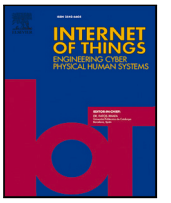
Internet of Things 25 (2024) 101036

Contents lists available at ScienceDirect

Internet of Things

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



Towards flexible data stream collaboration: Federated Learning in Kafka-ML

Antonio Jesús Chaves *, Cristian Martín, Manuel Díaz

ITIS Software, University of Malaga, Malaga, Spain

ARTICLE INFO

Dataset link: https://www.cs.toronto.edu/~kriz/cifar.html

Keywords:
Kafka-ML
Data streams
Deep learning
Internet of Things
Federated learning

ABSTRACT

Federated learning is applied in scenarios where organisations lack sufficient data volume for modelling their business logic and cannot share their data with external parties. Moreover, Industry 4.0 and IoT scenarios generate massive data streams, which normally are fed to ML/AI solutions for model training and prediction. However, in most cases, ML/AI frameworks are not prepared to work with these streaming pipelines. In this paper, we present an asynchronous federated learning solution based on the Kafka-ML data stream framework, which is able to combine federated learning and data stream capabilities within ML/AI applications. While most federated learning approaches are tailored to a specific ML model or a use case, the solution provided adapts itself to the availability of both data and ML models, achieving a flexible and dynamic federated learning solution. To validate its performance, an evaluation of the federated learning solution is carried out on different scenarios in a multi-node state-of-the-art infrastructure. Results show that this framework can work with multiple federated clients, being the resulting accuracy dependent on the amount of data and the behaviour of clients during training.

1. Introduction

A massive amount of data is generated every second thanks to the advances made in technology in recent years. Furthermore, this data are being generated by devices that are becoming more powerful in terms of computing power, such as smartphones, smart cameras, sensor networks in factories, etc. Thanks to the increasing computational power of these devices, it is possible to train models capable of predicting certain behaviours related to what is monitored by these devices (e.g. vehicle tracking, operational parameters monitoring, etc.) [1]. However, these devices sometimes have limited data, which makes it unviable to train a model with thousands of parameters and have an acceptable accuracy [2].

In general, in most Machine Learning (ML)/Artificial Intelligence (AI) scenarios, all the data produced from devices were transmitted to a central server where a common model was trained for all of them. However, this may lead to some hesitation by users about the privacy of their data. In certain cases, such as those involving medical or multiple organisation data, it should not be possible for anyone other than the model to see the information. To address this problem, federated learning appeared.

Federated learning [3] is a distributed machine learning methodology where a global model is created by aggregating locally trained models from multiple clients. Unlike centralised learning, where data are sent to a central server, federated learning only transmits model updates, ensuring data privacy and efficiency in communications. However, new challenges emerge, e.g., how the models will be aggregated, how we deal with non-independent and identically distributed data (non-IID), client disconnections, etc.

* Corresponding author.
E-mail addresses: chaves@uma.es (A.J. Chaves), cristian@uma.es (C. Martín), mdiaz@uma.es (M. Díaz).

https://doi.org/10.1016/j.iot.2023.101036
Received 27 September 2023; Received in revised form 27 October 2023; Accepted 14 December 2023
Available online 9 January 2024

The data generated by the IoT devices should potentially be sent through data streams, meeting IoT needs and ideal for tracking multiple features. One of the best-known data stream platforms is Apache Kafka [4]. It allows a multitude of replication, QoS, and partitioning configurations, making it easy to distribute load, and manage data streams.

Despite the evident advancements in IoT data streams and ML/AI fields in recent years, they currently lack adequate coordination with each other. For example, while most present ML/AI frameworks are well-suited for handling static datasets like files and storage systems, they struggle with data streams. This limitation becomes particularly evident when trying to manage the lifecycle of ML models from data streams, requiring a diverse array of software architectures, underlying technologies, and tools for tasks such as dataset creation, data storage, and model training. Moreover, ML/AI frameworks do not support the reuse of data streams for training and evaluating multiple models simultaneously, nor do they facilitate scalability in such processes. To address these shortcomings, Kafka-ML [5] was introduced. Kafka-ML is an open-source framework designed to manage the lifecycle of ML/AI applications using data streams on scalable platforms. Through a user-friendly web interface, Kafka-ML allows users to easily define ML models, train and evaluate their performance, and seamlessly deploy them for inference. Model training and inference are integrated with data streams, eliminating the traditional isolation of these data-consuming phases.

In this work, we have considered a federated learning solution powered by data streams through Kafka-ML. This capability is crucial for distributed machine learning, allowing multiple devices or clients to collaborate on model training while keeping their data decentralised and secure. Thanks to the usage of continuous data streams, we were able to devise an asynchronous federated learning solution. By leveraging Kafka's messaging system, we established a communication channel between the different clients and the Kafka-ML Platform, enabling them to exchange model parameters and weights efficiently. This enables IoT devices in decentralised scenarios to connect and collaborate in changing environments. This asynchronous approach not only enhanced the scalability of the system but also addressed privacy concerns by keeping sensitive data localised on individual clients, making the federated training process more robust and privacy-preserving. It also allows organisations to collaborate in federated tasks automatically and dynamically when they have a new dataset to train collaboratively. By doing so, they can collaborate on federated models as data becomes available, which, unlike many federated learning works, makes it a flexible federated learning solution.

The rest of the paper is structured as follows. Section 2 presents the related work and their differences against the implementation of federated training into Kafka-ML. Section 3 presents the new architecture of Kafka-ML. Section 4 explains the pipeline of a federated training job in Kafka-ML. Section 5 provides an evaluation over different test cases. Finally, Section 6 provides a concise overview of the study and explores potential avenues for future growth.

2. Background and related work

2.1. Background

Federated learning [6] is a machine learning approach designed to address the challenges of privacy and decentralisation in the data processing. It enables multiple clients to collectively train a shared machine learning model without sharing their data. Instead of centralising data, federated learning only communicates model updates, ensuring that individual data sources remain secure and private. Thanks to this, models are trained in a decentralised way, eliminating the need for extensive data transfers, reducing bandwidth usage, and enhancing overall efficiency. Thanks to its decentralised nature, the system can operate effectively even in the presence of network interruptions or device unavailability, ensuring the reliability of the learning process. Moreover, model updates can be securely aggregated on a central server employing cryptographic techniques to guarantee that individual data cannot be reconstructed from these updates, maintaining data privacy.

As part of federated learning, there are two main concepts, synchronous and asynchronous federated learning [7]:

- In synchronous federated learning, the participating clients are synchronised and follow a coordinated schedule, training at the same time slots and waiting for all the clients to end in order to aggregate all the client models together. Then the new aggregated model is sent back to the clients, and this is repeated until a stop flag is raised (such as a maximum number of rounds). This kind of federated learning ensures that all clients are on the same page at each training iteration. However, this approach requires strict synchronisation, which can introduce delays and communication bottlenecks if some clients have slower or unreliable network connections.
- In the case of asynchronous federated learning, clients operate independently and asynchronously. The training phase is then not synchronised between clients. In this case, each client trains its model with its data and sends its update back to a central server, which updates its reference model as it receives models from federated clients and sends it back to the clients for retraining. This allows clients to operate without synchronisation restrictions, which allows faster and more flexible training.

While federated learning offers several advantages, it also comes with its set of challenges and drawbacks. Understanding these limitations is crucial for a comprehensive evaluation of its suitability for specific use cases. One significant challenge is non-IID data management. Federated learning assumes that data on each client is similar and independently distributed. However, this assumption is usually not true in a real scenario. Data discrepancies across clients may interfere with model convergence, as the overall model may have difficulty understanding the diversity of data distributions. A similar challenge is outlier or noisy data management. Federated learning systems may be more susceptible to outliers or noisy data on individual clients, as the global model can be influenced by poorly performing local models. This can lead to challenges in maintaining the quality and accuracy of the global model. Also, resource constraints may limit the application of federated learning in certain settings. Clients participating in federated learning must possess the computational capabilities to perform local model updates. In resource-constrained environments, this requirement can be a limiting factor.

2.2. Related work

To the best of our knowledge, Kafka-ML is one of the first open-source tools to integrate the asynchronous federated learning pipeline and data streams. Nevertheless, there are other tools with comparable features that are mentioned here.

First of all, it is worth mentioning a certain number of tools that enable federated learning between organisations. These include TensorFlow Federated (TFF) [8], an open-source framework by Google for developing machine learning models on decentralised devices, enabling data privacy-preserving and collaborative learning across these devices. TFF provides APIs for defining federated algorithms, supporting various optimisation techniques and customisation options. There are other similar proposals, such as PySyft [9], which is an open-source Python library that enables privacy-preserving and secure machine learning. Developed by OpenMined, PySyft allows the implementation of privacy-enhancing techniques like federated learning, secure multi-party computation, and differential privacy. It extends PyTorch functionality, allowing distributed training on data across multiple parties while ensuring privacy protection. Other projects with similar objectives developed by large organisations include FATE [10], by FederatedAI, and OpenFL [11], developed in collaboration between Intel and the University of Pennsylvania. The major drawback of these frameworks/libraries is that they can be challenging for several reasons. Firstly, the environment has to be configured in order to work correctly between all the clients/organisations. Subsequently, users must adapt the classic machine learning workflow to these libraries, which is an additional effort and could be complex for users with limited knowledge of the area.

Flower [12] is another open-source federated learning framework designed to enable collaborative machine learning on decentralised data sources. It allows multiple parties to train a shared model without directly sharing their sensitive data, employing a client–server architecture, where clients represent data owners (e.g., devices or edge servers), and the server coordinates the training process. During training, clients compute gradients locally using their data and then share these gradients with the server. The server aggregates these gradients and updates the global model, which is then redistributed to the clients. This iterative process continues until the model converges. Although Flower has many strengths, including multiple model aggregation functions, a simple deployment, and a user-friendly API, there are also a few shortcomings. Flower only supports synchronous federated learning, which can limit the training process. Moreover, it does not have support for training from data streams.

FedLess [13] and Lambda-ML [14] are novel frameworks that leverage the power of serverless computing to address the challenges faced by conventional federated learning systems. They address challenges such as latencies or scalability by integrating federated learning with serverless functions. Although these federated learning frameworks are highly attractive for deploying and comparing their capabilities in cloud environments, they are outside the scope of our goal, which is to adapt the federated learning concept into Kafka-ML by using data streams, making it easy to use with IoT data.

Project Florida from Microsoft [15] is a system architecture and software development kit (SDK) designed for large-scale federated learning implementations in a heterogeneous device environment. Project Florida simplifies the deployment of cross-device federated learning solutions through cloud-hosted infrastructure, task management interfaces, and a multi-platform SDK supporting major programming languages. The platform abstracts complexities related to federated learning algorithms, communication protocols, and security mechanisms, providing user-friendly APIs for efficient, privacy-preserving, and robust federated learning deployment. While it is an ambitious platform with promising results, it is a proprietary platform developed for Microsoft's Azure cloud environment, unlike our work.

Chen et al. present ASO-Fed [16], which deals with both incremental training and streaming data, achieving great results. Since Kafka-ML enables the integration of incremental learning, future work could integrate this methodology with federated learning. Although great results were observed, we did not find a way to deploy the framework implementation in order to replicate the results.

FederatedScope is an innovative federated learning platform presented by Alibaba [17]. It addresses the challenges arising from the heterogeneity of federated learning solutions by employing an event-based architecture. FederatedScope allows clients to describe their behaviours when events arise (e.g. data arrival), enabling flexibility when training federated learning models. While it allows many interesting functionalities, such as attack protection or high customisation, we did not see support for training with data streams, ideal for Industry 4.0 use cases.

All the works under consideration demonstrate advancements in federated learning. Kafka-ML distinguishes itself by seamlessly integrating asynchronous federated learning with real-time data streams in a flexible, open-source manner, effectively addressing the needs of Industry 4.0 applications. Within the first tool block, including TFF, PySyft, FATE, and OpenFL, Kafka-ML stands out for its simplified implementation, ease of use, and support for data streams. Concerning Flower, besides the shortcomings mentioned in the previous block, we have not found support for asynchronous federated learning, thus deviating from our objective. FedLess and Lambda-ML are relevant works but outside of our scope. Project Florida and ASO-Fed are well-defined but lack open-source implementations and data stream focus. FederatedScope offers customisation but has deployment challenges and is not compatible with data streams. Consequently, we assert that Kafka-ML uniquely supports asynchronous federated learning with real-time data streams, offering flexibility, open-source availability, and ease of deployment.

3. Federated architecture with data streams

Kafka-ML manages ML/AI pipelines through data streams. In this work, a new module of Kafka-ML has been defined and some modifications have been introduced into the latest version of Kafka-ML to enable the possibility of train models in a federated way.

The Kafka-ML architecture is composed of service sets, following the single responsibility principle, to create a microservice-based system. Each component operates within Docker containers, ensuring isolation and easy portability. Managing and deploying this

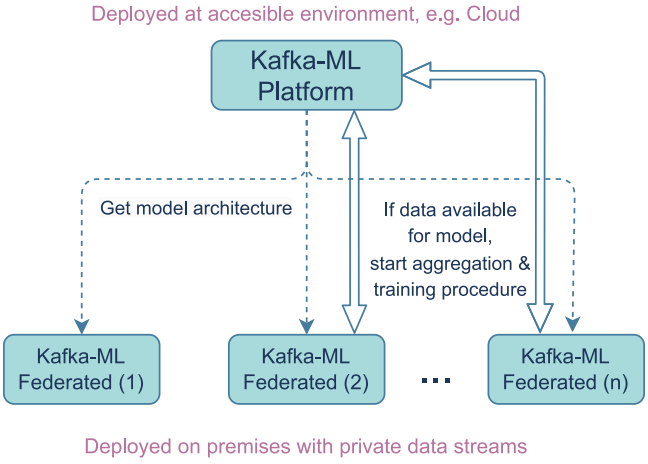


Fig. 1. Kafka-ML Federated architecture deployment overview.

platform across a cluster of nodes and distributed production infrastructures is accomplished through Kubernetes orchestration, facilitating continuous monitoring of Kafka-ML services and deployed tasks. All the new modules implemented for Kafka-ML Federated follow the same approach, allowing to manage all the clusters acting as clients from Kubernetes.

Fig. 1 presents the architecture of this work, showing the Kafka-ML Platform and the communications with the federated devices (Kafka-ML Federated). The Kafka-ML Platform is expected to be deployed in an accessible architecture by devices and/or organisations that are willing to collaborate in training, e.g. the Cloud. The Kafka-ML Platform will enable the deployment of ML models and the orchestration (and aggregation) of multiple Kafka-ML Federated clients for the federated process. Kafka-ML Federated will be deployed on training devices with access to data streams and will carry out the federated training with the ML models defined in the Kafka-ML Platform and the data available locally.

In the following subsections, we will describe the different modules of Kafka-ML as well as the changes and new developments that have been made.

3.1. Kafka-ML platform architecture

Fig. 2 shows the new architecture of the Kafka-ML framework, presenting the different functionalities separated by modules and highlighting the modifications presented in this work, emphasising the separation between Kafka-ML Platform (the original framework with slight modifications) and Kafka-ML Federated (a new module to enable federated model training using data streams).

The architecture comprises a frontend and backend module that handle user requests, such as model definition, configuration deployment, metrics visualisation, etc. These two modules have been slightly modified as well as the backend views that receive the user's information in order to collect additional data in case the user wants to deploy federated models. The definition of models has not changed, so models are defined in the way it was done in Kafka-ML, i.e., users can access the user-friendly web interface to define the models easily, even for non-technical users.

The rest of the modules of the Kafka-ML Platform architecture have not been modified. These are the "Platform Data Control Logger" (formerly "Control Logger"), which takes metadata of the user data sent to Kafka-ML Platform for training to generate the datasources which the models will target for training. Then, "Model training" and "Model inference" modules are deployed when the user wants to train or serve a model. In the case of training, these can be classic training in both TensorFlow and PyTorch [18], distributed model training [19], incremental model training, or the union of these with GPUs, if specified by the user. In the case of inference, the user can deploy the models by specifying brokers and Apache Kafka input/output topics, as well as replication configuration and deployment details in the case it is desired to deploy to another Kubernetes cluster.

"Federated Training Module" is the new module developed in this work. This module orchestrates the training on devices in a federated way. This module communicates with Apache Kafka to deposit the model metadata and weights. Once the training by the "Kafka-ML Federated" starts, this module will wait to receive the trained weights to aggregate and send them back afterwards, ensuring that every federated client receives the latest update. Then the weights between the various models trained from "Kafka-ML Federated" clients are aggregated and converge to a conforming model. In Section 4, the operation of asynchronous federated training using Apache Kafka will be explained in more detail.

Kafka-ML, along with its latest enhancements, implementation, configuration files, and several illustrative examples, can be found in the GitHub repository.[1]

[1] Kafka-ML GitHub repository: https://github.com/ertis-research/kafka-ml

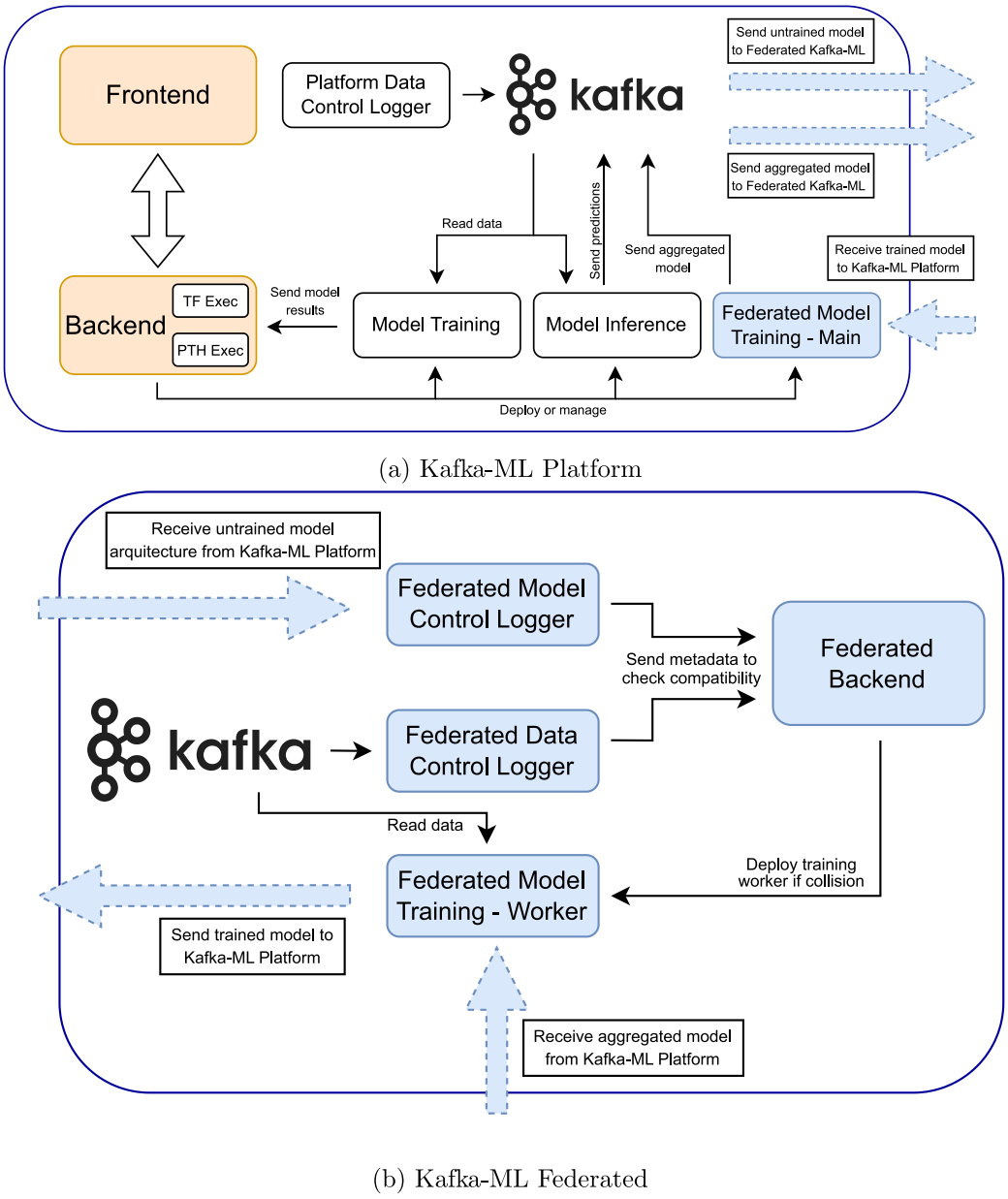


Fig. 2. Kafka-ML Architecture divided into Kafka-ML Platform and Kafka-ML Federated modules. Orange component: modified; blue component: added.

3.2. Kafka-ML federated architecture

The part of the architecture described as "Kafka-ML Federated" (Fig. 2(b)) consists of all the new modules developed to enable the asynchronous federated model training. Mainly, 4 modules have been developed: "Federated Data Control Logger", "Federated Model Control Logger", "Federated Backend" and "Federated Model Training - Worker". Also, each "Kafka-ML Federated" client deployed has its own Apache Kafka deployed to keep its data privately.

"Federated Data Control Logger" and "Federated Model Control Logger" are modules that monitor different Apache Kafka topics in order to track metadata, from models and data, to deploy the corresponding training task. In the case of "Federated Data Control Logger", the behaviour is very similar as the "Platform Data Control Logger", expecting a new field with metadata of the dataset so the training task can be automatically deployed when the right conditions are met. This monitoring is done on the Apache Kafka deployed in Kafka-ML Federated, unlike "Federated Model Control Logger", which monitors the Apache Kafka deployed in the Kafka-ML Platform for model updates. "Federated Model Control Logger" retrieves the metadata from the models published by the

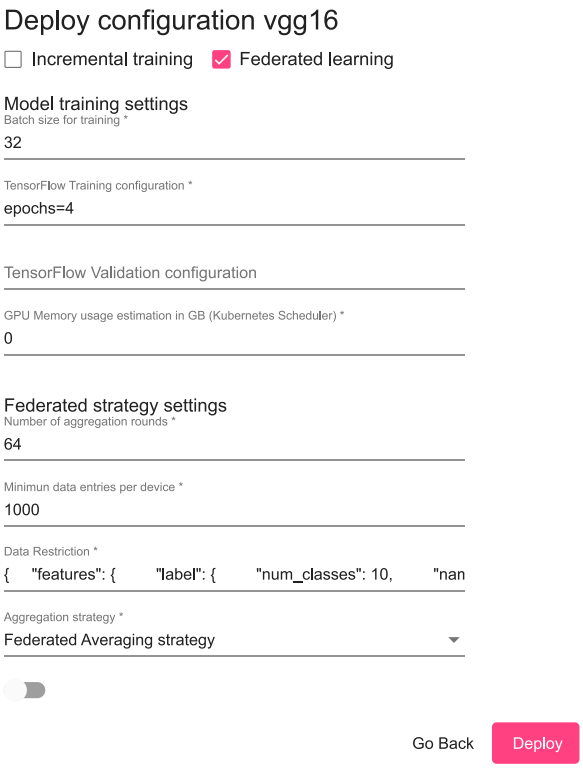


Fig. 3. Federated model training deployment form in Kafka-ML.

Kafka-ML Platform training module in a similar way. All the information collected is sent to the "Federated Backend", where it is analysed and if the conditions are met, a new training task is deployed.

"Federated Backend" is a tiny backend module developed with Django & Python. This module automatically manages the requests sent by the aforementioned control loggers to check if there is compatibility between the input data streams received in Apache Kafka locally and the defined models. If so, it will deploy a new Kubernetes job to start a federated training worker.

Finally, "Federated Model Training - Worker" is the module where models are received and trained with data streams from the federated section. Thanks to the metadata previously obtained from the "Federated Backend" about the available models in the Kafka-ML Platform, as well as the available data streams from Kafka-ML Federated (thus preserving the privacy of the information), it reconstructs the model and trains it through data streams. Finally, it sends it back the trained model to the Kafka-ML Platform for aggregation and to continue the federated training cycle in an asynchronous way.

4. Pipeline of an asynchronous federated training task

In this section, the workflow for carrying out an asynchronous federated training task in Kafka-ML is explained, highlighting the orchestration of various tasks and the integration of asynchronous model updates. The proposed workflow addresses the challenges caused by the heterogeneity of the participating clients and their data streams, and the possible instability of the network connections (they can reconnect), looking for model convergence and preserving the privacy and security of the data.

First of all, as in the rest of the Kafka-ML deployments, the user will define the architecture of the models to be deployed, as well as group them in configurations if desired to deploy them all simultaneously.

Once the configuration is created and the user wants to deploy the models for federated training, the user will have to fill in some additional fields. These fields mainly include the number of aggregation rounds, the aggregation technique to be used as well as certain restrictions that the data must meet for model training. We do this to determine, for example, the minimum amount of data we want to the model to be trained on, the class types, and other characteristics users want for input data streams to have. Fig. 3 shows how users can deploy their configurations for training in a federated way specifying the corresponding restrictions.

Once the model is deployed for federated training, its metadata as well as its architecture and initial weights are published in the Apache Kafka at Kafka-ML Platform. This allows all deployed Kafka-ML Federated clients, which permanently listen to the "Model Control Topic", to be alerted when a new model becomes available.

When a Kafka-ML Federated client detects a new model, it checks if there is any data available in its federated Apache Kafka that can be used for the training. This matching is done by checking dataset features like shape and properties (e.g., number of

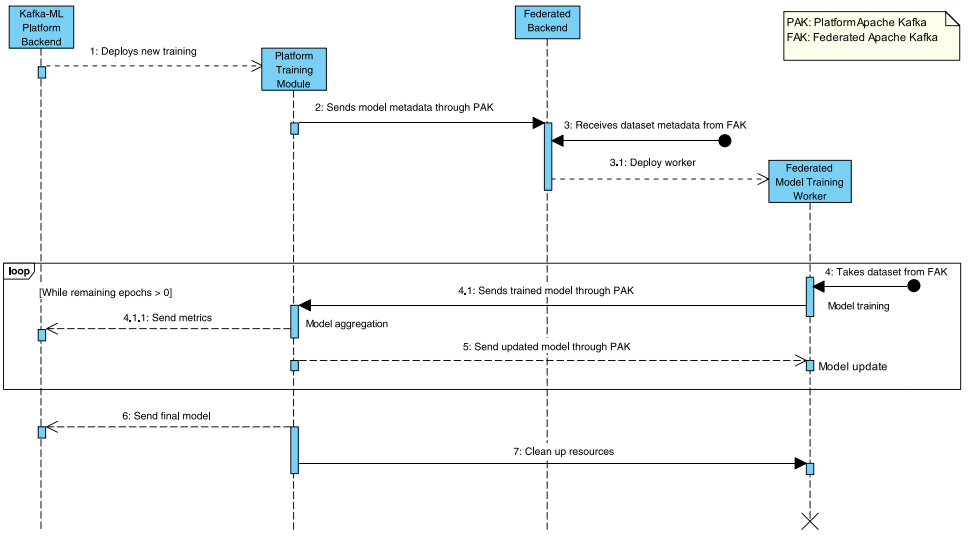


Fig. 4. Federated model training pipeline in Kafka-ML.

classes, labels, etc.) through schema matching (a JSON description of the dataset properties). For example, the matching schema for the MNIST dataset is shown in Listing 1. Matching schemes are defined by end-users in the Kafka-ML Platform.

```
{
  "features": {
    "label": {
    "num_classes": 10,
    "names":
      ["0", "1", "2", "3", "4",
       "5", "6", "7", "8", "9"]
    }
  },
  "supervised_keys": {
    "input": "image", "output": "label"
  }
}
```

Listing 1: Matching schema of the MNIST dataset.

If matching schemes are equivalent, and model input and dataset shapes match, the Kafka-ML Federated client will deploy a "Federated Model Training - Worker" task in which the model will be loaded, trained with data from the Apache Kafka Federated, and returned to the Apache Kafka at Kafka-ML Platform (publishing a control message and the trained weights). In the case there is no data available for that model, the model metadata will be stored in the "Federated Backend" and checked against future incoming data to determine whether the streaming data is valid for the model. If so, the federating learning starts as in the previous case.

Continuing with the training, once the weights are ready and published by the federated clients, the "Federated Model Training" located at Kafka-ML Platform (acting as an orchestrator) will retrieve the messages and weights in order from the control topic, and will aggregate them and send the updated model through the "Model Control Topic". Each of these actions is considered an aggregation round. These series of actions will be repeated as many times as rounds have been chosen. Kafka-ML Federated training clients always read the latest model message published by the Kafka-ML Platform training orchestrator, so in case they take longer to train or get temporarily disconnected when they request the model again, they will read the most updated version. Fig. 4 shows a sequence diagram describing the whole procedure.

Regarding model aggregation, we currently support an adaptation of FedAvg [20] for asynchronous federated learning. Therefore, some cases such as those with poorly distributed data would be complex to deal with. We have marked as future work to implement other aggregation techniques and make a comparison of which ones would be better in each case. Fig. 5 shows a diagram of how asynchronous model aggregation works.

In this diagram, it is shown how different devices cooperate asynchronously in a federated task. Here, Devices 1 and 2 get the initial weights and train the ML model defined simultaneously. Device 1 finishes training earlier, generating new global weights, which would be later updated with Device 2's weights. Device 3 joins later and receives the latest global model available for training, in which the previous devices have collaborated, and updates it including its information, always preserving the data confidentiality of each device. Algorithm 1 outlines the orchestrator and device workflow. Avoiding implementation details when loading and

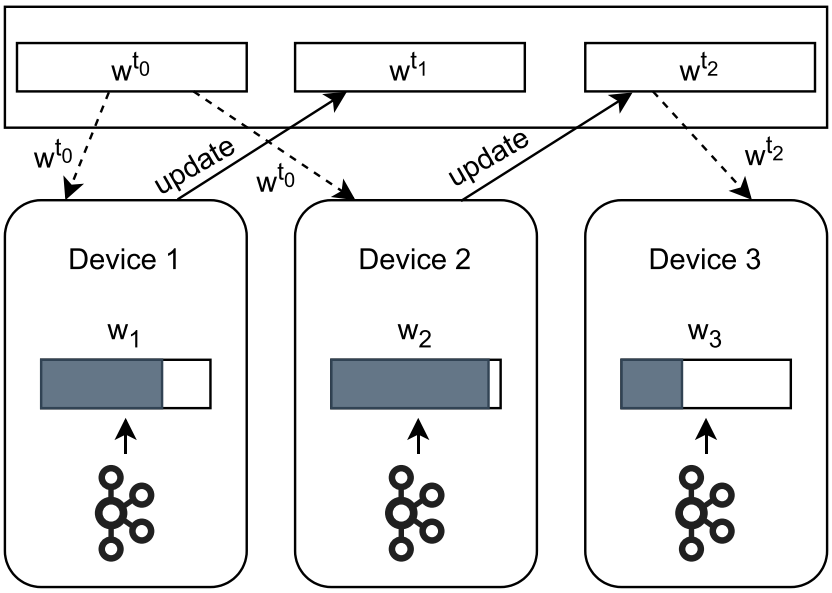


Fig. 5. Asynchronous aggregation schema.

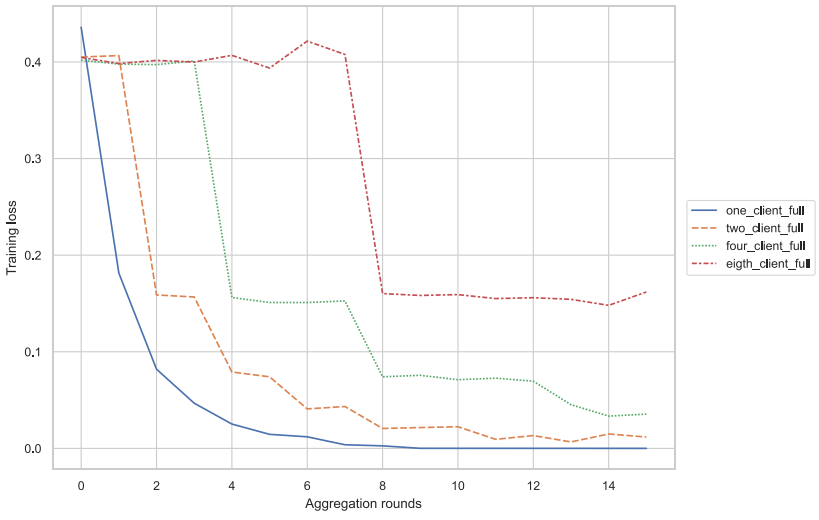
sending the models, it can be observed how models are handled in Apache Kafka, updating the global model given the trained versions available in the message queue, and the federated clients taking the last global model available.

Algorithm 1: Algorithm for Asynchronous FedAvg

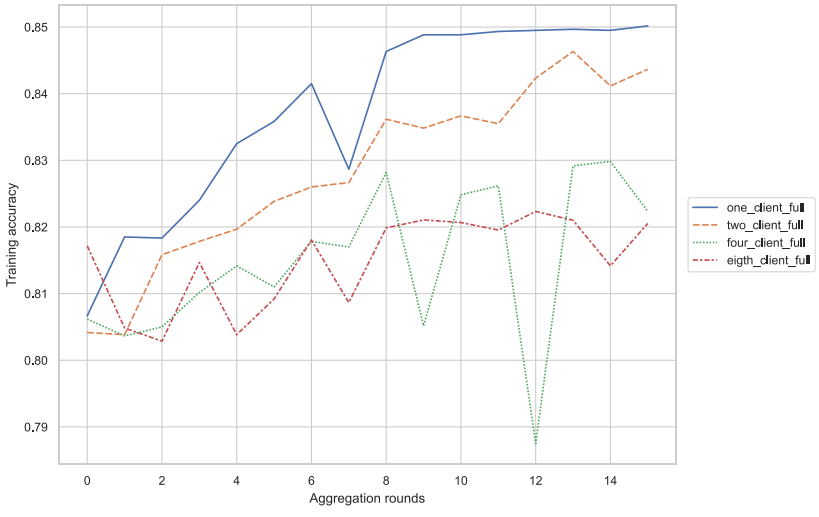
Generic Variables: $Topic_{Model}$, $Topic_{Agg}$, $Config_{Kafka}$
Procedure at Kafka-ML Platform
Input Data : $Model_0$, $maxAggRounds$, $Config_{Train}$, $currAggRound = 0$
Output Result : $Model_{Global}$, $Metrics$
$Model_{Global} = Model_0$, $Metrics = \{\}$
while $currAggRound \neq -1$ **do**
  sendModel($Model_{Global}$, $Config_{Train}$, $Topic_{Model}$, $Config_{Kafka}$, $currAggRound$)
  $Model_{currAggRound}$ = getLastUnreadTrainedModel($Topic_{Agg}$, $Config_{Kafka}$)
  $Model_{Global}$ = weightsAverage($Model_{Global}$, $Model_{currAggRound}$)
  if $currAggRound < maxAggRounds$ **then**
    $currAggRound = currAggRound + 1$
  else
    $currAggRound = -1$
    $Metrics$ = generateMetrics($Model_{maxAggRounds}$)
    sendModel($Model_{Global}$, $Config_{Train}$, $Topic_{Model}$, $Config_{Kafka}$, $currAggRound$) // Termination signal
  end
end
Procedure at Federated Kafka-ML Client
Input Data : $currAggRound = 0$
while $currAggRound \neq -1$ **do**
  $Model_{currAggRound}$, $Config_{Train}$, $currAggRound$ = getLastGlobalModel($Topic_{Model}$, $Config_{Kafka}$)
  $trainedModel$ = train($Model_{currAggRound}$, $Config_{Train}$)
  sendModel($trainedModel$, $Topic_{Agg}$, $Config_{Kafka}$)
end

5. Evaluation

To evaluate the behaviour of the proposed federated training solution as well as the ability of models to converge for all participating clients, a set of experiments has been carried out with different data as well as different numbers of clients.



(a) Training Loss



(b) Validation Accuracy

Fig. 6. Training and validation metrics at full dataset test case.

Kafka-ML is designed to facilitate the deployment of both user-created and predefined deep learning models. To assess its performance, we selected VGG16 [21] as a reference model due to its widespread applicability in machine learning applications and its computational complexity, which presents a significant challenge for the architecture.

To perform the training evaluation, we have considered an image classification use case using the CIFAR10 dataset. This set contains 60000 32 × 32 colour images and a total of 10 classes. The large number of images allows us to split the dataset in different ways in order to evaluate the convergence capability of the model over different tests.

5.1. Experimental setup

We are fortunate to possess a highly capable infrastructure for conducting a thorough evaluation. Next, a description of the system where the assessment will take place is presented:

- **Hardware Configuration**. The deployment of Kafka-ML Cloud has been performed on a computer with an Intel(R) Core(TM) i9-10900K CPU and 64 GB of RAM. The deployment of the multiple Kafka-ML Federated clients has been performed on a Kubernetes cluster with 7 state-of-the-art servers where each server has two Intel(R) Xeon(R) Gold 6230R CPU with two NVIDIA(R) Tesla(R) V100 GPUs as well as 384 GB of RAM.

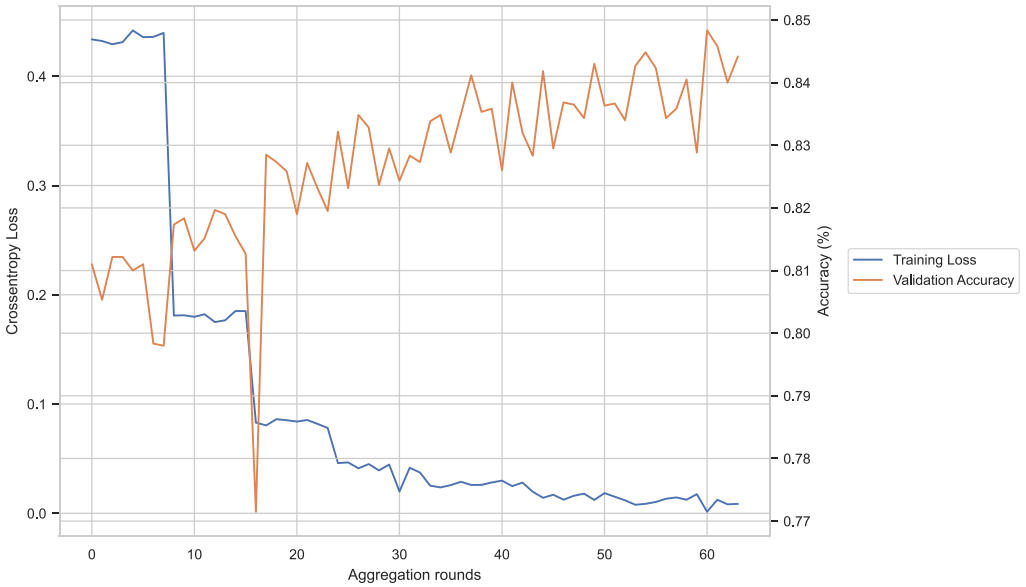


Fig. 7. Training and validation metrics at full dataset test case with 8 clients and 64 aggregation rounds.

- **Software configuration** The PC with the Kafka-ML runs Ubuntu 21.04 and K3s v1.25.7. Each node of the cluster runs Kubernetes v1.21.6 and Docker 20.10.8 on top of Ubuntu 20.04.3 LTS. A virtual machine (used as Kubernetes master) was deployed in one of the nodes. Every deployment of Kafka-ML Federated has been done in a separated namespace emulating different entities.

5.2. Training evaluation

To evaluate the training, the following experiments were planned. Starting from 1 to 8 federated clients, the VGG16 model will be trained on the CIFAR10 dataset in different configurations. In the first instance, each client will have the full dataset, looking to measure how the training metrics change as the number of clients increases. Subsequently, we will perform another test by splitting the dataset in a randomly distributed way. This should result in an accurate model, even if the classes are slightly unbalanced, and more robust than if the model had been trained individually for each client. Finally, a test will be performed with the same data as the previous one but by dropping and connecting some new clients during the training, in order to assess how the framework behaves under these conditions. All the tests, excluding some specific ones to check certain statements, have been carried out with a batch size of 32, 4 epochs, and 16 aggregation rounds.

5.2.1. Experiment 1: Training evaluation using the complete dataset

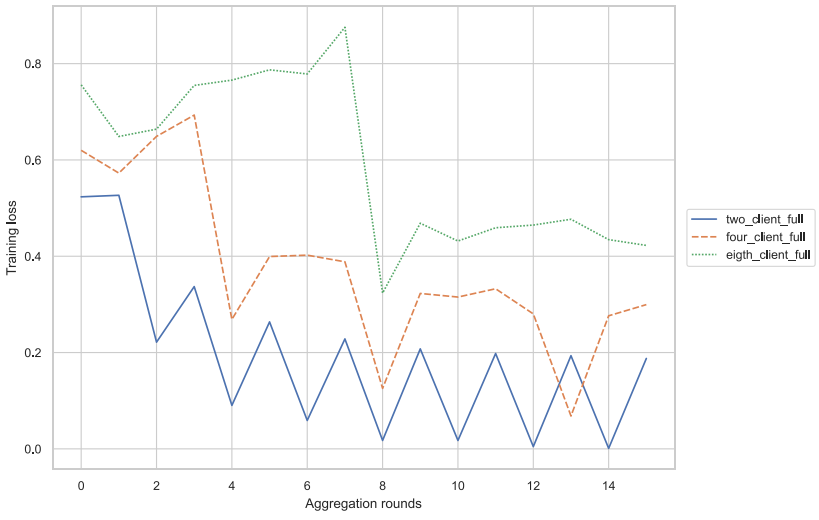
Firstly, the test has been performed on the full dataset to observe how, regardless of 1 or more customers, they all achieve similar results. The results of the test with the whole dataset can be seen in Fig. 6.

It can be seen that as we increase the number of clients, the precision slightly decreases. This is expected since we have kept the same number of aggregation rounds for a larger number of clients, thus making them have fewer update rounds with newer models. If we increase the number of aggregation rounds in cases with a higher number of clients, we can see that we converge to a model with similar results to those with fewer clients, as shown in Fig. 7.

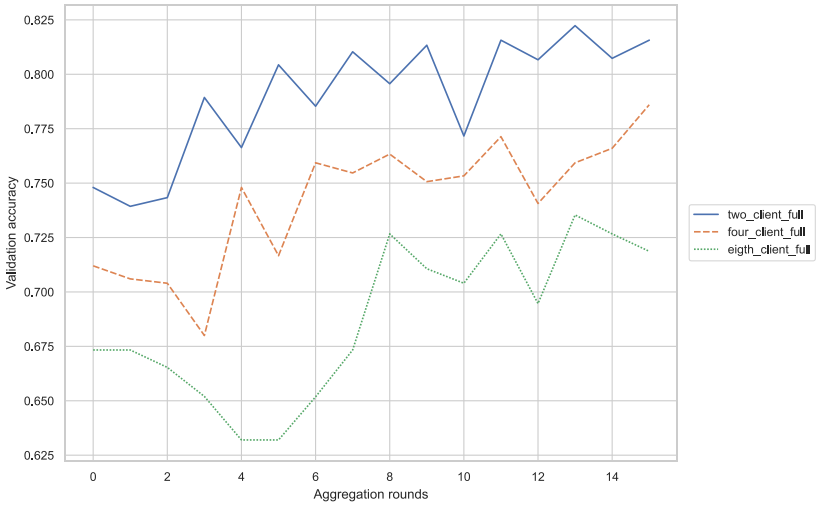
5.2.2. Experiment 2: Training evaluation with randomly splitting the dataset

In the next experiment, we randomly distributed the CIFAR10 dataset among the different federated clients. The results obtained are shown in Fig. 8.

It can be observed that, as the number of clients increases, the accuracy of the model decreases. In this case, the same test has been performed with 64 aggregation rounds but the result is similar. This could be due to several reasons such as that having less data means that less improvement is achieved and this does not propagate to the other models, or that using another aggregation function for unbalanced data is more appropriate. We will work further on imbalanced data cases by implementing new aggregation strategies in the future.



(a) Training Loss



(b) Validation Accuracy

Fig. 8. Training and validation metrics at random splitted dataset test case.

5.2.3. Experiment 3: Training evaluation with new connections and client drops

Finally, we have carried out a test with the same data as the previous one but by dropping and connecting some new clients during the training. For this experiment, in which we are going to operate with a total of 8 clients, we will carry out training with 64 aggregation rounds. Starting with two federated clients, every 8 rounds a new client will join the training and the oldest client will be dropped. Fig. 9 shows model metrics during this experiment.

It can be seen that there are some spikes which, looking at the client logs, are related to the joining of new clients. Dropping of older clients has caused the accuracy (especially visible in the training metric due to overfitting) to stop growing. After this experiment, we have observed that it is necessary to integrate new aggregation strategies based on client lifetime or other metric that considers the amount of data of a client to avoid generating model bias.

6. Conclusion and future work

In this paper, the implementation of asynchronous federated learning powered by data streams through the open-source Kafka-ML framework has been presented. Federated learning is the strategy of training ML/AI models between organisations data without sharing their information. The orchestration of Kafka-ML and this strategy has the potential to boost the adoption of federated learning in the IoT field thanks to the seamless integration of data streams achieved. Moreover, this approach allows organisations to dynamically collaborate when a new dataset is available, achieving a flexible solution.

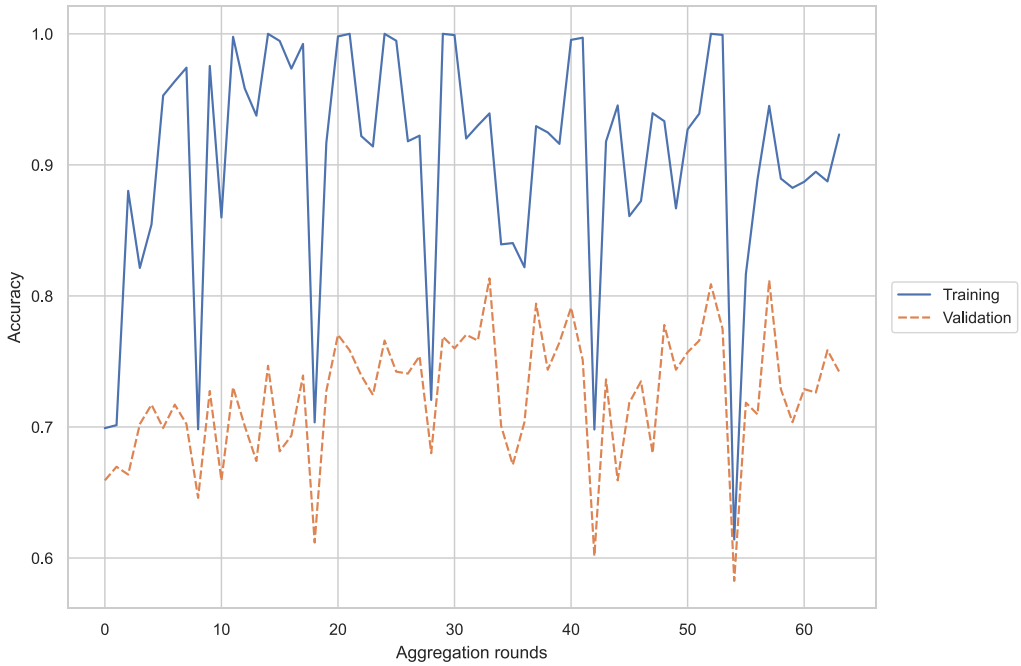


Fig. 9. Training and validation metrics at client join delay test case with 8 clients and 64 aggregation rounds.

To show the potential of the federated learning solution, an evaluation of this functionality has been carried out in several potential use cases on a state-of-the-art GPU infrastructure, demonstrating that proving the implemented model aggregation strategy works for these use cases. However, we observed that as the data becomes non-uniformly distributed across clients, the accuracy of the models is reduced, so it would be valuable to integrate new model aggregation strategies to mitigate this.

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

- Integrate new model aggregation techniques. Although FedAvg is a suitable aggregation strategy for the proposed use cases, there are other use cases where the data are not as uniformly distributed, or cases where there are customers are significantly slower and participate fewer times. Currently, these cases are not fully covered, which would lead to skewed models. However, there exist more aggregation techniques such as FedProx [22] or FedYogi [23] (in synchronous cases). It would be interesting to find new techniques applicable to asynchronous federated learning and non-IID data, and integrate them into Kafka-ML.
- Improving the user experience. There are some improvements that would be convenient to integrate into the user experience, such as improving the model definition view to make it simpler or integrating it with a more advanced text editor.
- Integrate other processing tasks. At times, it is necessary to integrate data processing tasks either before or after the data enters the model. It would be interesting if this processing could be applied directly to the data stream from Kafka-ML.
- Automatic optimisation of hyperparameters. This is another interesting functionality to integrate as it would save time when looking for a fine-grained solution while training ML/AI models.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Manuel Diaz reports financial support was provided by Spain Ministry of Science and Innovation.

Data availability

The data that support the findings of this study are available at https://www.cs.toronto.edu/~kriz/cifar.html. These data were derived from the tech report available online at the same site.

Acknowledgements

This work is funded by the Spanish projects CPP2021-009032 (‘ZeroVision: Enabling Zero impact wastewater treatment through Computer Vision and Federated AI’), TED2021-130167B-C33 (‘GEDIER: Application of Digital Twins to more sustainable irrigated farms’), TSI-063000-2021-116 (‘5G+TACTILE_2: Digital vertical twins for B5G/6G networks’), PID2022-141705OB-C21 (‘DiTaS: A framework for agnostic compositional and cognitive digital twin services’), and MIG-20221022 (‘GEDERA: Intelligent Flexible Energy Demand Management in Coupled Hybrid Networks’).

References

[1] M. Bansal, I. Chana, S. Clarke, A survey on iot big data: Current status, 13 v's challenges, and future directions, ACM Comput. Surv. 53 (6) (2020) 1–59.
[2] A. Imteaj, U. Thakker, S. Wang, J. Li, M.H. Amini, A survey on federated learning for resource-constrained IoT devices, IEEE Internet Things J. 9 (1) (2021) 1–24.
[3] A. Nilsson, S. Smith, G. Ulm, E. Gustavsson, M. Jirstrand, A performance evaluation of federated learning algorithms, in: Proceedings of the Second Workshop on Distributed Infrastructures for Deep Learning, 2018, pp. 1–8.
[4] A. Lazidis, K. Tsakos, E.G. Petrakis, Publish–subscribe approaches for the IoT and the cloud: Functional and performance evaluation of open-source systems, Internet Things 19 (2022) 100538.
[5] 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.
[6] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, Y. Li, X. Liu, B. He, A survey on federated learning systems: Vision, hype and reality for data privacy and protection, IEEE Trans. Knowl. Data Eng. (2021).
[7] C. Xu, Y. Qu, Y. Xiang, L. Gao, Asynchronous federated learning on heterogeneous devices: A survey, 2021, arXiv preprint arXiv:2109.04269.
[8] TensorFlow federated official webpage, 2023, Available from: https://www.tensorflow.org/federated. (Last accessed July 2023).
[9] Pysyft official webpage, 2023, Available from: https://github.com/OpenMined/PySyft. (Last accessed July 2023).
[10] FATE official webpage, 2023, Available from: https://fate.readthedocs.io/. (Last accessed July 2023).
[11] P. Foley, M.J. Sheller, B. Edwards, S. Pati, W. Riviera, M. Sharma, P.N. Moorthy, S.-h. Wang, J. Martin, P. Mirhaji, P. Shah, S. Bakas, OpenFL: The open federated learning library, Phys. Med. Biol. (2022) http://dx.doi.org/10.1088/1361-6560/ac97d9, URL http://iopscience.iop.org/article/10.1088/1361-6560/ac97d9.
[12] D.J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, H.L. Kwing, T. Parcollet, P.P.d. Gusmão, N.D. Lane, Flower: A friendly federated learning research framework, 2020, arXiv preprint arXiv:2007.14390.
[13] A. Grafberger, M. Chadha, A. Jindal, J. Gu, M. Gerndt, Fedless: Secure and scalable federated learning using serverless computing, in: 2021 IEEE International Conference on Big Data, Big Data, IEEE, 2021, pp. 164–173.
[14] K. Jayaram, V. Muthusamy, G. Thomas, A. Verma, M. Purcell, Lambda FL: Serverless aggregation for federated learning, in: International Workshop on Trustable, Verifiable and Auditable Federated Learning, 2022, p. 9.
[15] D.M. Diaz, A. Manoel, J. Chen, N. Singal, R. Sim, Project Florida: Federated learning made easy, 2023, arXiv preprint arXiv:2307.11899.
[16] Y. Chen, Y. Ning, M. Slawski, H. Rangwala, Asynchronous online federated learning for edge devices with non-iid data, in: 2020 IEEE International Conference on Big Data, Big Data, IEEE, 2020, pp. 15–24.
[17] Y. Xie, Z. Wang, D. Gao, D. Chen, L. Yao, W. Kuang, Y. Li, B. Ding, J. Zhou, Federatedscope: A flexible federated learning platform for heterogeneity, 2022, arXiv preprint arXiv:2204.05011.
[18] 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. (2023) e13287.
[19] 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.
[20] B. McMahan, E. Moore, D. Ramage, S. Hampson, B.A. y Arcas, Communication-efficient learning of deep networks from decentralized data, in: Artificial Intelligence and Statistics, PMLR, 2017, pp. 1273–1282.
[21] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, 2014, arXiv preprint arXiv:1409.1556.
[22] T. Li, A.K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, V. Smith, Federated optimization in heterogeneous networks, 2020, arXiv:1812.06127.
[23] S. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečný, S. Kumar, H.B. McMahan, Adaptive federated optimization, 2021, arXiv:2003.00295.