IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received September 19, 2019, accepted October 7, 2019, date of publication October 18, 2019, date of current version October 31, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2948160

ML4IoT: A Framework to Orchestrate Machine Learning Workflows on Internet of Things Data

JOSÉ M. ALVES[1], LEONARDO M. HONÓRIO[2], AND MIRIAM A. M. CAPRETZ[1], (Senior Member, IEEE)
[1]Department of Electrical and Computer Engineering, Western University, London, ON N6A 5B9, Canada
[2]Department of Electrical Energy, Federal University of Juiz de Fora, Juiz de Fora-MG 36036-900, Brazil

Corresponding author: Miriam A. M. Capretz (mcapretz@uwo.ca)

This work was supported by the NSERC STPGP, Western University, under Grant STPGP 506840-17.

ABSTRACT Internet of Things (IoT) applications generate vast amounts of real-time data. Temporal analysis of these data series to discover behavioural patterns may lead to qualified knowledge affecting a broad range of industries. Hence, the use of machine learning (ML) algorithms over IoT data has the potential to improve safety, economy, and performance in critical processes. However, creating ML workflows at scale is a challenging task that depends upon both production and specialized skills. Such tasks require investigation, understanding, selection, and implementation of specific ML workflows, which often lead to bottlenecks, production issues, and code management complexity and even then may not have a final desirable outcome. This paper proposes the Machine Learning Framework for IoT data (*ML4IoT*), which is designed to orchestrate ML workflows, particularly on large volumes of data series. The ML4IoT framework enables the implementation of several types of ML models, each one with a different workflow. These models can be easily configured and used through a simple pipeline. ML4IoT has been designed to use container-based components to enable training and deployment of various ML models in parallel. The results obtained suggest that the proposed framework can manage real-world IoT heterogeneous data by providing elasticity, robustness, and performance.

INDEX TERMS Big data, container-based virtualization, IoT, machine learning, machine learning workflow, microservices.

I. INTRODUCTION

Gartner [1], predicts that the Internet of Things (IoT) will reach 26 billion internet-connected devices by 2020, impacting a wide range of industries. The IoT is about connecting any device to the Internet, enabling the digitization of devices, vehicles, and other elements of the real world. Machine learning (ML) has been increasingly used with IoT data to create new applications, such as smart cities [2], smart homes [3], and smart grids [4]. An IoT application may provide vast amounts of real-time data through its application and business layers. Analysis of these temporal data series, through machine learning, can provide numerous benefits in several processes and services [5], [6].

More specifically, ML algorithms are used to convert data into valuable information [7]. In real-world IoT applications, ML development is divided into two phases: training and inference. The training phase typically begins with ingestion, storage, and preprocessing of IoT data. After this, ML models are trained using the preprocessed IoT data. The inference phase is also referred to in the literature as prediction serving, model score, or model prediction [7] and uses the trained ML models to infer information using new online IoT data.

The high volume, velocity, and variety of IoT data [8] require the use of several Big Data-enabling tools to ingest, store and preprocess the data even before selecting a given ML workflow. Furthermore, integrating those tools with ML software to create end-to-end workflows is a time-consuming task that requires specialized skills, and complex coding. For example, the development of typical ML applications using IoT data usually involves three steps. First, data engineers code data workflows that produce training data using Big Data tools. Second, data scientists downsample data and use ML frameworks in notebook environments (e.g., Jupyter [9], Zeppelin [10]) to develop and experiment with new models. Finally, software engineers work to deploy trained models

The associate editor coordinating the review of this manuscript and approving it for publication was Byung-Seo Kim.

in production systems. The hand-off between these steps leads to bottlenecks and production issues, which are signs of many of the machine learning anti-patterns described by Sculley *et al.* [11]. Moreover, applying machine learning to real IoT data includes challenges such as keeping models updated and running multiple ML workflows in parallel.

Most big players in machine learning have faced these challenges and started solving them internally with their own platforms. For example, Uber has built its ML orchestration platform called Michelangelo [12], Airbnb has Bighead [13], Netflix has developed the Meson platform [14], Google has introduced TensorFlow Extended (TFX) [15], and Facebook has implemented its data pipeline platform for generating and predicting models [16]. These are all in-house proprietary platforms to make sure that their time and money are not wasted in developing repetitive ML workflows and management tools. However, besides needing skilled users, these frameworks still present production issues related to treating and managing data and dealing with different methodologies and frameworks. Therefore, a unified framework that is capable of executing diverse ML workflows, handling and managing data, running configurations in parallel and not demanding advanced programming skills would be beneficial.

Considering the drawbacks mentioned above, the main contribution of this research is to overcome the challenges of integrating Big Data tools and machine learning software to provide a unified platform where ML applications can be developed using IoT data. Hence, a flexible, robust, and scalable Machine Learning Framework for IoT data (*ML4IoT*) has been designed and developed in this study to orchestrate machine learning workflows. The flexibility is obtained by organizing backend services, which uses containerized microservices to execute different tasks defined especially for each implemented workflow. In addition, to address the challenges of running multiple ML workflows in parallel, *ML4IoT* has been designed to use container-based components that provide a convenient mechanism to enable the training and deployment of numerous ML models in parallel. Furthermore, the use of containers provides process isolation between different ML workflows and ensures that a single workflow failure does not affect the execution of other workflows running in parallel, which make the process robust. Finally, to address common production issues faced during development of ML applications, such as *hard-to-maintain glue code*, *hidden dependencies*, *feedback loops*, and *pipeline jungles*, *ML4IoT* uses a microservices architecture to bring flexibility, reusability, and scalability to the framework.

ML4IoT was evaluated in three experiments using two types of real-world IoT data from the energy and traffic domains with four different objectives (forecasting for 15, 30, 45 and 60 minutes ahead), and two ML algorithms Random Forest (RF) [17], [18] and Long Short-Term Memory (LSTM) [19]. The first experiment investigated the feasibility of the framework by assessing its ability to orchestrate ML workflows on different IoT datasets. *ML4IoT* elasticity was evaluated in the second experiment by studying its ability to scale to support the execution of multiple ML workflows in parallel. Finally, the third experiment investigated the performance of the framework by analyzing the latency of the execution of ML workflows deployed to render predictions using online IoT data.

The rest of this paper is organized as follows. Section II provides an overview of the major approaches related to the proposed framework. Section III presents the proposed framework and the design of its components. Section IV shows the implementation details. Section V presents an evaluation of the framework. Finally, Section VI presents the final paper conclusions and future work.

II. RELATED WORK

This section presents research related to this paper divided into two categories: Machine Learning Platforms, and Big Data and Machine Learning tools applied to IoT.

A. MACHINE LEARNING PLATFORMS

This section presents a review of ML platforms that have been proposed to address challenges in developing and deploying ML models.

Wang *et al.* [20] introduced Rafiki, which is a system to provide the training and inference services for ML models. In their work, users are exempted from constructing ML models, tuning hyper-parameters, and optimizing prediction accuracy and speed. Instead, they upload their datasets, configure the service to conduct training and then deploy the model for inference.

Lee *et al.* [21] presented PRETZEL, a prediction serving system designed to serve predictions over trained pipelines originally developed in ML.Net [22]. Their work explored multi-pipeline optimization techniques to reduce resource utilization and improve performance.

Spell *et al.* [23] presented Flux, which is a system designed to deploy ML models in distributed batch mode and make them available as microservices for real-time use cases. The Flux system provides REST APIs for programmatic access to ML models and their executions. Their work supports multiple ML libraries and includes monitoring and archiving of ML models.

Studies [20], [21], [23] have been focused on providing generic ML platforms, which are in turn focused on achieving better efficiency during training and deployment of ML models. Unlike this work, these studies do not address the challenges posed by IoT data, such as high volume, high variety, and high velocity, which impacts the use of these studies in real-world scenarios. In addition, these existing machine learning platforms do not deal with IoT data preprocessing, which is a common step required during development of ML applications. Because of this, additional components need to be integrated with the previously cited platforms to create end-to-end machine learning applications on IoT data. This research extends previously cited approaches by providing a framework designed to orchestrate and automate the execution of ML workflows on IoT data.

B. BIG DATA AND MACHINE LEARNING TOOLS APPLIED TO IoT

This section presents a review of research that addressed the use of big data enabling tools and ML frameworks with IoT data.

Cecchinel *et al.* [24] presented an approach using machine learning to provide an optimal configuration that extended the battery lifetime of IoT sensors. In their work, middleware generates an energy-efficient sensor configuration based on live data observations, which dynamically optimizes sensors sampling frequency and network usage.

Yang *et al.* [25] proposed a semi-supervised method in association with a generative adversarial network [26] to support medical decision-making in an IoT-based health service system. Their approach was designed to solve problems involving both the lack of labelled sets and imbalanced classes, which are common in medical datasets collected from IoT-based platforms.

Kotenko *et al.* [27] introduced a framework combining Big Data processing and machine learning for security monitoring of the mobile Internet of Things. Their work defined several machine learning mechanisms intended to solve classification tasks using IoT data.

Lakshmanaprabu *et al.* [28] outlined a hierarchical framework for feature extraction in Social Internet of Things (SIoT) data using a MapReduce framework [29] along with a supervised classifier model. In addition, a Gabor filter [30] was used to reduce noise and unwanted entries in the SIoT data.

The studies cited above aimed to use ML techniques to solve specific problems in IoT environments, for example, IoT sensor efficiency [24], healthcare [25], security [27], and the Social Internet of Things [28]. Unlike these studies, this research proposes a general-purpose framework that supports machine learning on various IoT datasets using different ML algorithms.

Preuveneers *et al.* [31] proposed the SAMURAI framework, which is a batch and online data processing framework based on the Lambda architecture [32]. Their work integrates components for complex event processing, machine learning, and knowledge representation. Horizontal scalability is achieved with Big Data enabling technologies such as Apache Spark and Apache Storm.

Ta-Shma *et al.* [33] presented an architecture for extracting valuable historical insights and actionable knowledge from IoT data streams. Their architecture supports both real-time and historical data analytics using a hybrid data processing model. The main components used in their architecture instance (Node-Red, Apache Kafka, Apache Spark, and OpenStack Swift) were implemented using the microservices approach.

Almeida *et al.* [34] presented an architectural model to support scalability, flexibility, autonomy, and heterogeneity demands from IoT environments. Their work provides event collection, situation detection through on-the-fly event processing, and customizable dynamic reaction features. The architecture proposed was based on a middleware for ubiquitous computing called EXEHDA [35], which follows a multi-level strategy and consists of three hierarchically interconnected modular components.

In contrast to previous studies [31], [33], [34], this work focusses on providing methods to enable efficient development of ML applications with IoT data. Unlike the studies mentioned previously, this research examines the orchestration and automation of end-to-end ML workflows to support parallel training and deployment of multiple ML models with IoT data. Existing platforms are dedicated to a specific ML algorithm, either to provide solutions to a particular problem, or to provide a big data solution to one of the previously-mentioned domains. This work, however, introduces a novel framework that is agnostic of deployed ML algorithms and can handle high volume of IoT data.

III. ML4IoT FRAMEWORK

The proposed *ML4IoT* framework, shown in Figure 1, uses containerized microservices to automate the execution of tasks specified in the ML workflows. Each container is designed through a REST API-based component and managed by the orchestrator and scheduler. *ML4IoT* is generally implementation-agnostic, and therefore can be easily expanded to support new components such as data preprocessing tasks, algorithm types, and ML frameworks. To accomplish this, the microservices architecture and container-based virtualization were combined to deal with challenges such as the multitude of ML frameworks available and the parallel execution of ML workflows.

More specifically, the framework's *Core* sets and manages containers responsible for training and deploying ML models. Each model is coded into a different *docker image* and becomes available for use. The *workflow designer* is responsible for building workflow batch jobs. The selection of each workflow job requires the definition of dataset connections, preprocessing data treatments, and the setting up of a given available ML model. The *Orchestrator* reads the workflow batch jobs and uses the *Container Management System* (CMS) to generate, for each workflow, a docker container microservice. Moreover, each microservice runs its tasks through the *Distributed Data Processing Engine*.

The IoT data used by the containers are ingested and stored in the *ML4IoTData Management* (DM) component. The DM is composed of three sub-components: a Messaging System, a Distributed File System, and a NoSQL database. Finally, the DM also supports the storage of trained models and predicted data, as well as providing temporary storage for preprocessed data.

This architecture enables the decoupling of software components into small and specialized services, bringing flexibility, reusability, and extensibility to the framework.

The following sub-sections detail the components and sub-components of the *ML4IoT* framework.

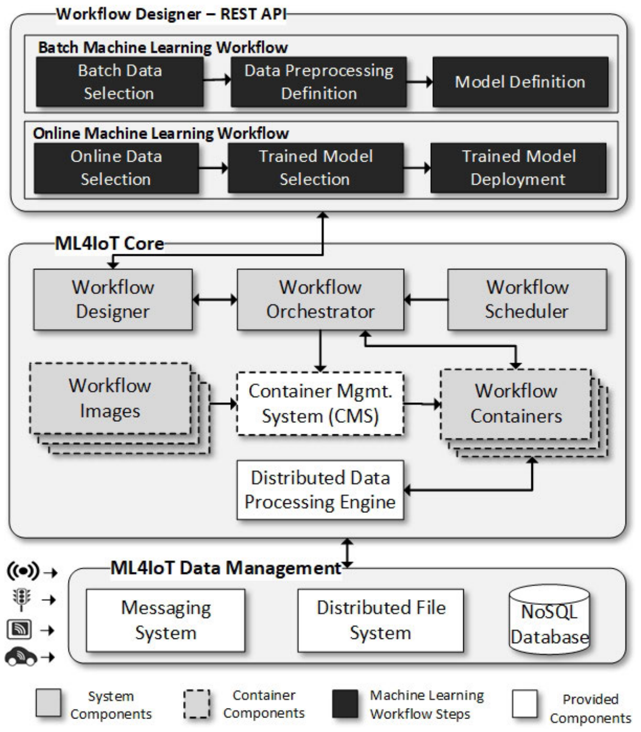


FIGURE 1. Machine learning framework for IoT data - ***ML4IoT***.

A. ML4IoT CORE

Although the Core has two different functionalities (training and deploying ML models), the same pipeline is applied to both. The Core is divided into five major components, as described next.

1) WORKFLOW DESIGNER

The *Workflow Designer* microservice is responsible for selecting and configuring the two types of ML workflows that are described below.

The *batch machine learning workflow* involves selecting and configuring historical datasets, preprocessing tasks, and ML algorithms. These tasks are executed in the three steps shown in Figure 1 and are discussed below. Note that these are the only three steps in which a user must interact with ML4IoT to obtain a final trained ML workflow. An intuitive GUI ensures that no major programming knowledge is necessary.

- *Batch Data Selection* defines the parameters used to create historical datasets during the execution of batch ML workflows. For instance, when this type of workflow is executed, historical datasets are created using IoT data and are stored in the *Distributed File System* (DFS). Examples of these parameters include the date range, the location in the DFS, and the attributes of the dataset.
- *Data Preprocessing Definition* specifies the preprocessing tasks to be applied to the datasets defined in the previous step. These tasks are defined for each dataset along with their parameters. For example, data aggregation is a common preprocessing task used for IoT data and can be specified in this step.
- *Model Definition* defines the ML algorithms along with their parameters to be used to train ML models. For example, the type of algorithm (e.g., LSTM, Linear Regression), specific algorithm configurations, and how the datasets are split between training and testing are some of the parameters defined in this step.

The *online machine learning workflow* is used to deploy trained ML models to infer new information using online IoT data. It involves selecting and configuring the online datasets, trained ML models, and deployment parameters required to execute this type of workflow. These tasks are performed in the three steps discussed below.

- *Online Data Selection* defines the parameters used to create datasets composed of online data during execution of online ML workflows. These online datasets are created using the IoT data that are ingested in the *Messaging System*.
- *Trained Model Selection* is the step where a trained ML model, built by the batch ML workflow, is selected for use to infer new information. The chosen model uses the previously created online datasets as input to issue predictions during execution of online ML workflows.
- *Trained Model Deployment* defines the parameters related to the deployment of a trained ML model. For example, execution of online ML workflows uses a micro-batch approach in which the steps defined in this type of workflow are executed repeatedly at a specific pre-defined interval, such as every 30 seconds, every minute, every 30 minutes, and so on. This interval is one of the parameters configured in this step. The micro-batch approach was chosen because IoT sensors have different duty cycles and intermittent connectivity, meaning that data are often delayed or misaligned. Executing the online ML workflows at small intervals helps to deal with these issues and is also useful for data preprocessing tasks where a sample data point needs to be correlated with previous values, such as the sliding window technique.

2) WORKFLOW ORCHESTRATOR

This component is a microservice that provides the operational capabilities needed to train and deploy the ML workflows. It automates execution, ensures consistent ML practices across the development lifecycle, and optimizes the computational power required to execute these workflows. By taking advantage of the conceptual data model used to store the ML workflows, the *Workflow Orchestrator* models the workflows as a sequence of steps, where each step can be executed by a containerized microservice. Figures 2 and 3 show the steps performed during the orchestration of batch and online ML workflows, respectively. It is important to note that each combination set during the designer step generates an individual container that will run independently. Figure 2 exemplifies that the number of ML models built is a combination of the number of datasets times the number of ML algorithms.

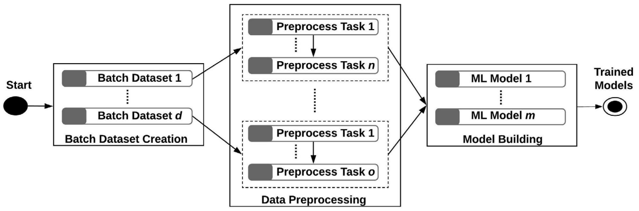


FIGURE 2. Orchestration steps of batch ML workflows.

The execution of batch ML workflows involves three steps: batch dataset creation, data preprocessing, and model building as explained next.

- *Batch dataset creation* is the first step performed in the execution of batch ML workflows. In Figure 1, the *Workflow Orchestrator* sends a request to the CMS to create a new containerized microservice. The new containerized microservice interacts with the *Distributed Data Processing Engine* to create a new dataset, using historical IoT data that are stored in the *Distributed File System*. When the *Distributed Data Processing Engine* finishes execution, the containerized microservice sends a request to the *Workflow Orchestrator* to execute the next step defined in the batch ML workflow.
- *Data Preprocessing* is the step where preprocessing tasks are applied to each dataset defined in the workflow. Similarly to the previous step, Figure 1 shows that the *Workflow Orchestrator* orchestrates the creation of a new containerized microservice, which uses the *Distributed Data Processing Engine* to apply data preprocessing tasks to a previously created dataset. Examples of preprocessing tasks include removal of null values, removal of repeated values, sliding window, data aggregation, and normalization.
- *Model Building* is the step where tasks are executed to train and test ML models according to the parameters defined in batch ML workflows. Figure 1 illustrates that the *Distributed Data Processing Engine* is used to train and test an ML model using a preprocessed dataset. The trained ML model generated at the end of its execution is stored in the *Distributed File System* and can be used in online ML workflows.

The management of *online ML workflows* also involves executing three computation steps: online dataset creation, data preprocessing, and model inference. Online dataset creation follows the same principle as the batch workflow and the preprocessing tasks are the same ones that were used to train the model being deployed. Moreover, Figure 3 shows that, for each trained model, the orchestration flow is executed repeatedly to render new predictions.

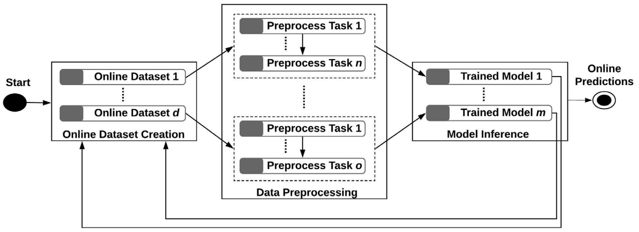


FIGURE 3. Orchestration steps of online ML workflows.

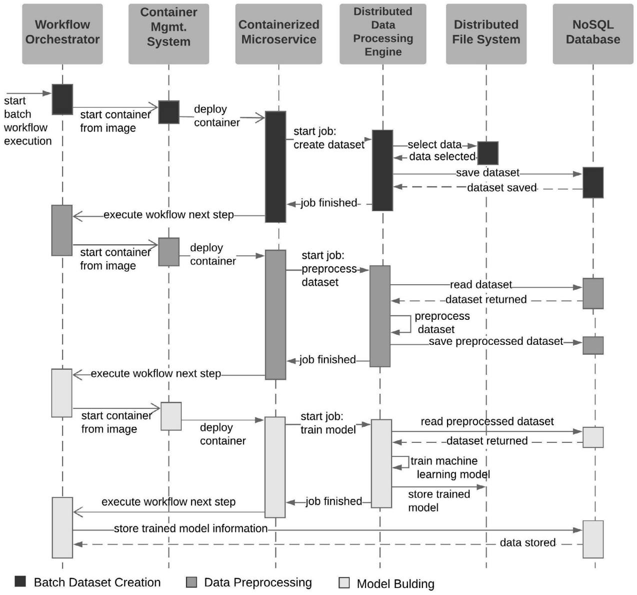

Workflow Orchestrator
Container Mgmt. System
Containerized Microservice
Distributed Data Processing Engine
Distributed File System
NoSQL Database
start batch workflow execution
start container from image
deploy container
start job: create dataset
select data
data selected
save dataset
dataset saved
job finished
execute wokflow next step
start container from image
deploy container
start job: preprocess dataset
read dataset
dataset returned
preprocess dataset
save preprocessed dataset
job finished
execute wokflow next step
start container from image
deploy container
start job: train model
read preprocessed dataset
dataset returned
train machine learning model
store trained model
job finished
execute workflow next step
store trained model information
data stored
■ Batch Dataset Creation
▨ Data Preprocessing
□ Model Bulding

FIGURE 4. Sequence diagram: execution of batch ML workflows.

More specifically, the *Workflow Orchestrator* manages the execution of these ML workflows by sending requests to the API of the *Container Management System*, which creates containerized microservices to run the tasks defined in each step. In addition, execution of the steps shown in Figures 2 and 3 involves interactions among the dynamically created containerized microservices and the other components of the framework. The sequence diagram depicted in Figure 4 shows an example of the interactions that occur during orchestration of a batch ML workflow. The diagram has three main parts, which are highlighted by different shades of grey in the activation bars and indicate the execution of tasks related to the steps depicted in Figure 2. The orchestration of online ML workflows uses a similar approach; however, the *Messaging System* is used to provide the online IoT data, and the predictions generated during execution of these workflows are stored in the *NoSQL* database. Moreover, the execution of online ML workflows is repeated at defined intervals.

3) WORKFLOW SCHEDULER

This component can re-execute batch ML workflows automatically at intervals defined in these workflows. In some cases, the trained ML models become outdated because they were trained with past data that do not represent the actual data distribution. Because data distribution can drift over time, building an ML model is not a one-time exercise, but rather a continuous process. The *Workflow Scheduler* can execute batch ML workflows at scheduled intervals to retrain ML models to keep them updated.

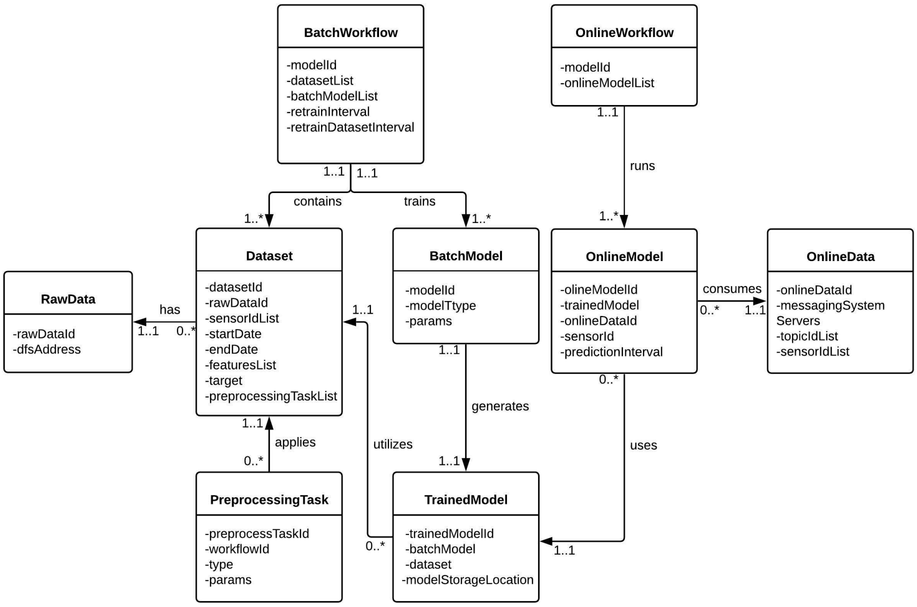


FIGURE 5. Conceptual data model for the ML4IoT.

The *Workflow Scheduler* is a microservice, which provides scheduling services in the framework by exposing a REST API where the re-execution of batch ML workflows can be configured. By allowing to schedule the re-execution of batch ML workflows, this component enables retraining of ML models during non-critical business hours and also facilitates the distribution of the processing workload along different periods of the day. To trigger the re-execution of batch ML workflows, the *Workflow Scheduler* sends a REST request to the *Workflow Orchestrator*, which orchestrates the necessary steps.

4) CONTAINER-BASED COMPONENTS: WORKFLOW IMAGES, CONTAINER MANAGEMENT SYSTEM, AND WORKFLOW CONTAINERS

Workflow Images, *Container Management System*, and *Workflow Containers* are container-based software used to support the execution of the ML workflows. The details of these sub-components are described below.

Workflow Images are Docker images in which reusable software code is implemented to execute the tasks defined in the ML workflows.

Container Management System (CMS) is the software responsible for deploying containers according to requests sent by the *Workflow Orchestrator*.

Workflow Containers are Docker containers created dynamically by the CMS based on requests from the *Workflow Orchestrator*. In *ML4IoT*, containers can be destroyed when they finish their execution, which optimizes the use of computational resources.

5) DISTRIBUTED DATA PROCESSING ENGINE

The *Distributed Data Processing Engine* is a distributed system designed to process large volumes of data. In the *ML4IoT* framework, the *Distributed Data Processing Engine* component executes data preprocessing tasks that require high computational power and need to be performed in a distributed way. *ML4IoT* was designed to use Apache Spark [36] as the *Distributed Data Processing Engine*.

B. *ML4IoT* DATA MANAGEMENT

ML4IoT Data Management provides components to ingest and store IoT data. These IoT data are used during the execution of batch and online ML workflows to train and deploy ML models. This component is made up of three sub-components: a *Messaging System*, a *Distributed File System*, and a *NoSQL* database. These sub-components are Big Data tools that can deal with the high-volume, high-speed, and high-variety characteristics of IoT data. The *Messaging System* is designed to ingest massive amounts of IoT data, the *Distributed File System* is used to store the data permanently, and the *NoSQL* database supports the temporary storage of preprocessed data during the orchestration of ML workflows. *ML4IoT* was designed to use Apache Kafka [37] as the *Messaging System*, Apache Hadoop as the *Distributed File System* [38], and MongoDB [39] as the *NoSQL* database.

In the document-oriented *NoSQL* database, the notion of a schema is dynamic: each ML workflow can contain different fields. This flexibility is particularly helpful for modeling the various workflows that can be created in *ML4IoT*. Moreover, having a representation of the ML workflows stored in unified and hierarchical documents adds an audit trail to these workflows and facilitates versioning and debugging. For instance, by using JSON documents to represent ML workflows, users can record executions and keep track of model parameters, code, and datasets for each ML workflow. Figure 5 shows an entity-relationship (ER) diagram that illustrates the

TABLE 1. Container images and services implemented in each image.

Workflow Type	Orchestration Step	Image Name	Description
Batch	Batch D. Creation	data-selection	Creates historical datasets
Batch / Online	Data Preprocessing	remove-nulls	Removes null values from datasets
Batch / Online	Data Preprocessing	remove-repeated	Removes repeated values from datasets
Batch / Online	Data Preprocessing	aggregate-values	Aggregates time-series data
Batch / Online	Data Preprocessing	sliding-window	Generates sliding-window features
Batch / Online	Data Preprocessing	normalize-values	Normalizes values of datasets
Batch	Model Building	lstm-building	Builds predictive models using LSTM
Batch	Model Building	rf-building	Builds predictive models using RF
Online	Online D. Creation	online-processing	Creates online datatsets
Online	Model Inference	lstm-online	Render predictions using LSTM models
Online	Model Inference	rf-online	Render predictions using RF models

conceptual data model used to store the ML workflows in the *NoSQL* database.

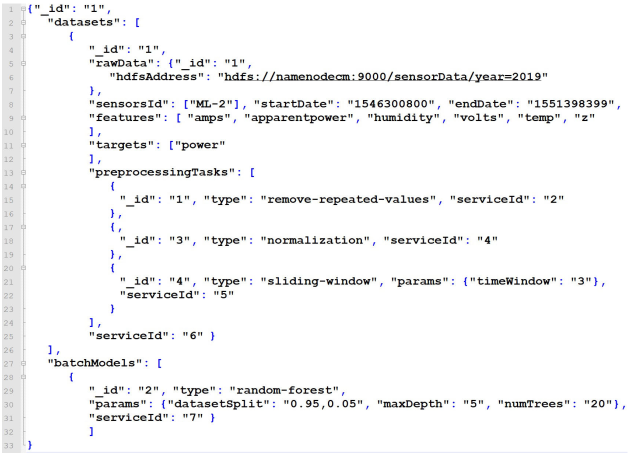
A description of the entity types depicted in Figure 5 is given below:

- **BatchWorkflow** contains a list of the *Dataset* entities, a list of *BatchModels* entities, and scheduling parameters for batch ML workflows.
- **RawData** describes the metadata of the IoT data stored in the distributed file system.
- **Dataset** describes the metadata of a dataset created in a batch ML workflow.
- **PreprocessingTask** describes a preprocessing task associated with a *Dataset*.
- **Batch Model** describes a ML algorithm and its parameters in a batch ML workflow.
- **TrainedModel** describes a trained ML model generated after the execution of a batch ML workflow.
- **OnlineData** describes the metadata of online IoT data ingested in the *Messaging System*.
- **OnlineModel** contains a *TrainedModel* entity, an *OnlineData* entity, and deployment parameters for each trained model.
- **OnlineWorkflow** contains a list of *OnlineModel* entities.

In relational databases, a table contains data about just one entity, whereas in the document-oriented *NoSQL* database, a document can contain one or more of the entities depicted in Figure 5. An example of a batch ML workflow stored in a JSON document is shown in Figure 6.

IV. IMPLEMENTATION DETAILS

A prototype system based on the proposed framework was built for use as a proof of concept. The implementation details of the prototype system are shown in Figure 7 and described next.



```
{"_id": "1",
   "datasets": [
       {
          "_id": "1",
          "rawData": {"_id": "1",
             "hdfsAddress": "hdfs://namenodecm:9000/sensorData/year=2019"
          },
          "sensorsId": ["ML-2"], "startDate": "1546300800", "endDate": "1551398399",
          "features": [ "amps", "apparentpower", "humidity", "volts", "temp", "z"
          ],
          "targets": ["power"
          ],
          "preprocessingTasks": [
             {
              "_id": "1", "type": "remove-repeated-values", "serviceId": "2"
             },
             {,
              "_id": "3", "type": "normalization", "serviceId": "4"
             },
             {
              "_id": "4", "type": "sliding-window", "params": {"timeWindow": "3"},
              "serviceId": "5"
             }
          ],
          "serviceId": "6" }
   ],
   "batchModels": [
       {
          "_id": "2", "type": "random-forest",
          "params": {"datasetSplit": "0.95,0.05", "maxDepth": "5", "numTrees": "20"},
          "serviceId": "7" }
          ]
}
```

FIGURE 6. Batch ML workflow represented in a JSON file.

ML4IoT Core. The three microservices, *Workflow Designer*, *Workflow Orchestrator*, and *Workflow Scheduler*, are implemented in Java and Spring Boot. The *Container Management System* (CMS) and the *Distributed Data Processing Engine* used in this evaluation were Docker Swarm and Apache Spark Framework, respectively. The *Distributed Data Processing Engine* uses Spark. Table 1 shows the container images that were implemented to support the orchestration of ML workflows.

ML4IoT Data Management. ML4IoT was designed to use Apache Kafka [37] as the *Messaging System*.The *Distributed File System* uses Apache Hadoop [38], and finally, MongoDB [39] is used for the *NoSQL* database.

Three servers were used to deploy the prototype system and are described in Table 2.

The servers were connected to an HP MSA 2040 SAN 12 TB storage system, which was divided into four logical partitions of 2.5 TB each. The storage system was configured to allow parallel access to all configured partitions. The big

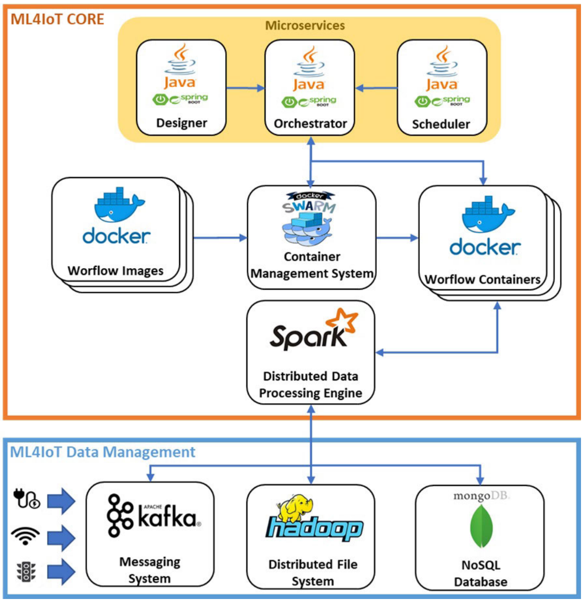


FIGURE 7. Implementation details of the *ML4IoT*.

TABLE 2. Hardware environment.

Server	CPU	RAM
Server 1	2 x Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz	96 GB
Server 2	2 x Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz	96 GB
Server 3	2 x Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz	96 GB

TABLE 3. Clusters configuration.

Application	Configuration
HDFS	1 name node, 3 data nodes
Apache Kafka	1 zookeeper, 3 data nodes
MongoDB	3 shards, 3 configuration servers, 3 routers
Spark	1 master node, 3 slave nodes

data tools used to implement the prototype system were configured in clusters running on Linux containers. Table 3 describes the initial cluster's configuration of each distributed application.

The data are ingested into Apache Kafka at an approximate average rate of 520 records/sec. Data are retained for one month, which gives about 275 GB of stored data. The HDFS has the capacity to store more than five billions sensor readings in a compressed format (ORC), reducing the data size to 44 GB.

Finally, each entry of the dataset used to train the ML models is a 5-minute average of each variable read.

V. EVALUATION

This section presents an evaluation of *ML4IoT* focused on three main aspects: machine learning orchestration, elasticity, and performance. It is important to note that this work does not aim to develop or improve any individual ML algorithm, instead it provides a framework that can easily evaluate several models/workflows for a given dataset. In this way, the goal is the overall performance comparison for a given available metric. This section first details the IoT data used in the experiments. Then, it describes the tests conducted to evaluate the framework.

TABLE 4. Features and description of the energy data.

Features	Description
Apparent Power	The power supplied to the electric circuit
Reactive Power	The energy generated/absorbed to maintain a constant voltage
Power	The real power that the electric circuit is consuming
PF	The ratio of real power to apparent power
Amps	The electric current flow in the circuit
Volts	The potential energy that the electric circuit can provide
Frequency	The number of cycles per second in an alternating current
Z	The electric circuit impedance
Humidity	The quantity of water vapour present in the air
Temp	The external temperature of the sensor's location

A. INTERNET OF THINGS DATA

In the experiments presented in this section, two distinct sets of real-world IoT data were used; energy consumption, and traffic flow. Energy consumption prediction is crucial for improved decision-making to reduce energy consumption and CO_2 emission. Predicting traffic flow can also help reduce air pollution and provide more efficient traffic conditions.

The details of each experiment are given below.

1) ENERGY DATA

The energy IoT data were collected in collaboration with T-innovation Partners, a company that offers the use of innovative products for monitoring and control of electrical systems. The data were provided by nine sensors that collect energy features from a building located at Western University, London, Canada. The IoT data were pulled from a REST API, which sampled the data at two-second intervals. For each sensor, 60 distinct features were collected. These features were composed of ten distinct energy domain measures shown in Table 4 multiplied by six mathematical operators.

These mathematical operators were standard deviation (sd), average (avg), last value (last), minimum value (min), maximum value (max), and sum. The energy data generated an average of 40,563,069 sensor readings per day. The data were collected for four months, generating a total of 4,867,568,280 data samples.

2) TRAFFIC DATA

The IoT traffic data were provided by the Madrid Council, which has deployed roughly 3000 traffic sensors in fixed locations around the city of Madrid on the M30 ring road. Madrid Council published the data using a REST API [40], where the data were refreshed every 5 minutes. The features available in the traffic data are described in Table 5.

TABLE 5. Features and description of the traffic data.

Features	Description
Occupancy (ocupacion)	Percentage occupancy of the location
Service level (nivelServicio)	Indicates whether the road is congested
Load (carga)	Based on intensity and occupation
Velocity (velocidad)	Average speed of the vehicles
Intensity (intensidad)	Number of vehicles per hour

The traffic data consisted of an average of 1,006,319 sensor readings per day. The data were collected for four months, creating 120,758,326 data samples.

B. MACHINE LEARNING ORCHESTRATION EVALUATION

To demonstrate how *ML4IoT* can automate ML workflow orchestration in IoT data, batch ML workflows were created to train and predict short-term energy consumption and traffic flow ML models. The ML workflows were defined by using the designer component; no programing skills were required. Moreover, the tasks defined in each workflow were executed automatically while being orchestrated by backend services. Details of the batch and online ML workflows created in this experiment are described next.

1) Batch ML workflows

Four batch ML workflows were built to train the ML models with prediction horizons of 15, 30, 45 and 60 minutes ahead. The number of historical entries used to train each batch is proportional to its prediction horizon, and provides the immediate next data entry prediction as output. The full time ahead prediction was obtained by using a sliding window approach. For instance, consider the following ML model used to predict 15 minutes ahead[1]

$$\hat{X}_{t+1} = f(X_t, X_{t-1}, X_{t-2}) \tag{1}$$

where $t \in \mathbb{N}$ is a dataset entry, $X_t \in \mathbb{R}^n$ is a vector of all n features of t, $\hat{X}_{t+1} \in \mathbb{R}^n$ is the prediction output and $f : \mathbb{R}^{3n} \mapsto \mathbb{R}^n$ is a trained ML model that uses the last three entries to predict the next one. The sliding window approach uses an output prediction as input to the next step. Therefore, to predict the 15 minutes ahead two more predictions are required as shown next

$$\hat{X}_{t+2} = f(\hat{X}_{t+1}, X_t, X_{t-1}) \tag{2}$$
$$\hat{X}_{t+3} = f(\hat{X}_{t+2}, \hat{X}_{t+1}, X_t) \tag{3}$$

Finally each batch was defined to execute both datasets (energy and traffic) at the same time.

TABLE 6. Datasets description.

Dataset	Inputs	Target	Readings
Energy	Power, ApparentPower, ReactivePower, PF, Amps, Volt, Temp Frequency, Z, Humidity,	Power	252,447,882
Traffic	Velocity, Occupancy, Service Level, Load, Intensity	Velocity	12,828

Algorithm 1 Training of Machine Learning Models

Input: *Historical Dataset (D), Model (M), Training Parameters (TP)*

Output: *List of trained models(TM)*

```
$D_{train}, D_{test} \leftarrow$ splitDataset($D$)
**if** output from $Model(M)$ is multiple **then**
    $trainedModel \leftarrow$modelBuilding($D_{train}$, $D_{test}$,$model$, $TP$)
    append $trainedModel$ to $TM$

**else if** output from $Model(M)$ is single **then**
    **for each** $attribute$ in $D_{train}$ **do**
        $trainedModel \leftarrow$ modelBuilding($D_{train}$, $D_{test}$, $model$, $TP$)
        append $model$ to $TM$
```

Batch Data Selection. The energy dataset uses 10 features where the target is total power. The traffic dataset uses 5 features and its target is velocity. In this case the target indicates which variable the training process must minimize during ML model creation. Both datasets contained two months of historical data. Table 6 shows details of the energy and traffic datasets used in this experiment. The datasets were split between a training set (95%) and a test set (5%).

Data Preprocessing Definition. The tasks of removing null values, removing repeated values, sliding-window rearrangement, and normalization were defined in the workflows and applied to both historical datasets. The data aggregation task was performed only on the energy data because the collected data were aggregated into two-second intervals. For this reason, the energy data were aggregated into five-minute intervals.

Model Definition. Each batch ML workflow was set to run two ML algorithms, Random Forest and Long Short-Term Memory. In this way, four models (2 datasets x 2 ML algorithms) were trained in each workflow. The details of the parameters used in each algorithm are described below.

Random Forest (RF): was trained using 20 trees with a maximum depth of five. The number of features to consider for splits at each tree node was one-third of the total number of features. The MLlib framework provided the algorithm.

Long Short-Term Memory (LSTM): was trained using an LSTM network with five layers and five neurons in each layer. A dropout rate of 0.25 was applied to the dense layers. The Tensorflow and Keras libraries provided the algorithm.

The steps performed to train the ML models are described in Algorithm 1, which contains an if statement to check the

[1] it gives a total of 3 entries considering a 5 minute entry timeframe

Algorithm 2 Inference of Machine Learning Models

Input: *Online data (OD), List of trained models(TM), Time Steps to Predict (T)*

Output: *Predicted Data(PD)*

```
for $T$ interactions do
    if trainedModelOutputType is Multiple then
        $nextPredictedStep_{list}$ ←modelPrediction
(onlineData, trainedModel)
        append $nextPredictedStep_{list}$ to onlineData
        append nextPredictedStep to $predictedOutput_{list}$

    else if trainedModelOutputType is single then
        for each attribute in onlineData do
            nextPredictedStep ←modelPrediction(
onlineData, trainedModel)
            append nextPredictedStep to
$nextPredictedStep_{list}$
        append $nextPredictedStep_{list}$ to onlineData
        append $nextPredictedStep_{list}$ to
$predictedOutput_{list}$
```

output type of the model. Some ML frameworks provide models that produce one single output, such as the RF implemented by MLlib. In this case, because of the type of strategy used to predict the steps, one model should be trained for each attribute in order to predict all the attributes to feed the model recursively during the inference phase. After the batch ML workflows were executed, the trained ML models were used to infer online IoT.

2) Online ML workflows

Four online ML workflows were created to apply the previously built models, using the batch ML workflows, with online IoT data. Results were generated for each model after executing 500 predictions.

Online Data Selection. The online data used in the online ML workflows were chosen from the same sensors that provided the data to train the models with the batch ML workflows.

Each online ML workflow was configured to deploy four models, which were the same models trained on each batch ML workflow. The same preprocessing tasks used in each trained ML model were automatically applied to the online data. Algorithm 2 shows the steps performed during the inference of new data using the trained models. Inference is performed using a recursive strategy. In this strategy, one or more models are trained to perform a one-step-ahead forecast. When the first prediction is rendered, the results are input into the models to predict the subsequent time step. This action is repeated until the desired number of steps has been predicted. The main concern of this strategy is whether it can produce a sequence of successful predictions to prevent errors from propagating.

3) RESULTS AND DISCUSSION

The prediction performance of the ML models was evaluated using the root mean squared error (RMSE), mean absolute error (MAE), and the mean square error (MSE) metrics defined in Eqs. 4, 5 and 6, where y_k and $\hat{y_k}$ are the actual and predicted values respectively. The MAE is an average of the absolute prediction errors, the MSE is the average of the squares of the prediction errors, and the RMSE is the square root of the MSE. These are three of the most common metrics used to measure accuracy for continuous variables, and they are negatively oriented scores, meaning that lower values are better. These metrics are also scale-dependent, which provides a straightforward way to quantify the prediction error:

$$RMSE = \sqrt{\frac{\sum_{k=1}^{n}(y_k - \hat{y_k})^2}{n}} \tag{4}$$

$$MAE = \frac{\sum_{k=1}^{n}|y_k - \hat{y_k}|}{n} \tag{5}$$

$$MSE = \frac{\sum_{k=1}^{n}(y_k - \hat{y_k})^2}{n} \tag{6}$$

Table 7 shows the results for the prediction accuracy of the ML models deployed on the online ML workflows. An important observation is that the results are normalized between 0-1 corresponding to 0-150km/h and 0-20Kw/h for the traffic control and energy datasets. Finally, it is also important to note that this paper does not focus on finding the best ML model for each one of these scenarios; the goal is to demonstrate the functionality and usability of the proposed framework.

Using traffic data, the results showed that the RF model achieved the best performance in predicting the next 30 minutes, whereas the LSTM model gave the best performance for the next 60 minutes. Looking at the energy data, it is clear that both the RF and LSTM models demonstrated the best results for the next 60 minutes. Models trained with the LSTM algorithm achieved lower RMSE, MAE, and MSE than models trained with the RF algorithm on traffic data. Overall, the models trained with the RF algorithm achieved lower RMSE, MAE, and MSE than models trained with the LSTM algorithm on traffic data. The plots depicted in Figs. 8, 9, 10, and 11 show that the RF models achieved more success in capturing the evolving energy use and traffic flow conditions when they fluctuated widely, because of the intricate patterns present in these two types of data.

Notably, with the energy data, the LSTM models were not successful in following the real measured data. The results obtained with the LSTM models could be enhanced using preprocessing techniques that were not applied in this evaluation. For example, both types of IoT data presented strong fluctuations, which can impact the performance of ML models. The application of preprocessing tasks such as removal of noise and outliers can help to smooth the data and improve their quality. Nevertheless, implementing this type of technique can be challenging in online IoT data because of several factors such as the number of online samples available

TABLE 7. Prediction accuracy of ML models rendering predictions using online IoT data.

Minutes Ahead	Traffic - Km/h						Energy - Kw/h					
	RMSE		MAE		MSE		RMSE		MAE		MSE	
	RF	LSTM	RF	LSTM	RF	LSTM	RF	LSTM	RF	LSTM	RF	LSTM
15	0.054	0.066	0.091	0.098	0.030	0.017	0.036	0.087	0.074	0.106	0.017	0.023
30	**0.046**	0.064	**0.086**	0.091	**0.027**	0.015	0.040	0.088	0.035	0.057	0.004	0.007
45	0.061	0.064	0.121	0.084	0.052	0.013	0.040	**0.081**	0.049	**0.048**	0.005	**0.006**
60	0.064	**0.063**	0.094	**0.076**	0.030	**0.011**	**0.0327**	0.086	**0.030**	0.050	**0.004**	0.005

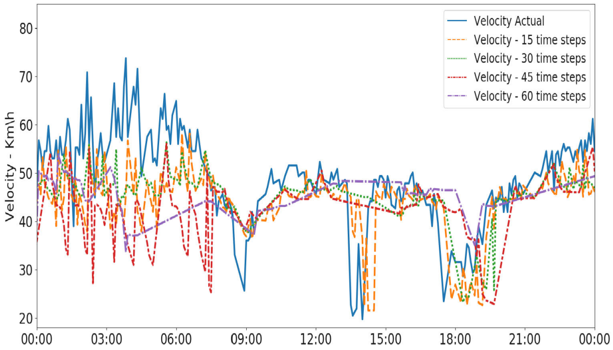


FIGURE 8. Traffic data - results of RF models.

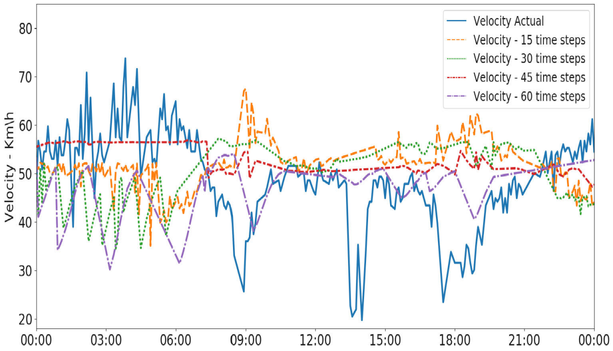


FIGURE 9. Traffic data - results of LSTM models.

and time and processing constraints. Tuning and retraining the ML models can also help increase overall prediction quality. In *ML4IoT*, the *Workflow Scheduler* can be used to schedule ML model retraining at fixed intervals, which can augment the accuracy of predictions rendered by the proposed framework. When *Workflow Scheduler* re-executes batch ML workflows, the trained ML models that are deployed in the online ML workflows are automatically updated to the revised model.

The results achieved in this evaluation demonstrated that *ML4IoT* managed to provide orchestration services to automate the execution of ML workflows to train and infer IoT data. By providing reusable and standardized ML workflows, *ML4IoT* supported the development of end-to-end ML applications in two IoT use cases. The automated execution of ML workflows involved the orchestration of several tasks executed on top of Big Data tools (Kafka, Hadoop, MongoDB, Spark) using different ML frameworks (MLlib, Tensorflow, and Keras). The results obtained for online prediction of energy and traffic data showed that the framework is a feasible solution for orchestrating ML workflows on IoT data.

C. ELASTICITY EVALUATION

The goal of this evaluation was to validate whether the framework could dynamically allocate resources to match the demands of orchestrating multiple ML workflows in parallel. The experiment evaluated execution time, container allocation, and memory and CPU utilization during the execution of batch ML workflows in four scenarios with different workloads. In each workload, increasing numbers of batch ML workflows were executed in parallel. The batch ML workflows and workloads used in this experiment are described below.

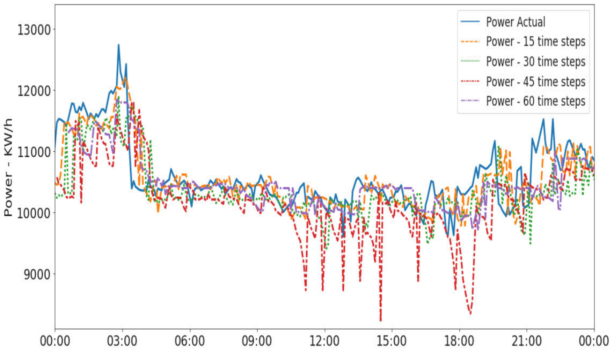


FIGURE 10. Energy data - results of RF models.

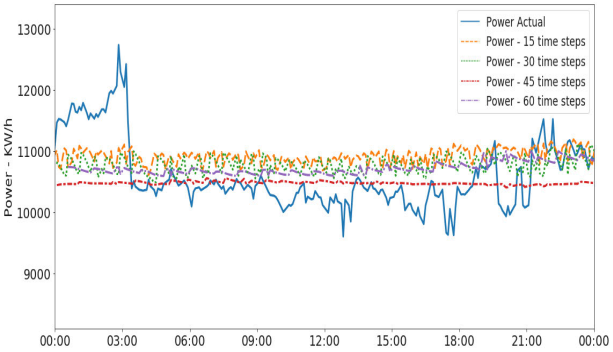


FIGURE 11. Energy data - results of LSTM models.

- **Batch workflows**. Each workflow was composed of the two datasets described in Table 6. Five preprocessing tasks were also applied to the energy dataset and four to the traffic dataset. Moreover, each batch ML workflow was configured to build an RF and an LSTM model for each dataset.

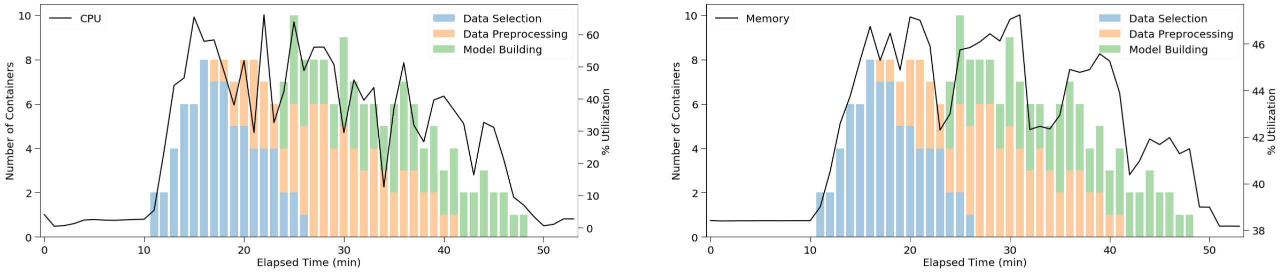


FIGURE 12. Workload 1 - 4 batch ML workflows training 16 models in parallel.

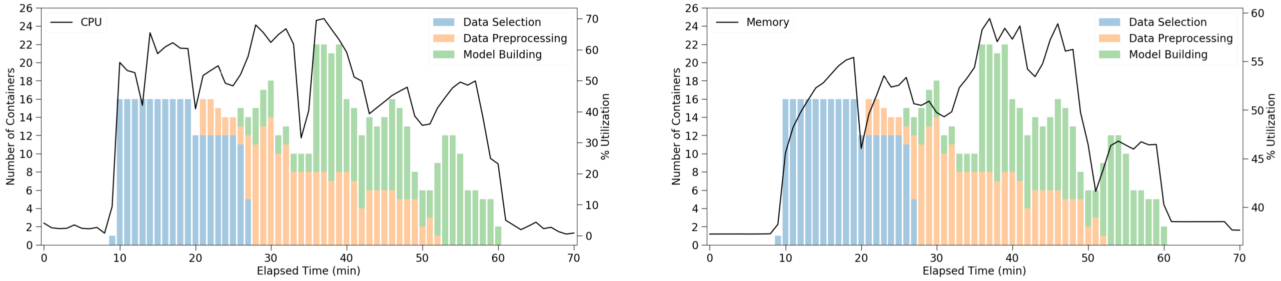


FIGURE 13. Workload 2 - 8 batch ML workflows training 32 models in parallel.

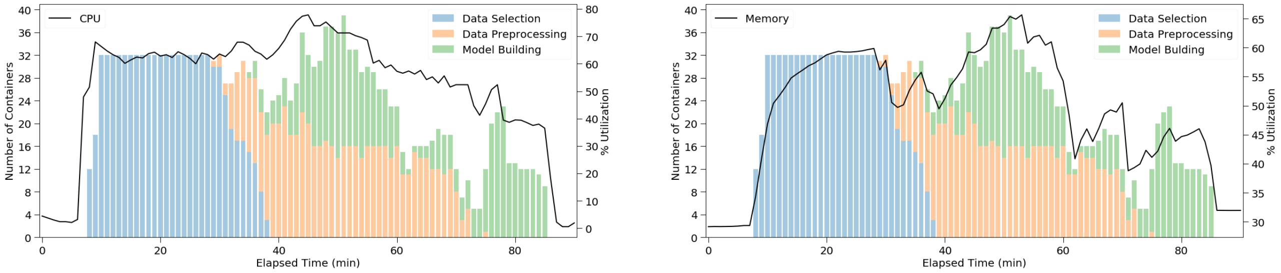


FIGURE 14. Workload 3 - 16 batch ML workflows training 64 models in parallel.

- **Workloads**. Table 8 describes the four workloads used in this experiment. In each workload, the numbers of workflows increased from 4 to 32. In each workflow, two datasets were created, nine preprocessing tasks were executed (five on the energy dataset and four on the traffic dataset), and two models were trained for each dataset. For this reason, from workloads 1 to 4, the number of datasets varied from 8 to 64, and the number of preprocessing tasks varied from 36 to 576. Moreover, the number of ML models being trained in parallel varied from 16 to 128 from workload 1 to workload 4.

TABLE 8. Description of workloads used in the elasticity evaluation.

Workload No	Workflows	Datasets	Preprocessing	Models
1	4	8	36	16
2	8	16	72	32
3	16	32	288	64
4	32	64	576	128

1) RESULTS AND DISCUSSION

Figures 12, 13, 14, and 15 show the results of each workload execution. The maximum numbers of containers running in parallel in the workloads 1, 2, 3, and 4 were 12, 22, 38, and 96, respectively. The results concerning CPU and memory utilization show that these resources were provisioned according to the numbers of containers running in parallel. The peaks of CPU and memory utilization happened when the workloads reached the maximum number of containers running concurrently. The graphs also show that at the end of each workload, when the containers were destroyed, the CPU and memory utilization returned to their original levels, releasing server resources. Although workloads 2, 3, and 4 presented size ratios of 1:2, 1:4, and 1:8 with workload 1, the execution time ratios were 1:1.21, 1:1.76, and 1:3.38 respectively. The execution time results increased slower than linearly, showing that allocating containers to execute ML workflows dynamically is a valid strategy to

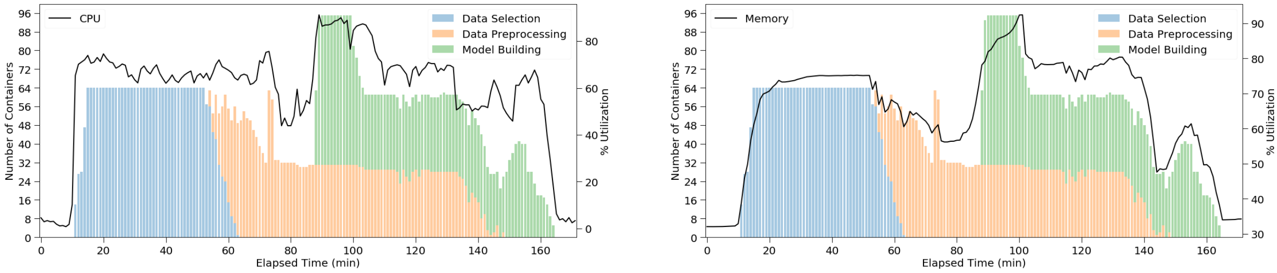


FIGURE 15. Workload 4 - 32 batch ML workflows training 128 models in parallel.

TABLE 9. Description of workloads used in the performance evaluation.

Workload No	Workflows	Datasets	Preprocessing	Models
1	4	4	18	4
2	8	8	36	8
3	16	16	72	16
4	32	32	144	32
5	64	64	288	64

provide an elastic solution that can support execution of multiple ML workflows in parallel.

D. PERFORMANCE EVALUATION

Because the framework was designed to deal with the execution of multiple ML workflows in parallel, one experiment was carried out to validate the performance of the *ML4IoT* framework in this scenario. In the experiment, the performance was evaluated by measuring the latency of rendering online predictions as the workload was increased. Five workloads were assessed in this experiment, with the number of online ML workflows running in parallel varying from 4 to 64. The online ML workflows and workloads used in this experiment are described below.

- **Online machine workflows.** Each workflow was configured to deploy one previously trained ML model to render predictions using online IoT data. Two types of model were presented in the workflows, RF, and LSTM. Each online ML workflow executed a prediction cycle every 10 minutes, which involved the creation of one online dataset, the application of preprocessing tasks (five for energy and four for the traffic dataset), and the prediction of new values using one trained ML model.
- **Workloads.** Table 9 describes the five workloads used in this experiment. The workloads contained an equal number of workflows running LSTM and RF models and rendering predictions on energy and traffic IoT data. For example, in workload 1, the four workflows are defined with different configurations formed by combining the two types of models (LSTM and RF) applied to the two types of IoT data (energy and traffic). The same combination was then applied to the other workloads. From workload 1 to 5, the number of online ML workflows running in parallel increased from 4 to 64.

  At each prediction cycle of an online ML workflow, one online dataset was created, and four or five preprocessing tasks were executed, depending on the dataset (five for energy and four for traffic). One trained ML model also rendered predictions for each online dataset. For this reason, from workload 1 to 5, the number of datasets increased from 4 to 64, and the amount of preprocessing tasks varied from 18 to 288. Moreover, the number of ML models rendering online predictions in parallel went from 4 to 64 from workload 1 to workload 5.

1) RESULTS AND DISCUSSION

The boxplot presented in Figure 16 illustrates the latency time of the online ML workflows. The latency of online ML workflows measures the time spent over the execution of a complete prediction cycle, including the selection of the online data, application of preprocessing tasks, and prediction of new values.

The median latency time of the online ML workflows started at 97.56 seconds when 8 online ML workflows were running in parallel and increased to a maximum of 185.30 seconds when 64 online ML workflows were being executed at the same time. According to the results, the median latency time increased by 89.93% from 4 to 64 workflows, although the number of workflows in parallel increased by 400%. Figure 16 also shows that when the number of online ML workflows running in parallel was 64, the long tail issue started to appear in the latency distributions. The long tail issue is the term used to identify latency measures that refer to the higher percentiles in comparison to the average latency time. For example, in Figures 16 and 17, the scatter points represent the latency measures in the 99$^{\text{th}}$ percentile.

Figure 17 depicts a boxplot of the latency time for model inference only, which is a step performed during execution of online ML workflows, in which trained ML models render predictions taking as input previously processed datasets. One observed trend was that the higher the number of online ML workflows running in parallel, the longer the model inference step takes, and the higher is the number of results with values that are well beyond the average, as shown by the long tail issue in the graph. However, the results demonstrated that increasing the number of online ML workflows

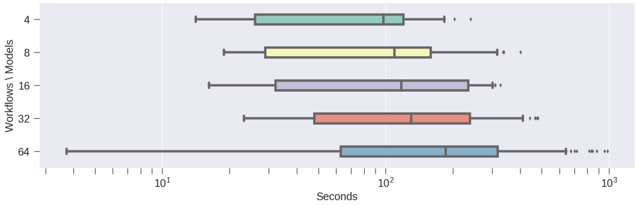


FIGURE 16. Latency of online ML workflows.

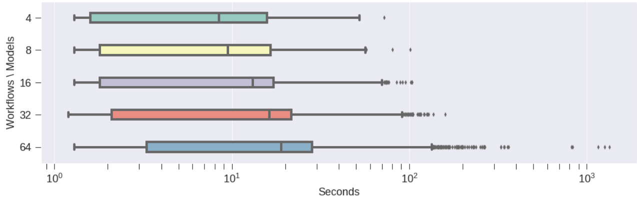


FIGURE 17. Latency of the model inference step.

has little effect on model inference latency. For example, when 64 online ML workflows were running in parallel, only 1.01% of the model inference latency measures were in this worst-case scenario (99th percentile).

Overall, results demonstrated that *ML4IoT* performance was not affected by an increase in the number of online ML workflows rendering predictions simultaneously. This experiment also showed that the framework can manage the execution of multiple online ML workflows in parallel and that it can be used to deploy trained ML models to render predictions using online IoT data.

VI. CONCLUSION AND FUTURE WORK

This study has proposed the Machine Learning Framework for IoT data (*ML4IoT*) to address the challenges involved in integrating Big Data enabling tools and ML frameworks to provide a unified platform for executing end-to-end ML workflows on IoT data. Its main goal was to provide orchestration services for training and inference of ML models on IoT data, which enables automated execution of ML workflows on top of various big data tools and ML frameworks.

To demonstrate the applicability of the *ML4IoT* framework, a prototype system was built to perform 3 experiments using two real-world IoT datasets. Experimental results have shown that *ML4IoT* can simplify the definition and automate the execution of ML workflows running on top of heterogeneous Big Data tools and ML frameworks. The results also demonstrated that *ML4IoT* can manage the orchestration of ML workflows by providing scalability and elasticity to enable execution of multiple ML workflows in parallel.

The use of the framework can potentially improve the accuracy of ML algorithms for the following reasons: a) it can be used for the execution of several models in parallel, which in turn makes it possible to train different models and then choose the ones that can provide better results; b) parallel training also provides efficient optimization of hyper parameters, hence significantly improving results; c) the framework was built to handle IoT data, and therefore the more data are used for training, the better the chances for the models to present improved results; and d) the framework enables re-training of models with more current data, which can improve results, especially when the data distribution has changed over time.

This work has successfully used two ML frameworks, however, the number of potential algorithms and available tools is extensive. Therefore, it is challenging to ensure that the *ML4IoT* can deal with common abstraction layers for all of them. Therefore, one possible future research direction is to perform a deeper analysis and evaluation of the limitations of the proposed framework. A future study can focus on extending the proposed framework to automate selection and tuning of ML models, which is a method known as Automatic ML (AutoML) [41]. Another interesting research extension would be to explore the use of this framework as part of cloud computing services. The deployment of ML applications needs to deal with conflicting priorities such as speed, uptime, and costs. In this way, *ML4IoT* could be implemented in cloud services as an *ML-as-a-Service platform*, which could bring benefits such as preventing downtime, optimizing data center costs, and reducing response latency.

REFERENCES

[1] M. Hung. *Leading the IoT*. Accessed: Jan. 17, 2019. [Online]. Available: http://www.gartner.com/imagesrv/books/iot/iotEbook_digital.pdf

[2] H. Arasteh, V. Hosseinnezhad, V. Loia, A. Tommasetti, O. Troisi, M. Shafie-Khah, and P. Siano, "Iot-based smart cities: A survey," in *Proc. IEEE 16th Int. Conf. Environ. Elect. Eng. (EEEIC)*, Jun. 2016, pp. 1–6.

[3] B. L. R. Stojkoska and K. V. Trivodaliev, "A review of Internet of Things for smart home: Challenges and solutions," *J. Cleaner Prod.*, vol. 140, no. 3, pp. 1454–1464, Jan. 2017.

[4] L. Li, K. Ota, and M. Dong, "When weather matters: IoT-based electrical load forecasting for smart grid," *IEEE Commun. Mag.*, vol. 55, no. 10, pp. 46–51, Oct. 2017.

[5] F. Chen, P. Deng, J. Wan, D. Zhang, A. V. Vasilakos, and X. Rong, "Data mining for the Internet of Things: Literature review and challenges," *Int. J. Distrib. Sensor Netw.*, vol. 11, no. 8, Aug. 2015, Art. no. 431047.

[6] Y. Sun, H. Song, A. J. Jara, and R. Bie, "Internet of Things and big data analytics for smart and connected communities," *IEEE Access*, vol. 4, pp. 766–773, 2016.

[7] D. Crankshaw and J. Gonzalez, "Prediction-serving systems," *Queue*, vol. 16, no. 1, p. 70, Jan. 2018.

[8] D. E. O'Daniel, "'Big data', the 'Internet of Things' and the 'Internet of signs'," *Intell. Syst. Accounting, Finance Manage.*, vol. 20, no. 1, pp. 53–65, Jan./Mar. 2013.

[9] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. E. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. B Hamrick, J. Grout, and S. Corlay, "Jupyter notebooks-a publishing format for reproducible computational workflows," in *Proc. ELPUB*, May 2016, pp. 87–90.

[10] Y. Cheng, F. C. Liu, S. Jing, W. Xu, and D. H. Chau, "Building big data processing and visualization pipeline through apache zeppelin," in *Proc. Pract. Exper. Adv. Res. Comput.*, Jul. 2018, Art. no. 57.

[11] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 2503–2511.

[12] *Uber Michelangelo*. Accessed: Jan. 17, 2019. [Online]. Available: https://eng.uber.com/michelangelo

[13] *Airbnb Bighead*. Accessed: Jan. 17, 2019. [Online]. Available: https://databricks.com/session/bighead-airbnbs-end-to-end-machine-learning-platform

[14] *Netflix Meson*. Accessed: Jan. 17, 2019. [Online]. Available: https://medium.com/netflix-techblog/meson-workflow-orchestration-for-netflix-recommendations-fc932625c1d9

[15] D. Baylor *et al.*, "TFX: A tensorflow-based production-scale machine learning platform," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2017, pp. 1387–1395.

[16] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, J. Law, K. Lee, J. Lu, P. Noordhuis, M. Smelyanskiy, L. Xiong, and X. Wang, "Applied machine learning at facebook: A datacenter infrastructure perspective," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2018, pp. 620–629.

[17] W. Lin, Z. Wu, L. Lin, A. Wen, and J. Li, "An ensemble random forest algorithm for insurance big data analysis," *IEEE Access*, vol. 5, pp. 16568–16575, 2017.

[18] D. Zhang, L. Qian, B. Mao, C. Huang, B. Huang, and Y. Si, "A data-driven design for fault detection of wind turbines using random forests and XGboost," *IEEE Access*, vol. 6, pp. 21020–21031, 2018.

[19] F. Karim, S. Majumdar, H. Darabi, and S. Chen, "LSTM fully convolutional networks for time series classification," *IEEE Access*, vol. 6, pp. 1662–1669, 2017.

[20] W. Wang, J. Gao, M. Zhang, S. Wang, G. Chen, T. K. Ng, B. C. Ooi, J. Shao, and M. Reyad, "Rafiki: Machine learning as an analytics service system," *Proc. VLDB Endowment*, vol. 12, no. 2, pp. 128–140, Oct. 2018.

[21] Y. Lee, A. Scolari, B.-G. Chun, M. D. Santambrogio, M. Weimer, and M. Interlandi, "PRETZEL: Opening the black box of machine learning prediction serving systems," in *Proc. 13th USENIX Symp. Operating Syst. Des. Implement. (OSDI)*, 2018, pp. 611–626.

[22] *ML.NET*. Accessed: Jan. 17, 2019. [Online]. Available: https://dotnet.microsoft.com/apps/machinelearning-ai/ml-dotnet

[23] D. C. Spell, X.-H. T. Zeng, J. Y. Chung, B. Nooraei, R. T. Shomer, L.-Y. Wang, J. C. Gibson, and D. Kirsche, "Flux: Groupon's automated, scalable, extensible machine learning platform," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2017, pp. 1554–1559.

[24] C. Cecchinel, F. Fouquet, S. Mosser, and P. Collet, "Leveraging live machine learning and deep sleep to support a self-adaptive efficient configuration of battery powered sensors," *Future Gener. Comput. Syst.*, vol. 92, pp. 225–240, Mar. 2019.

[25] Y. Yang, F. Nan, P. Yang, Q. Meng, Y. Xie, D. Zhang, and K. Muhammad, "Gan-based semi-supervised learning approach for clinical decision support in health-IoT platform," *IEEE Access*, vol. 7, pp. 8048–8057, 2019.

[26] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.

[27] I. Kotenko, I. Saenko, and A. Branitskiy, "Framework for mobile Internet of Things security monitoring based on big data processing and machine learning," *IEEE Access*, vol. 6, pp. 72714–72723, 2018.

[28] S. K. Lakshmanaprabu, K. Shankar, A. Khanna, D. Gupta, J. J. P. C. Rodrigues, P. R. Pinheiro, and V. H. C. De Albuquerque, "Effective features to classify big data using social Internet of Things," *IEEE Access*, vol. 6, pp. 24196–24204, 2018.

[29] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[30] J. P. Jones and L. A. Palmer, "An evaluation of the two-dimensional Gabor filter model of simple receptive fields in cat striate cortex," *J. Neurophysiol.*, vol. 58, no. 6, pp. 1233–1258, Dec. 1987.

[31] D. Preuveneers, Y. Berbers, and W. Joosen, "Samurai: A batch and streaming context architecture for large-scale intelligent applications and environments," *J. Ambient Intell. Smart Environ.*, vol. 8, no. 1, pp. 63–78, 2016.

[32] N. Marz and J. Warren, *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*. New York, NY, USA: Manning, 2015.

[33] P. Ta-Shma, A. Akbar, G. Gerson-Golan, G. Hadash, F. Carrez, and A. Moessner, "An ingestion and analytics architecture for IoT applied to smart city use cases," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 765–774, Apr. 2018.

[34] R. B. Almeida, V. R. Junes, R. da Silva Machado, D. Y. L. da Rosa, L. M. Donato, A. C. Yamin, and A. M. Pernas, "A distributed event-driven architectural model based on situational awareness applied on Internet of Things," *Inf. Softw. Technol.*, vol. 111, pp. 144–158, Jul. 20190.

[35] A. Yamin, I. Augustin, L. C. D. Silva, R. A. Real, and C. F. Geyer, "EXEHDA: Adaptive middleware for building a pervasive grid environment," in *Proc. Conf. Self-Org. Autonomic Inform.*, 2005, pp. 203–219.

[36] *Apache Spark*, Accessed: Jan. 17, 2019. [Online]. Available: https://spark.apache.org

[37] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A distributed messaging system for log processing," *Proc. NetDB*, Jun. 2011, pp. 1–7.

[38] *Apache Hadoop*. Accessed: Jan. 17, 2019. [Online]. Available: https://hadoop.apache.org/

[39] Y.-S. Kang, I.-H. Park, J. Rhee, and Y.-H. Lee, "Mongodb-based repository design for IoT-generated rfid/sensor big data," *IEEE Sensors J.*, vol. 16, no. 2, pp. 485–497, Jan. 2016.

[40] *Madrid Council Traffic Data*. Accessed: Jan. 17, 2019. [Online]. Available: http://informo.munimadrid.es/informo/tmadrid/pm.xml

[41] C. Wong, N. Houlsby, Y. Lu, and A. Gesmundo, "Transfer learning with neural AutoML," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 8366–8375.

JOSÉ M. ALVES received the B.Sc. degree in information systems from USP, Brazil, in 2010. He is currently pursuing the M.E.Sc. degree in software engineering with Western University, Canada. From 2010 to 2017, he was involved in numerous software development and data analytics projects for telecommunications companies in Brazil. His current research interests include big data, machine learning, the IoT, and cloud computing.

LEONARDO M. HONÓRIO received the B.Sc. degree from the Federal University of Juiz de Fora (UFJF), in 1993, and the M.Sc. and Ph.D. degrees from EFEI, Brazil, in 1999 and 2002, respectively, all in electrical engineering. He was a Visiting Researcher with Porto and Irvine California University, in 2006 and 2012, respectively. He is currently an Associate Professor with UFJF. His current research interests include evolutionary algorithms, probabilistic methods, optimal power flow, robotics, autonomous vehicles, fuzzy logic, pattern recognition, and optimization.

MIRIAM A. M. CAPRETZ received the B.Sc. and M.E.Sc. degrees from UNICAMP, Brazil, and the Ph.D. degree from the University of Durham, U.K. She was with the University of Aizu, Japan. She is currently a Professor with the Department of Electrical and Computer Engineering, Western University, Canada. She has been involving in the software engineering area for more than 35 years. She has been involved with the organization of workshops and symposia and has been serving on program committees in international conferences. Her current research interests include cloud computing, big data, machine learning, service–oriented architecture, privacy, and security.