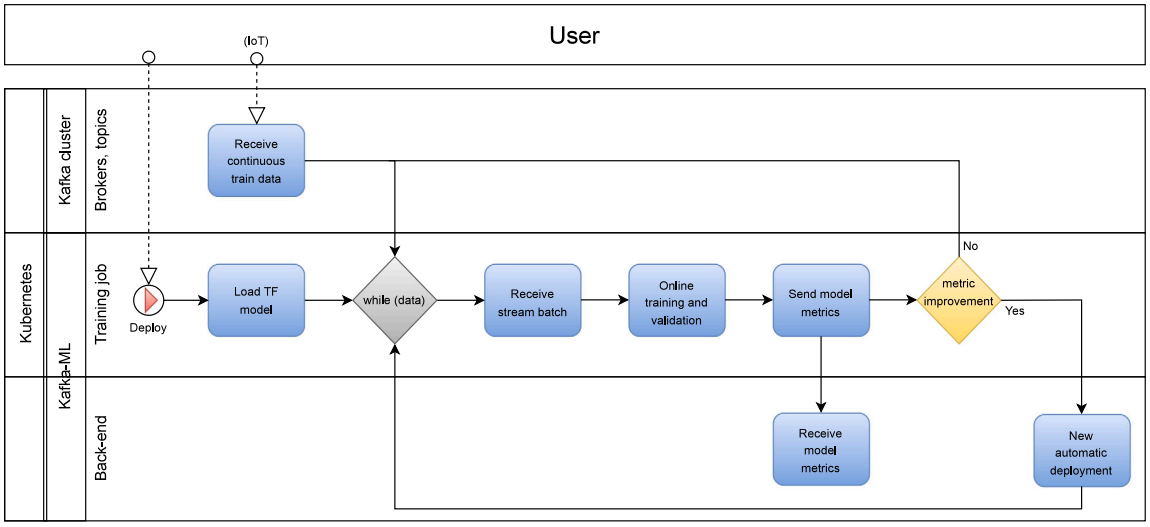


Fig. 6. Indefinite training flowchart.

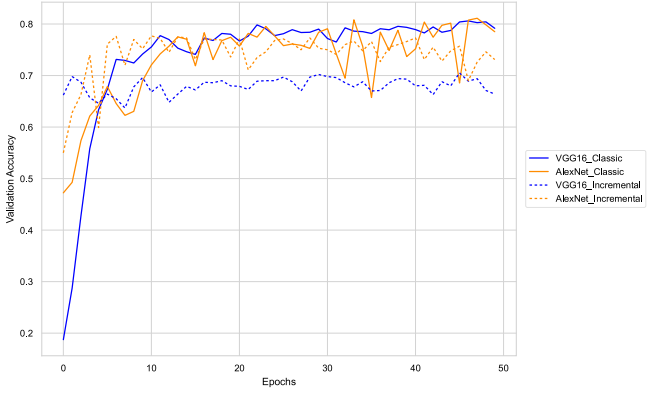


Fig. 7. Single models validation accuracy.

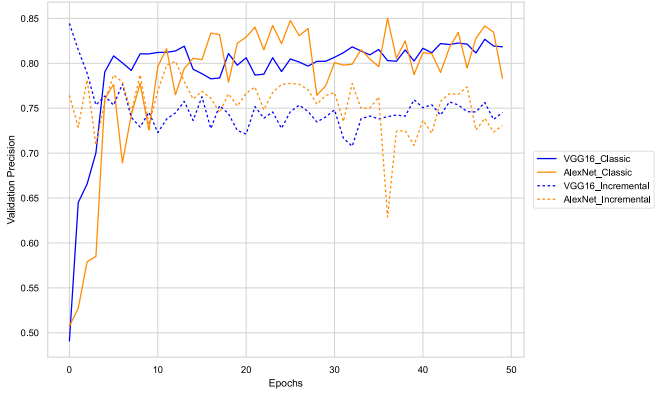


Fig. 8. Single models validation precision.

5.1. Single models

This test compares the validation accuracy, precision, and recall of the two models using the traditional training method and the incremental method. These metrics have been chosen in this test to demonstrate different evaluation options. The aim is to find out which learning method performs best when we have a limited dataset over time.

The progression of the validation accuracies of the single models is shown in Fig. 7. As can be seen, both models together with both approaches achieve values of around 0.7 and 0.8 for accuracy. For the incremental training case, lower values are obtained than in the classical training case, but they are quite close.

On the other hand, the validation precision obtained by the models with both training approaches is shown in Fig. 8. As the graph shows, we have a similar behavior as before, with values close to 0.75 for incremental deployments and 0.8, for classical deployments.

Finally, the validation recall of both models is shown in Fig. 9 using the two training procedures. In this case, values between 0.7 and 0.8 are again obtained for both types of deployment, leaving a very small difference between them.

The comparison of the training times between the two approaches is shown in Fig. 10. On the one hand, it can be seen that the times obtained from incremental training are a bit higher than those obtained in classical training. This is normal since the data are sent in an interrupted way in time, and also because the incremental approach is training with small batches in each iteration and not with the whole dataset, so it takes longer to complete its training. On the other hand, it can be seen that the AlexNet model is a bit faster than the VGG16 model.

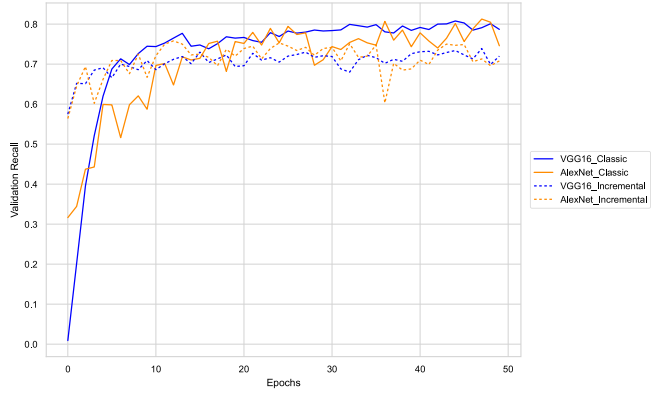


Fig. 9. Single models validation recall.

5.2. Distributed models

This test aims to compare the progression of the validation accuracy of the two partitioned models by applying the traditional and incremental training methods. The purpose of this test is the same as the previous one but in this case evaluating the models in a distributed way.

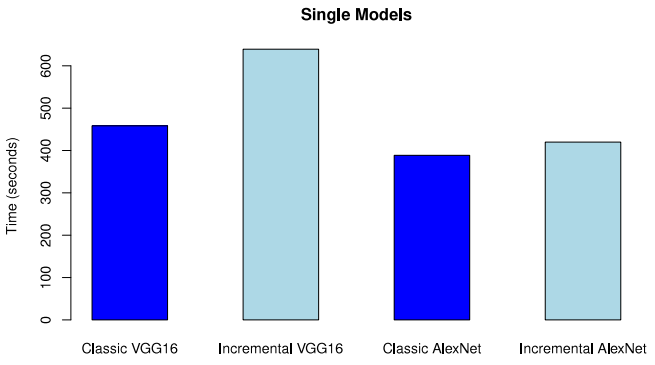


Fig. 10. Single models training time.

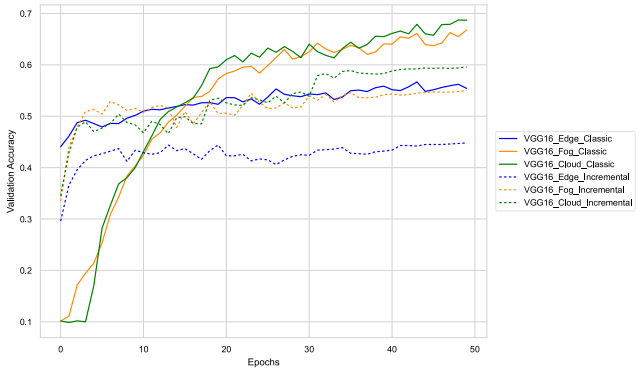


Fig. 11. Distributed VGG16 validation accuracy.

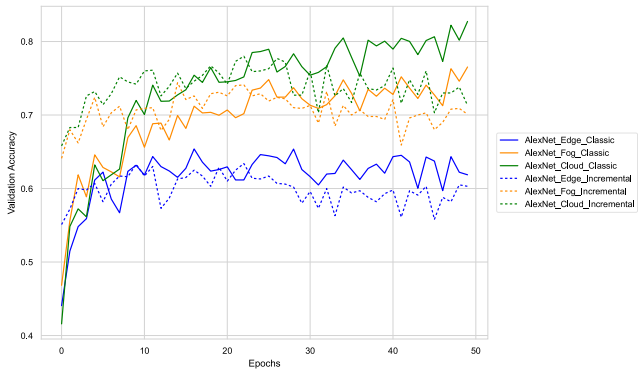


Fig. 12. Distributed AlexNet validation accuracy.

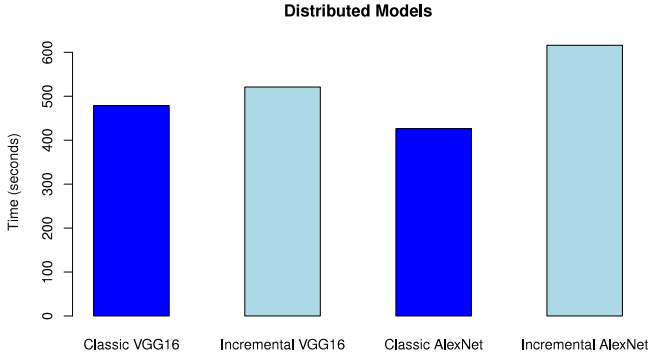


Fig. 13. Distributed models training time.

The succession of the validation accuracies of the partitioned VGG16 model can be seen in Fig. 11. Although the results are around 0.6, the accuracy in both approaches has an upward trend over time. As we go up the distributed chain, better results for accuracy are obtained. This is normal since, in the overall computation, we have more layers of the network. On the other hand, comparing each level separately for both approaches, we can see that the classical training scenario obtains slightly better results. As discussed in previous sections, for not very large datasets and in the case of having a moderately powerful computing infrastructure, the classical approach will normally obtain better results. However, if these assumptions were not met, the incremental approach would be more appropriate and better suited to the problem.

The progression of the validation accuracies of the partitioned AlexNet model is shown in Fig. 12. As can be seen, the accuracy values are around 0.7. As before, we note that when comparing the different levels of the partitions with each other and both approaches in general, the same behavior is obtained as in the VGG16 model.

The comparison of the training times between the two approaches is shown in Fig. 13. As can be seen, the same happens in the case of the unpartitioned models. On the one hand, training the VGG16 model takes longer than training the AlexNet model, and on the other hand, we see that the incremental approach still takes longer than the classical approach for the same reason of having to train with small batches of data in each iteration and the interrupted sending of data. However, the time difference is small, and in case we had a dynamic environment and data were continuously arriving over time, the incremental approach would allow the model to adapt to it in real-time and better results in terms of prediction confidence would be obtained.

5.3. Indefinite training

This section aims to demonstrate the Kafka-ML ability to perform automatic upgrades of ML models over time for subsequent inference when improvements in its validation metrics are obtained during training. The purpose of this test is to show the continuous refinement of the ML models performance as new data arrives indefinitely. In this case, the VGG16 and AlexNet models are again used in their single versions, as well as the dataset and the deployment parameters. Three new variables now come into play compared to the previous tests. These are the monitoring metric, which in this test has been set to *sparse_categorical_accuracy* as it is one of the most common metrics for assessing the performance of models; the direction in which that metric improves, being upwards for accuracy; and finally, the improvement that the metric has to have to carry out the automatic deployment of the models, which has been set to *0.05*. Fig. 14 shows the Kafka-ML training results screen with the automatically deployed models, each with their training metrics, validation metrics, and deployment times. In this way, together with the metrics visualization window (Fig. 4) and the tools provided by Kubernetes for monitoring their working containers, we can perfectly keep track of the state of our framework as a whole. Table 1 also summarizes and compares these results for correct reading.

Table 1
Incremental results comparative.

Model	Accuracy (val)	Deployment time (s)
1st AlexNet	0.646	120
1st VGG16	0.579	141
2nd AlexNet	0.715	203
2nd VGG16	0.734	234

As can be seen in Table 1, the AlexNet model performs its first deployment at 120 s, with a value for the validation accuracy of 0.646. From here, at 203 s, it performs its second deployment, moving to an accuracy value of 0.715, leaving between both deployments a difference for the metric of 0.069, which exceeds the limit set for the test. On the other hand, the VGG16 model takes slightly longer

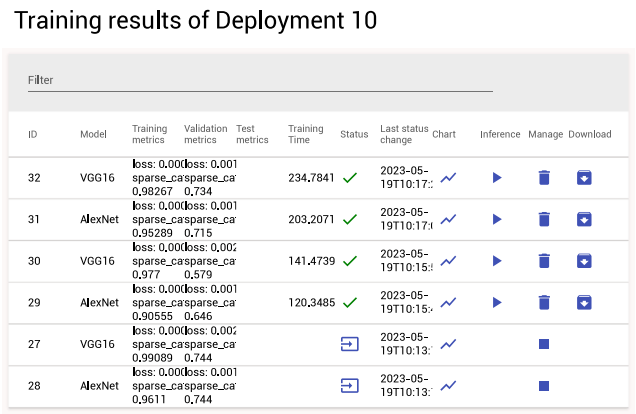
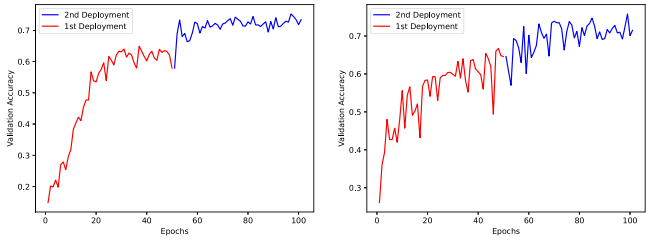


Fig. 14. Incremental results automatically deployed in Kafka-ML.



(a) VGG16 (b) AlexNet

Fig. 15. Incremental validation accuracy in automatic deployments.

to perform its automatic deployments, taking 141 s for the first one with a value for validation accuracy of 0.579. Its second deployment takes 234 s, with an accuracy value of 0.734, showing a difference of 0.155. This demonstrates the ability of the framework to carry out dynamic and automatic upgrades. In the meantime, initial deployments of these models are still awaiting data to keep improving their metrics during their training indefinitely over time. A better visualization of the behavior of the validation metrics can be seen in Fig. 15(a) for VGG16 and Fig. 15(b) for AlexNet, in which the first deployment is painted in red and the second in blue, both trained 50 epochs.

5.4. Petrochemical industry use case

This section presents the results of the evaluation of a model applied to a real use case of Industry 4.0. Specifically, it is a model developed for Cepsa, which is the largest Spanish petrochemical company. The objective of the model is to predict the freezing point of a lubricant produced by Cepsa based on different operating conditions and certain properties of the raw material they use. This freezing point is important for the production process, so the aim is to predict it in real-time to have better control over the production chain. The dataset used consists of a monitoring process which has several variables corresponding to different important parameters in the process of creating the lubricant in question. The idea for this test arose from the need to also evaluate the new incremental feature of our Kafka-ML framework using a regression problem, and in a Industrial use case. As before, the evaluation will have two training approaches, classical and incremental.

The neural network architecture used is a fully connected dense model. The parameters for the evaluation of the model, in this case, are different from those used in previous sections. The batch size has been set to 8, the chosen optimizer is Adam, the learning rate is 0.001, and the number of epochs has been 50. Then, the selected metrics are MAE (Mean Absolute Error), which serves as an easy-to-understand quantifiable measurement of errors for regression problems; MSE (Mean Squared Error), to assess the quality of the model predictions by measuring how closely they align with the ground truth; and MAPE (Mean Absolute Percentage Error), to measure forecast accuracy. In turn, when sending the data for incremental training, the dataset has been divided into six equal-sized parts and each part is sent progressively in RAW format, leaving a 30-second interval between each subset to simulate the continuous trickle of data.

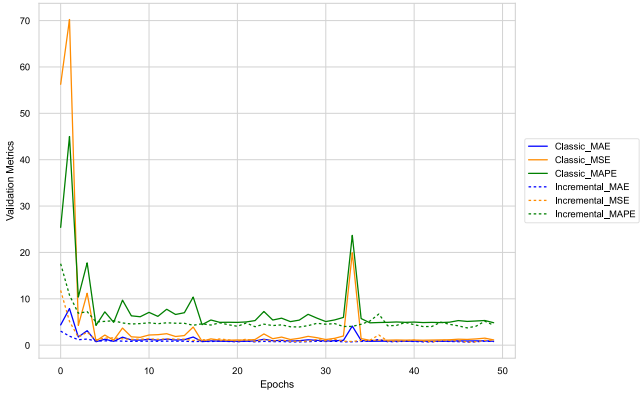


Fig. 16. Cepsa model validation metrics.

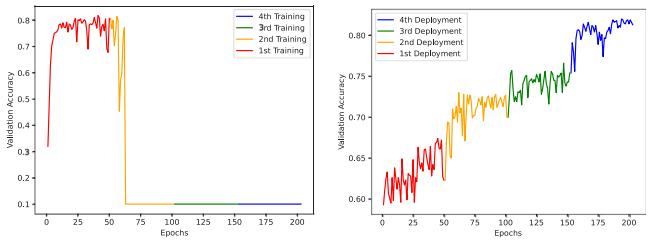
The comparison of the validation metrics, which all concern errors, between the two approaches, is shown in Fig. 16. As can be seen, they all have the same downward trend, approaching very close to zero and staying there as the epochs progress. This means that the model produces a very small error when predicting data that are outside its training set, so these are considered good results. On the other hand, the training times of both models were also compared, with the classical training taking 92.50 s and the incremental training taking 229.15 s.

5.5. Learning with the same data distribution over time

This section aims to study the improvement capacity of the incremental and traditional learning paradigms when working on data that have the same distribution over time. Therefore, the objective of this test is to find out the performance behavior of the models under the aforementioned assumption. The test is intended to assess the continuous arrival of data that share the same distribution, such as those captured by a sensor that is indefinitely collecting data within the domain under study.

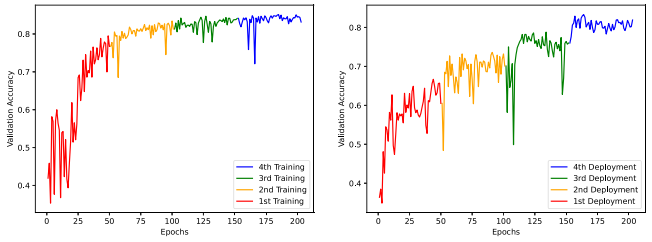
To compare the two approaches, normal and incremental deployments have been considered, and the same dataset has been sent repeatedly for training. As in the previous tests, the models used were VGG16 and AlexNet, the dataset was CIFAR-10, and the metric to be compared was also the accuracy in the validation phase. Specifically, the dataset was sent to the deployments a total of four times in a row, and in each iteration, the models were trained for 50 epochs. The difference in sending the data remains the same as in the previous sections, that is, for the traditional method, the entire dataset is sent at once, while for the incremental method, subsets of it are sent little by little.

The results of the VGG16 model for the traditional and the incremental approach are shown in Figs. 17(a) and 17(b), respectively. As can be seen, the traditional training method for this neural network architecture is not appropriate when repeatedly using the same dataset. From the second transmission of data for this model onwards, its validation accuracy drops off completely around 0.1, and this is because it overlearns the training dataset and is unable to extrapolate that knowledge to examples it has not seen yet. However, incremental training for this type of architecture does improve its validation accuracy, which is over 0.8, as it continues to receive data that have the same distribution over time. This shows how the model is able to extrapolate its knowledge for new inputs with the incremental approach over time.



(a) Traditional (b) Incremental

Fig. 17. VGG16 validation accuracy with repeated dataset.



(a) Traditional (b) Incremental

Fig. 18. AlexNet validation accuracy with repeated dataset.

On the other hand, the results obtained for the AlexNet model are shown in Fig. 18(a) for the traditional case and Fig. 18(b) for the incremental case. As can be seen, with this architecture, both training methods yield very similar results. The validation accuracies, which also reach values above 0.8, continue to increase as more data with the same distribution arrive and training epochs elapse. In the case of this model and with this continuous data delivery, it appears that the traditional approach manages, for the training time employed, to keep extrapolating the knowledge gained at this stage for those examples within the validation set that it has not studied yet. The behavior of both models under the traditional approach varies, most surely because they have completely different architectures, which directly influence the way these networks learn and the time needed to do so. On the other hand, in the case of incremental training, the AlexNet model has the same behavior as in past tests, and the model continues to improve.

6. Challenges and limitations of online training

OL offers advantages such as adaptability to changing data and the ability to handle large datasets, but it also has some limitations and challenges. The most important of these are set out below:

- **Catastrophic forgetting**: Neural networks trained incrementally are prone to catastrophic forgetting, which occurs when the network forgets previously learned information as it is trained on new data. This issue arises because updating the network with new data can overwrite the existing knowledge captured in the model's parameters [32].
- **Stability-plasticity dilemma**: The stability-plasticity dilemma refers to the trade-off between the stability of learned representations and the ability to learn new information. OL must strike a balance between preserving existing knowledge and accommodating new knowledge. It is challenging to maintain stability while adapting to new patterns [33].
- **Concept drift**: In many real-world scenarios, the underlying data distribution may change over time, known as concept drift. OL must be able to detect and adapt to these changes to maintain model performance. Failure to handle concept drift can lead to a significant degradation in the model's accuracy [12].
- **Computational complexity**: Incremental learning can be computationally expensive, especially for large-scale neural networks. Updating the model parameters with each new data point requires substantial computational resources and can slow down the learning process [34].
- **Data availability and labeling**: OL assumes a continuous stream of data, but obtaining labeled data in real time can be challenging. It may require additional mechanisms for data collection, labeling, and verification to ensure the accuracy and quality of the training data [35].
- **Balancing exploration and exploitation**: In OL, it is crucial to balance exploration (learning from new data) and exploitation (using the learned knowledge to make predictions). Determining the appropriate exploration–exploitation trade-off can be tricky as overly focusing on either can hinder the learning process [36].
- **Limited memory**: OL typically operates with limited memory as retaining an extensive history of past data can be impractical or infeasible. Deciding how much past data to store and utilize for training poses a challenge [37].

Most recent works on OL attempt to address these difficulties by using regularization techniques [38], incorporating memory mechanisms [39], developing adaptive learning algorithms [40], and designing strategies to handle concept drift [41], among others. Future work intends to address these challenges in the adoption of incremental learning with data streams.

7. Conclusions and future work

Learning algorithms in real-world applications face challenges like handling large data sets, continuous data streams, and changing data generation processes. Traditional learning methods may not be suitable for these environments due to their assumptions of stationary and independent data. To meet the requirements of the learning process, systems should be able to adapt both their structures and their parameters. Online learning is a powerful technique that enables models to adapt to changing data and improve their performance over time. Unlike batch learning, where all training examples are processed at once, online learning updates the weights and biases incrementally as new examples arrive, allowing the model to converge over time. This makes online learning especially useful in real-time applications where data are constantly changing or arriving in streams.

In this work, we have extended our Kafka-ML framework to provide an infrastructure for deploying and managing OL pipelines and models that can train indefinitely, allowing for continuous improvements in their performance. Kafka-ML enables models to learn from streaming data in real-time, making it useful in applications where data are continuously generated, such as IoT sensor networks. With Kafka-ML, developers can easily manage the lifecycle of online learning models, including monitoring, scaling, and updating, while ensuring the accuracy and consistency of the models over time. By supporting online learning, Kafka-ML provides a powerful framework for building adaptive and streaming applications that can evolve and improve with the data they are processing.

In this way, our framework allows automatic upgrading of ML models as they continue to be trained. As seen in Section 5, the results of the tests carried out in this online setting are close to the traditional learning approach, and show the effective capacity of Kafka-ML to carry out incremental training processes for different use cases. Therefore, given all the features, advantages and possibilities now offered by our open-source ML tool, this research and its result, together with the lack of similar platforms, represent a significant advance within this field of study.

Future work for Kafka-ML includes the support for federated incremental training of models and the capability for addressing the challenges mentioned in Section 6. To address these problems, the

following solutions have been envisaged: (1) the use of Elastic Weight Consolidation (EWC) to assign different levels of importance to various parameters and penalize more for changes in important parameters to preserve knowledge; (2) utilize adaptive learning rate methods to dynamically adjust the learning rate according to the importance of parameters; (3) investigate and implement neural network architectures that dynamically adjust their structure based on the task complexity; (4) use ensemble methods to combine the predictions of several models, which provides a more robust performance against conceptual drift; and (5) leverage both labeled and unlabeled data to enhance model training, especially in scenarios where labeled data is scarce. Also, the support of new ML frameworks, such as Paddle[9] and Jax.[10] To support the upcoming frameworks it would be necessary to develop a new executor validator to check the implementation of the models, the corresponding new training module as well as the inference module for each framework. On the other hand, we also considered the option of supporting KSQL or Kafka Streams for the pre-processing of our system data. This way, user could control the window he/she wants to use, he/she can define functions and apply them to the different data streams, etc.

CRediT authorship contribution statement

Alejandro Carnero: Writing – original draft, Validation, Software, Investigation, Data curation, Conceptualization. **Cristian Martín:** Writing – review & editing, Supervision, Investigation, Funding acquisition, Conceptualization. **Gwanggil Jeon:** Writing – review & editing, Supervision. **Manuel Díaz:** Writing – review & editing, Supervision, Project administration, Investigation, Funding acquisition, Conceptualization.

Declaration of competing interest

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

Data availability

Data will be made available on request.

Acknowledgments

This work is funded by the Spanish projects Grant TSI-063000-2021-116 ('5G+TACTILE_2: Digital vertical twins for B5G/6G networks') funded by MICIU/AEI/10.13039/501100011033/ and by 'European Union NextGenerationEU/PRTR', Grant TED2021-130167B ('GEDIER: Application of Digital Twins to more sustainable irrigated farms'), funded by MICIU/AEI/10.13039/501100011033/ and by 'European Union NextGenerationEU/PRTR', Grant CPP2021-009032 ('ZeroVision: Enabling Zero impact wastewater treatment through Computer Vision and Federated AI') funded by MICIU/AEI/10.13039/501100011033/ and by 'European Union NextGenerationEU/PRTR', and Grant PID2022-141705OB-C21 ('DiTaS: A framework for agnostic compositional and cognitive digital twin services') funded by MICIU/AEI/10.13039/501100011033/ and by 'FEDER'. This project has received funding from the European Union's Horizon Europe research and innovation programme under the Marie Skłodowska-Curie grant agreement EVOLVE No 101086218. Funding for open access charge: Universidad de Málaga/CBUA.

[9] https://github.com/PaddlePaddle/Paddle.

[10] https://github.com/google/jax.

References

[1] D. Saad, On-line learning in neural networks, J. Amer. Statist. Assoc. 95 (2000) http://dx.doi.org/10.2307/2669811.

[2] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, C. Liu, A survey on deep transfer learning, in: V. Kůrková, Y. Manolopoulos, B. Hammer, L. Iliadis, I. Maglogiannis (Eds.), Artificial Neural Networks and Machine Learning – ICANN 2018, Springer International Publishing, Cham, 2018, pp. 270–279.

[3] L.C. Jain, M. Seera, C.P. Lim, A review of online learning in supervised neural networks, Neural Comput. Appl. 25 (2014) http://dx.doi.org/10.1007/s00521-013-1534-4.

[4] D.R. Wilson, T.R. Martinez, The general inefficiency of batch training for gradient descent learning, Neural Netw. 16 (10) (2003) 1429–1451, http://dx.doi.org/10.1016/S0893-6080(03)00138-2.

[5] B. Pérez Sánchez, O. Fontenla-Romero, B. Guijarro-Berdiñas, A review of adaptive online learning for artificial neural networks, Artif. Intell. Rev. 49 (2018) http://dx.doi.org/10.1007/s10462-016-9526-2.

[6] J. Wang, A. Belatreche, L. Maguire, M. McGinnity, Online versus offline learning for spiking neural networks: A review and new strategies, in: 2010 IEEE 9th International Conference on Cyberntic Intelligent Systems, 2010, pp. 1–6, http://dx.doi.org/10.1109/UKRICIS.2010.5898113.

[7] C. Martín, P. Langendoerfer, P.S. Zarrin, M. Díaz, B. Rubio, Kafka-ML: Connecting the data stream with ML/AI frameworks, Future Gener. Comput. Syst. 126 (2022) 15–33, http://dx.doi.org/10.1016/j.future.2021.07.037, URL https://www.sciencedirect.com/science/article/pii/S0167739X21002995.

[8] Apache samoa, 2021, Available online: https://samoa.incubator.apache.org/. (Accessed 13 May 2021).

[9] Kubeflow, 2021, Available online: https://www.kubeflow.org/. (Accessed 13 May 2021).

[10] D. Sahoo, Q. Pham, J. Lu, S.C.H. Hoi, Online deep learning: Learning deep neural networks on the fly, 2017, http://dx.doi.org/10.48550/ARXIV.1711.03705, URL https://arxiv.org/abs/1711.03705.

[11] D. Le, M. Greene, W. Makumi, W. Dixon, Real-time modular deep neural network-based adaptive control of nonlinear systems, IEEE Control Syst. Lett. PP (2021) 1, http://dx.doi.org/10.1109/LCSYS.2021.3081361.

[12] H. Guo, S. Zhang, W. Wang, Selective ensemble-based online adaptive deep neural networks for streaming data with concept drift, Neural Netw. 142 (2021) 437–456, http://dx.doi.org/10.1016/j.neunet.2021.06.027, URL https://www.sciencedirect.com/science/article/pii/S0893608021002598.

[13] B. Bejeck, Kafka Streams in Action: Real-Time Apps and Microservices with the Kafka Streams API, Simon and Schuster, 2018.

[14] A. Besedin, P. Blanchart, M. Crucianu, M. Ferecatu, Evolutive deep models for online learning on datastreams with no storage, in: IOTSTREAMING@PKDD/ECML, 2017.

[15] Y. Qian, L. Pan, Variety-aware GAN and online learning augmented self-training model for knowledge graph entity alignment, Inf. Process. Manage. 60 (5) (2023) 103472, http://dx.doi.org/10.1016/j.ipm.2023.103472, URL https://www.sciencedirect.com/science/article/pii/S0306457323002091.

[16] T. Ergen, S.S. Kozat, Efficient online learning algorithms based on LSTM neural networks, IEEE Trans. Neural Netw. Learn. Syst. 29 (8) (2018) 3772–3783, http://dx.doi.org/10.1109/TNNLS.2017.2741598.

[17] G. Bombara, C.A. Belta, Online learning of temporal logic formulae for signal classification, in: 2018 European Control Conference, ECC, 2018, pp. 2057–2062.

[18] Z. Wang, H.-X. Li, C. Chen, Incremental reinforcement learning in continuous spaces via policy relaxation and importance weighting, IEEE Trans. Neural Netw. Learn. Syst. 31 (6) (2020) 1870–1883, http://dx.doi.org/10.1109/TNNLS.2019.2927320.

[19] S. Bose, M. Huber, Incremental learning of neural network classifiers using reinforcement learning, in: 2016 IEEE International Conference on Systems, Man, and Cybernetics, SMC, 2016, pp. 002097–002103, http://dx.doi.org/10.1109/SMC.2016.7844549.

[20] J. Wang, S. Wang, IRDA: Incremental reinforcement learning for dynamic resource allocation, IEEE Trans. Big Data PP (2020) http://dx.doi.org/10.1109/TBDATA.2020.2988273.

[21] C. Ezenkwu, A. Starkey, An unsupervised autonomous learning framework for goal-directed behaviours in dynamic contexts, Adv. Comput. Intell. 2 (2022) http://dx.doi.org/10.1007/s43674-022-00037-9.

[22] G. Dulac-Arnold, D. Mankowitz, T. Hester, Challenges of real-world reinforcement learning, 2019, arXiv:1904.12901.

[23] J. He, R. Mao, Z. Shao, F. Zhu, Incremental learning in online scenario, 2020.

[24] R. Vuorio, D.-Y. Cho, D. Kim, J. Kim, Meta continual learning, 2018, http://dx.doi.org/10.48550/ARXIV.1806.06928, URL https://arxiv.org/abs/1806.06928.

[25] A. Carnero, C. Martín, D.R. Torres, D. Garrido, M. Díaz, B. Rubio, Managing and deploying distributed and deep neural models through kafka-ML in the cloud-to-things continuum, IEEE Access 9 (2021) 125478–125495, http://dx.doi.org/10.1109/ACCESS.2021.3110291.

[26] A.J. Chaves, C. Martín, M. Díaz, The orchestration of machine learning frameworks with data streams and GPU acceleration in kafka-ML: A deep-learning performance comparative, Expert Syst. e13287, http://dx.doi.org/10.1111/exsy.13287, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/exsy.13287.

[27] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, http://dx.doi.org/10.48550/ARXIV.1412.6980, URL https://arxiv.org/abs/1412.6980.
[28] D.R. Torres, C. Martín, B. Rubio, M. Díaz, An open source framework based on kafka-ML for distributed DNN inference over the cloud-to-things continuum, J. Syst. Archit. 118 (2021) 102214.
[29] A. Krizhevsky, Learning multiple layers of features from tiny images, 2009.
[30] D. Vohra, Apache avro, in: Practical Hadoop Ecosystem: A Definitive Guide to Hadoop-Related Frameworks and Tools, A Press, Berkeley, CA, 2016, pp. 303–323, http://dx.doi.org/10.1007/978-1-4842-2199-0_7.
[31] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2017, arXiv:1412.6980.
[32] R. Kemker, M. McClure, A. Abitino, T. Hayes, C. Kanan, Measuring catastrophic forgetting in neural networks, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 32, 2018.
[33] D. Kim, B. Han, On the stability-plasticity dilemma of class-incremental learning, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023, pp. 20196–20204.
[34] P.J. Freire, S. Srivallapanondh, A. Napoli, J.E. Prilepsky, S.K. Turitsyn, Computational complexity evaluation of neural network applications in signal processing, 2022, arXiv:2206.12191.
[35] H.M. Nguyen, E.W. Cooper, K. Kamei, Online learning from imbalanced data streams, in: 2011 International Conference of Soft Computing and Pattern Recognition, SoCPaR, 2011, pp. 347–352, http://dx.doi.org/10.1109/SoCPaR.2011.6089268.
[36] A. Triche, A.S. Maida, A. Kumar, Exploration in neo-hebbian reinforcement learning: Computational approaches to the exploration–exploitation balance with bio-inspired neural networks, Neural Netw. 151 (2022) 16–33, http://dx.doi.org/10.1016/j.neunet.2022.03.021.
[37] S. S.B., A. Garg, P. Kulkarni, Dynamic memory management for GPU-based training of deep neural networks, in: 2019 IEEE International Parallel and Distributed Processing Symposium, IPDPS, 2019, pp. 200–209, http://dx.doi.org/10.1109/IPDPS.2019.00030.
[38] A. Agarwal, A. Rakhlin, P. Bartlett, Matrix Regularization Techniques for Online Multitask Learning, Tech. Rep. UCB/EECS-2008-138, EECS Department, University of California, Berkeley, 2008.
[39] C.-C. Chang, J.-C. Liu, Y.-L. Shen, T. Chou, P.-C. Chen, I.-T. Wang, C.-C. Su, M.-H. Wu, B. Hudec, C.-C. Chang, C.-M. Tsai, T.-S. Chang, H.-S.P. Wong, T.-H. Hou, Challenges and opportunities toward online training acceleration using RRAM-based hardware neural network, in: 2017 IEEE International Electron Devices Meeting, IEDM, 2017, pp. 11.6.1–11.6.4, http://dx.doi.org/10.1109/IEDM.2017.8268373.
[40] A. Nagabandi, C. Finn, S. Levine, Deep online learning via meta-learning: Continual adaptation for model-based RL, 2018, http://dx.doi.org/10.48550/ARXIV.1812.07671, URL https://arxiv.org/abs/1812.07671.
[41] S. si Zhang, J. wei Liu, X. Zuo, Adaptive online incremental learning for evolving data streams, Appl. Soft Comput. 105 (2021) 107255, http://dx.doi.org/10.1016/j.asoc.2021.107255, URL https://www.sciencedirect.com/science/article/pii/S1568494621001782.

Alejandro Carnero received his Bachelor's degree in Software Engineering and his M.Sc. in Software Engineering and Artificial Intelligence from the University of Málaga, Spain, in 2020 and 2021, respectively. Since the end of 2020 he has been a research assistant in the ERTIS research group at the University of Málaga. Currently, he is a member of the ITIS Software Institute of the University of Málaga and a Ph.D. student researching structural health monitoring of civil infrastructures based on computer vision techniques and distributed deep machine learning in the Cloud-to-Things continuum. His research interests focus on distributed and incremental deep machine learning and artificial vision along with the IoT field.

Cristian Martín is an Associate Professor at the University of Malaga (UMA), and he is part of the ERTIS research group and the ITIS Software. Cristian Martín obtained a Ph.D. in Computer Science in 2018 at UMA, with an extraordinary Ph.D. thesis award. His research interests are focused on the IoT, machine-learning applied, digital twins, as well as paradigms such as Fog and Edge Computing. Previously he has been working in several technology companies on RFID technology and software development. He has participated in more than 20 research projects and contracts with companies. He has carried out four international stays, one at the University of Ghent, Belgium (2016, pre-doctoral), two at the IHP research institute, Frankfurt Oder, Germany (2020-2021, post-doctoral), and the last one in Incheon, Korea (2022).

Gwanggil Jeon (Member, IEEE) received the B.S., M.S., and Ph.D. degrees from Hanyang University, Seoul, South Korea, in 2003, 2005, and 2008, respectively., From 2009 to 2011, he was a Postdoctoral Fellow with the University of Ottawa, Ottawa, ON, Canada, and from 2011 to 2012, he was an Assistant Professor with Niigata University, Niigata, Japan. He is currently a Professor with Incheon National University, Incheon, South Korea; and Xidian University, Xi'an, China. His research interests fall under the umbrella of image processing, particularly image compression, motion estimation, demosaicking, and image enhancement as well as computational intelligence, such as fuzzy and rough sets theories.

Manuel Díaz is a University Full Professor in the Department of Languages and Computer Sciences at the University of Malaga, where he directs the ERTIS research group, integrated within the Software Engineering Group of the University of Malaga and from 2016 he is the CEO of Software for Critical Systems, S. L. of which he was co-founder in 2009. Between 1987 and 1991 he worked in the private sector (Olivetti Spain and R&D department of Fujitsu in Malaga). Since 1991 he belongs to the Dept. of Languages and Computer Sciences. His main lines of research focus on distributed systems, real-time embedded systems, and IoT and, more specifically, on the aspects related to middleware for this type of applications and the development of innovative applications in critical systems through the use of disruptive technologies as deep learning and distributed ledger technologies. He has been a principal researcher in 40 research contracts with private companies (Tecnatom, Abengoa, Indra, Adif, ...), 6 National Plan projects and 8 European projects (one of them as coordinator).