Received: 29 July 2022 | Revised: 29 January 2023 | Accepted: 3 March 2023

DOI: 10.1111/exsy.13287

ORIGINAL ARTICLE

Expert Systems WILEY

The orchestration of Machine Learning frameworks with data streams and GPU acceleration in Kafka-ML: A deep-learning performance comparative

Antonio Jesús Chaves | Cristian Martín | Manuel Díaz

ITIS Software Institute, University of Málaga, Málaga, Spain

Correspondence
Antonio Jesús Chaves, ITIS Software Institute, University of Málaga, Málaga, Spain.
Email: chaves@uma.es

Funding information
Consejería de Economía, Innovación, Ciencia y Empleo, Junta de Andalucía; European Commission; Ministerio de Ciencia, Innovación y Universidades

Abstract

Machine Learning (ML) applications need large volumes of data to train their models so that they can make high-quality predictions. Given digital revolution enablers such as the Internet of Things (IoT) and the Industry 4.0, this information is generated in large quantities in terms of continuous data streams and not in terms of static datasets as it is the case with most AI (Artificial Intelligence) frameworks. Kafka-ML is a novel open-source framework that allows the complete management of ML/AI pipelines through data streams. In this article, we present new features for the Kafka-ML framework, such as the support for the well-known ML/AI framework PyTorch, as well as for GPU acceleration at different points along the pipeline. This pipeline will be described by taking a real Industry 4.0 use case in the Petrochemical Industry. Finally, a comprehensive evaluation with state-of-the-art deep learning models will be carried out to demonstrate the feasibility of the platform.

KEYWORDS
artificial intelligence, data streams, Kafka-ML, machine learning, PyTorch, TensorFlow

1 | INTRODUCTION

The Internet of Things (IoT) Díaz et al. (2016) has driven the acquisition and monitoring of information from the physical world with the aim of bridging the gap with the digital one and enabling paradigms such as cyber-physical systems and digital twins. The IoT evolution, resulting in the increase of monitored physical information, has also promoted the growth of other areas that are directly dependent on the information available, such as artificial intelligence and machine learning. An example of this intersection is deep learning, which requires large amounts of data to detect complex patterns and behaviours in information sources with minimal human intervention.

Despite the clear evolution of these fields over the last years, nowadays they are not adequately coordinated with each other. For instance, most current Machine learning (ML)/Artificial Intelligence (AI) frameworks are well suited to work with static datasets such as files and storage systems but not with data streams. For the simple use case of managing the lifecycle of ML models only from data streams, it is currently necessary to resort to a wide range of software architectures (and underlying technologies and tools) for dataset creation, data storage, and model training and so on. Not to mention AI frameworks do not allow reusing data streams to train and evaluate several models in parallel or do this in scalable architectures. For these reasons, we defined "Kafka-ML: connecting the data stream with ML/AI frameworks" (2022), an open-source framework that manages the lifecycle of ML/AI applications through data streams in scalable platforms. Through an accessible and user-friendly

Web interface Kafka-ML users can easily define ML models so as to train them and evaluate their performance, and deploy them for inferences. In addition, both training and model inference can be performed with data streams. These data-consuming phases are no longer isolated steps.

Despite the valuable contributions provided by Kafka-ML, it had a number of limitations. For instance, Kafka-ML did not support GPU acceleration. This limited the training and inference of machine learning models, specially deep learning ones, leading to long days of training. Furthermore, initially, Kafka-ML only offered support for the ML framework TensorFlow, but there are nowadays other popular frameworks such as PyTorch, which does not provide any official support for dataset creation and data consumption from Apache Kafka, the data stream distribution system backbone of Kafka-ML. Another desirable thing is to evaluate different models, even in different ML frameworks, and compare their performance with the same data streams. Finally, regarding visualization, learning-curve graphs to better assess when ML models are ready for inference were not available. In this paper, we present an open-source update of Kafka-ML that addresses the previous limitations, in addition to an extensive and multi-framework evaluation of Kafka-ML on a state-of-the-art high-performance infrastructure. Therefore, this open-source framework has a place in Industry 4.0, and to validate this, we will present a use case in it, specifically in the Petrochemical Industry, in which we will predict through Kafka-ML the freezing point in the production of lubricants in real time. The main contributions of this article are the following:

1. Support for the management of the lifecycle of PyTorch models, with the development of a missing open-source connector for data stream ingestion from Apache Kafka[1] .
2. Incorporation of support for GPU acceleration for training and inference phases.
3. Improvement of the assessments of ML models with the generation of descriptive learning curves.
4. Extensive evaluation of Kafka-ML with state-of-the-art deep learning models (both TensorFlow and PyTorch) in a high-performance GPU infrastructure.
5. Validation of the Kafka-ML framework in a Petrochemical Industry 4.0 use case.

The rest of the paper is structured as follows. Section 2 presents the related work and their potential differences with Kafka-ML. Section 3 presents the architecture of Kafka-ML. Section 4 explains the regular workflow in Kafka-ML by using an Industry 4.0 use case. Section 5 provides an evaluation of several key elements of the framework. Finally, Section 6 summarizes the work, with a discussion on possible future expansions.

2 | RELATED WORK

To the best of our knowledge, Kafka-ML remains one of the first open-source tools providing an ML/AI pipeline solution that tightly integrates ML/AI and data streams. However, other tools serve similar purposes or provide some of the functionalities offered by Kafka-ML.

KFIML Wan et al. (2022) is a tool that integrates ML applications with big data streaming processing. In addition, they provide a comprehensive way to process data from IoT devices as well as online ML. However, while we consider that it is a suitable tool for handling data from IoT devices, it does not allow the definition of new models. This is something that Kafka-ML does allow, being able to train and do inference of state-of-the-art models from frameworks such as TensorFlow or PyTorch with the desired characteristics.

NVIDIA Deep Learning GPU Training System (DIGITS) NVIDIA DIGITS Official Webpage (n.d.) is a web-based tool for training and inference of deep neural networks using GPUs. DIGITS is not a framework per se, but a wrapper for several well-known deep-learning libraries such as TensorFlow or PyTorch. Some of the main advantages of DIGITS are the support of multi-GPUs, the deployment of pre-trained models for inference, and the ability to view training metrics in real-time. Unfortunately, it does not support the training and inference via data streams that Kafka-ML does, whereas our solution also currently supports training using GPUs.

Kubeflow Official Webpage (n.d.) is one of the most powerful ML tools available today. Kubeflow users can define a multitude of models from a variety of frameworks with multiple configurations. Kafka-ML makes transparent certain steps that the user must perform in Kubeflow, such as the construction of training containers. In our tool, the user can train and deploy a model for inference by simply dealing with the Web User Interface. Regarding the data stream management, although Kubeflow has limited support for the Feast library Feast Official Webpage (n.d.), we have not seen any example or mention of it being applied to data streams. Feast is an operational data system for managing and serving machine-learning features to models in production. Feast is able to serve feature data to models from a low-latency online store for real-time prediction or from an offline store for model training. However, we also do not observe that data obtained from data streams can be used for training in an effective way. In Kafka-ML, everything is based on the data stream concept, from training to inference. There are several open-source platforms similar to Kuberflow and Kafka-ML in functionality, such as Metaflow Official Webpage (n.d.) and MLflow Official Webpage (n.d.), and with ample support. However, these do not have the possibility of sending data via data stream.

MLRun Official Webpage (n.d.) is an open-source ML operation (MLOps) platform for a quick build and continuous management of ML applications throughout their lifecycle. This platform is integrated into the development and CI/CD environment, automating tasks such as data delivery and ML pipeline management. Additionally, users can easily integrate MLRun into their preferred IDE, whether it be local or cloud-based, resulting in faster adoption and deployment times. While MLRun shares some similarities with Kafka-ML, our solution has a web user interface

that allows even non-technical users to deploy models by simply defining the model architecture, unlike MLRun where models, data, training, and inference parameters are registered via an API. Moreover, although MLRun has a data structure for Kafka data streams, we have just found an implementation of it without examples, where it seems to only be used for inferencing production data.

Pachyderm Official Webpage (n.d.) is a MLOps solution that provides similar capabilities to MLRun. These capabilities include automatic model deployment, data versioning, and data lineage tracking. Despite operating in a similar way to prior platforms (through an API), Pachyderm also has data stream support (for Apache Kafka and AWS Kinesis). Besides inferring data streams, this feature allows for incremental training of models as new data becomes available. However, it is important to note that, similar to MLRun, Pachyderm's API usage and data stream connectivity may not be suitable for users with limited technical expertise. In contrast, Kafka-ML offers a more user-friendly approach by automating these processes, making it accessible to a broader range of users. Furthermore, if the user wants to deploy Pachyderm, he/she will need a cluster with high capabilities, as well as the knowledge to deploy its components. In Kafka-ML the components are clearly differentiated, lightweight, and easy to deploy.

Ray Moritz et al. (2018) is an open-source framework designed to scale AI and Python workloads. It offers a comprehensive set of libraries and tools that support a wide range of applications, including reinforcement learning, deep learning, model serving, hyperparameter tuning, and data preprocessing. One limitation of Ray is that it does not support data stream management and it requires knowledge of distributed computing and parallel programming, which could be a barrier for some users. Despite this, the library remains a valuable tool for anyone looking to speed up the performance of their AI and Python workloads, and it could be integrated into other ML pipeline tools such as Kafka-ML.

ClearML (2019) is an MLOps platform that allows data scientists and engineers to track, compare, and optimize their machine learning models and experiments. It provides experiment tracking, reproducibility, and model management features. Combining the integration with Jupyter Notebooks and the ClearML UI, users can work with those features in a comfortable way. However, compared to Kafka-ML, our proposal has a browser-integrated UI (especially useful for model definition), making it more convenient for non-technical users. ClearML does not support the management of data streams, and it is available in different licences, and if we consider only the open-source version, it has more limitations when deploying certain tasks.

There are also enterprise solutions, such as Amazon SageMaker What is Amazon SageMaker? (n.d.). It is a suite of applications that is part of Amazon Web Services, where a CI/CD service is offered for ML. It has support for several ML frameworks, such as TensorFlow, PyTorch, or MXNet. SageMaker provides a multitude of services and functionalities, including hyperparameter optimization, incremental training, and elastic training. Although it offers support for data streams, these can only be used during inference, requiring data lakes to take the training data, unlike Kafka-ML, where both training and inference are carried out with data streams, so we do not need an additional data lake.

In addition to SageMaker, there are other paid options, such as Algorithmia MLOps Official Webpage (n.d.), a platform especially focused on inference; cnvrg.io cnvrg.io Official Webpage (n.d.), a platform that can be deployed in a multitude of infrastructures and with support for a wide variety of ML/IA frameworks; or iguazio Official Webpage (n.d.) and SigOpt Official Webpage (n.d.), with similar characteristics to the previous one. However, these platforms do not have a pure focus based on data streams. Also, it must be taken into account that they are not open source.

Google Cloud AutoML Official Webpage (n.d.) and Azure Machine Learning Official Webpage (n.d.) are tools similar to SageMaker, where the user can request an automatic estimation of hyperparameters to obtain the ones that best fit their use case. While this is interesting, it is a part of the Kafka-ML's goals having an accessible platform where, with a few lines of code, both expert and non-expert users can work with the classic Machine Learning pipeline through data streams.

Finally, River Montiel et al. (2021) is an ML tool that allows online training as well as the use of data streams. While noting that it achieves great results in terms of performance, it does not go into complex Deep Learning models. In addition, this tool does not offer a simple interface for non-expert users to work with it.

3 | ARCHITECTURE

Kafka-ML is a novel and open-source framework designed to manage ML pipelines through data streams. In this work, Kafka-ML architecture has been redefined to enable the management and deployment of models from a new widely used framework, PyTorch, whose models can also be deployed in combination with TensorFlow ones. In addition, some extensions have been applied to allow GPU acceleration during the training and inference steps. Kafka-ML offers an open-source and easy-to-use Web User Interface (UI) for both specialists and non-specialists to deal with the ML/AI model pipeline in a user-friendly way.

Users simply have to define the model in a similar way as they would locally. In some cases (e.g., in PyTorch), they will have to define some extra functions, in which they will specify the loss function, the optimizer, and the metrics, so that these can then be used during training. Once the models have been defined, they can be trained, evaluated, and inferred from them. Furthermore, this structure works with the data streams in such a way that they can be reused and avoids having the information in external systems and data warehouses for datasets (we will discuss this further in Sections 4 and 5).

The Kafka-ML architecture comprises sets of services based on the single responsibility principle, forming a microservice architecture. All components run as Docker containers in the architecture, enabling component isolation and portability. The management and deployment of the platform in a cluster of nodes and in a distributed manner and production infrastructures is achieved through the orchestration provided by Kubernetes. Kubernetes allows a continuous monitoring of the Kafka-ML services and deployed tasks (e.g., training and inference), ensuring that they are in the state defined.

Kafka-ML, with all its new features, its implementation, its configuration files, and some examples, can be found in our GitHub repository (Kafka-ML official Github repository, n.d.)[2] . An overview of the Kafka-ML architecture, its modifications and the new components is shown in Figure 1. The new components created in this architecture are shown in blue and the components modified in orange. The control logger is responsible for managing the control messages and sending them to the backend. Apache Kafka and Zookeeper (required by Kafka) are also deployed as Docker containers and managed through Kubernetes.

3.1 | Frontend

The frontend component provides a Web User Interface where users can manage all Kafka-ML functionalities. This component interacts with the RESTful API defined in the backend and has been developed in Angular, the popular TypeScript-based web development framework. Slight changes have been introduced in this module that allow the user to define which ML framework he/she wants to work on and the GPU use, as well as to automatically manage the content to be displayed in each case.

3.2 | Backend

The backend developed in Django/Python component provides a RESTful API to manage all the information contained in Kafka-ML. In addition, this component works by using the Kubernetes API to manage and deploy all the necessary components for ML/AI pipelines.

There are some differences in this component with respect to the previous version. This is due to the fact that, previously, since there was only support for TensorFlow, the code for this framework was in the backend itself, making it difficult to incorporate new frameworks.

To avoid this, the parts executing the ML framework's own code have been extracted to new components, named ML Code Executor (e.g., TensorFlow Executor and PyTorch Executor for the currently supported frameworks). Now, when the backend needs something related to the models, such as checking if the user input is valid, it sends this input to one of these modules, where this check will be performed. This also allows for greater scalability because Kubernetes can distribute these subcomponents on different machines so the load they may have can be balanced. In addition, the modifications made enable the integration of a new ML framework into the system without many problems in the future.

3.3 | Training and inference of ML models

Algorithm 1 describes the procedure of the training job in Kafka-ML, in which, initially, the model is first retrieved from the backend and awaits a control message indicating the arrival of data. Once that message arrives, the data are loaded and divided into the subsets established by the user, in order to train and validate the model with these subsets. Finally, both the trained model and its metrics are returned to the backend. Algorithm

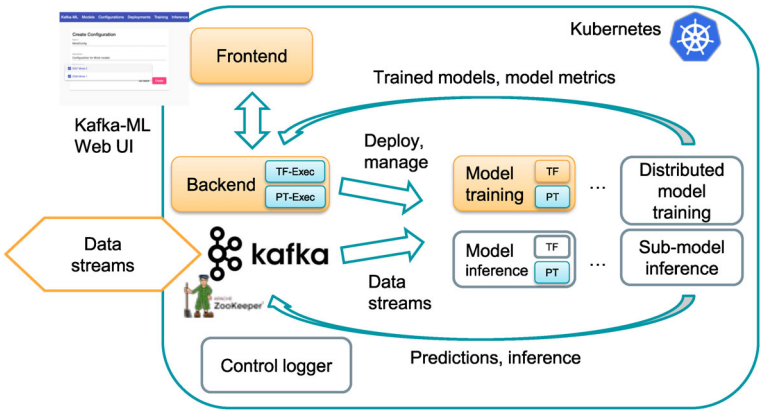

FIGURE 1 Overview of the new Kafka-ML architecture. Orange component: modified; blue component: added.

ALGORITHM 1 **Training algorithm for PyTorch ML Model in Kafka-ML.**

```
Input: model_architecture_url, model_weights_url, train_kwargs, val_kwargs, test_kwargs, deployment_id, data_stream
Output: trained_model_weights, training_metrics, val_metrics, test_metrics, confusion_matrix
model ← downloadModelArchitecture(model_architecture_url)
while not trained do
  trained ← False
  msg ← readControlStreams()
  if deployment_id == msg.deployment_id then
    data_stream ← readStream(msg.topic)
    training_size ← (1-(msg.validation_rate+msg.test_rate))*msg.total_msg
    validation_rate ← msg.validation_rate*msg.total_msg
    test_rate ← msg.test_rate*msg.total_msg
    train_set, val_set, test_set ← split(data_stream, training_size, validation_size, test_size)
    train_res, val_res ← trainModel(model, train_kwargs, train_set, val_kwargs, val_set)
    if msg.test_rate > 0 then
      test_res ← evaluateModel(model, test_kwargs, test_set)
      confusion_matrix ← createConfusionMatrix(test_res, test_set)
    end if
    uploadTrainedModelAndMetrics(model_weights_url, training_res, val_res, test_res, confusion_matrix)
    trained ← True
  end if
end while
```

ALGORITHM 2 **Inference algorithm for PyTorch ML Model in Kafka-ML.**

```
Input: model_architecture_url, model_weights_url, input_topic, output_topic, input_configuration, data_stream
Output: Predictions to Apache Kafka
model ← downloadModelArchitecture(model_architecture_url)
weights ← downloadModelWeightsFromBackend(models_urls)
model ← loadModelWeights(model, weights)
deserializer ← getDeserializer(input_configuration)
while True do
    stream ← readStreams(input_topic)
    data ← decode(deserializer, stream)
    predictions ← predict(model, data)
    sendToKafka(predictions, output_topic)
  end while
```

2 describes the procedure of the inference job. First of all, the model and its trained weights are retrieved from the backend. Afterwards, the weights are loaded into the model, and the corresponding deserializers are created in order to process the data received by the given input topic into a suitable format for the model. When data are received, they are processed and sent to the model, generating a prediction that is sent through the configured output topic. Except for slight changes to add new functionalities that will be discussed later, the training and inference modules work in a similar way as in Kafka-ML: connecting the data stream with ML/AI frameworks (2022).

It should be noted that specially the training process requires large computational capacities. Moreover, there are components such as GPUs that considerably accelerate the computation in this respect. Therefore, support for the use of GPUs has been added to Kafka-ML through Aliyun's "GPU Share Scheduler Extender" library.[3] This is a major breakthrough as it allows model training to be greatly accelerated. While there is an official NVIDIA plugin[4]that also allows this, it reserves an entire GPU for each job we run that requires a GPU, whereas Aliyun's option seemed more appropriate as it allows us to manage the memory of the GPUs as an indication.

3.4 | Management of data streams in Kafka-ML

The management of data streams in Apache Kafka provides fault tolerance and load balancing, as well as the ability not to depend on a storage system to keep the data (thanks to Kakfa topic logs) and to reuse them.

PyTorch does not have a library like TensorFlowIO that allows the creation of a dataset from Kafka data streams. This is very useful in TensorFlow training to have all the data grouped together and to train the model in a regular way. Therefore, it was necessary to make an implementation according to the PyTorch Dataset, which, given a control message[5], collects the data from one or more Kafka topics/partitions and applies the relevant transformations to them.

The implementation of this PyTorchKafkaDataset is publicly available on GitHub[6] .

4 | PIPELINE OF PYTORCH ML MODEL IN KAFKA-ML APPLIED TO INDUSTRY 4.0 USE CASE

In this section, the pipeline of an PyTorch ML model in Kafka-ML is presented using an Industry 4.0 use case in collaboration with the Spanish company CEPSA. Before starting, the configuration concept[7] used in Kafka-ML is still maintained, but its structure has been extended to accept models from various frameworks. This is convenient because, in addition to the previous advantages, it is now possible to deploy and compare similar models across frameworks in parallel using only one data stream. Kafka-ML's pipeline is the following:

(1) implementation and registration of PyTorch/TensorFlow ML models; (2) creation of training configurations for a set of ML models to be trained; (3) deployments of a specific training configuration with certain parameters; (4) feeding the deployed configuration with training data streams; (5) obtaining and comparing the result metrics from the training phase; (6) deploying a selected ML model for inference and prediction making; and lastly, (7) inference phase to feed the deployed trained models with data streams to obtain their predicted results.

Figure 2 shows the full sequence of the pipeline steps. Next, each of the steps is detailed.

4.1 | Use case: CEPSA virtual analyser

In order to describe the pipeline, it will be illustrated by a use case applied to the company CEPSA.

The objective of the use case is to define a virtual analyser which is able to predict the freezing point of one CEPSA end product (lubricant) based on the operating conditions and the properties of the feedstock. This process is carried out in the San Roque (Spain) Energy Park of CEPSA, one of the world's largest energy companies. The freezing point is an important parameter, which due to its characteristics, must be measured in laboratory. Based on the monitoring of different components, such as filters speed and other sensors of the machinery and the type of feedstock, the aim is to predict, through deep learning, the state of the freezing point in real time for better control of the process.

4.2 | Definition and registration of PyTorch ML models

While supporting a new framework like PyTorch, the aim is follow a simple approach that allows programmers to focus on the most important details, such as the definition of a good ML model. For this reason, the same approach has been maintained when defining ML models, allowing an easy validation of model code.

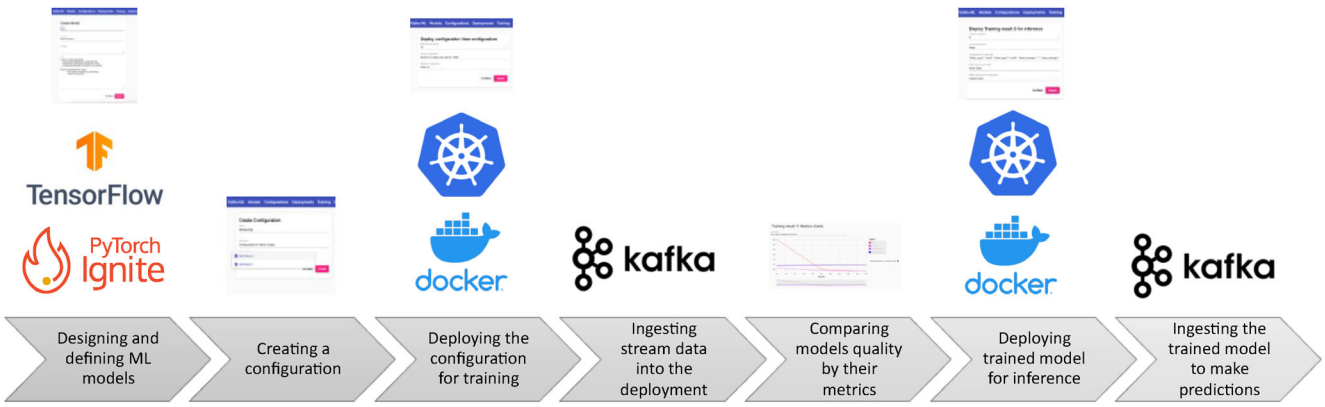

FIGURE 2 ML/AI pipeline in Kafka-ML.

FIGURE 3 Definition of a PyTorch model in Kafka-ML.

Given the peculiarities of training models in PyTorch, extra functions must be defined in order to train a model using Kafka-ML. However, except for the verbosity involved in defining a model in this ML framework, it is defined in a somewhat similar way to how it would in TensorFlow. An example of a PyTorch model declaration in Kafka-ML is shown in Figure 3.

It should be noted that, for confidentiality reasons, the original architecture of the model for the use case will not be shown, but an equivalent architecture is shown to describe the process. The only noticeable reason will be the quality of training results.

If we need to add any other required function to the model, it can be inserted in the imports field. Once the model is submitted, its source code will be checked as a valid model by the PyTorchExecutor and incorporated into Kafka-ML. If the model has been successfully created, the pipeline can be continued to the next step.

4.3 | Creation of a training configuration

As mentioned before, a configuration is a set of ML models to be deployed and trained together. There is now the possibility to mix ML models from different ML frameworks in a way that is transparent to the user. This can be seen in Figure 4a. Therefore, this makes it possible to achieve the best model regardless of the framework as well as benchmark different models under the same conditions. Returning to the CEPSA use case, this allows us to evaluate various models and hyperparameter combinations quickly and easily. Therefore, we could swiftly obtain the model that best suits our objective.

4.4 | Deploying a configuration for training

After creating our configuration, it is time to deploy the models from it. To do this, it will be necessary to establish a series of parameters related to the training and evaluation phases that will be common to all models of the same ML framework. The parameters to be defined are the batch size, the training arguments (in these arguments, the number of epochs must be defined), and the validation arguments.

When the form is submitted, a task is deployed for each model. The first step each job takes is to fetch the ML model from the existing models in Kafka-ML, load it and start the training. Given the peculiarities of PyTorch's model storage, it has been decided to take a different approach in order to fetch the ML models. Now, instead of bringing an ML model compiled from the backend, we bring the code with the model

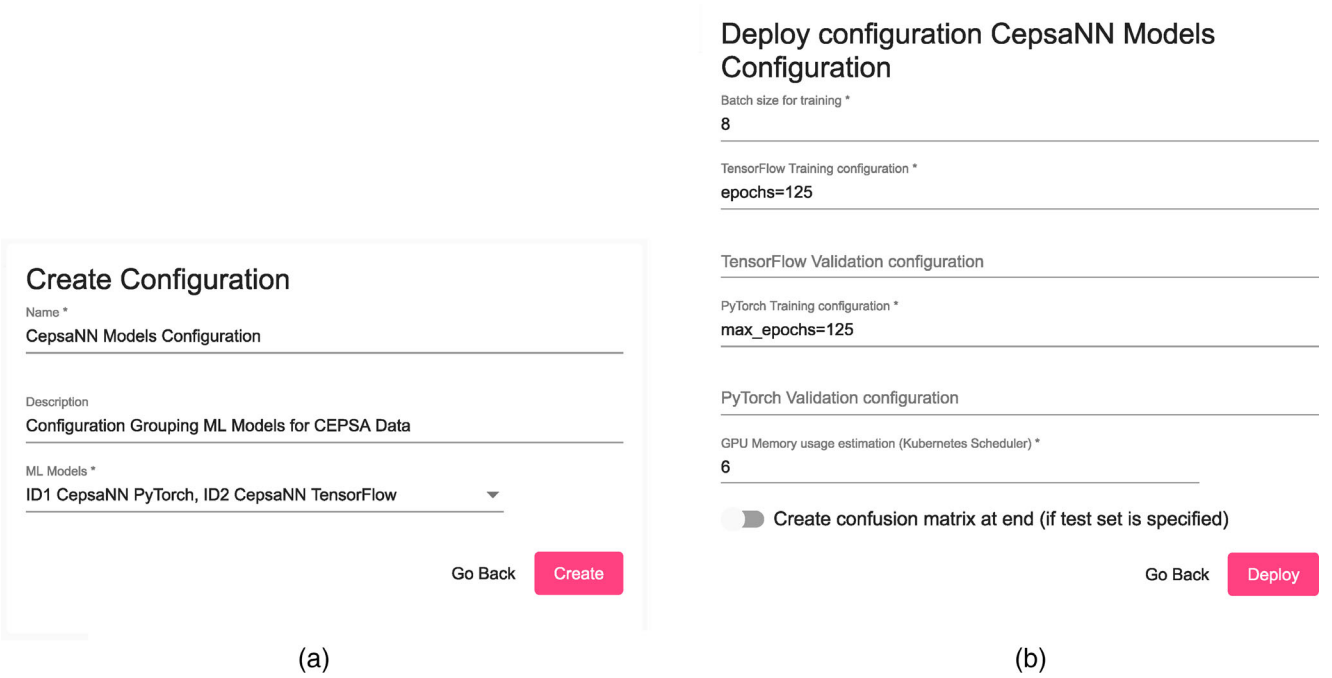

FIGURE 4 (a) Creation of a configuration in Kafka-ML. (b) Deployment of a configuration in Kafka-ML.

architecture, and it is executed in the job. This is because PyTorch's model saving is based on Pickle[8], which serializes and saves the model object but also part of its context, which can cause issues when moving it to another container. However, this can be prevented by separating the model structure and weights, so we can deploy it in another container without complications.

PyTorch Ignite library[9] has been used for PyTorch given the convenience of having a training or validation engine as well as a large number of metrics already defined for its use.

Once the model is ready, the next step of the job is to wait until a data stream is received for training and optionally for evaluation through Apache Kafka. This allows us to have ready-to-train models and train them directly if there is already a data stream available in Kafka.

A deployment can be created in the Kafka-ML Web UI as shown in Figure 4b.

4.5 | Feeding the deployment with data stream

The next step after having the models deployed and waiting for training data is to send the data stream to them. As mentioned above, the training will not start until the data stream is available in Kafka. Two Kafka topics are used for this purpose: the first is the data topic, which only contains the data for the training and evaluation phases. The second topic is the control topic, used to specify to which deployed model the data are destined. This is done through control messages that dictate when and where data streams are available for training and evaluation. Control messages are further detailed in Kafka-ML: connecting the data stream with ML/AI frameworks (2022).

After sending a data stream to the corresponding deployment using the libraries provided in Kafka-ML, which in the presented use case contains the data from various sensors and process control variables, all models belonging to that configuration will start their training.

4.6 | Obtaining the metrics resulting from the training

After training and evaluation phases, Kafka-ML allows users to visualize the metrics they have previously defined in the model. An example can be seen in Figure 5. For each model, the user can download the trained model (the weights in the case of PyTorch), delete it and deploy it for inference (which would be our next phase).

Compared to the previous version of Kafka-ML, several changes have been introduced in the view regarding this section. Firstly, since now it is possible to have up to three subsets of data (a training set used to train the model, a validation set used to evaluate the model performance while training, and a test set used to evaluate the model after being trained and to generate, if indicated, a confusion matrix), and there is one field

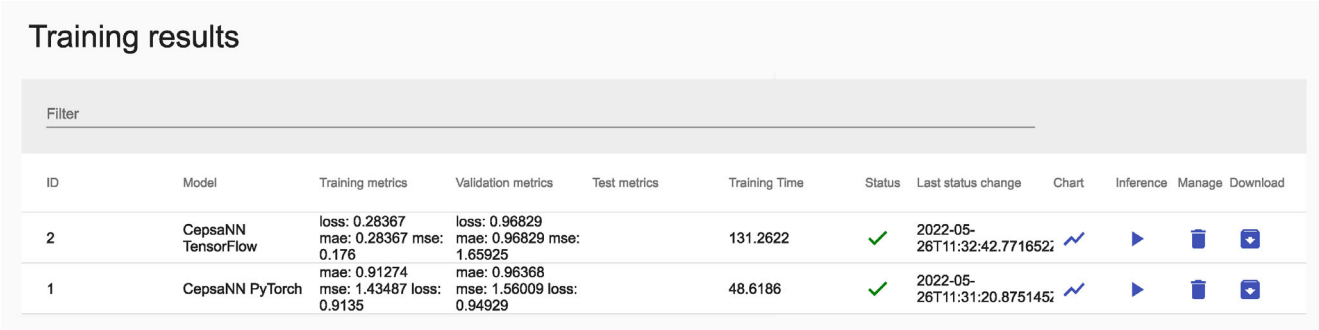

FIGURE 5 Training management and visualization in Kafka-ML.

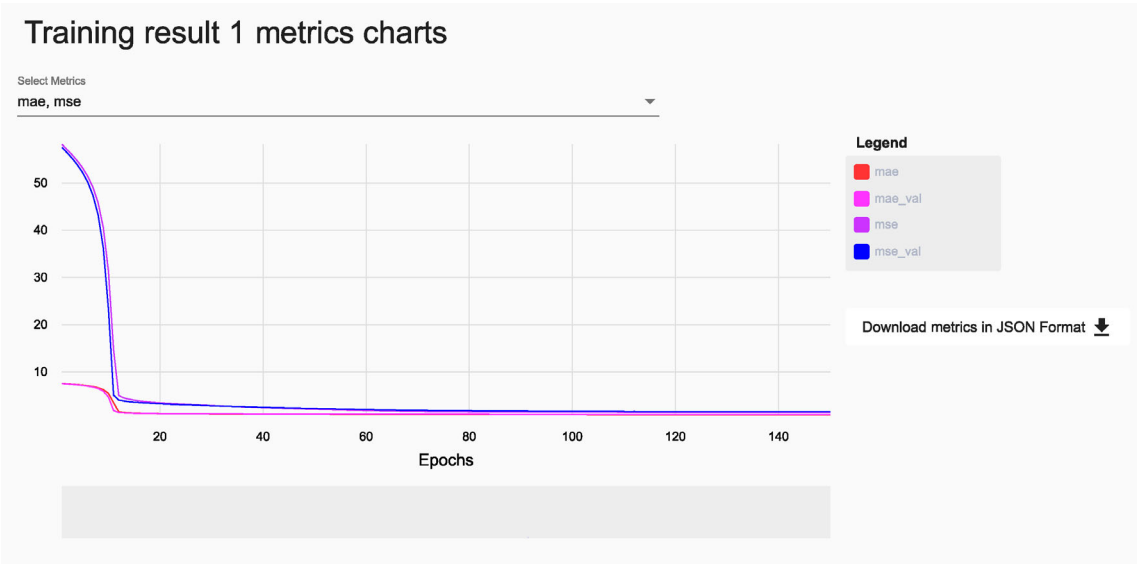

FIGURE 6 Metrics visualization in Kafka-ML.

for each subset in each model where the last calculated metrics are shown. In the previous version, there were only training and validation subsets.

In addition, a new view has been included where users can visualize how the model has been improving in each epoch, displaying the curves of the different metrics specified for both training and validation. This can be seen in Figure 6. Moreover, in case the user has specified it and the model being utilized is a classifier, he/she will also be able to visualize the confusion matrix with respect to the test subset. An example of confusion matrix can be seen in Figure 7.

This is very useful as it allows users to be more certain that their models have not been overtrained and that they are capable of making quality predictions on input data that they have not been trained on. This is ideal in all areas but is of particular importance when applying an ML/AI-based model in production, as would be the case in our collaboration with CEPSA.

4.7 | Deploying an ML model for inference

After training the models, they can be deployed for inference from the Kafka-ML UI itself, as shown in Figure 8.

This form asks for the number of inference replicas to be deployed. Replicas allow for load balancing and fault tolerance.

Users will have to specify several details in the form: (1) input topic, for values to predict on; (2) output topic, for predictions (topics are key elements for producers and consumers to interact with Apache Kafka; each of these topics is identified with a name); (3) Kubernetes credentials can also be included for the deployment of the model into another Kubernetes cluster in the continuum.

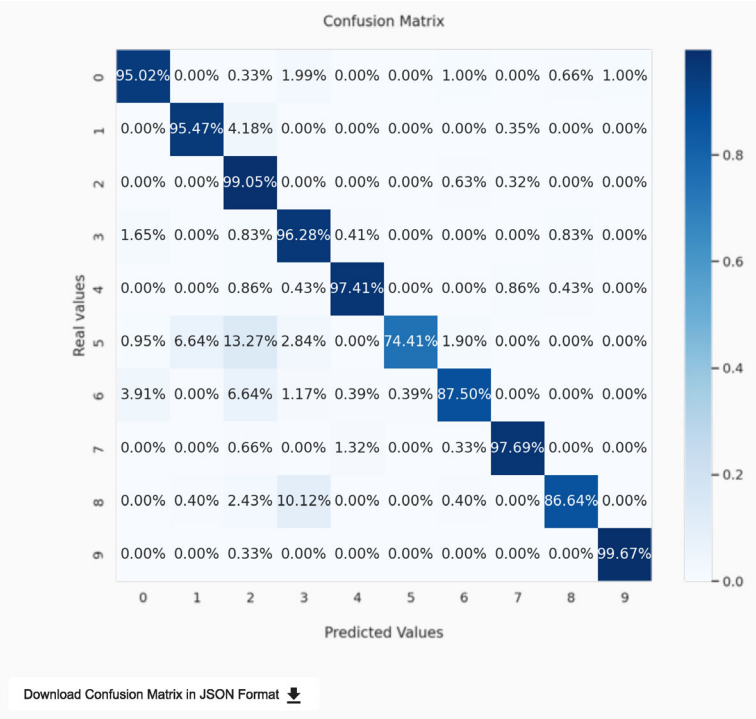

FIGURE 7 Metrics visualization and confusion matrix display in Kafka-ML.

FIGURE 8 Deployment of a trained model to inference.

Returning to the use case in collaboration with CEPSA, the fact that the inference is fault-tolerant is very important, as the results of this model can lead them to make decisions about the quality of their compound and therefore the optimization of their profit.

4.8 | Feeding deployed trained models with data streams for inference

Finally, the ML pipeline ends once the model is trained and deployed to perform predictions through data streams. In this case, it is not necessary to send control messages, because the necessary information for inference [(e.g., input and output topics) was already defined in the Kafka-ML Web UI].

Users and systems will only have to send the data to the input topic via data streams using the appropriate data format. The results of the inference of these data will be sent almost immediately to the output topic when the models make the predictions.

5 | EVALUATION

In order to evaluate the performance of the new architecture as well as the implementation of the new extensions on the platform, a set of experiments have been carried out to evaluate tasks such as training time and influence on the data stream configuration, together with high availability and load balancing in two popular ML frameworks: TensorFlow and PyTorch.

This evaluation is organized into two parts. In the first one, we will focus on training, evaluating different deep learning models with different datasets, and comparing results such as training time or accuracy between frameworks.

In the second part, we will be focused on evaluating the performance of data inference on trained models from the different frameworks, in order to evaluate factors related to fault tolerance and response time.

While Kafka-ML allows the deployment of user-created models, the tool is also prepared for the deployment of pre-defined deep learning models compatible with TensorFlow or PyTorch. For the evaluation, we have chosen some of these models because of their applicability in machine learning applications, as well as for the computational challenge they represent for the architecture.

The state-of-the-art deep-learning models that have been considered for evaluation are listed below.

1. VGG16 Simonyan and Zisserman (2014)
2. ResNet-101 He et al. (2015)
3. Densenet-201 Huang et al. (2016)
4. EfficienceNet-B7 Tan and Le (2020)

To perform the training and inference tests, the eurosat dataset Helber et al. (2017) and the so2sat dataset Zhu et al. (2018) have been used. Those datasets allow us to quickly evaluate different aspects of the models and the frameworks.

Specifically, the images from the eurosat dataset are 64×64 and the ones from the so2sat dataset are 32×32, both in colour format. There are a total of 10 classes in the eurosat dataset and 17 classes in the so2sat dataset. By having different sizes and number of classes, this allows us to evaluate the performance of the framework during training and inference under different conditions.

5.1 | Experimental setup

For this evaluation, we are fortunate to have a highly capable infrastructure to carry out a comprehensive evaluation. The following provides a brief description of the system where the assessment is to be carried out.

1. **Hardware configuration**. The deployment of the tool as well as all the tests have been carried out on our own 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. The client that sent the information and where the results were measured from was a PC with an Intel(R) Core(TM) i9-10900K CPU and 64 GB of RAM.
2. **Software configuration**. 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 PC with the client runs Ubuntu 21.04.

5.2 | Training evaluation

To evaluate both frameworks, each pre-trained model will be trained for the two datasets mentioned above. Of interest in these first tests will be the training time as well as the quality of the models. The training configuration consists of a batch_size selection of 256 and a total of 50 epochs.

Two tests will be carried out; in the first one, we will only use one partition per Kafka topic, and all the models will read from it. Subsequently, we will do a similar test by setting up four partitions for each Kafka topic, looking to see how this might affect training time.

The results with the eurosat dataset are shown in Figure 9. Looking at the training time, it can be seen that PyTorch models were slightly faster than TensorFlow models, in line with the results obtained in Florencio et al. (2019) and El Shawi et al. (2021). However, it can be seen in Figure 10a that the accuracy of these models is particularly lower. The plot depicting the validation loss (a metric used to evaluate the performance of deep learning models and determine if further adjustments are needed; also used to compare the performance of different models or configurations), visible in Figure 10b, has also been generated. Here, despite the difference in values of the loss due to the diverse ways of working between frameworks, a better training of the TensorFlow models is observed. These results are also similar to the ones obtained in El Shawi et al. (2021) and Dai et al. (2022).

In the case of the second test, the results obtained regarding time are similar to those shown above and are shown in Figure 11. It is noticeable how the training time has been slightly reduced in the TensorFlow models, while in some PyTorch models, the time has increased slightly. One potential reason for this is that in PyTorchKafkaDataset, consumers are created sequentially per topic and partition, which could generate some extra latency.

The results of the so2sat dataset, shown in Figure 12, are slightly different from the above. If we consider the training time, it can be seen that it is not possible to determine which of the two frameworks is faster. However, it can be still seen in Figure 13a that the accuracy of PyTorch models is slightly lower compared to TensorFlow models. In Doleck et al. (2020), similar results are obtained. It should be noted that this dataset has a larger number of data, and this might be a determining factor in PyTorch models. Regarding the loss, shown in Figure 13b, it can be seen that the models in TensorFlow (except for EfficientNet-B7), are doing some overfitting, which makes their loss grow. This can also be seen in the

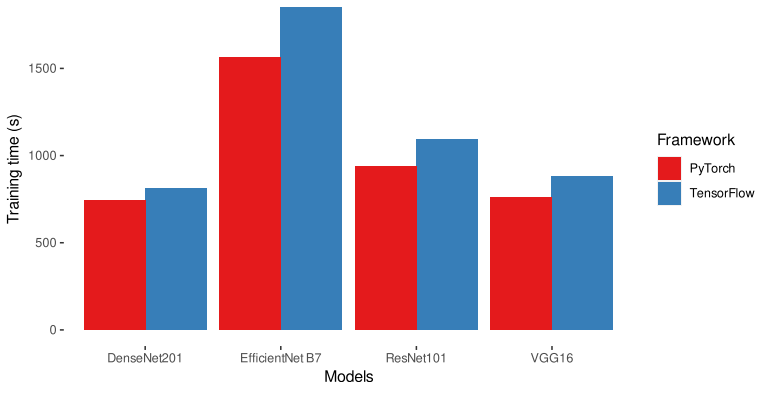

FIGURE 9 Training time comparison on the eurosat dataset between the different frameworks, using different models and 1 partition per Kafka topic.

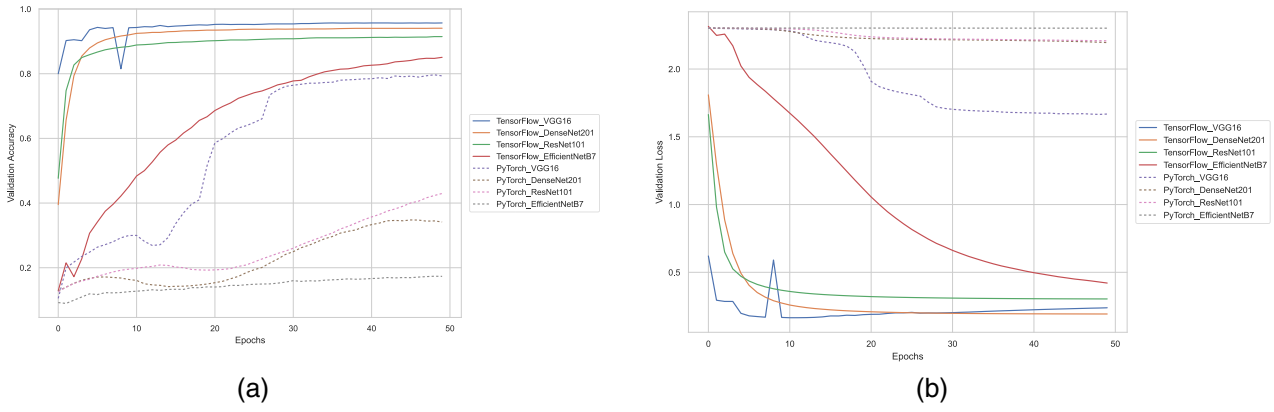

FIGURE 10 (a) Validation accuracy and (b) validation loss over epochs comparison on the eurosat dataset between the different frameworks, using different models.

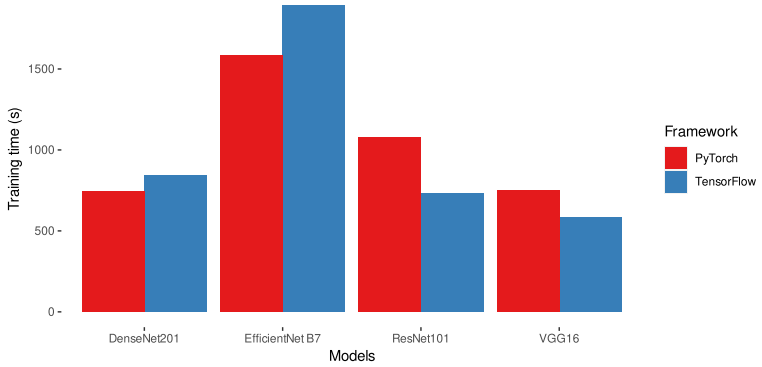

FIGURE 11 Training time comparison on the eurosat dataset between the different frameworks, using different models and 4 partitions per Kafka topic.

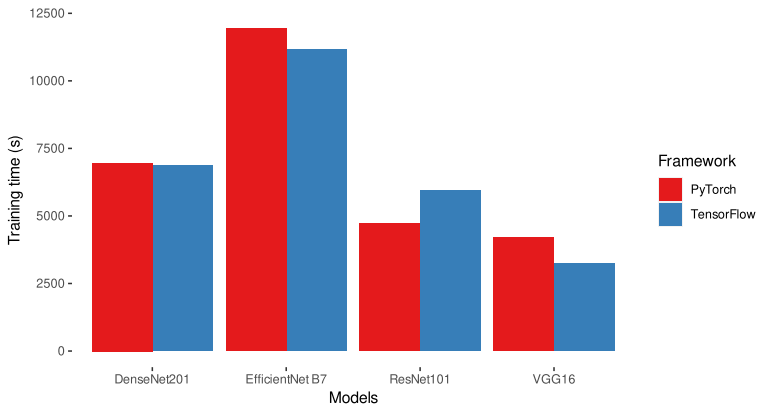

FIGURE 12 Training time comparison on the so2sat dataset between the different frameworks, using different models and 1 partition per Kafka topic.

accuracy plot, where, practically as soon as the training started, these models already had a high percentage of success, and many epochs in this case may have been counterproductive.

As for test results with 4 partitions per Kafka topic with so2sat dataset, they are quite similar to those shown above, with similar implications. This can be seen in Figure 14.

For the second test, it has been decided not to plot the training of the model in terms of accuracy and loss, since the results were pretty similar to those obtained previously.

We have also performed the same benchmarks training only on the CPU, in order to show the improvement that the architecture obtains using GPU acceleration. There is a clear enhancement through the use of GPUs, with an average improvement of more than 15× over CPU usage. See Figure 15a for the eurosat dataset and Figure 15b for the so2sat dataset.

As first conclusions, it is possible to surmise that, as a general rule, training in PyTorch models is slightly faster than in TensorFlow models. However, it is also noteworthy that TensorFlow models have always achieved the best results in terms of accuracy. It is also worth noting that there is a significant performance improvement when using GPU acceleration.

5.3 | Inference evaluation

In the evaluation of the inference phase, we will make a performance comparison between the frameworks available in Kafka-ML on different scenarios.

For the evaluation, we measure the latency response in both frameworks. In order to measure this, we have performed latency measurements with various configurations. In these configurations, we varied the number of clients sending information, as well as the number of model replicas and Kafka partitions used.

The configurations used are as follows:

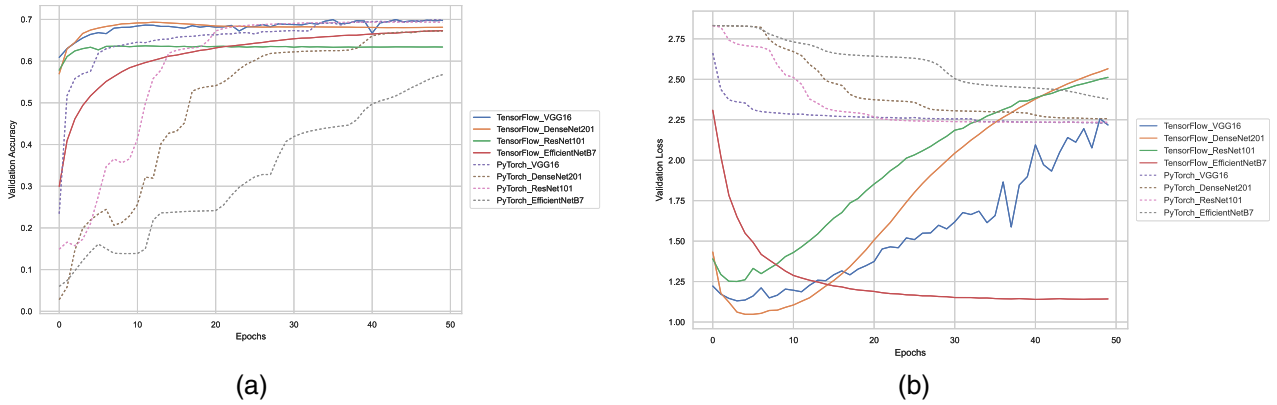

FIGURE 13 (a) Validation accuracy and (b) validation loss over epochs comparison on the so2sat dataset between the different frameworks, using different models.

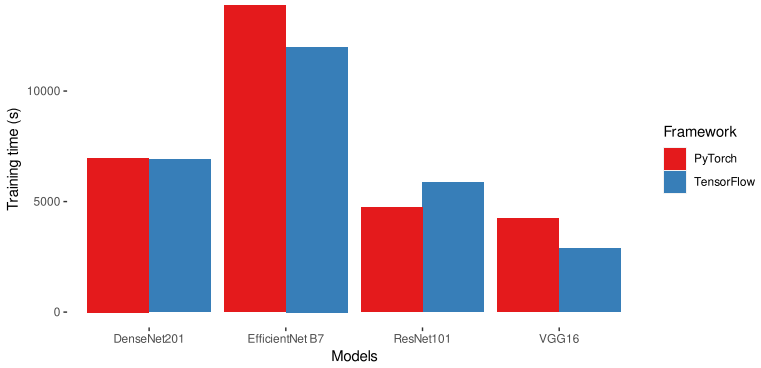

FIGURE 14 Training time comparison on the so2sat dataset between the different frameworks, using different models and 4 partitions per Kafka topic.

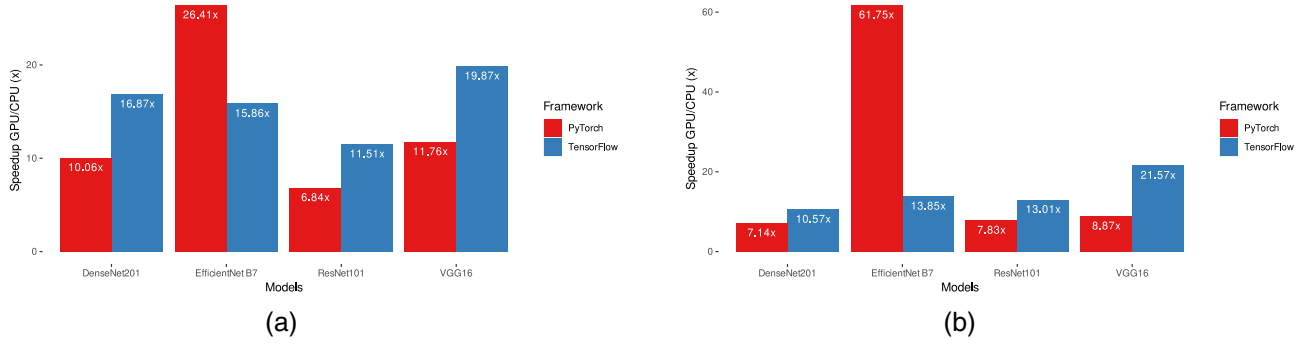

FIGURE 15 (a) GPU speedup against CPU time consumption of the eurosat dataset. (b) GPU speedup against CPU time consumption on the so2sat dataset)

1. One unpartitioned topic and each model replicated once.
2. One topic with two partitions and each model replicated twice.
3. One topic with four partitions and each model replicated four times.
4. One topic with eight partitions and each model replicated eight times.

Given the large number of models available and the small difference in terms of inference speed between them, we have decided to only evaluate the model with the lowest training time, which was VGG16-based model, as well as the model with the highest training time, which was EfficientNet-B7-based model.

Each test consists of sending 256 images and measuring how long the model takes to predict. This is done 20 times in order to obtain an average result. These tests will be performed with various configurations of replicas of inference components and various partitions in the Kafka topics. In addition, each test will be replicated several times with different numbers of clients (specifically with 1, 2, 4, 8, 16, and 32 clients for all cases and additionally 64 and 96 clients for higher availability scenarios), making requests, seeking to evaluate how the system behaves when faced with a multitude of requests. Furthermore, these tests will be carried out on the datasets presented above, providing us with a large battery of tests from which to draw some conclusions.

5.3.1 | Configuration 1: One unpartitioned Kafka topic and model replicated once

In this configuration, the inference components have been configured to read information from 1 broker, and the topics were created with 1 partition. Also, we deployed just1 component containing the model ready for inference. This scenario is the basic Kafka-ML use case: the use of a single broker and one partition per topic. Figure 16 depicts the average latency response with different number of clients.

It can be clearly observed that, as more simultaneous requests are made using more clients, the response latency shoots up, which indicates that the inference component or the Kafka topic is overloaded, and having a greater number of replicas of the model and a greater distribution of the data in Kafka would, a priori, improve this result. It can also be observed that, in terms of latency, there is no significant difference between the two frameworks.

For the subsequent conclusions, we will take the best and the worst result obtained from this one (which are 'TensorFlow EB7 so2sat' and 'PyTorch VGG16 eurosat' respectively), taking them as a reference for the comparison.

5.3.2 | Configuration 2: One topic in two partitions and each model replicated twice

With this configuration, two replicas of the inference component have been deployed, and the input and output topics have been partitioned into two. In this scenario (Figure 17a), a clear improvement is obtained, especially in cases of greater overload, such as those with a larger number of clients. Moreover, a significant reduction in latency in the stress cases can be seen.

5.3.3 | Configuration 3: One topic in four partitions and each model replicated four times

In this scenario, four replicas of the inference component have been deployed, and the Kafka topics have been partitioned into four partitions. A priori, this case should give us a clear improvement in the overload cases, although the results might worsen slightly with few clients. Looking at Figure 17b, it can be seen that our deduction is close to reality.

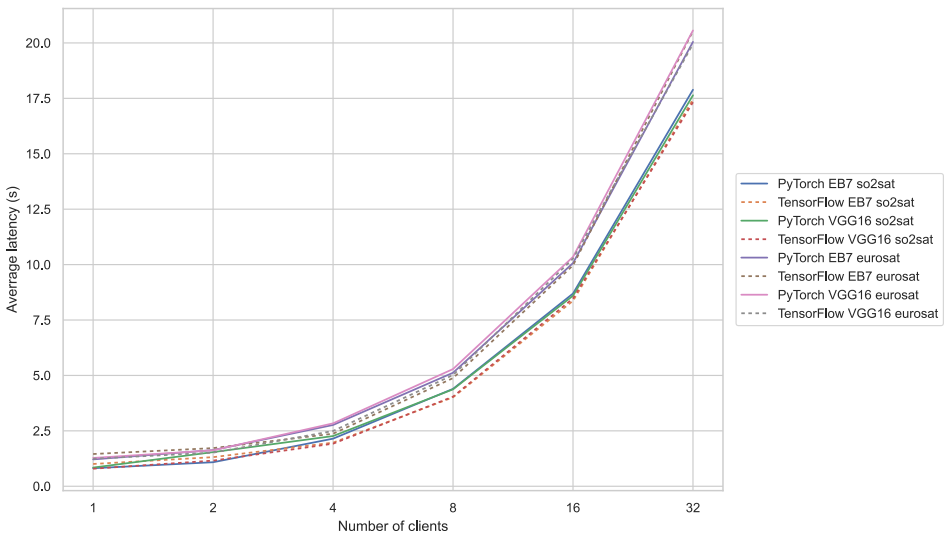

FIGURE 16 Average inference latency response with different number of clients (1 topic partition).

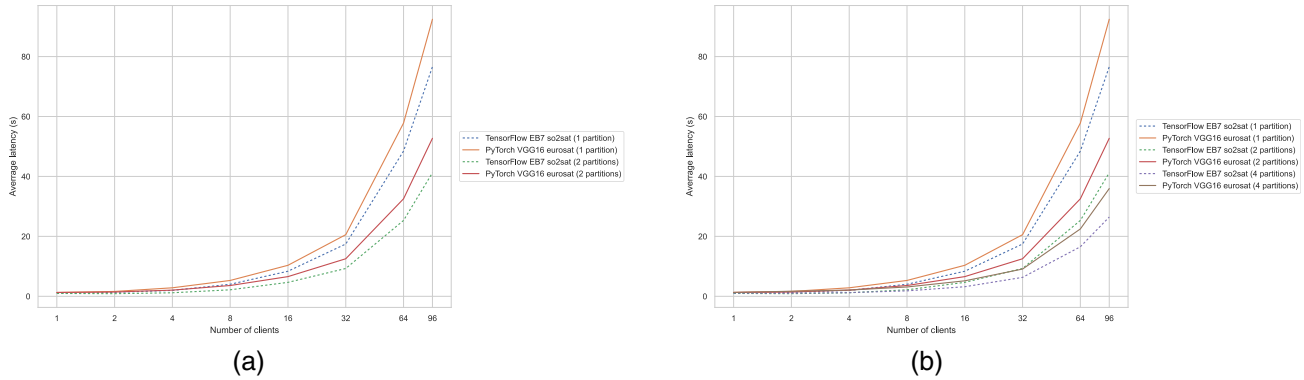

FIGURE 17 Average inference latency response with different number of clients (a) (2 topic partitions); (b) (4 topic partitions).

5.3.4 | Configuration 4: One topic in eight partitions and each model replicated eight times

In the last scenario, eight replicas have been deployed, and the topics have been divided into eight. In this case, slight improvements can already be observed for the established number of customers, but in case of a higher overload, given the previous results, it would perform better than the other ones.

Figure 18 shows a comparison between all the tests performed and the results obtained, and Tables 1 and 2 show the latency of all models evaluated with eight replicas and eight partitions grouped by dataset.

It can be concluded that having more replication and partitioning has considerably reduced latency in all the proposed cases. Furthermore, as a general rule, PyTorch models are faster in the lower demand cases and TensorFlow models in the higher demand cases, but the differences are barely noteworthy.

Regarding the deep learning models and unlike the training time, the EfficientNet-B7 model is slightly faster than VGG16. This is probably due to the implementation of the models and their management of resources, but what we should highlight is that the inference results for both, regardless of the dataset, are quite similar. Therefore, in most cases, we should not pay much attention at this slight difference but rather at the quality of the model during training, seeking to choose the one that gives us the best results for the objective we are pursuing.

In terms of fault tolerance, the deployment of Kafka-ML on Kubernetes and the usage of Kafka offers several advantages. Kubernetes provides self-healing mechanisms that automatically detect and recover from failures, ensuring that applications are always available. Additionally, Kafka provides data replication, which allows data to be read from multiple replicas in case of a broker failure. The combination of both ensures high availability and resilience to failures, making the application more robust, as shown in Kafka-ML: connecting the data stream with ML/AI frameworks (2022).

6 | CONCLUSIONS AND FUTURE WORK

In this article, Kafka-ML, a tool for the management of the ML/AI application pipeline with data streams, was redefined with the support of one of the most popular frameworks of recent times, PyTorch, the inclusion of GPU acceleration for faster training and inference, and improved user-friendly visualizations. PyTorch was not prepared to work with data streams and Apache Kafka, the backbone of Kafka-ML, and a new open-source data stream connector was released for this framework. Kafka-ML, unlike many other frameworks, which only support static data, is an open-source tool that allow ML/AI researchers to handle common ML/AI tasks with data streams, enabling them to work directly with data streams generators such as IoT devices. Kafka-ML can also work interchangeably and simultaneously with TensorFlow and PyTorch models via data streams.

To illustrate the potential of Kafka-ML, a Kafka-ML pipeline in the Petrochemical Industry 4.0 has been presented. Furthermore, the performance of the redefined architecture has been thoroughly evaluated in a state-of-the-art GPU infrastructure with popular deep learning models and in both TensorFlow and PyTorch frameworks using the capabilities of Kafka-ML. The evaluation has been carried out both at training and inference time, taking into account several scenarios that could be of interest in the platform. In addition to this, it has been evaluated that Kafka-ML is prepared for high availability if necessary.

During training tests, it has been noticed that while TensorFlow models obtain better accuracy results, PyTorch models are generally faster. The use of GPU acceleration has significantly improved the performance of the architecture. Concerning the inference tests, it is noted that with

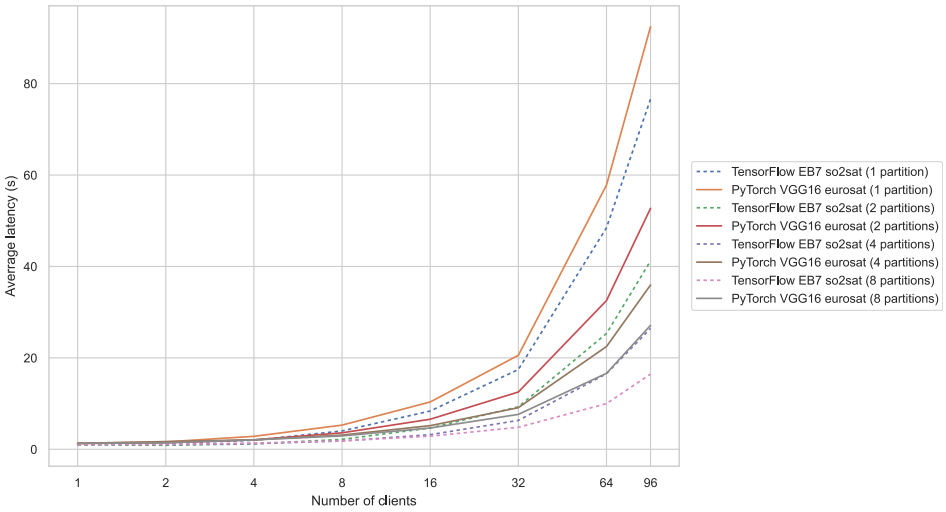

FIGURE 18 Average inference latency response with different number of clients, comparing different amount of replicas and partitions.

TABLE 1 Average inference latency (s) in a scenario with 8 partitions per topic and 8 replicas; so2sat dataset.

No. of clients	PTH EB7 so2sat	TF EB7 so2sat	PTH VGG16 so2sat	TF VGG16 so2sat
1	0.836123	1.005989	0.858307	1.001753
2	1.198585	1.097265	1.159442	0.97078
4	1.453084	1.347875	1.471743	1.18575
8	1.902492	1.84455	1.901407	1.663036
16	2.781597	2.859153	2.934127	2.580357
32	4.688964	4.807515	4.85404	4.593488
64	9.89461	9.995645	10.033936	9.812559
96	16.369561	16.459052	16.490979	16.297757

TABLE 2 Average inference latency (s) in a scenario with 8 partitions per topic and 8 replicas; eurosat dataset.

No. of clients	PTH EB7 eurosat	TF EB7 eurosat	PTH VGG16 eurosat	TF VGG16 eurosat
1	1.275998	1.427651	1.279919	1.280507
2	1.58691	1.395471	1.42447	1.503619
4	2.044637	1.794421	2.046851	1.975691
8	3.000125	2.691809	2.92808	2.895541
16	4.608588	4.251507	4.657033	4.532421
32	7.67565	7.250512	7.634325	7.550847
64	16.651685	16.286778	16.615675	16.544145
96	27.149051	26.830539	27.118188	27.054735

a higher number of replicas and partitions, better results are obtained in cases of higher demand regardless of the framework used, being the results among them pretty similar.

As future work, we have set ourselves the following improvements to Kafka-ML:

1. Automatic optimization of hyperparameters. This is a very useful functionality applicable to the development of ML/AI models because of the convenience and time-saving nature of finding a relatively fine-grained solution. For easy integration of this functionality, one option is to integrate parts of the Ray Tune Official Webpage (n.d.) tool together with one of the Kafka-ML modules.
2. Online training. Another functionality that we think could be interesting to add would be the option to allow online training. This means that the model would never stop training until certain conditions are met, and it would automatically publish its improvements from time to time.

This would be very useful for Industry 4.0 models that could be trained with real time data from the process and obtaining better and better results.

3. Integration with new frameworks. Given the new component breakdown, Kafka-ML now allows easy integration with new frameworks that may appear or be of interest to users.
4. Improving the user experience. There are several options where we could improve the average user experience. For example, designing a new view where the user can get a clear graphical view of the results of the inference predictions. Another section to improve is the definition of models, by embedding a code editor that allows to define models in a simpler way in the platform itself.
5. Integrate other processing tasks. Many applications, such as Industry 4.0 cases, may use ML/AI but also other statistical and preprocessing or/and postprocessing tasks that may require the same data stream. Therefore, Kafka-ML could also manage these non-ML/AI tasks to integrate them with the data stream utilized.
6. Improving information tracking. A convenient functionality would be the ability to link the datasets to the deployed models as well as to their hyperparameters or other related data, allowing users to track and filter the models according to these data.

AUTHOR CONTRIBUTIONS

Antonio Jesús Chaves: Software development and evaluation, First manuscript draft. **Cristian Martín**: Supervised the research, Conceptualization, Manuscript review. **Manuel Díaz**: Supervised the research, Conceptualization, Manuscript review, Funding.

ACKNOWLEDGEMENTS

This work is funded by the Spanish projects RT2018-099777-B-100 ('rFOG: Improving Latency and Reliability of Offloaded Computation to the FOG for Critical Services'), PY20_00788 ('IntegraDos: Providing Real-Time Services for the Internet of Things through Cloud Sensor Integration'), TSI-063000-2021-116 ('Digital vertical twins for B5G/6G networks'), TED2021-130167B-C33 ('GEDIER: Application of Digital Twins to more sustainable irrigated farms'), and CPP2021-009032('ZeroVision: Enabling Zero impact wastewater treatment through Computer Vision and Federated AI'); and the European project LIFEWATCH-2019-11-UMA-01-BD ('EnBiC2-Lab - Environmental and Biodiversity Climate Change Lab'). This project has received funding from the European Union's Horizon Europe research and innovation proframme under the Marie Skłodowska-Curie grant agreement No 101086218. Funding for open access charge: Universidad de Málaga / CBUA.

CONFLICT OF INTEREST STATEMENT

The authors declare no potential conflict of interest.

DATA AVAILABILITY STATEMENT

The open data used in this article and their usage are referenced in our GitHub official repository. The CEPSA use case data are private for confidentiality reasons.

ORCID


Antonio Jesús Chaves https://orcid.org/0000-0002-4523-7461

ENDNOTES

[1] PyTorch Kafka Dataset GitHub. https://github.com/ertis-research/PyTorchKafkaDataset

[2] Kafka-ML Github. https://github.com/ertis-research/kafka-ml/

[3] GPU Share Scheduler Extender GitHub. https://github.com/AliyunContainerService/gpushare-scheduler-extender/

[4] NVIDIA device plugin for Kubernetes Github. https://github.com/NVIDIA/k8s-device-plugin/tree/v0.7.0

[5] A message used in Kafka to indicate a job there are training data available.

[6] PyTorch Kafka Dataset Github. https://github.com/ertis-research/PyTorchKafkaDataset

[7] Logical group of models to deploy them together to train in parallel.

[8] Pickle Python Library Documentation. https://docs.python.org/3/library/pickle.html

[9] PyTorch Ignite Official Website. https://pytorch.org/ignite/

REFERENCES

Algorithmia MLOps Official Webpage. (n.d.). Available from: https://algorithmia.com/mlops [last accessed May 2022].

Azure Machine Learning Official Webpage. (n.d.). Available from: https://azure.microsoft.com/es-es/services/machine-learning/ [last accessed May 2022].

ClearML. (2019). Clearml - your entire mlops stack in one open-source tool. Retrieved from https://clear.ml/ Software available from http://github.com/allegroai/clearml

cnvrg.io Official Webpage. (n.d.). Available from: https://cnvrg.io/ [last accessed May 2022].

Dai, H., Peng, X., Shi, X., He, L., Xiong, Q., & Jin, H. (2022). Reveal training performance mystery between tensorflow and pytorch in the single gpu environment. *Science China Information Sciences*, *65*, 1–17. https://doi.org/10.1007/s11432-020-3182-1

Díaz, M., Martín, C., & Rubio, B. (2016). State-of-the-art, challenges, and open issues in the integration of internet of things and cloud computing. *Journal of Network and Computer Applications*, *67*, 99–117.

Doleck, T., Lemay, D., Basnet, R., & Bazelais, P. (2020). Predictive analytics in education: a comparison of deep learning frameworks. *Education and Information Technologies*, *25*, 1–13. https://doi.org/10.1007/s10639-019-10068-4

El Shawi, R., Wahab, A., Barnawi, A., & Sakr, S. (2021). Dlbencsh: a comprehensive experimental evaluation of deep learning frameworks. *Cluster Computing*, *24*, 1–22. https://doi.org/10.1007/s10586-021-03240-4

Feast Official Webpage. (n.d.). Available from: https://feast.dev/ [last accessed May 2022].

Florencio, F., Silva, T., Ordonez, E., & Júnior, M. (2019). Performance analysis of deep learning libraries: Tensorflow and pytorch. *Journal of Computer Science*, *15*, 785–799. https://doi.org/10.3844/jcssp.2019.785.799

Google Cloud Automl Official Webpage. (n.d.). Available from: https://cloud.google.com/automl [last accessed May 2022].

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. *arXiv*. https://arxiv.org/abs/1512.03385

Helber, P., Bischke, B., Dengel, A., & Borth, D. (2017). Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification.

Huang, G., Liu, Z., van der Maaten, L., & Weinberger, K. Q. (2016). Densely connected convolutional networks. *arXiv*. https://arxiv.org/abs/1512.03385

iguazio Official Webpage. (n.d.). Available from: https://www.iguazio.com/ [last accessed May 2022].

Kafka-ML Official Github Repository. (n.d.). Available from: https://github.com/ertis-research/kafka-ml/ [last accessed May 2022].

Kubeflow Official Webpage. (n.d.). Available from: https://www.kubeflow.org/ [last accessed May 2022].

Martín, C., Langendoerfer, P., Zarrin, P. S., Díaz, M., & Rubio, B. (2022). Kafka-ml: Connecting the data stream with ML/AI frameworks. *Future Generation Computer Systems*, *126*, 15–33.

Metaflow Official Webpage. (n.d.). Available from: https://metaflow.org/ [last accessed May 2022].

MLflow Official Webpage. (n.d.). Available from: https://mlflow.org/ [last accessed May 2022].

MLRun Official Webpage. (n.d.). Available from: https://www.mlrun.org/ [last accessed January 2023].

Montiel, J., Halford, M., Mastelini, S. M., Bolmier, G., Sourty, R., Vaysse, R., Zouitine, A., Gomes, H. M., Read, J., Abdessalem, T., & Bifet, A. (2021). River: Machine learning for streaming data in python. *Journal of Machine Learning Research*, *22*(110), 1–8 Retrieved from http://jmlr.org/papers/v22/20-1380.html

Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z., Paul, W., Jordan, M. I., & Stoica, I. (2018). Ray: A distributed framework for emerging ai applications. In *13th usenix symposium on operating systems design and implementation (osdi 18)* (pp. 561–577). USENIX Association.

NVIDIA DIGITS Official Webpage. (n.d.). Available from: https://developer.nvidia.com/digits [last accessed May 2022].

Pachyderm Official Webpage. (n.d.). Available from: https://www.pachyderm.com/ [last accessed January 2023].

Ray tune official webpage. (n.d.). Available from: https://docs.ray.io/en/latest/tune/index.html [last accessed June 2022].

SigOpt Official Webpage. (n.d.). Available from: https://sigopt.com/ [last accessed January 2023].

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv*. https://arxiv.org/abs/1409.1556

Tan, M., & Le, Q. V. (2020). Efficientnet: Rethinking model scaling for convolution neural networks. *arXiv*. https://arxiv.org/abs/1905.11946

Wan, Z., Zhang, Z., Yin, R., & Yu, G. (2022). Kfiml: Kubernetes-based fog computing iot platform for online machine learning. *IEEE Internet of Things Journal*, *9*, 19463–19476.

What is Amazon Sagemaker? (n.d.). Available from: https://docs.aws.amazon.com/sagemaker/latest/dg/whatis.html [last accessed May 2022].

Zhu, X., Hu, J., Qiu, C., Shi, Y., Bagheri, H., Kang, J., Mou, L., Bagheri, H., Häberle, M., Hua, Y., Huang, R., Hughes, L., Li, H., Sun, Y., Zhang, G., Han, S., Schmitt, M., & Wang, Y. (2018). So2sat lcz42 [Dataset]. Technical University of Munich. Retrieved from https://mediatum.ub.tum.de/1454690

AUTHOR BIOGRAPHIES

Antonio Jesús Chaves received his BSc in Computer Science Engineering and his M.Sc. in Software Engineering and Artificial Intelligence from the University of Málaga, Spain, in 2021 and 2022, respectively. He is currently a PhD student at the ERTIS Research Group, University of Málaga, and a member of the ITIS Software Institute, University of Málaga. His research interests include modern deep-learning techniques, such as object recognition, fault detection, federated learning, and artificial intelligence applied to the IoT field.

Cristian Martín received a BSc in Computer Engineering, a MSc in Software Engineering and Artificial Intelligence and a PhD in Computer Science from the University of Málaga in 2014, 2015 and 2018 respectively. Currently, he is an Assistant Professor at the University of Málaga and at the ITIS Software. Previously, he has been working as a software engineer in various tech companies with the RFID technology and software development. His research interests focus on the adoption of the Internet of Things, cloud, fog and edge computing, deep learning, low-latency applications and structural health monitoring.

Manuel Díaz is Full Professor in the Computer Science Department at the University of Málaga, Head of the ERTIS research group, and a member of the ITIS software Institute. His research interests are in distributed and real-time systems, Internet of Things, and P2P, especially in the context of middleware platforms and critical systems. In the last years, his main area of work has been WSN and monitoring systems, especially energy monitoring (FP7 e-balance project), water infrastructure monitoring (FP7 SAID project), and energy efficient buildings (FP7 SEEDS). He has collaborated in many technology transfer projects with different companies such as Tecnatom, Telefónica, Indra, or Abengoa.

He was the coordinator of the FP6 SMEPP project and main researcher for UMA in the WSAN4CIP (ICT FP7). He is also co-founder of the spin-off Softcrits and head of its RandD department.

How to cite this article: Chaves, A. J., Martín, C., & Díaz, M. (2023). The orchestration of Machine Learning frameworks with data streams and GPU acceleration in Kafka-ML: A deep-learning performance comparative. *Expert Systems*, e13287. https://doi.org/10.1111/exsy.13287