A methodology for the development of soft sensors with Kafka-ML

Antonio Jesús Chaves, Cristian Martín, Luis Llopis Torres,
Enrique Soler, and Manuel Díaz

Abstract
Advances in the Internet of Things (IoT) field have allowed a wide variety of devices to be connected and send information continuously to the Internet. Thanks to this increase in data communication, Machine Learning (ML) and data science have been able to be applied to analyze and extract valuable intelligence from the IoT. In this sense, the IoT has also contributed to improving the design and implementation of soft sensors. Soft sensors are used to predict features that are difficult to measure directly because the sensor to do so does not exist or is very expensive. IoT real-time monitoring can be used in conjunction with ML techniques to infer those parameters that are difficult to achieve with specific sensors. There exist methodologies for the development of soft sensors but there is a lack of a common tool to support the design and implementation of them, covering the phases from model training to visualization of predictions. In this chapter, we present a methodology to support soft sensor development based on Kafka-ML, an open-source framework to manage ML pipelines. Kafka-ML will allow researchers to develop, train, and validate ML models, and visualize real-time predictions using streaming data. To demonstrate the viability of our proposal, we developed a soft sensor which predicts nitrate levels from river watersheds.

1 Introduction

The Internet of Things (IoT) has definitely contributed to the application of data science due to data growth in recent years [1]. Any kind of device is able to send data periodically, even with high rates, to servers, cloud, or fog platforms. This has made

ITIS Software Institute, University of Málaga,
Málaga, Spain
`{chaves,cristian,lmllopis,esoler,mdiaz}@uma.es`
`http://itis.uma.es/`

it possible to have large amounts of data on the platforms and has given them the possibility of applying different processes to improve decision-making. For example, analytic techniques can be applied in order to estimate, predict or monitor values and actions. Machine learning (ML) and Artificial Intelligence (AI) are the main fields where data are treated to carry out these predictions and recommendations. In this sense, the data availability and ML techniques can contribute to improving the decision makings and the automated response of the IoT systems.

However, there are two challenges that can difficult data management. Sensors, which are mainly the data generators, are very expensive in some cases (expensive sensors can make product development unprofitable) or, simply, there is no sensor to produce this data. In this case, ML techniques also can help to infer this data from the information provided by other sensors that can have some correlation with the necessary data.

A soft sensor is the combination of hardware sensors and software (mostly through machine learning) to estimate parameters which cannot be calculated directly or it is expensive to obtain [2]. The development of soft sensors has been greatly facilitated by achievements in data science, computing and communication technologies, statistical tools, and machine learning techniques [3]. Therefore, soft sensor development is a low-cost alternative to manage variables that can be inferred when there is no data available to use in a formula or no such formula exists. Soft sensors can be designed as a source of variable estimation.

Kafka-ML [4] is an open-source framework to manage the pipeline of TensorFlow and PyTorch Machine Learning (ML) models on Kubernetes [5]. This tool allows the design, training, and inference of ML models by using data streams (with Apache Kafka as the data provider), enabling scalability, fault tolerance, and distributed models [6]. Since Kafka-ML is designed to work with data streams (both for training and inference), the framework can be used in an IoT ecosystem to manage real-time data pipelines, typical of sensorization in IoT, and to integrate ML models to take decisions and predict values. In this sense, Kafka-ML arises as a powerful tool for the design and development of soft sensors.

There is a large literature about soft sensors, each applying different tools but, from our point of view, there is a lack of a generic platform to support soft sensor development. In this chapter, we present a methodology based on Kafka-ML for soft sensor design and development. In order to show the suitability of the platform, we present the design, implementation, and prototype of a soft sensor to calculate harmful nitrate levels from underground water.

The main contributions of this chapter are:

- Presenting Kafka-ML as an open-source framework for the design and development of soft sensors.
- Providing a streaming framework for the continuous prediction and visualization of soft sensors.
- Using Kafka-ML to obtain a fast evaluation and validation of different models and scenarios, given the complexity of soft sensor design.
- Demonstrating through a use case the prediction of harmful nitrate levels from underground water.

The rest of the chapter is structured as follows. Section 2 presents the related work regarding Kafka-ML and soft sensor design. Section 3 presents the proposed methodology of soft sensor design using Kafka-ML. Section 4 describes the proposed methodology by using a water pollution monitoring use case and evaluates several key elements of the proposed approach. Finally, Section 5 summarizes the work, with a discussion on further potential extensions.

2 Related Work

The IoT has contributed to increasing the development of hardware sensors. Almost any IoT device is equipped with sensors and it is able to send information using wireless communication technology such as Bluetooth or WiFi. However, when the necessary knowledge is unavailable, or the modeling process is too complicated, the construction of soft sensors is a promising complementary technology [3].

The design of a soft sensor can be carried out by applying different techniques, namely model-driven and data-driven [2]. Model-driven soft sensors are based on First Principle Models (FPM), extended Kalman filter [7] and adaptive observer [8]. FPMs describe the physical and chemical background of the process but it is difficult to model malfunctions or changing conditions. Data-driven soft sensors have been presented as a design alternative.

The design of data-driven soft sensors can apply statistical methods and ML techniques. Some examples are principal component and partial least squares regression, support vector machine, artificial neural network, and deep learning [3],[9],[10]. There are many different application fields for data-driven soft sensors such as:

- On-line prediction. Combining real-time data and estimated variables to carry out predictions about the processes.
- Monitoring. Using estimation to monitor variables without sensor devices.
- Fault detection. Applying the on-line prediction to detect malfunctions in the monitored system.

Neural network techniques are broadly used to estimate values by minimizing the distance between the network and target outputs. When the model is trained, it is possible to predict the output simulating a real sensor. Soft sensors using this technique are designed in many different fields. For example, in [11] a control of distillation column soft sensor is designed. Kadlec et al. [2] present data-driven soft sensors for the industry. The chemical industry is applying techniques for variable estimations to substitute expensive sensors or data which cannot be directly sensorized. One example, in this context, is the estimation of the nitrate concentration as described in [12], [13].

Many works have researched the forecasting of nitrate concentration based on ML techniques. In [14] the ability of artificial neural networks to model groundwater nitrate of the Arak Aquifer is presented. Nitrate monitoring is very important in cities. In [15], an array of soft sensors to improve nitrate concentration accuracy is proposed. Soft sensors for nitrate estimation are also designed for treatment water plants as shown in [16]. Another complementary work [17] applies soft computing models to simulate nitrate contamination.

All these proposals apply the traditional stages for the application of machine learning techniques, i.e, feature extraction, model training, and inference. Additionally, data can be used in an offline or real-time mode but it is necessary to adapt the framework to the particular characteristics of the design. We propose a methodology based on the Kafka-ML framework to simplify soft sensor design.

Apache Kafka [18] is an open-source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications. In particular, Kafka can be used as a real-time data provider (fault-tolerant and scalable streaming data) to the machine learning models in order to take decisions or forecasting. Many works deal with Kafka and ML, such as [19], [20] or [21].

Kafka-ML [4] is a framework to manage the pipeline of Tensorflow/Keras and PyTorch (Ignite) ML models on scalable containerized platforms (Kubernetes). The Kafka-ML pipeline allows the design, training, validation, and inference of ML models. The training and inference datasets for the ML models can be fed through Apache Kafka, thus they can be directly connected to data streams like the ones provided by the IoT. Other approaches such as Kubeflow [22], NVIDIA DIGITS [23], Amazon SageMaker [24], Algorithmia [25], and Valohai [26] have similar goals or have provided some of the functionalities but, Kafka-ML is the first open-source framework to provide an ML/AI pipeline solution to integrate ML/AI and data streams. This chapter proposes a methodology to use this framework to design and implement soft sensors.

3 Proposed methodology for soft sensor development with Kafka-ML

Kafka-ML provides an open-source framework that enables the management of the pipeline of machine learning techniques with data streams including steps such as training and inference. Soft sensors are precisely, in most cases, data stream generators that infer a unique property where machine learning and deep learning techniques are adopted. In this section, a methodology to integrate these two paradigms is presented. The proposed methodology (Fig. 1) for the design of soft sensors through Kafka-ML includes:

1. Soft sensor design and data ingestion into Kafka-ML for the training of ML models.
2. Design, training, and validation of ML models (even multiple in parallel) with the data streams received.

3. Trained model deployment for soft sensor inference, where real-time sensor data will be ingested into the model as input to obtain predictions of the desired property.
4. Soft sensor prediction visualization through a user-friendly interface in Kafka-ML.

As you can see, you only need the Kafka-ML framework for the design of soft sensors with this methodology. Next, each step of the methodology is described in detail.

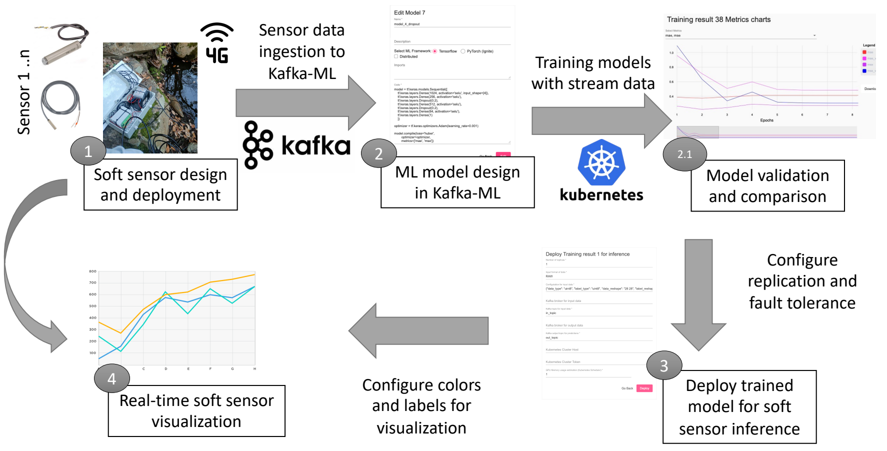

Sensor 1 ..n
4G
Sensor data ingestion to Kafka-ML
kafka
1
Soft sensor design and deployment
2
ML model design in Kafka-ML
Training models with stream data
kubernetes
2.1
Model validation and comparison
Configure replication and fault tolerance
3
Deploy trained model for soft sensor inference
Configure colors and labels for visualization
4
Real-time soft sensor visualization

Fig. 1 Proposed soft sensor development methodology in Kafka-ML.

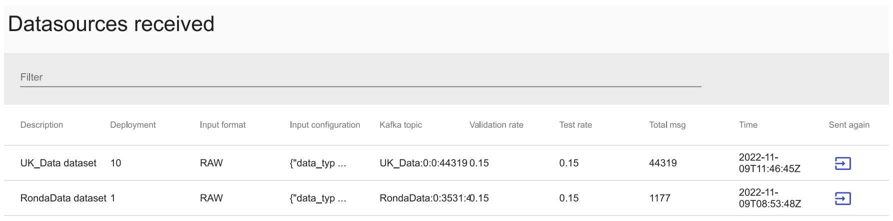
3.1 Soft-sensor design and data ingestion to Kafka-ML

The first decision to take into account in this methodology is the selection of the physical sensors to design the soft sensor. This decision can be influenced by several factors, such as the sensors available to predict the desired feature, their cost, real-time response, and the combination of all of them. Although in the inference phase it will only be necessary to send the measurements to predict the target variable, in the training phase it is also necessary to send the real value of the measurement to be predicted to carry out supervised learning. This can be done in a laboratory, or by using a sensor (probably very expensive) that will later be replaced by the developed soft sensor. Once the sensors have been selected and properly installed in the study area, the sensor data can be sent to the Kafka-ML framework for further analysis and training. Two of the libraries developed in Kafka-ML can be used for this purpose: RAW (binary format) and Apache Avro [27].

On the one hand, the RAW library can be used to send and encode measurements directly in binary. Apache Kafka works with a binary format for high performance. The RAW library handles the data types and encodes them transparently to the user. The user only needs to code an array or matrix with a list of measurements to obtain a prediction from the soft sensor, and the library handles the encoding, shape, and dispatching.

On the other hand, users can also use the most-popular library for encoding data streaming pipelines, Apache Avro. Using a defined Avro schema for the studied dataset, users only have to send a simple JSON with the Avro library to Kafka-ML. The advantage of this data format is that most big data and data stream platforms support Apache Avro, and therefore, this facilitates hypothetical integration with other systems.

Unlike many streaming data platforms, sending measurements for feature prediction does not mean that these measurements are consumed and discarded. Kafka-ML exploits the Apache Kafka feature for data retention, and stream data can be reused as many times as necessary. Thanks to this, Kafka-ML not only allows the reuse of the generated datasets for the training of multiple models in parallel, thus optimizing time and resources, but also for training new models later on. Figure 2 shows the Kafka form for data stream management. Once data streams are sent to Kafka-ML they are registered so they can be easily dispatched to other training models in this form.


Datasources received

Filter

Description	Deployment	Input format	Input configuration	Kafka topic	Validation rate	Test rate	Total msg	Time	Sent again
UK_Data dataset	10	RAW	{"data_typ ...	UK_Data:0:0:44319	0.15	0.15	44319	2022-11-09T11:46:45Z	⇥
RondaData dataset	1	RAW	{"data_typ ...	RondaData:0:3531:4	0.15	0.15	1177	2022-11-09T08:53:48Z	⇥

Fig. 2 Kafka-ML data stream management form.

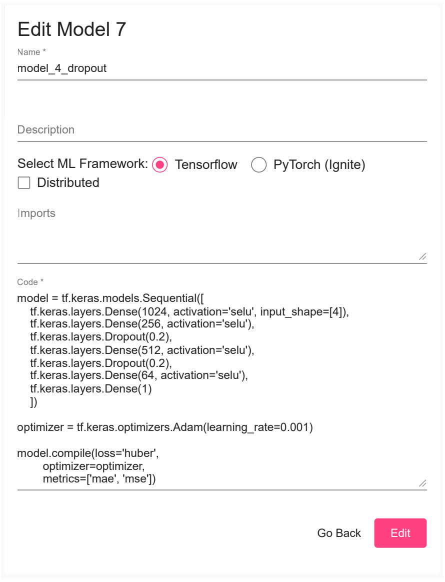
3.2 Model design, training, and validation

The second step of the methodology covers the definition and model training through the data stream sent to Kafka-ML in the previous step. In Kafka-ML, users can easily define multiple models in popular machine learning and deep learning frameworks. Currently, Kafka-ML supports TensorFlow and PyTorch, allowing developers to use the one that best suits their needs. Once users have defined a set of models to evaluate, they can make them available for training. One of the common practices in machine learning is to continuously evaluate different models, hyperparameters, and architectures until reaching the one that best fits the studied dataset. Kafka-

ML facilitates this thanks to its data stream management and simultaneous model deployment. In the soft sensor design, users can create several models to compare their performance with data streams previously received from physical sensors.

Kafka-ML also supports model deployment on infrastructures with both CPUs and GPUs. In the case of computational intensive models such as deep learning models, Kafka-ML exploits GPU acceleration for fast training. For advanced users who want to exploit Kafka-ML's features to their fullest, another alternative is to deploy partitioned models in a distributed way [6], [28]. This would enable the deployment of deep learning models in the cloud-to-things continuum for optimizing response times.

The definition of models in Kafka-ML is a very easy task. Figure 3 shows how to define a new machine learning model. In the creation form, a name for the model, optional imports for required libraries, and the source code of the model itself in the selected framework have to be specified. For instance, Listing 1 shows a Kafka-ML TensorFlow code example for the definition of a soft sensor model with 4 input variables and 3 neural layers of 64, 32, and 16 neurons, respectively. This code also defines the optimizer (Adam), the loss function (mean squared error), and finally the metrics that will be displayed when the model is trained. In this case, the Mean Absolute Error (MAE) and Mean Squared Error (MSE) are used. In the soft sensor design usually regression models are involved so the output layer of the model is a continuous value.


Edit Model 7

Name *
model_4_dropout

Description

Select ML Framework: [✓ Tensorflow] PyTorch (Ignite)
☐ Distributed

Imports

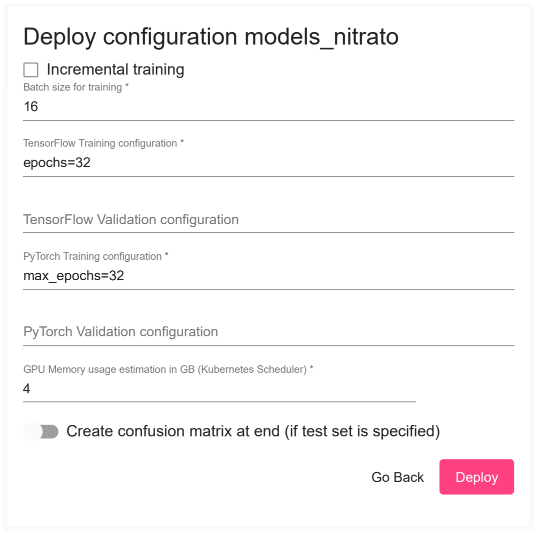
Code *
```
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(1024, activation='selu', input_shape=[4]),
    tf.keras.layers.Dense(256, activation='selu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(512, activation='selu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(64, activation='selu'),
    tf.keras.layers.Dense(1)
    ])

optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)

model.compile(loss='huber',
        optimizer=optimizer,
        metrics=['mae', 'mse'])
```

Fig. 3 Create model form in Kafka-ML.

Listing 1 Example soft sensor TensorFlow/Keras ML code with 4 sensor inputs.

```
model = Sequential([
Dense(64, activation='relu', input_shape=[4]),
Dense(32, activation='relu'),
Dense(16, activation='relu'),
Dense(1)
])
model.compile(loss='mse',
optimizer='adam', metrics=['mae', 'mse'])
```

Once a set of models for the soft sensor has been defined in Kafka-ML, it is time to create a configuration. A configuration is a logical union of one or more models that will be trained together. The configuration is then deployed with the corresponding training parameters such as the batch size and the number of epochs. Figure 4 shows the form where the user can set those parameters. Once deployed, a Docker container will be created in Kafka-ML (Kafka-ML is a microservice architecture) and will start reading the data stream sent previously from Apache Kafka. It is not a problem if the data has not been sent yet because the models will wait until the data stream is available.


Deploy configuration models_nitrato

☐ Incremental training

Batch size for training *
16

TensorFlow Training configuration *
epochs=32

TensorFlow Validation configuration

PyTorch Training configuration *
max_epochs=32

PyTorch Validation configuration

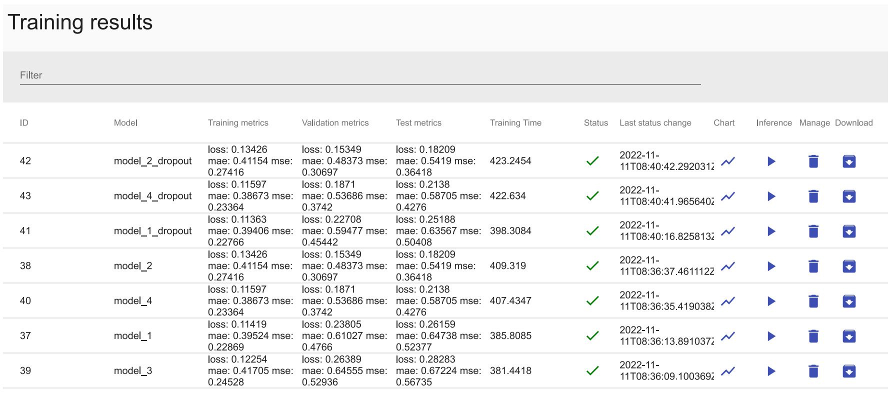
GPU Memory usage estimation in GB (Kubernetes Scheduler) *
4

Create confusion matrix at end (if test set is specified)

Fig. 4 Kafka-ML model deployment form.

Users can check the training metrics and compare the results obtained by the defined models in real-time since the first epoch. This allows users not only to visualize and compare the performance of their models but also to stop the training of a model as soon as they realize that it is not getting acceptable results and not to wait until the end of the training. Once all the models have been trained, users evaluate the best-performing models, download them in h5 format, and prepare the selected ones for soft sensor inference (next step).

Figure 5 shows the resulting metrics form where the trained models can be compared. These metrics are the ones defined when designing the model. Furthermore, metrics can be studied in more detail (e.g. learning curves) by selecting the chart option. In this case, a chart with the metrics such as the one displayed in Fig. 6 shows the progression of the metrics with respect to the epochs. In this interactive graph, users can select which metrics are displayed and compare the results of training and validation, for example, to identify whether overfitting or underfitting has occurred.


Training results

Filter

ID	Model	Training metrics	Validation metrics	Test metrics	Training Time	Status	Last status change	Chart	Inference	Manage	Download
42	model_2_dropout	loss: 0.13426 mae: 0.41154 mse: 0.27416	loss: 0.15349 mae: 0.48373 mse: 0.30697	loss: 0.18209 mae: 0.5419 mse: 0.36418	423.2454	✓	2022-11-11T08:40:42.292031	📈	▶	🗑	⬇
43	model_4_dropout	loss: 0.11597 mae: 0.38673 mse: 0.23364	loss: 0.1871 mae: 0.53686 mse: 0.3742	loss: 0.2138 mae: 0.58705 mse: 0.4276	422.634	✓	2022-11-11T08:40:41.965640	📈	▶	🗑	⬇
41	model_1_dropout	loss: 0.11363 mae: 0.39406 mse: 0.22766	loss: 0.22708 mae: 0.59477 mse: 0.45442	loss: 0.25188 mae: 0.63567 mse: 0.50408	398.3084	✓	2022-11-11T08:40:16.825813	📈	▶	🗑	⬇
38	model_2	loss: 0.13426 mae: 0.41154 mse: 0.27416	loss: 0.15349 mae: 0.48373 mse: 0.30697	loss: 0.18209 mae: 0.5419 mse: 0.36418	409.319	✓	2022-11-11T08:36:37.461112	📈	▶	🗑	⬇
40	model_4	loss: 0.11597 mae: 0.38673 mse: 0.23364	loss: 0.1871 mae: 0.53686 mse: 0.3742	loss: 0.2138 mae: 0.58705 mse: 0.4276	407.4347	✓	2022-11-11T08:36:35.419038	📈	▶	🗑	⬇
37	model_1	loss: 0.11419 mae: 0.39524 mse: 0.22869	loss: 0.23805 mae: 0.61027 mse: 0.4766	loss: 0.26159 mae: 0.64738 mse: 0.52377	385.8085	✓	2022-11-11T08:36:13.891037	📈	▶	🗑	⬇
39	model_3	loss: 0.12254 mae: 0.41705 mse: 0.24528	loss: 0.26389 mae: 0.64555 mse: 0.52936	loss: 0.28283 mae: 0.67224 mse: 0.56735	381.4418	✓	2022-11-11T08:36:09.100369	📈	▶	🗑	⬇

Fig. 5 ML models and training metric result table in Kafka-ML.

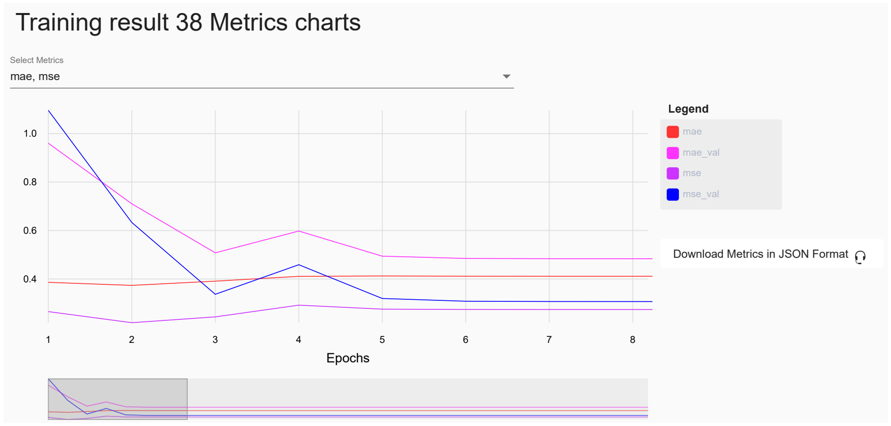


Fig. 6 Metric visualization chart in Kafka-ML.

3.3 Model deployment for soft sensor inference

Once the model or models have been trained, validated and selected, users can deploy them to make data-driven inferences and obtain real-time predictions for the soft sensor. In this way, they do not have to use another tool for this purpose (e.g. TensorFlow Serving [29], BentoML [30], etc). Kafka-ML offers the possibility to deploy these models directly from the application and through Kafka topics, receive and return the output as appropriate. To deploy the models, users can configure the number of replicas of the model, the Kafka topic where the measurement data will arrive, and the output topic where the soft sensor predictions will be sent, among other parameters. From the moment of deployment, the deployed models expect sensor data in an input topic, and they will send the soft sensor prediction result to the configured output topic.

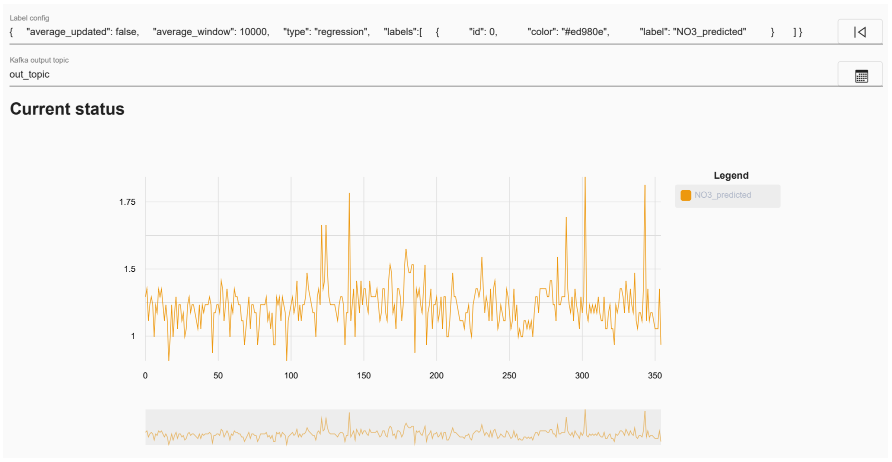
This closes a loop where the data reaches Kafka-ML and is used in the different phases of the pipeline using data streams. In addition, the application allows the deployment of these models with the possibility of being fault-tolerant if necessary, deploying several replicas of the necessary components, or partitioning the input and output topics for load balancing.

3.4 Final application: Kafka-ML visualization

The last step when users have a trained and deployed model is to visualize the soft sensor data over time. Normally in these situations, an ad-hoc API is developed to visualize the soft sensor predictions via a web form (e.g., a Python Flask API that imports the trained model and serves the predictions through a graphical interface). Kafka-ML has also considered this case, and it is prepared for fast application delivery. Finally, users can visualize the soft sensor inferences in real time.

Figure 7 shows an example of the visualization of predictions on one of the models that we will later observe in the evaluation. To use this tool, users only need to configure the output topic they configured in the inference, and design aspects such as the colour of the graph, the number of outputs, and the type of model (regression in this case). Once connected, the soft sensor results will be displayed as shown in the previous example. Note that the data displayed is real-time data and there is no system to display historical data. This API is currently intended for fast data visualizations as presented in this case.

Using this visualization means that users do not have to leave Kafka-ML and move everything previously developed to other tools such as Grafana [31] to visualize predictions in real time, accelerating the development and usage of soft sensors with data streams.


Label config

```
{    "average_updated": false,    "average_window": 10000,    "type": "regression",    "labels":[   {        "id": 0,        "color": "#ed980e",        "label": "NO3_predicted"     }   ] }
```

Kafka output topic

out_topic

Current status

Fig. 7 Real-time soft sensor prediction visualization in Kafka-ML.

4 Water pollution monitoring use case

To demonstrate how the methodology works, we introduce the use of this methodology for the development of a soft sensor for water pollution monitoring. In particular, we present a soft sensor for the prediction of nitrate ($NO_3{}^-$) concentration in river watersheds. To do so, we will carry out this development with a public dataset taken from the United Kingdom (UK) rivers in an attempt to implement a soft sensor capable of detecting the amount of nitrate in water, a parameter that is difficult and costly to measure continuously.

In the following subsections, we will discuss the dataset used, as well as the models to be evaluated in order to compare the results and choose the best one to deploy in a soft sensor.

4.1 Methods and datasets

Considering the challenge of measuring nitrate levels in water due to the lack of data and sensors; the first step is to look for a public dataset to address the problem and study whether it is viable to do so or not. In this first step, we have made use of the UK government's Open Data on river water quality monitoring [32]. This dataset is public and open, with a large amount of data available.

The dataset contains multiple information about the water quality monitoring stations, such as their geographical coordinates, station identifier name, as well as different variables measured over time. The variables change depending on the station, although the main ones include the following:

- Alkalinity
- Biochemical Oxygen Demand
- Conductivity
- Dissolved Copper
- Dissolved Oxygen
- Dissolved Iron
- Nitrate
- Nitrite
- Ammonium
- pH
- Dissolved Phosphor
- Suspended Solids
- Dissolved Zinc

Among these measured characteristics is the nitrate, which is difficult to measure directly, and the sensors to do so are very expensive. A clear candidate for a soft sensor. From the rest of the variables, we must select which ones that, when assembling a real device with attached sensors, are, as a whole, easier and cheaper to obtain than the nitrate sensor we are trying to simulate.

Therefore, from the previously mentioned variables, we selected alkalinity, conductivity, pH, and suspended solids, as these are the ones that are easier to measure with low-cost sensors.

After selecting the variables, we slightly pre-process the dataset by cleaning possible outliers, normalizing the data for better training, and sending it to Kafka-ML simulating that the data has been sent by an IoT device.

Regarding machine learning models, various model architectures have been evaluated in order to determine which of these best fits the data and the problem to be solved. Given the data, the architecture that suits us best is a fully connected (known as dense) one. One of these architectures can be seen in Figure 8.

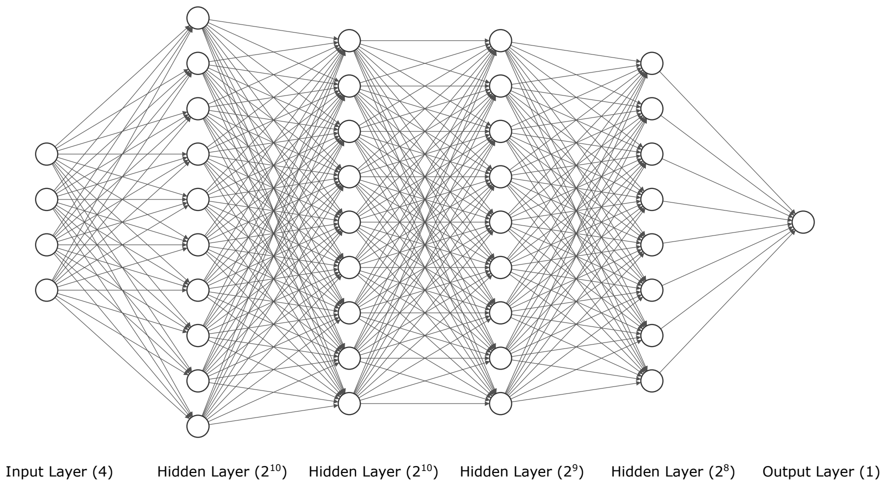


Fig. 8 Proposed soft sensor ML model architecture.

4.2 Evaluation

For the evaluation of the soft sensor performance in Kafka-ML, the tool has been deployed on a cluster architecture with the following computation capacities:

- **Hardware configuration**. Kafka-ML has been deployed on our own cluster of 7 state-of-the-art servers. Each server 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 data and where the results were measured has an Intel(R) Core(TM) i9-10900K CPU and 64 GB of RAM.
- **Software configuration**. Each node 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 of the nodes, while the remaining are Kubernetes workers. The client PC runs Ubuntu 21.04.

We evaluated a total of 4 different model architectures, each of which different number of layers and neurons per layer. In addition, the previous ones have been slightly modified by adding Dropout layers [33]. All models have been trained using Huber Loss as loss function, Adam optimizer, and Scaled Exponential Linear Units (SELU) as activation function in hidden layers. Table 1 shows the different model architectures evaluated.

Table 1 Models architectures used during evaluation

#	Description	Architecture
1	Model with 3 layers	512 x 128 x 64
2	Model 1 with a dropout layer	512 x 128 x Dropout(0.2) x 64
3	Model with 4 layers	1024 x 512 x 512 x 256
4	Model 3 with dropout layer	1024 x 512 x Dropout(0.2) x 512 x 256
5	Similar to Model 1, changing neurons distribution	64 x 1024 x 64
6	Model with 4 layers with fewer neurons	1024 x 256 x 512 x 64
7	Model 6 with dropout layers	1024 x 256 x Dropout(0.2) x 512 x Dropout(0.2) x 64

For this evaluation, these models have been defined in Kafka-ML, grouped, and deployed with the same training setup. Specifically, this consists of a training of 32 epochs with a batch size of 16 samples. MAE and Mean metrics have been used to compare the precision of the models. Figure 9 shows the MAE decrease for the different models and Figure 10, the corresponding for the MSE.

As can be seen in the MAE and MSE validation, we found that `model_2` as well as the modified model with Dropout layer obtained the best performance. Its architecture can be seen in Figure 8. Once selected `model_2` as the baseline model, we move on to the inference experiments using this model.

For the inference evaluation, we measure the response time of the inference service. In order to measure this, we have carried out different experiments by

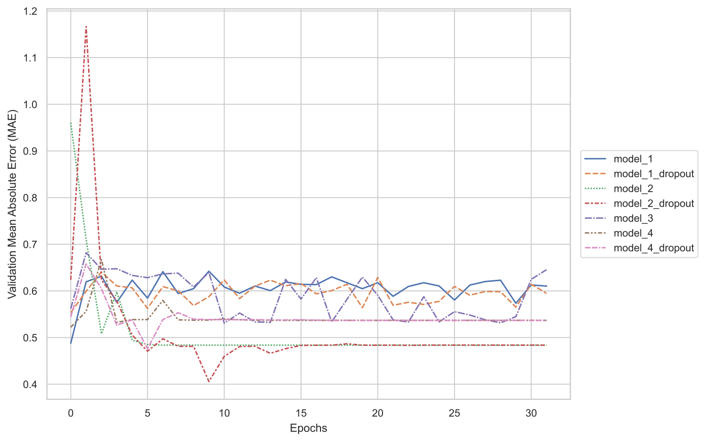


Fig. 9 Validation MAE decrease of the evaluated models during training.

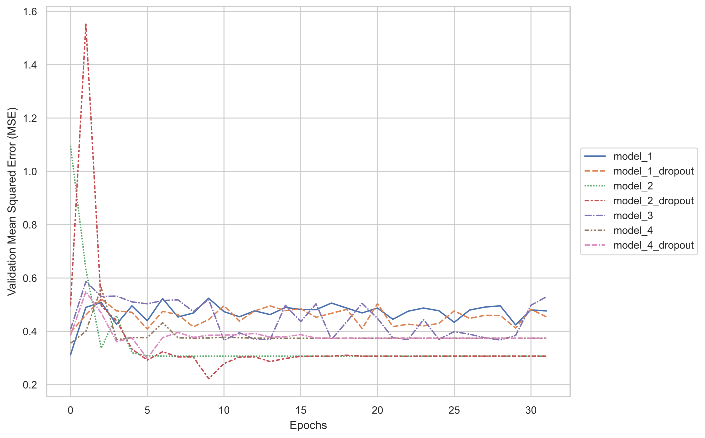


Fig. 10 Validation MSE decrease of the evaluated models during training.

deploying the service in different setups. In these setups, we have varied the number of clients sending sensor data to be inferred. In addition, the number of replicas, and the number of Kafka partitions were also modified.

These benchmarks are important since it is possible that in some use cases the user may require high availability of the predictive models due to the need of having multiple sensors deployed across a measurement area.

The setups used are as follows:

- One un-partitioned topic and the model replicated once.
- One topic with two partitions and the model replicated twice.
- One topic with four partitions and the model replicated four times.
- One topic with eight partitions and the model replicated eight times.

Each test consists of sending 512 messages with sensor data and measuring how long the model takes to predict. This was done 32 times in order to obtain an average result. These tests have been carried out with various inference replicas setups, and various partitions in the Kafka topics. Furthermore, each test has been replicated with different numbers of clients (specifically with 1, 2, 4, 8, 16, and 32 clients, respectively), making requests, seeking to evaluate how the system behaves when faced with a multitude of requests.

The most basic use case of Kafka-ML is the setup where we use one topic in a single partition and no replication of the inference module. It can be observed in Figure 11 that this setup works fine with a few clients but as soon as we scale the number it gets overloaded and the latency increases.

Having a greater number of replicas of the model and a greater distribution of the data in Kafka improve this result. This can be seen in cases requiring higher availability, where the latency is clearly lower (except when we have already reached a limit that cannot be improved by replication and partitioning). Therefore, it can be concluded that having more replication and partitioning of data can reduce latency in cases where higher availability is required.

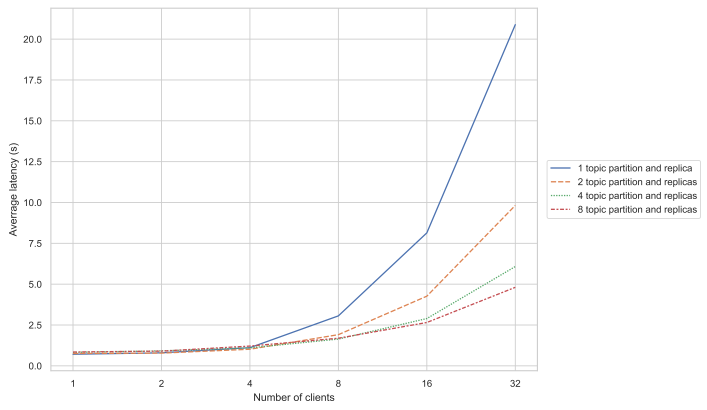


Fig. 11 Average inference latency response with different number of clients, topic partitions, and replicas.

Another test was carried out to measure the response capacity of Kafka-ML at different data transmission frequencies. In this test, clients (simulating IoT devices) generated data at certain time intervals and we seek to observe the responsiveness of Kafka-ML as well as the point where there is no longer a bottleneck in the prediction.

As a result (Figure 12), it was found that from approximately 2.25 seconds onwards, Kafka-ML is able to respond to data sent at this frequency without any bottleneck.

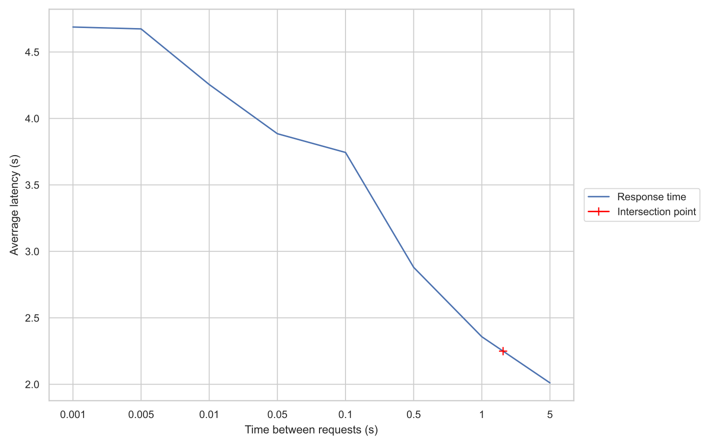


Fig. 12 Responsiveness of Kafka-ML towards data transmission at different time intervals.

5 Conclusions and future work

In this chapter, we have presented a methodology for the definition and deployment of soft sensors using Kafka-ML, a framework for the management of ML/AI pipelines with data streams. In this methodology, we have presented step by step how to design a soft sensor including: 1) soft sensor design and IoT data dispatching and integration into Kafka-ML; 2) ML model design, training, and validation for predicting a feature of interest; 3) trained ML model deployment and data ingestion for real-time soft sensor inference; and finally, 4) a fast-delivery application with the Kafka-ML visualization. All these steps only require the Kafka-ML framework, which reduces implementation efforts and makes the design and implementation of soft sensors much easier.

A use case with open data from the water quality of UK rivers has been presented to show the potential of Kafka-ML for the deployment of soft sensors. In addition, a series of experiments in different scenarios have been carried out on the proposed methodology, resulting in a low-latency product that can be configurable for fault tolerance and high availability.

As a future work, the following improvements to Kafka-ML and the proposed methodology are considered:

- User experience improvement. From a user's point of view, we have observed some possible improvements that could enhance Kafka-ML for the proposed methodology. For example, the model creation view could be redefined to simplify it, if necessary, so that users outside the computer science domain are able to define ML/AI models in an intuitive way.
- Automatic hyperparameter optimization. The automatic selection of the best hyperparameters for the model would improve the evaluation of different models, allowing to automatically obtain the best combination of training parameters.

Furthermore, and related to the aforementioned, this would allow users external to the domain to create fine-grained solutions in a fully automated way. In order to integrate this functionality, one of our choices is to adapt certain features of frameworks related to this area, such as Ray Tune [34] or Optuna [35], and integrate it into a new module to use at any time.

- Pretrained ML models inclusion. Researchers in other areas may already have ML models that perform the job of a soft sensor thanks to previous research efforts. Kafka-ML could open the possibility that these models could be integrated and usable for real-time data reception from sensors.
- Data preprocessing and postprocessing. In most ML application use cases, data preprocessing or postprocessing is necessary for better model performance. While this is achievable before sending the data to the model, either for training or for inference, a potential functionality for Kafka-ML and the proposed methodology would be the integration of data processing activities as well as statistical analysis for this.

References

1. M. Díaz, C. Martín, and B. Rubio, "State-of-the-art, challenges, and open issues in the integration of internet of things and cloud computing," *Journal of Network and Computer applications*, vol. 67, pp. 99–117, 2016.
2. P. Kadlec, B. Gabrys, and S. Strandt, "Data-driven soft sensors in the process industry," *Computers & Chemical Engineering*, vol. 33, no. 4, pp. 795–814, 2009.
3. Q. Jiang, X. Yan, H. Yi, and F. Gao, "Data-driven batch-end quality modeling and monitoring based on optimized sparse partial least squares," *IEEE Transactions on Industrial Electronics*, vol. 67, no. 5, pp. 4098–4107, 2020.
4. C. Martín, P. Langendoerfer, P. S. Zarrin, M. Díaz, and B. Rubio, "Kafka-ml: connecting the data stream with ml/ai frameworks," *Future Generation Computer Systems*, vol. 126, pp. 15–33, 2022.
5. "Kubernetes." Available online: `https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/`, (accessed on 9 Oct 2022.
6. A. Carnero, C. Martín, D. R. Torres, D. Garrido, M. Díaz, and B. Rubio, "Managing and deploying distributed and deep neural models through kafka-ml in the cloud-to-things continuum," *IEEE Access*, vol. 9, pp. 125478–125495, 2021.
7. G. Welch and G. & Bishop, "An introduction to the kalman filter," *University of North Carolina, Department of Computer Science*, 2001.
8. G. Bastin and D. Dochain, *On-line estimation and adaptive control of bioreactors*, vol. 1. Elsevier, 1990.
9. Q. Sun and Z. Ge, "A survey on deep learning for data-driven soft sensors," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 9, pp. 5853–5866, 2021.
10. W. Yan, H. Shao, and X. Wang, "Soft sensing modeling based on support vector machine and bayesian model selection," *Computers & Chemical Engineering*, vol. 28, no. 8, pp. 1489–1498, 2004.
11. A. Rani, V. Singh, and J. Gupta, "Development of soft sensor for neural network based control of distillation column," *ISA Transactions*, vol. 52, no. 3, pp. 438–449, 2013.
12. S. Singh, A. G. Anil, V. Kumar, D. Kapoor, S. Subramanian, J. Singh, and P. C. Ramamurthy, "Nitrates in the environment: A critical review of their distribution, sensing techniques, ecological effects and remediation," *Chemosphere*, vol. 287, p. 131996, 2022.

13. H. Haimi, F. Corona, M. Mulas, L. Sundell, M. Heinonen, and R. Vahala, “Shall we use hardware sensor measurements or soft-sensor estimates? case study in a full-scale wwtp,” *Environmental Modelling & Software*, vol. 72, pp. 215–229, 2015.
14. A. H. Zare, V. Bayat, and A. P. Daneshkare, “Forecasting nitrate concentration in groundwater using artificial neural network and linear regression models,” *International Agrophysics*, vol. 25, 2011.
15. F. Corona, M. Mulas, H. Haimi, L. Sundell, M. Heinonen, and R. Vahala, “Monitoring nitrate concentrations in the denitrifying post-filtration unit of a municipal wastewater treatment plant,” *Journal of Process Control*, vol. 23, no. 2, pp. 158–170, 2013. IFAC World Congress Special Issue.
16. H. Haimi, M. Mulas, F. Corona, and R. Vahala, “Data-derived soft-sensors for biological wastewater treatment plants: An overview,” *Environmental Modelling & Software*, vol. 47, pp. 88–107, 2013.
17. M. Haghbin, A. Sharafati, and B. e. a. Dixon, “Application of soft computing models for simulating nitrate contamination in groundwater: Comprehensive review. assessment and future op portunities,” *Computat Methods Eng*, vol. 28, 2021.
18. J. Kreps, N. Narkhede, J. Rao, *et al.*, “Kafka: A distributed messaging system for log processing,” in *Proceedings of the NetDB*, vol. 11, pp. 1–7, 2011.
19. Z. Wan, Z. Zhang, R. Yin, and G. Yu, “Kfiml: Kubernetes-based fog computing iot platform for online machine learning,” *IEEE Internet of Things Journal*, vol. 9, no. 19, pp. 19463–19476, 2022.
20. K. Kumar, N. A. Sharma, and A. B. M. S. Ali, “Machine learning solutions for investigating streams data using distributed frameworks: Literature review,” in *2021 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)*, pp. 1–6, 2021.
21. R. A. Ariyaluran Habeeb, F. Nasaruddin, A. Gani, M. A. Amanullah, I. Abaker Targio Hashem, E. Ahmed, and M. Imran, “Clustering-based real-time anomaly detection—a breakthrough in big data technologies,” *Transactions on Emerging Telecommunications Technologies*, vol. 33, no. 8, p. e3647, 2022. e3647 ett.3647.
22. “Kubeflow,” 2022. Available online: `https://www.kubeflow.org/`.
23. L. Yeager, J. Bernauer, A. Gray, and M. Houston, “Digits: the deep learning gpu training system,” in *ICML 2015 AutoML Workshop*, pp. 1–4, 2015.
24. E. Liberty, Z. Karnin, B. Xiang, L. Rouesnel, B. Coskun, R. Nallapati, J. Delgado, A. Sadoughi, Y. Astashonok, P. Das, *et al.*, “Elastic machine learning algorithms in amazon sagemaker,” in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pp. 731–737, 2020.
25. “Algorithmia - the enterprise mlops platform,” 2022. Available online: `https://algorithmia.com/`.
26. “Valohai,” 2022. Available online: `https://www.valohai.com/`.
27. D. Vohra, “Apache avro,” in *Practical Hadoop Ecosystem*, pp. 303–323, Springer, 2016.
28. D. R. Torres, C. Martín, B. Rubio, and M. Díaz, “An open source framework based on kafka-ml for distributed dnn inference over the cloud-to-things continuum,” *Journal of Systems Architecture*, vol. 118, p. 102214, 2021.
29. C. Olston, N. Fiedel, K. Gorovoy, J. Harmsen, L. Lao, F. Li, V. Rajashekhar, S. Ramesh, and J. Soyke, “Tensorflow-serving: Flexible, high-performance ml serving,” *arXiv preprint arXiv:1712.06139*, 2017.
30. “Bentoml: A faster way to ship your models to production,” 2022. Available online: `https://www.bentoml.com/`.
31. “Grafana: The open observability platform,” 2022. Available online: `https://grafana.com/`.
32. OpenDataNI, “River water quality monitoring 1990 to 2018..” Available at: `http://data.europa.eu/88u/dataset/river-water-quality-monitoring-1990-to-2018`.
33. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

34. R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, "Tune: A research platform for distributed model selection and training," *arXiv preprint arXiv:1807.05118*, 2018.
35. T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyper-parameter optimization framework," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2623–2631, 2019.