4-24-2019 10:30 AM

Orchestration of machine learning workflows on Internet of Things data

Jose Miguel Alves
The University of Western Ontario

Supervisor
Dr. Miriam A. M. Capretz
The University of Western Ontario

Graduate Program in Electrical and Computer Engineering
A thesis submitted in partial fulfillment of the requirements for the degree in Master of Engineering Science

Abstract

Applications empowered by machine learning (ML) and the Internet of Things (IoT) are changing the way people live and impacting a broad range of industries. However, creating and automating ML workflows at scale using real-world IoT data often leads to complex systems integration and production issues. Examples of challenges faced during the development of these ML applications include glue code, hidden dependencies, and data pipeline jungles.

This research proposes the Machine Learning Framework for IoT data (*ML4IoT*), which is designed to orchestrate ML workflows to perform training and enable inference by ML models on IoT data. In the proposed framework, containerized microservices are used to automate the execution of tasks specified in ML workflows, which are defined through REST APIs.

To address the problem of integrating big data tools and machine learning into a unified platform, the proposed framework enables the definition and execution of end-to-end ML workflows on large volumes of IoT data. In addition, to address the challenges of running multiple ML workflows in parallel, the *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. Finally, to address the common production issues faced during the development of ML applications, the proposed framework used microservices architecture to bring flexibility, reusability, and extensibility to the framework.

Through the experiments, we demonstrated the feasibility of the (*ML4IoT*), which managed to train and deploy predictive ML models in two types of IoT data. The obtained results suggested that the proposed framework can manage real-world IoT data, by providing elasticity to execute 32 ML workflows in parallel, which were used to train 128 ML models simultaneously. Also, results demonstrated that in the *ML4IoT*, the performance of rendering online predictions is not affected when 64 ML models are deployed concurrently to infer new information using online IoT data.

Keywords: IoT, Machine Learning, Big Data, Machine Learning Workflow Automation, Orchestration, Time-series Forecasting, Container-based Virtualization, Microservices

Acknowledgements

First, I would like to thank my supervisor, Dr. Miriam Capretz. Dr. Capretz believed in my potential and gave me the opportunity to study under her guidance. I will always be thankful for the opportunities she gave me and for entrusting me with the freedom to pursue my own research interests.

This thesis would also not have been possible without the support of my wife, Carla Alves, who left everything behind to join me in this challenge. Thank you for being with me and for guiding me during these years. I love you.

Infinite thanks to my mother, Vilma Alves. Thank you for your hard work to support our family. Thanks to my sisters, my friends, my niece Isabella, and my nephew Antenor, who have provided me with numerous moments of laughter and joy.

I would also like to extend my heartfelt appreciation to my fantastic research team for helping me throughout my research: Wander Queiroz, Norman Tasfi, Willamos Aguiar, Alexandra LHeureux, and Santiago Gomez. Special thanks to Dr. Hany ElYamany and Dr. Mahmoud ElGayyar for lending their expertise to my research and for all the time they spent helping me. Thank you.

Lastly, I would also like to thank members of the industry I worked in the past, especially Icaro Technologies. Thank you for giving me a chance to work with an outstanding team who inspired me to pursue my dreams and accept new challenges.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Motivation

Gartner [5], predicts that the Internet of Things (IoT) will reach 26 billion internet-connected devices by 2020, impacting a wide range of industries. The Internet of Things (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 [6], smart homes [7], and smart grids [8].

Machine Learning is an approach used to convert data into applications by obtaining models that generalize to new data [9]. 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 [9]. In IoT scenarios, the inference phase involves the deployment of trained ML models to infer information using online IoT data that were not used previously to train the models.

IoT devices produce massive amounts of data, at high speed, and from a vast variety of sources [10]. The high volume, high velocity, and high variety of the data require the use of

several Big Data-enabling tools to ingest, store, and preprocess the IoT data before they are used for training of and inference by ML models. In addition, depending on the goals of the ML application being developed, different frameworks and libraries must be used. For example, a company may need to use three different ML frameworks if they want to train deep learning, tree-based, and linear models. Big data enabling tools such as messaging brokers [11–13], distributed file systems [14, 15], NoSQL databases [16–20], and data processing engines [21–25] have been used to support the ingestion, storage, and processing of IoT data. Libraries and frameworks such as TensorFlow [26], MLlib [27], Pytorch [28], Deeplearning4j [29], and Keras [30] have also been used to apply machine learning to IoT data.

However, integrating Big Data enabling tools with ML software to create end-to-end workflows to train and infer IoT data is a time-consuming task that requires specialized skills, and code that is repetitive and complex to manage. Given the heterogeneity of tools, many are incompatible with each other. The advances in software for IoT data have not yet converged on standard formats and interfaces across Big Data tools and ML frameworks. In this way, creating ML applications using large volumes of historical and online IoT data can be prohibitively complex and expensive.

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 [31, 32] to develop and experiment with new models. Finally, software engineers work to deploy trained models in production systems. The hand-off between these steps leads to bottlenecks and production issues such as *hard-to-maintain glue code*, *hidden dependencies*, *feedback loops*, and *pipeline jungles*, which are signs of many of the machine learning anti-patterns described by Sculley *et al.* [33]. 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 these

problems internally with their own platforms. For example, Uber has built its ML orchestration platform called Michelangelo [34], Airbnb has Bighead [35], Netflix has developed the Meson platform [36], Google has introduced TensorFlow Extended (TFX) [37], and Facebook has implemented its data pipeline platform for generating and predicting models [38]. These are all in-house proprietary platforms to make sure that their time and money are not wasted on developing repetitive ML workflows and management tools. Nevertheless, not every company has the capabilities to invest in ML orchestration, and many large corporations still fail to see the significant impact it will have on their business. Furthermore, Cloud providers like Amazon (AWS) [39], Google [40], and Microsoft (Azure) [41] have built services on their cloud platforms that cover IoT data ingestion and preprocessing as well as, training and deployment of ML models, but none provides services to do all of the above using integrated and orchestrated solutions.

The primary goal of this research is to overcome the challenges of integrating Big Data enabling tools and machine learning software to provide a unified platform where end-to-end ML workflows can be orchestrated for both training of and inference by ML models on IoT data.

1.2 Contribution

The main contribution of this thesis is the Machine Learning Framework for IoT data (*ML4IoT*), which is designed to orchestrate machine learning workflows to perform training and enable inference by ML models on IoT data.

To address the problem of integrating big data tools and machine learning into a unified platform, the proposed framework enables the definition and execution of end-to-end ML workflows using REST APIs. The definition of ML workflows using high-level APIs abstracts from users and developers the complexities of creating complex and repetitive code to integrate the numerous tools required to apply ML to IoT data. Two types of ML workflows can be

created in this framework: batch and online ML workflows. These workflows are composed of a set of configurations that define the sequential tasks and parameters required to train and deploy ML models to infer online IoT data.

Moreover, to address the challenges of running multiple ML workflows in parallel, the *ML4IoT* was designed to use container-based components that provide a convenient mechanism for horizontally and independently scaling the *ML4IoT* to execute multiple ML workflows in parallel. The automated execution of the ML workflows is orchestrated by backend services provided by the *ML4IoT*, which uses containerized microservices to execute the tasks defined in the workflows. Packaging software code into portable containerized microservices enables ML4IoT to run ML workflows with different software and hardware specifications in parallel. For example, ML workflows can use models provided by different frameworks (e.g., TensorFlow [26], MLlib [27]) running on a mix of CPUs and GPUs. 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.

Finally, to address the common production issues faced during the development of ML applications, the proposed framework used microservices architecture to bring flexibility, reusability, and extensibility to the framework. For example, machine learning is evolving at a rapid pace, and ML libraries and frameworks can become outdated quickly. In the *ML4IoT*, these frameworks and libraries used to build ML models can be added, replaced, or updated without affecting the other components of the framework. Also, splitting the design of ML workflows into small and specialized microservices facilitates the reuse of these software components. Lastly, it contributes to the extensibility of the ML4IoT, because when the design of ML workflows is divided into well-defined components, it creates natural points of extension for new functionalities.

ML4IoT was evaluated in three experiments using two types of real-world IoT data from the energy and traffic domains. The first experiment investigated the feasibility of the framework by assessing its ability to orchestrate ML workflows using different ML libraries and IoT data. The

results achieved in this evaluation demonstrated that *ML4IoT* managed to automate the execution of ML workflows, which were used to perform short-term prediction of energy consumption and traffic flow using two ML models, Random Forest (RF) and LSTM.

The framework elasticity was evaluated in the second experiment by studying its ability to scale to support the execution of multiple ML workflows in parallel. The obtained results suggested that the proposed framework can manage real-world IoT data, by providing elasticity to execute 32 ML workflows in parallel, which were used to train 128 ML models simultaneously.

Finally, the third experiment investigated the performance of the framework by analyzing the latency of the execution of ML workflows deployed to render prediction using online IoT data. Also, results demonstrated that the performance of rendering online predictions is not affected when 64 models are deployed in parallel to infer new information using online IoT data.

1.3 Organization of the Thesis

The remainder of this thesis is organized as follows:

- Chapter 2 provides background information that is useful in understanding this work as well as a literature review of related studies. This chapter first provides an introduction to the technical terms and concepts that are used throughout this thesis. Second, this chapter presents a review of current research that addresses the application of machine learning to IoT data. Finally, it contrasts the contribution of this thesis with existing practice.

- Chapter 3 presents the components of the *ML4IoT*. Besides providing an overview of the function of each component, this chapter also describes how each part interacts with the others.

- Chapter 4 presents an evaluation of the *ML4IoT*. First, it introduces an implementation of a prototype system based on the proposed framework, followed by descriptions of the

experimental set-up and the IoT data used in the experiments. Finally, the experiments and a discussion of the preliminary results are presented.

- Chapter 5 provides the conclusions of this thesis along with a discussion of future work possibilities.

Chapter 2

Background and Literature Review

This chapter serves a dual purpose: first, it introduces various terms related to the topics discussed in this thesis, and second, it provides an overview of existing research on topics related to the application of machine learning to IoT data.

2.1 Background

2.1.1 Internet of Things

The Internet of Things (IoT) is a network of internet-connected devices, including sensors, machines, vehicles, and other elements of the real world. Gartner [5] predicts that the IoT will reach 26 billion units by 2020, impacting a wide range of industries. The availability of IoT data produced by this giant network of devices has created new opportunities for innovative applications, for example, smart cities [6], smart agriculture [42], and smart grids [8].

IoT data

The actual value of the IoT for companies can be fully realized when valuable business information is extracted from the data produced by IoT devices. However, the high volume, high speed, and high variety of IoT data sources pose challenges to retrieving relevant information from

these data. Dealing with IoT data usually involves the steps of ingesting [43], storing [44], and processing historical and online data [45]. These steps are discussed below.

- **Ingesting IoT data** is a usual step required to manage IoT data [25]. It involves ingesting the data into a platform where they are processed and analyzed. For example, in a smart city traffic scenario where thousands of sensors are deployed to collect measures of the traffic flow,congestion on a highway is detected by correlating information produced by different sensors spread over the city. Therefore, the data generated by IoT devices need to be ingested into a centralized platform where they can be analyzed and correlated. Tools such as message brokers [11–13], data ingestion tools [46], extract-transform-load solutions [47], and machine-to-machine communication frameworks [48] have been used to create interfaces where IoT data can be ingested for later processing.

- **Storing IoT data** is challenging because of the large volume of data produced by IoT devices. However, a broad range of ML algorithms can use past data to build models that are applied later to both historical and online IoT data. Even in scenarios where the data distribution changes with time, past data are useful to help create evolving ML models that can adapt to drift in data patterns. Distributed File Systems [14, 15] and NoSQL databases [16–20] are examples of technologies that have been used for historical storage of IoT data.

- **Processing IoT data** when they are still in motion is critical because the detection of patterns and anomalies in real time can have a considerable impact on event outcomes. In addition, IoT sensors are a valuable source of online data. For example, during building operation, a significant amount of energy may be wasted due to equipment and human faults, but ML models can be used to detect abnormal consumption patterns using online data collected from IoT sensors. In this case, energy waste is reduced if appropriate energy-saving procedures are adopted based on information provided by the online data. Tools such as stream processing systems [21–25] and ML libraries [26–30] have been

used to process historical and online IoT data.

2.1.2 Machine Learning

Machine Learning (ML) is a technique that uses algorithms to learn complex patterns automatically from the data [49]. Because of the enormous potential of automating tasks that do not require human intervention, machine learning has become a ubiquitous technology. The development of ML applications is divided into two phases: training and inference. The training phase involves optimizing the parameters of a function, called the loss or cost function, which generates a trained ML model. The inference phase of machine learning is also referred to in the literature as prediction serving, models score, or model prediction [9]. This phase involves the deployment of trained ML models to infer information using new data. The quality of data impacts the results in both the training and inference phases of machine learning. For this reason, data preprocessing tasks are often used to improve the overall quality of the data used in ML applications.

First, this section introduces the data preprocessing techniques used to prepare data before training of and inference by ML models. Then, an overview is provided of how ML applications are classified according to their training type. Next, two ML algorithms used in the evaluation section of this research are discussed. Finally, the concept of ML workflow is discussed.

Data Preprocessing

Data preprocessing techniques are applied to outliers, noisy data, and missing data because these can affect the results of the ML model [50]. Preprocessing the data helps to improve the data quality, for example, by removing null and duplicate values of the data. Moreover, preprocessing a dataset prepares the data for the application of ML algorithms. For instance, new features can be created in the datasets, helping the algorithms to achieve better and more accurate results. Data preprocessing tasks can be grouped into four categories: data integration, data reduction, data cleaning, and data transformation [50].

- **Data Integration** seeks to merge the data from disparate data sources. The application of ML generally involves the use of datasets that are stored in different places. In addition, most of the data are produced by heterogeneous data sources and have various data structures. In these cases, data integration methods help to assure the data quality when datasets are merged.

- **Data Reduction** is used to obtain a smaller representation of the original dataset. In this way, it contributes to increasing the efficiency of ML algorithms because fewer resources are necessary to process the smaller datasets. The datasets produced after the application of data reduction methods yield results similar to the original dataset.

- **Data Cleaning** replaces or corrects incomplete, noisy, and inconsistent data. Incomplete data contain missing attributes or just aggregate data; noisy data present errors or values that are not expected; inconsistent data include corrupt values. Data cleaning methods are focussed on solving these three problems by filling in incomplete data, smoothing out or removing noisy data, and fixing data inconsistencies.

- **Data Transformation** is used to transform data into other formats that can improve the performance of ML models. Methods for data transformation include:

  – **Normalization** rescales the values of the attributes to lie within a smaller range. Normalization is required by some ML algorithms because attributes with larger values can have more influence than smaller ones during model training. One common method to normalize the data is to rescale the dataset attributes to range within [0 1] using Eq. (2.1), where $\mathbf{X}$ is the dataset and n is the number of dataset samples:

$$\tilde{x_n} = \frac{x_n - min(\mathbf{X})}{max(\mathbf{X}) - min(\mathbf{X})} \tag{2.1}$$

  – **Aggregation** summarizes or aggregates values of attributes. For example, some IoT

sensors periodically sense data and transmit them in short intervals, such as seconds or minutes. Depending on the ML model goal, the data may need to be made to represent longer ranges using data aggregation methods.

- **Discretization** replaces numeric attributes by mapping values to intervals or labels. For example, if an attribute is used to represent the amount of time that an equipment or sensor was faulty, instead of serving the exact numeric value of each fault, this attribute can store discrete values, for example, less than 30 minutes, more than 30 minutes or less than one hour, more than one hour and so on.
- **Feature engineering** is used to rearrange or create new attributes that can help train ML models. For instance, a popular feature engineering technique used in time-series data involves rearranging data by representing each input instance using a sliding data window instead of a single input value.

Machine Learning Types

The algorithms used in machine learning are classified according to how the learning process is conducted. Four common classes are used to group ML algorithms: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning [49].

- **Supervised learning** uses a set of labelled data to train an algorithm. Labelled data imply that each input data point in the training dataset has a defined output, which is called a label value. At the end of the training process, for each input data point, the trained algorithm should come up with an output similar to the label value. Supervised learning is applied to solve two types of problems: classification problems and regression problems. In classification problems, algorithms are used to infer a discrete value by classifying the input data as part of a category or group. Regression problems involve inferring a continuous value from each input data point.
- **Unsupervised learning** creates ML models using datasets that do not contain labels.

Hence, there is no correct answer or output for each input data point. The algorithms try to find useful patterns and structure in the data. However, because there are no data labels, it is challenging to evaluate the accuracy of algorithms trained using unsupervised learning.

- **Semi-supervised learning** trains ML models using datasets that contains both labelled and unlabelled data. In some cases, labelling all the input data points is a prohibitive task because of the volume and velocity of the data, which require the use of semi-supervised learning. In semi-supervised learning, data labels are still needed to validate algorithm accuracy.

- **Reinforcement learning** finds an optimal way to accomplish a particular goal or improve performance on a specific task. If the algorithm takes action that moves toward the goal, it receives a reward; otherwise, it is given a punishment. The overall goal of the algorithm is to predict the best next step to take to earn the biggest final reward. Training algorithms using reinforcement learning is an iterative process in which the more rounds of feedback are used, the better the agents strategy becomes.

Machine Learning Algorithms

The two ML algorithms used in this thesis are described below.

- **Random Forest (RF)**, an algorithm proposed by Breiman [51], is an ensemble algorithm that joins multiple decision trees to decrease the chance of overfitting. The algorithm introduces randomness into the training process so that, in the set of decision trees, each one is a little different. Combining the predictions from each tree reduces the variance of the predictions, improving the performance of test data. The training of an RF involves a technique called *bagging* [52], which uses an ensemble of C trees $\{T_1(X), ..., T_C(X)\}$ to produce C outputs $\{Y_1 = T_1(X), ..., Y_C = T_C(X)\}$, where $X = \{x_1, ..., x_n\}$ is an n-dimensional feature vector and $Y_a, a = 1, ..., C$ is the value predicted by the a^{th} tree. The output value

Y is obtained by averaging the predictions from all the individual regression trees or by taking a majority vote when the algorithm is applied to classification problems, as shown in Figure 2.1.

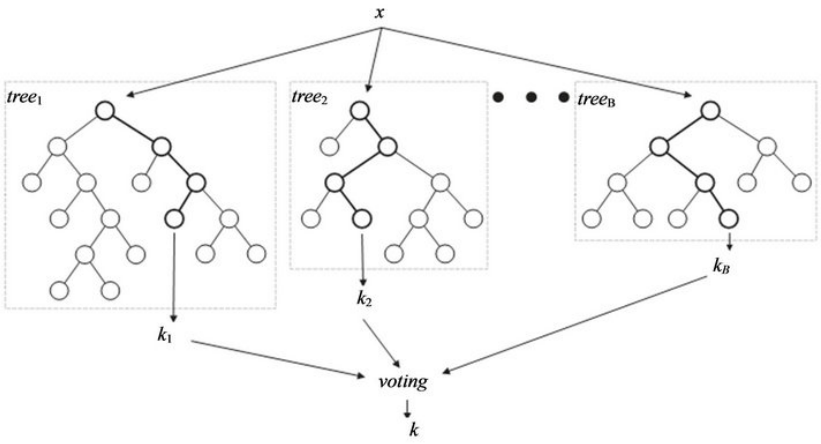

Figure 2.1: Random Forest structure [1].

- **Long Short-Term Memory (LSTM)** is the most popular type of recurrent neural networks, which are algorithms that can perform parallel and sequential computation. Because of its capacity to learn long sequences and deal with the vanishing gradient problem [53], this algorithm is widely used in domain-specific applications such as time-series prediction [54], speech recognition [55], and robot control [56]. As illustrated in Figure 2.2, each neuron of a long short-term memory (LSTM) networks has four components called the *cell*, *input gate*, *output gate*, and *forget gate*. The cell is the memory component, and the gates (input, forget, output) controls how the information flows inside the LSTM neuron. LSTM networks can retain errors across time steps and layers, which enables them to learn even when a large number of time steps are represented in the input data.

Machine Learning Workflow

The terms *machine learning workflow* or *machine learning pipeline* are commonly used in the industry to define a sequence of steps performed during the development of ML

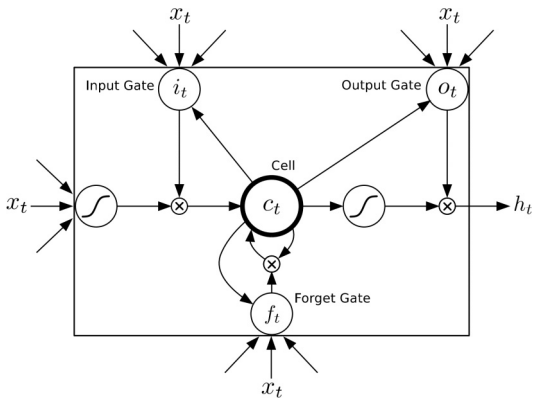

Figure 2.2: Long Short-Term Memory Neuron [2].

applications. Although there is no official definition of these terms, research published by crucial players in machine learning such as Google [37], Facebook [38], and Twitter [57] has used these terms in their publications.

The development of ML applications involves two distinct phases, a training phase, and an inference phase [9], which are discussed below:

- **The training phase** typically begins with collecting and preprocessing a training dataset. The attributes of the dataset are used as criteria for choosing from a wide range of model designs (e.g., linear regression, random forest, LSTM, etc.) and their corresponding training algorithms (e.g., supervised learning, unsupervised learning, semi-supervised learning, reinforcement learning). After a model and training algorithm have been selected, there are often additional hyper-parameters that must be tuned by repeatedly training and evaluating the model.
- **The inference phase** is also referred to in the literature as prediction serving, models score, or model prediction. In this phase, a new dataset is also collected and preprocessed, and the ML models trained previously in the training phase are deployed to take data as input and emit predictions. The inference phase requires integrating ML software with other systems, for example, user interface applications, live databases, and high-volume data streams.

In the development of ML applications, a sequence of steps performed during the training, or the inference, or both phases is called a *machine learning workflow* or a *machine learning pipeline*. Figure 2.3 depicts an example of a generic *machine learning workflow/pipeline*.

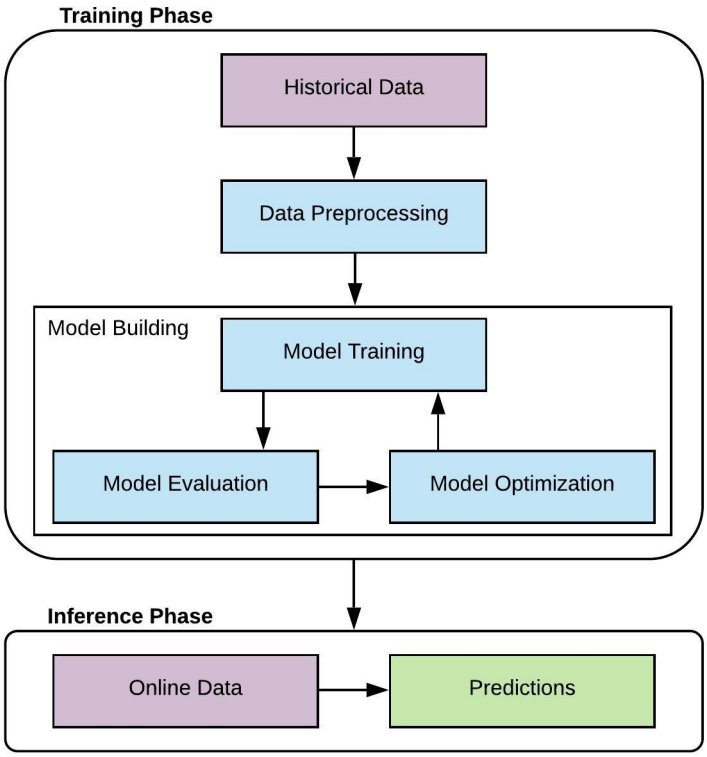

Figure 2.3: Example of a generic machine learning workflow/pipeline.

The steps that compose these *workflows* or *pipelines* depends on several aspects, for example, the use case, data domain, type of machine learning tasks, etc. Although the terms *workflow* and *pipeline* are used interchangeably, this research has adopted the term *machine learning workflow* because, in software engineering, the term *workflow* is closely related to automation [58], which is one of the topics explored in this work. Moreover, the term *machine learning workflow* is broadly defined in the industry, and it also encompasses the term *machine learning pipeline*. In addition, in this research, the terms *batch machine learning workflow* and *online machine learning* are used to define the sequence of steps performed during the training and inference phases of machine

learning applications, respectively.

2.1.3 Time-series Forecasting

Time-series forecasting is a growing field of interest because of the large amount of data produced daily by IoT devices can be classified as time-series data. A time series is a chain of historical measures y_t of an observable variable y at regular periods. For example, an IoT sensor measuring the traffic conditions on a road produces data at specific intervals, which constitute a time series. Time-series forecasting can be performed for both single and multiple periods. Forecasting a single period, which is also known as one-step-ahead, means forecasting the next value of the time-series sequence. Multi-step-ahead forecasting consists of predicting the next H values $[y_{N+1}, ..., y_{N+H}]$ composed of N observations, where $H > 1$ denotes the forecasting horizon. Unlike one-step-ahead forecasting, multistep-ahead forecasting tasks are more difficult [59], because this type of forecasting must deal with additional complexities, such as accumulated errors, reduced accuracy, and increased uncertainty [60].

Figure 2.4 shows the five strategies that have been proposed in the literature [3] to tackle multi-step ahead forecasting tasks: Recursive, Direct, Multi-input Multi-output (MIMO), Recursive and Direct (DirREC), and Direct and Multi-input Multi-output (DirMO).

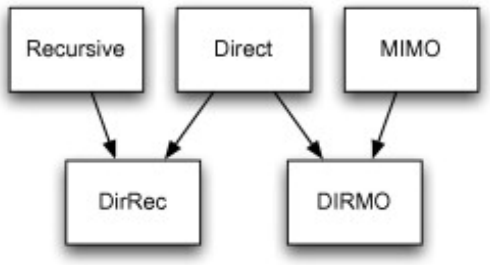

Figure 2.4: Time-series forecasting strategies and their relationship [3].

– **Recursive** starts by predicting a value, which is then used for forecasting the subsequent time steps.

- **Direct** predicts each step independently of the others, using specific models for each step ahead.
- **Multi-input Multi-Output (MIMO)** differs from Recursive and Direct strategies because it returns multiple outputs, where a value is predicted for each attribute of the input data.
- **Recursive and Direct (DirRec)** is a combination of the Direct and Recursive strategies.
- **Direct and Multi-input Multi-output (DIRMO)** is a combination of the Direct and MIMO strategies.

2.1.4 Microservices

A microservices architecture is a software architecture composed of small and independent services that work together [61]. Each service supports a specific function and uses a simple, well-defined interface to communicate with other sets of services. A microservices architecture is the opposite of the traditional monolithic architecture in which, although different components or modules are used, the application is developed and deployed as one. A short discussion of some of the vital differences between microservices and monolithic architecture is presented next.

- **Technology heterogeneity.** Because the services are independent in a microservices architecture, each component can adopt a different technology or framework. This also enables new technologies to be adopted more quickly. With a monolithic architecture, any change in one of the architectures components impacts a large part of the system. On the other hand, in a microservices architecture, replacing a service does not affect the other architectural components, enabling easy integration and adoption of new technologies.
- **Scalability.** In monolithic architectures, the components need to be scaled together. On the other hand, because the microservices architecture is composed of smaller services, it

is easier to scale just those services that need scaling.

- **Ease of deployment.** Usually, changes in software implemented using monolithic architectures require the redeployment of the whole application to release the new software version. In a microservices architecture, a modification can be made to a single service. Then the changed service is deployed independently of the rest of the system.

2.1.5 Container-based Virtualization

Resource virtualization is a technique which uses an intermediary software layer on top of an underlying host server in order to provide abstractions of multiple virtual resources. In general, virtualized resources are called virtual machines (VM). Hypervisor-based virtualization and container-based virtualization are the two most popular types of virtualization used in cloud computing [62]. In hypervisor-based virtualization, a set of virtual machines (VM) is created to share physical hardware resources, but each VM executes distinct operating systems [63]. On the other hand, in container-based virtualization, the virtual instances created to share the physical resources of the host server are called containers. Each container executes as a standalone operating system, but they share a single operating system kernel. Because the containers share the kernel of the host operating system, container-based virtualization leads to lower overhead, and instantiating, relocating, and optimizing hardware resources is far easier [64].

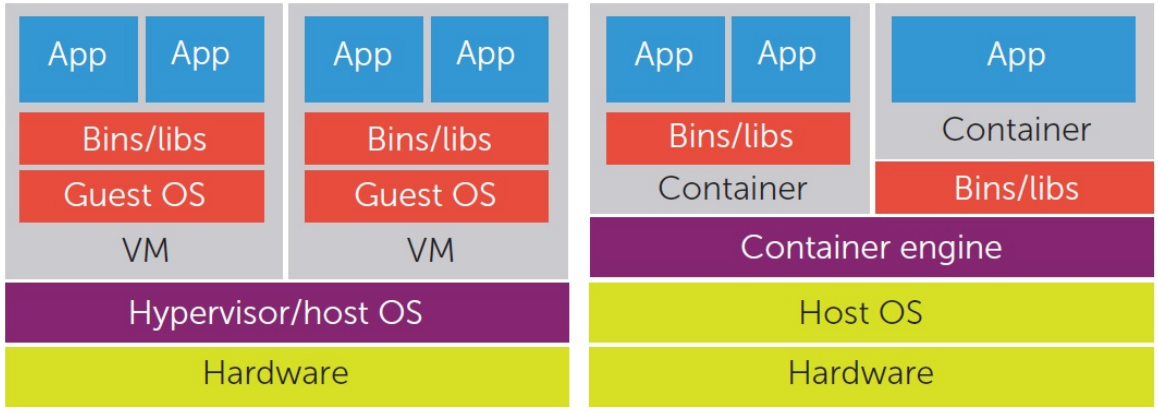

Figure 2.5: Comparison of virtualization architectures: hypervisor-based versus container-based [4].

Figure 2.5 shows the contrast between hypervisor-based and container-based virtualization. Hypervisor-based virtualization provides an abstraction for the guest operating systems (one per virtual machine). On the contrary, container-based virtualization works at the operating system level, providing abstractions directly for guest processes.

Microservices architecture does not require containers to deploy the software components [65]. Nevertheless, because containers are smaller, faster to instantiate, and faster to scale than traditional virtual machines, containers have become a popular approach to combine microservices architecture and container-based virtualization.

2.2 Literature Review

This section presents research related to this thesis divided into three categories: Machine Learning Platforms, Data Processing Frameworks, and Machine Learning and Big Data Frameworks applied to IoT.

2.2.1 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.* [66] introduced Rafiki, which is a system to provide both the training and inference services for ML models. In their work, users are exempted from constructing the ML models, tuning the hyper-parameters, and optimizing the prediction accuracy and speed. Instead, they upload their datasets, and configure the service to conduct training and then deploy the model for inference. Docker containers are used to implement ML workflows.

Lee *et al.* [67] presented PRETZEL, a prediction serving system designed for serving predictions over trained pipelines originally developed in ML.Net [68]. Their work explored multi-pipeline optimization techniques to reduce resource utilization and improve performance.

Crankshaw *et al.* [69] introduced Clipper, which is a general-purpose low-latency prediction

serving system. Their work provided a model abstraction layer and a common prediction interface that isolates applications from variability in ML frameworks and simplifies the process of deploying a new model or framework to a running application. In their work, models are encapsulated in containers to achieve process isolation.

Studies [66, 67, 69] have been focussed on providing generic ML platforms, which are in turn focussed 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. This research extends previously cited approaches by providing a framework designed to orchestrate and automate the execution of ML workflows with IoT data.

Zhao *et al.* [70] proposed a ML platform to support the sharing of ML models developed across various ML libraries. Their platform packs pre-trained ML models into Docker images, which can be deployed to an appropriate run-time environment. Besides packaging ML models in Docker images, the work described in this thesis enables training and deployment of models using container-based virtualization.

Predict I/O [71] is an-open source project that breaks down the ML workflows into four components: Data Source and Data Preparator, Algorithm, Serving, and Evaluation Metrics. It deploys models as Web services and makes REST APIs available for prediction queries. In its current implementation, only Spark MLlib is supported as a model engine. The framework proposed in this thesis provides generic support for ML frameworks, which avoids being tied to a specific library or implementation.

Cloud providers, like Amazon AWS [39], Microsoft Azure [41], and IBM Watson [72] have already included services to train and deploy ML models on IoT data. However, their services require integration of diverse components to preprocess the data and to train and deploy the models to infer online IoT data. This research proposes an approach in which the components required to apply ML models to IoT data are already integrated into a unified framework, which enables orchestration and automation of ML workflows.

Uber [34], Airbnb [35], Netflix [36], Google [37], and Facebook [38], which are some of the key players in ML have implemented their own ML orchestration platforms. Unfortunately, their platforms are proprietary or not entirely publicly available.

To summarize the differences and similarities between *ML4IoT* and existing platforms, a comparison is presented in Table 2.1.

Features	***ML4IoT***	**Rafiki**	**PRETZEL**	**Clipper**	**Acumos**	**Predict I/O**	**AWS IoT Azure IoT IBM IoT**
Easy cloud deployment	Yes	Yes	Yes	No	Yes	No	Yes
Training of ML models	Yes	Yes	No	No	No	Yes	Yes
Deployment of ML models	Yes	Yes	Yes	Yes	Yes	No	Yes
Integrated platform for ML on IoT data	Yes	No	No	No	No	No	No
Data preprocessing tasks sharing	Yes	No	No	No	No	No	No
ML model sharing	Yes	No	No	No	Yes	No	No
Support multiple ML libraries	Yes	Yes	No	Yes	Yes	No	Yes

Table 2.1: Comparison between *ML4IoT* and existing platforms.

2.2.2 Data Processing Frameworks

This section presents a review of data processing frameworks which have been used to apply ML to IoT data.

Morales *et al.* [73] proposed the Scalable Advanced Massive Online Analysis (SAMOA) framework, which enables standard ML algorithms to execute on top of distributed stream processing engines. It provides a set of distributed streaming algorithms for the most popular ML tasks such as classification, clustering, and regression. SAMOA also offers programming abstractions that support the development of new algorithms.

Hido *et al.* [74] presented a generic computational framework called Jubatus for online and distributed online ML. In their work, only the ML models are shared between distributed

servers, rather than data. Their framework can achieve high throughput for both online training and prediction on Big Data streams.

Apache Spark [75] is a project proposed in 2009 by a group at the University of California, Berkeley. Apache Spark has a programming model comparable to MapReduce [76], but extends it with a data-sharing abstraction called Resilient Distributed Datasets (RDD). Spark can work with a broad range of data processing workloads that previously required different engines, such as SQL, streaming, machine learning, and graph processing [77].

These studies [73–75] provided generic computational processing frameworks that can be used as the data processing engine for large amounts of data. Because the previously cited works were designed to be generic, they required adaptation and additional architecture layers to create effective and efficient solutions for IoT data. In contrast, this work focusses on providing an end-to-end solution for ML orchestration of IoT data that can be flexibly applied to different IoT use cases. The framework proposed in this work does not require adaptation and extra components for use with IoT data.

2.2.3 Big Data and Machine Learning in 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.* [78] 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.* proposed a semi-supervised method in association with a generative adversarial network [79] for supporting medical decision-making in an IoT-based health service system. Their approach was designed to solve problems involving both lack of labelled sets and imbalanced classes, which are common in medical datasets collected from IoT-based platforms.

Kumar *et al.* [15] proposed a scalable IoT-based three-tier architecture to process sensor

data and identify the most significant clinical parameters for patients to get heart disease. In their work, the first tier focussed on collection of IoT data, the second tier stored the data, and finally, the third tier applied a regression-based prediction model for heart disease.

Chou *et al.* [80] introduced a data-driven framework for fault detection in cellular-assisted IoT networks. Their framework uses ML techniques to analyze crowd-sourced measurements uploaded from several IoT devices. Then ML models are used to construct a global view of a radio environment, namely, the radio environment map (REM) for diagnosis and management purposes.

Shen *et al.* [81] presented a privacy-preserving SVM training scheme called secureSVM. Their approach tackled the challenges of data privacy and data integrity by using blockchain techniques to build a secure SVM training algorithm in multi-part scenarios where IoT data are collected from multiple data providers.

Sun *et al.* [82] outlined a framework for modelling and clustering attacker activity patterns based on IoT data collected from honeypots [83] deployed on a global scale. Their work combines a ML model and a graph-based clustering algorithm for analyzing attacker patterns without advanced feature engineering.

Wan *et al.* [48] designed an architecture focussed on manufacturing Big Data for active preventive maintenance. They used an Industry 4.0 designed solution called UPC UA [84] to ingest the data produced by IoT sensors. In addition, they proposed methods for collecting data and applying ML algorithms to both historical and online manufacturing data.

Javed *et al.* [85] presented a machine-learning-based smart controller for HVAC (heating, ventilation, and air conditioning systems) in commercial buildings. Data collected from IoT sensors were used to train a recurrent neural network model that could recognize when a room was unoccupied and switch off the HVAC, decreasing energy consumption.

The studies cited above aimed to use ML techniques to solve specific problems in IoT environments, for example, IoT sensors efficiency [78], fault-detection [80], medical decisions [15,86], security and privacy [81,82], preventive maintenance [48], and energy management [85].

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.* [87] proposed the SAMURAI solution, which is a batch and online data processing framework based on the Lambda architecture [88]. Their work integrates components for complex event processing, machine learning, and knowledge representation. Horizontal scalability is achieved with Big Data enabling technologies such as Spark and Apache Storm [89].

Ta-Shma *et al.* [25] 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.

Strohbach *et al.* [90] proposed a framework for addressing the volume and velocity challenges [91] of the IoT data. The components of their framework were organized according to the Lambda architecture design. However, they extended the Lambda architecture by supporting the creation of statistical and machine-learning models using historical data that were later applied to the online IoT data.

Mishra *et al.* [92] outlined a cognitive-oriented framework for IoT Big Data. Their framework was designed for effective data management and knowledge discovery over IoT data by applying the principles of data-centric architecture. A set of subsystems was combined to construct their final framework, which provided a real-time platform for IoT Big Data and ML capabilities for large-scale automation.

Sezer *et al.* [93] proposed a framework that provides support for storing IoT data, performing semantic rule reasoning, and applying ML methods. The goal of their work was to provide better management features for IoT sensors by means of data semantics. In addition, their framework explores interoperability issues among existing platforms designed to deal with IoT data.

Pham *et al.* [12] presented a self-adaptive cloud-based framework for real-time IoT Big

Data analytics. Their framework collects and analyzes data for IoT services using existing components such as M2M gateways [94], message brokers, and Big Data enabling tools such as Apache Storm and Cassandra.

In contrast to previous studies [25, 87, 90, 92, 93], this research 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.

2.3 Summary

This chapter has provided an overview of the concepts related to various topics that assist in understanding the framework proposed in this research. More specifically, an introduction to IoT, machine learning, microservices, and container-based virtualization terminologies has been presented. In addition, current studies on various topics related to the application of machine learning to IoT data were discussed and contrasted with the methods used in this work. The methods used in this research are discussed in more detail in the following chapter.

Chapter 3

ML4IoT Framework

This chapter describes the Machine Learning Framework for IoT data (*ML4IoT*). First, an introduction of the goals of the framework is presented in Section 3.1, followed by an overview of its components and sub-components in Section 3.2. Finally, the orchestration of ML workflows is discussed in Section 3.3.

3.1 Introduction

IoT devices produce massive amounts of data, at high speed, and from and from a vast variety of sources. The high volume, high velocity, and high variety of data require several Big Data enabling tools to ingest, store, and preprocess the IoT data before the training and inference of ML models can take place. Moreover, depending on the goals of the IoT application being developed, different ML frameworks must be used. However, creating and automating ML workflows at scale to train and infer real-world IoT data are challenging tasks, often leading to bottlenecks, production issues, and complex and repetitive code to manage. Also, applying ML to real-world IoT data includes challenges such as keeping ML models updated and running multiple ML workflows in parallel.

This thesis proposes the Machine Learning Framework for IoT data (*ML4IoT*), which is designed to orchestrate ML workflows to perform training and to enable inference by machine

learning models on IoT data. The *ML4IoT* uses containerized microservices to automate the execution of tasks specified in ML workflows, which are defined through REST APIs. The *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.

The overview of the framework is shown in Figure 3.1. In the *ML4IoT User Interface*, two types of ML workflows can be defined: batch and online machine learning workflows. These workflows are composed of a set of configurations, which define sequential tasks and parameters required to train and to deploy ML models using IoT data.

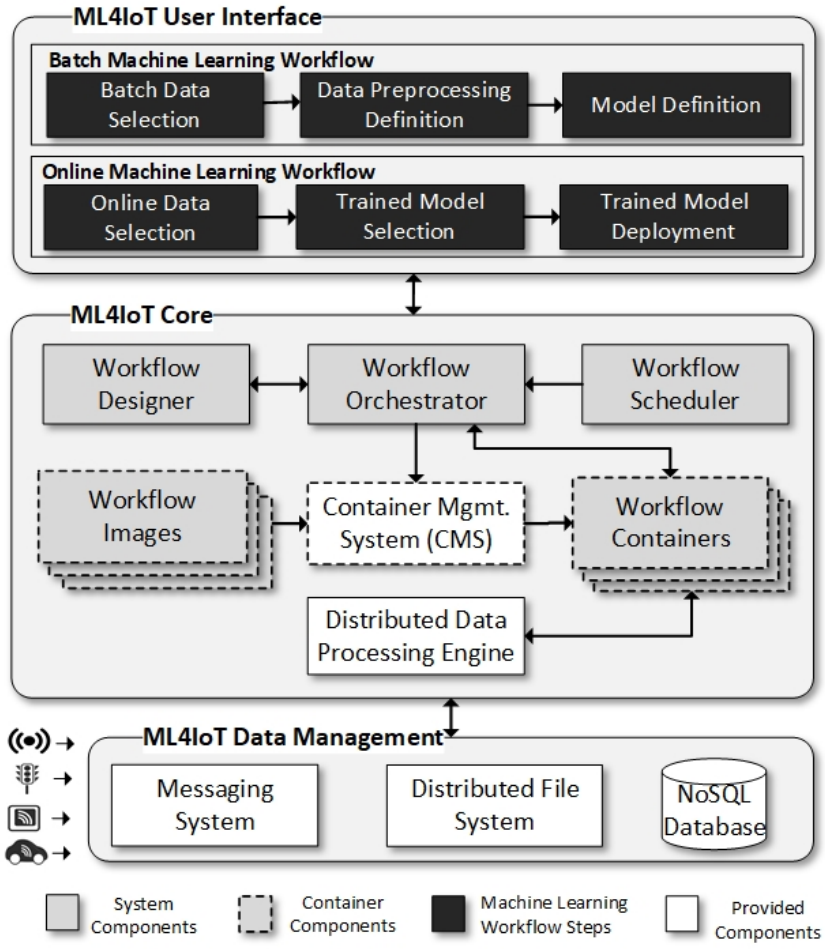

Figure 3.1: Machine Learning Framework for IoT data - ML4IoT.

To support the definition, orchestration, and scheduling of ML workflows, two key software engineering technologies are used in the *ML4IoT Core*: microservices architecture and container-

based virtualization. The microservices architecture enables the decoupling of *ML4IoT* software components into small and specialized services, bringing flexibility, reusability, and extensibility to the framework. In addition, container-based virtualization and microservices are combined to deal with challenges such as the multitude of ML frameworks available and the parallel execution of ML workflows.

In Figure 3.1, the *Workflow Designer*, *Workflow Orchestrator*, and *Workflow Scheduler*, are three microservices that are used to define, orchestrate, and schedule ML workflows, respectively. To orchestrate the execution of ML workflows, the *Workflow Orchestrator* communicates with the *Container Management System* (CMS) to create containerized microservices that perform the required data processing tasks. The software code needed to run these tasks is packaged in *Workflow Images* (Docker images), which are used by the CMS to deploy the *Workflow Containers* (Docker containers). The containerized microservices created by the CMS run the tasks defined in the ML workflows by themselves or using the *Distributed Data Processing Engine*.

The IoT data used during the execution of ML workflows are ingested and stored in the *ML4IoT Data Management* component, which is composed of three sub-components: a *Messaging System*, a *Distributed File System*, and a *NoSQL* database. *ML4IoT Data Management* also supports the storage of trained models and predicted data, and provides temporary storage for preprocessed data produced during ML workflow orchestration. The following sub-sections detail the components and sub-components of the *ML4IoT* framework.

3.2 ML4IoT Components

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

3.2.1 ML4IoT User Interface

The *ML4IoT User Interface* is a front-end component that provides users with a graphical interface to define ML workflows. Figure 3.2 shows an example of an implementation of the *ML4IoT User Interface*.

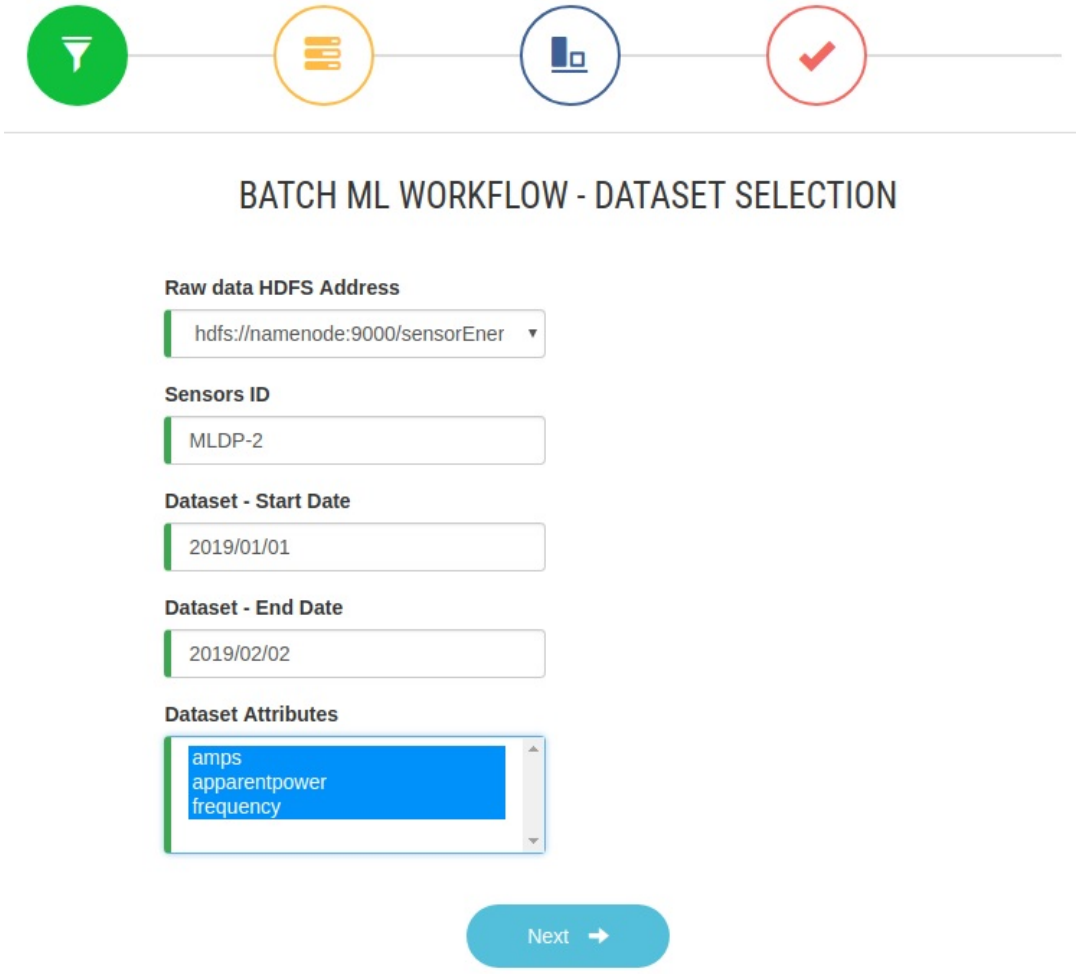

Figure 3.2: Prototype of the ML4IoT User Interface.

This component simplifies the tasks of creating these workflows because all the steps and parameters are defined in a simplified user interface. For example, to create an ML application to predict energy consumption, first ML models need to be trained, and then, these models need to be deployed to predict new information using online IoT data. The training of ML models involves the execution of three common steps. First, a set of data is selected, then,

it is preprocessed to remove inconsistencies, and finally, ML models are trained using the preprocessed data. The deployment of models also requires the execution of sequential steps, such as selecting online data, applying preprocessing tasks to these data, and using trained ML models to predict new values. Creating custom software code to execute these steps often leads to duplicate and hard-to-maintain glue code. In this way, defining ML workflows using a graphical interface helps to avoid unnecessary duplication of code and produces standardized representations of these workflows. The ML workflows defined in the *ML4IoT User Interface* are persisted in a *NoSQL* database and executed according to user requests. The steps performed during the definition of batch and online ML workflows are detailed in the next sub-sections.

Batch Machine Learning Workflow

Batch machine learning workflow is a logical representation of a set of sequential tasks and parameters required to build ML models. The execution of this type of workflow produces models, which can be used in online ML workflows.

The definition of a batch ML workflow involves the selection and configuration of historical datasets, preprocessing tasks, and ML algorithms. These tasks are executed in three steps shown in Figure 3.3 and discussed below.

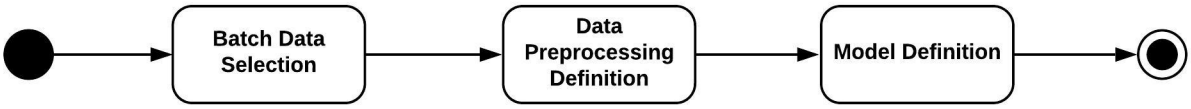

Figure 3.3: Steps performed during the creation of batch ML workflows.

- *Batch Data Selection* defines the parameters used to create historical datasets during 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.

Online Machine Learning Workflow

Online machine learning workflow is a logical representation of a set of sequential tasks and parameters required to deploy trained ML models to infer new information using online IoT data. The definition of an online ML workflow involves the selection and configuration of online datasets, trained ML models, and deployment parameters required to execute this type of workflow. These tasks are performed in three steps shown in Figure 3.4 and discussed below.

Figure 3.4: Steps performed during the creation of online ML workflows.

- *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 executing a 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 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.

3.2.2 ML4IoT Core

The *ML4IoT Core* uses the microservices architecture and container-based virtualization to support the definition, orchestration, and scheduling of ML workflows. The microservices architecture is an approach used in the *ML4IoT* that improves the flexibility of the framework. For example, machine learning is evolving at a rapid pace, and ML libraries and frameworks can appear and become outdated quickly. In *ML4IoT*, these frameworks and libraries used to build ML models can be added, replaced, or updated without affecting the other components of the framework. Moreover, splitting the design of ML workflows into small, specialized microservices facilitates the reuse of these software components. Lastly, it contributes to the extensibility of *ML4IoT* because when the design of ML workflows is divided into well-defined components, this creates natural points of extension for new functionalities.

Moreover, the combined use of container-based virtualization and microservices addresses common issues found during the development of ML applications in real-world scenarios. For instance, packaging software code into portable containerized microservices enables the *ML4IoT* to run ML workflows with conflicting software and hardware requirements concurrently. For example, ML workflows can use algorithms provided in different frameworks (e.g., TensorFlow

[95], MLlib [96], Scikit-learn [97]) running on a mix of CPUs and GPUs. It also provides process isolation between different ML workflows and ensures that a single failure does not affect the parallel execution of other workflows. Ultimately, the use of containerized microservices provides a convenient mechanism for horizontally and independently scaling the *ML4IoT* framework to execute multiple ML workflows in parallel. In the next subsections, the sub-components of the *ML4IoT Core* are described in detail.

Workflow Designer

This component is a microservice that provides a REST API service to store logical representations of ML workflows in a *NoSQL* database that stores the data in documents. These documents use a JSON (JavaScript Object Notation) structure and provide an intuitive and natural way to model ML workflows that is closely aligned with object-oriented design. 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 the *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 3.5 shows an entity-relationship (ER) diagram that illustrates the conceptual data model used to store the ML workflows in the *NoSQL* database.

The description of the entity types depicted in Figure 3.5 is given below:

- **BatchWorkflow** contains a list of the *Dataset* entities, a list of *BatchModels* entities, and 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.

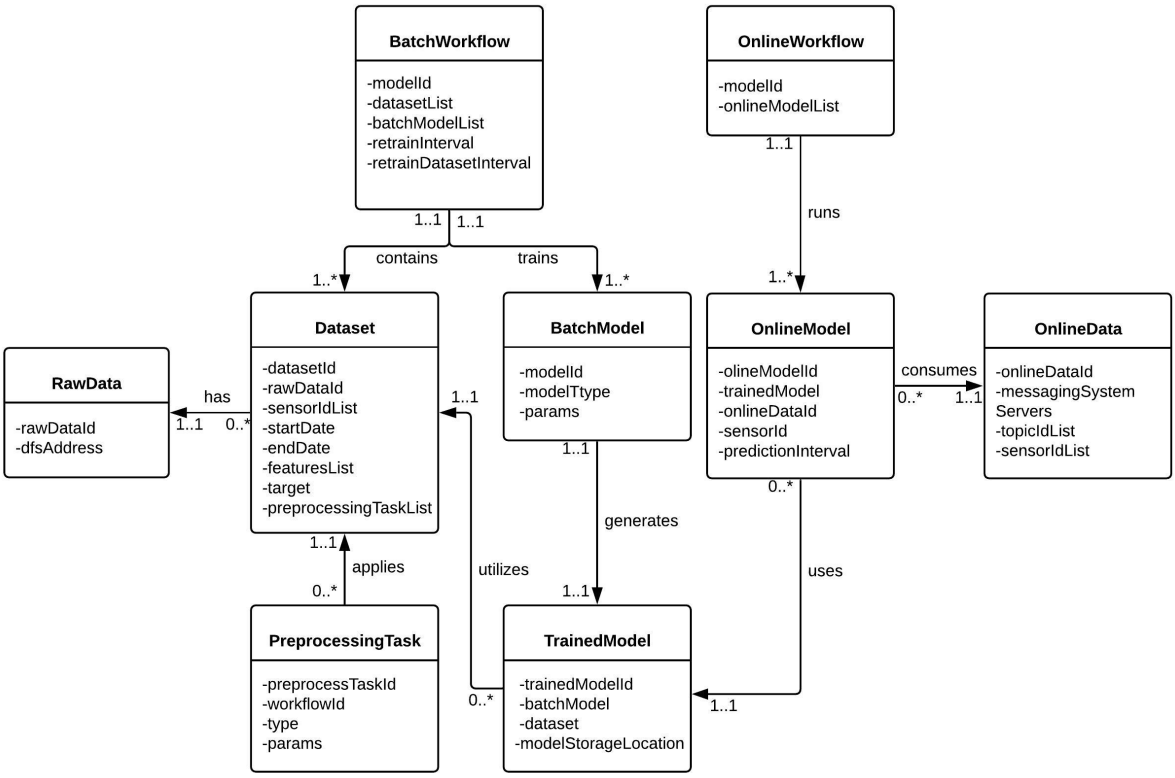

Figure 3.5: Conceptual data model for the ML4IoT.

- **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.

The model presented in Figure 3.5 identifies the following relationships between the entities.

- **Contains.** A *BatchWorkflow* contains one or more *Dataset* entities, and each *Dataset* can be associate to only one *BatchWorkflow*.

- **Has.** A *Dataset* has one *RawData*, and each *RawData* can be associated to many *Dataset* entities.

- **Trains.** A *BatchWorkflow* trains one or more *BatchModel*, and each *BatchModel* is trained by only one *BatchWorkflow*.

- **Applies.** A *PreprocessingTask* is applied to only one *Dataset*, and each *Dataset* can be applied many *PreprocessingTask* entities.

- **Generates.** A *BatchModel* generates only one *TrainedModel*, and a *TrainedModel* is generated by one *BatchModel*.

- **Utilizes.** A *TrainedModel* utilizes one only one *Dataset*, and a *Dataset* can be utilized by many *TrainedModel* entities.

- **Runs.** An *OnlineWorkflow* runs one or more *OnlineModel*, and each *OnlineModel* is executed by only one *OnlineWorkflow*.

- **Uses.** An *OnlineModel* uses only one *TrainedModel*, and each *TrainedModel* can be used by many *OnlineModel* entities.

- **Consumes.** An *OnlineModel* consumes only one *OnlineData*, and each *OnlineData* can be consumed by many *OnlineModel* entities.

In relational databases, a table contains data about just one entity, while in the document-oriented *NoSQL* database, a document can contain one or more of the entities depicted in Figure 3.5. An example of a batch ML workflow stored in a JSON document is shown in Listing 3.1.

```
{"_id": "1",
  "datasets": [{
    "_id": "1",
    "rawData": {"_id": "1",
    "hdfsAddress": "hdfs://namenode: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": "2",
      "type": "normalization", "serviceId": "4"},
    { "_id": "3",
      "type": "sliding-window", "params": {"timeWindow": "3"},  "serviceId": "5" }]
      } ],
  "batchModels": [
    {
      "_id": "1",
      "type": "random-forest",
      "params": {"datasetSplit": "0.95,0.05", "maxDepth": "5", "numTrees": "20"},
      "serviceId": "6" }
  ]
}
```

Listing 3.1: Batch ML workflow represented in a JSON file.

Workflow Orchestrator

This component is a microservice that provides orchestration capabilities in the framework described here. The *Workflow Orchestrator* automates execution, which accelerates ML application development, ensures consistent ML practices across the development lifecycle, and

optimizes the computational power required to execute these workflows. In addition, automating ML workflow execution can help users build and evaluate more models more rapidly, which can improve the quality of the trained ML models. 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 3.6 and 3.7 shows the steps performed during the orchestration of batch and online ML workflows, respectively.

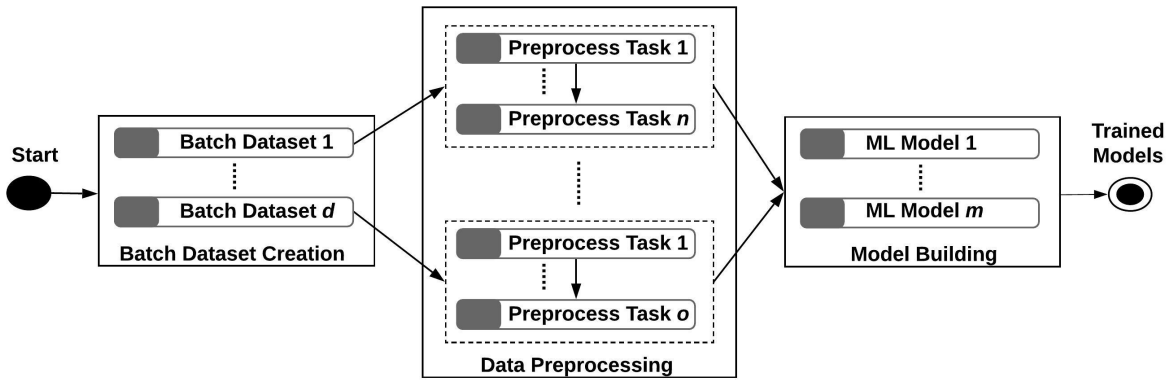

Figure 3.6: Orchestration steps of batch ML workflows.

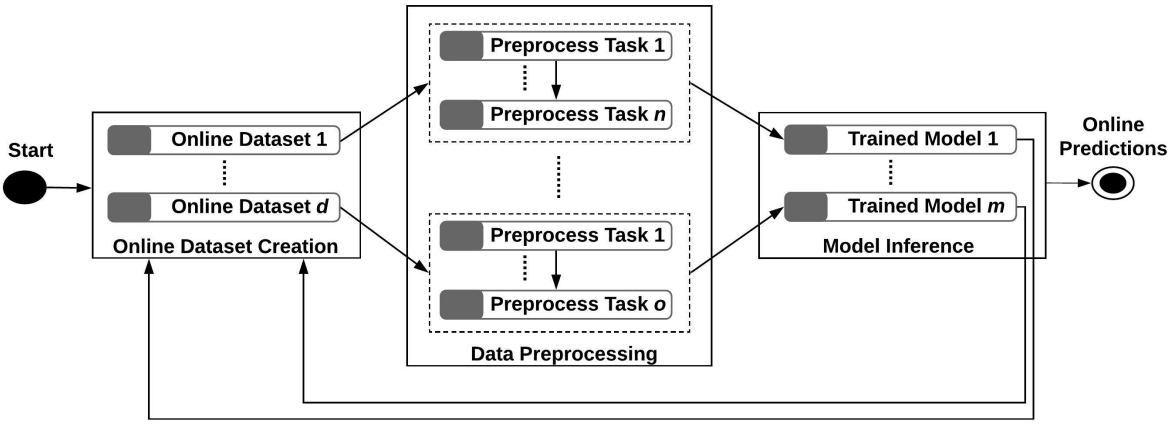

Figure 3.7: Orchestration steps of online ML workflows.

The orchestration of batch ML workflows involves the execution of three steps: batch dataset creation, data preprocessing, and model building. In addition, the data preprocessing step can be composed of multiple sub-steps. In the first step, datasets are created, then preprocessing

tasks are applied to these datasets, and in the third step, ML models are trained according to the algorithms defined in each workflow. In Figure 3.6, the number of ML models built at the end of the orchestration of the batch ML workflow is equal to the number of defined datasets times the number of ML algorithms. For example, if a batch ML workflow is defined with three datasets and two ML algorithms, then six ML models are trained when this workflow is executed.

The orchestration of online ML workflows also involves executing three computation steps: online dataset creation, data preprocessing, and model inference. In the first step, online datasets are created, then preprocessing tasks are applied to these datasets, and finally trained ML models are used to render new predictions using the preprocessed data. The preprocessing tasks applied to the online IoT data are the same preprocessing tasks that were used to train the model being deployed. For example, when a trained ML model is deployed in an online ML workflow, the preprocessing tasks applied to the batch dataset, which were used to train this model, are automatically configured and applied to the online dataset. Examples of preprocessing tasks implemented in the prototype system built to evaluate the *ML4IoT* include removal of null values, removal of repeated values, sliding window, data aggregation, and normalization. Moreover, Figure 3.7 shows that, for each trained model, the orchestration flow is executed repeatedly to render new predictions. This prediction cycle is determined by a parameter configured for each deployed model when defining the online ML workflow.

The *Workflow Orchestrator* orchestrates 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. For instance, the batch ML workflow shown in Listing 3.1 is composed of one dataset, four preprocessing tasks (removal of repeated values, data aggregation, normalization, sliding window), and two ML models (LSTM and RF). During the execution of this workflow, seven containers are created dynamically. The first container creates the batch dataset, then four containers execute the data preprocessing tasks sequentially, and the last two containers perform the building of the two ML models in parallel. The details of the orchestration of batch and online ML workflows are discussed in Section 3.3.

Workflow Scheduler

This component can re-execute batch ML workflows automatically at intervals defined in each one of these workflows. The goal is to provide a method to retrain ML models by executing batch ML workflows at regular intervals. 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 distributions can be assumed to 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.

The *ML4IoT* keeps the information about which configuration choices were made to build ML workflows in a *NoSQL* database. That ensures that all the steps that led to actual ML models can be reproduced. By simply re-executing the batch ML workflows to new IoT data, ML models are retrained. Also, if a retrained ML model is deployed in a online ML workflow, this workflow is updated automatically.

The parameters used to retrain batch ML workflows include a configuration to update the interval range of the datasets and the frequency of re-execution of the workflows. For example, a batch ML workflow can be scheduled to execute every day at midnight with each dataset contained data from the past 30 days. Then, each execution re-executes the entire workflow, which generates a new version of the previous trained ML models.

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 on non-critical business hours and also facilitates the distribuition of the processing workload along different periods of the day. To start de re-execution of batch ML workflows, the *Workflow Scheduler* sends a REST request to the *Workflow Orchestrator*, which orchestrates the necessary steps. The orchestration steps performed during the re-execution of batch ML workflows are similar to the steps performed during the first execution, but, the batch datasets used to train the ML models are updated and created with recent data according to the

parameters defined in the batch ML workflows.

Container-based Components: Workflow Images, Container Management System, and Workflow Containers

Workflow Images, *Container Management System*, and *Workflow Containers* are container-based software employed to support the execution of the ML workflows. *Workflow Images* are Docker images, which are instantiated by the *Container Management System* and become Workflow Containers (Docker containers). The details of these sub-components are described below.

- ***Workflow Images*** are Docker images where reusable software code is implemented to execute tasks defined in ML workflows. Besides the software code to execute specific tasks, the Docker images also allow for packaging together all the software dependencies needed to run them as an isolated process (e.g., libraries and binaries). In this way, each step of computation required to execute batch and online ML workflows can be implemented on a separated Docker image. An example of a Dockerfile used to create Docker Images is shown in Listing 3.2.

```
1 FROM jalves7/tensorflow−hadoop:latest
2 MAINTAINER Jose Miguel <jalves7@uwo.ca>
3 ADD ./modelFactory.py /app/trainLSTM.py
4 ADD ./docker−entrypoint.sh /app/docker−entrypoint.sh
5 ENTRYPOINT [”sh”, ”/app/docker−entrypoint.sh”]
```

Listing 3.2: Example of a Dockerfile that is used to define Docker images.

The first line in Listing 3.2 is used to prepare and set up the Docker image for the execution. With the help of the inheritance capacity of Docker, it is possible to start with an environment that had an ML framework already installed, as indicated in line (1), by picking the image *tensorflow-hadoop:latest*. Line (3) copies to the Docker image a Python code that trains an ML model. Lines (4-5) define a start point for the Docker image, that is a script which executes the Python code responsible for training an LSTM model.

- ***Container Management System*** is the software responsible for deploying containers according to the requests sent by the *Workflow Orchestrator*. The *ML4IoT* was designed to be executed in an environment controlled by a CMS, specifically Docker Swarm [98]. From an architectural standpoint, no specific CMS implementation is required as long as it supports all functionalities expected. In theory, distinct implementations can be used simultaneously, for example, Kubernetes [99], requiring only the adaptation of the code in the *Workflow Orchestrator* that interact directly with the CMS. The following list summarizes the features that must be provided by the CMS:

  - *Docker support:* the containerized microservices used to execute ML workflows have an associated Docker image, and the CMS must support the creation of multiple containers based on this image.

  - *API access:* the CMS must provide an API that the *Workflow Orchestrator* can use to request the creation of containers to run tasks defined in the ML workflows.

  - *Container scheduling:* once a container is created, it must be automatically scheduled to execute in one available server.

- ***Workflow Containers*** are Docker containers created dynamically by the CMS according to requests from the *Workflow Orchestrator*. In the *ML4IoT*, the containers can be destroyed when they finish their execution, which optimizes the use of computational resources. Also, no code needs to be rewritten to move the containers from experimentation to production, or to deploy them in multiple platforms, such as locally or in cloud services. Figure 3.8 shows the simplified lifecycle of Docker containers.

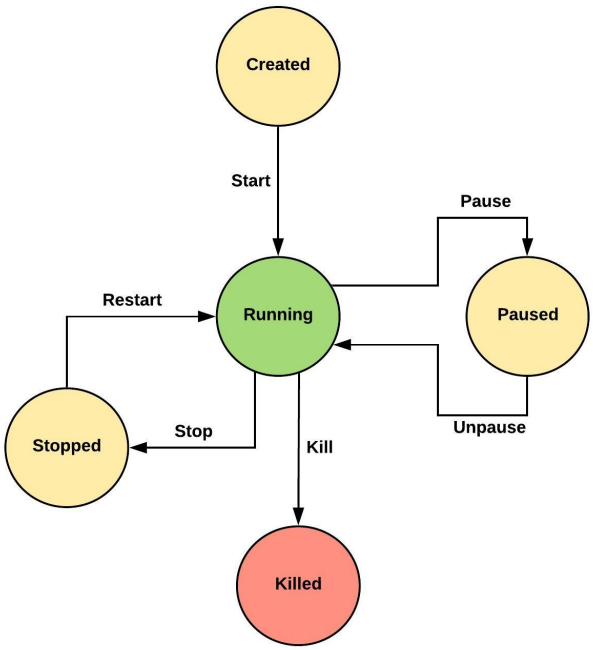

Figure 3.8: Simplified lifecycle of a Docker Container.

When new containers are created, the first state of the containers is *created*. Then, the CMS starts their execution, moving them to the *running* state. When the containers finish their execution, they send a message to the REST API provided by the *Workflow Orchestrator* to indicate that their execution is done. Then, the containers go to the *stopped* state. The *paused* and *kill* states are used for the CMS to pause or to force the stop of the containers execution.

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. The *ML4IoT* was designed to use Apache Spark [75] as the *Distributed Data Processing Engine*. The extension of the framework to support new implementations of the *Distributed Data Processing Engine* involves extending the code implemented in

the *Workflow Images*.

3.2.3 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 formed 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 deal with massive amounts of IoT data ingestion, 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. The *ML4IoT* was designed to use Apache Kafka [100] as the *Messaging System*, Apache Hadoop as the *Distributed File System* [101], and MongoDB [16] as the *NoSQL* database. Similarly to the *Distributed Data Processing Engine*, *ML4IoT* can use different implementations of the *Messaging System*, *Distributed File System*, and *NoSQL* database, by extending the *Workflow Images* in which is packaged the code used to execute data processing tasks that interacts with these three components.

Messaging System

The *Messaging System* is a distributed application where the IoT data is ingested in the platform. The framework requires that the IoT data have a time-series format, which means that each inputted data needs to have a timestamp attribute. This attribute helps to store the IoT data efficiently in the *Distributed File System*, by allowing the organization of the data according to temporal information. Also, the timestamp attribute is used to select historical and online datasets during the execution of ML workflows. Popular data preprocessing tasks applied to IoT data, such as data aggregation, slide-window, and feature generation, also require timestamp attributes during their execution. The *Messaging System* can handle a high volume of IoT data

ingestion, and it can provide mechanisms that enable the parallel consumption of data.

Distributed File System

The *Distributed File System* (DFS) is the component where the IoT data is stored permanently because the *Messaging System* is not optimized to store a large volume of data. Also, the DFS allows the storage of any data format and size. Hence, the DFS also stores the trained ML models produced by the execution of batch ML workflows. In the *ML4IoT*, the data is pulled from the *Messaging System* to the *Distributed File System* using batch processing jobs performed by the *Distributed Processing Engine*.

NoSQL Database

The *NoSQL* database stores temporary data generated during the execution of batch and online ML workflows. The execution of ML workflows involves intensive reading and writing tasks, and *NoSQL* databases usually present better reading and writing rates than other traditional tools designed to handle data storage such as *RDBMS* or *Distributed File Systems*.

3.3 ML4IoT Design

The development of ML applications in IoT data usually involves the creation of multiple ML workflows first to build ML models, and second to deploy these models to render predictions using online data. Moreover, to build ML models, several ML workflows need to be tested and executed with different configurations in order to find the optimal combination of these parameters. Besides, the hyper-parameters of ML algorithms, components such as datasets and preprocessing tasks employed in the ML workflows also affect the ML models accuracy.

Orchestrating the execution of ML workflows allows the automated running, testing, and tuning of multiple ML workflows in parallel, which accelerates the development of ML applications. Also, the orchestration of ML workflows helps to provide an established path to deploying

ML models into production environments, which enables users to build ML applications at scale easily. Lastly, the automated execution of ML workflows is closely aligned with one of the foundational pillars of DevOps [102], which is a popular methodology focused on increasing the organizations ability to deliver applications and services at high-velocity.

The following sub-sections discuss the orchestration details of batch and online ML workflows.

3.3.1 Batch ML Workflow Orchestration

The pseudo-code used in the *Workflow Orchestrator* to orchestrate the execution of batch ML workflows is outlined in Algorithm 1, which highlights the execution of three steps: batch dataset creation, data preprocessing, and model building. Some instructions of the Algorithm 1 contains high-level abstractions (e.g., *batchWorkflow, dataset*) that are based on entities described in the conceptual data model shown in 3.5.

Algorithm 1: Orchestration of Batch ML Workflows

```
    /* Batch Dataset Creation [line 1 to 3] */
 1: procedure StartBatchWorkflowExecution(batchWorkflow)
 2:     for each dataset in batchWorkflow do
 3:         createHistoricalDataset(dataset)

 4: procedure ExecuteBatchWorkflowNextStep(batchWorkflow, dataset, lastStep)
 5:     workflowNextStep ← getWorkflowNextStep(batchWorkflow,lastStep )

    /* Data Preprocessing [line 6 to 8] */
 6:     if workflowNextStep is DataPreprocessing then
 7:         preprocessingTask ← getNextPreprocessingtask(batchWorkflow,lastStep )
 8:         applyPreprocessingTask(dataset,preprocessingTask, batchWorkflow )

    /* Model Building [line 9 to 11] */
 9:     else if workflowNextStep is ModelBuilding then
10:         batchModel ← getWorkflowNextModel(batchWorkflow,lastStep )
11:         buildMachineLearningModel(batchModel, dataset, batchWorkflow )
```

In Algorithm 1, lines (1-3) execute the creation of historical datasets defined in the batch ML workflows. Line (3), calls a procedure that interacts with the CMS to create a containerized microservice that executes the creation of a new historical dataset. It is necessary to mention that

the requests sent to the CMS use *asynchronous* calls, which allow the creation and execution of multiple containerized services simultaneously. For example, if a batch ML workflow is defined with three datasets, the execution of the tasks to create these datasets happens in parallel. When a containerized microservice created in the first step (batch dataset creation) finishes its execution, it sends a REST request to the *Workflow Orchestrator* that executes the procedure, *ExecuteBatchWorkflowNextStep* (line 4). Then, depending on the workflow definitions, data preprocessing (lines 6-8) or the model building (lines 9-11) steps are executed.

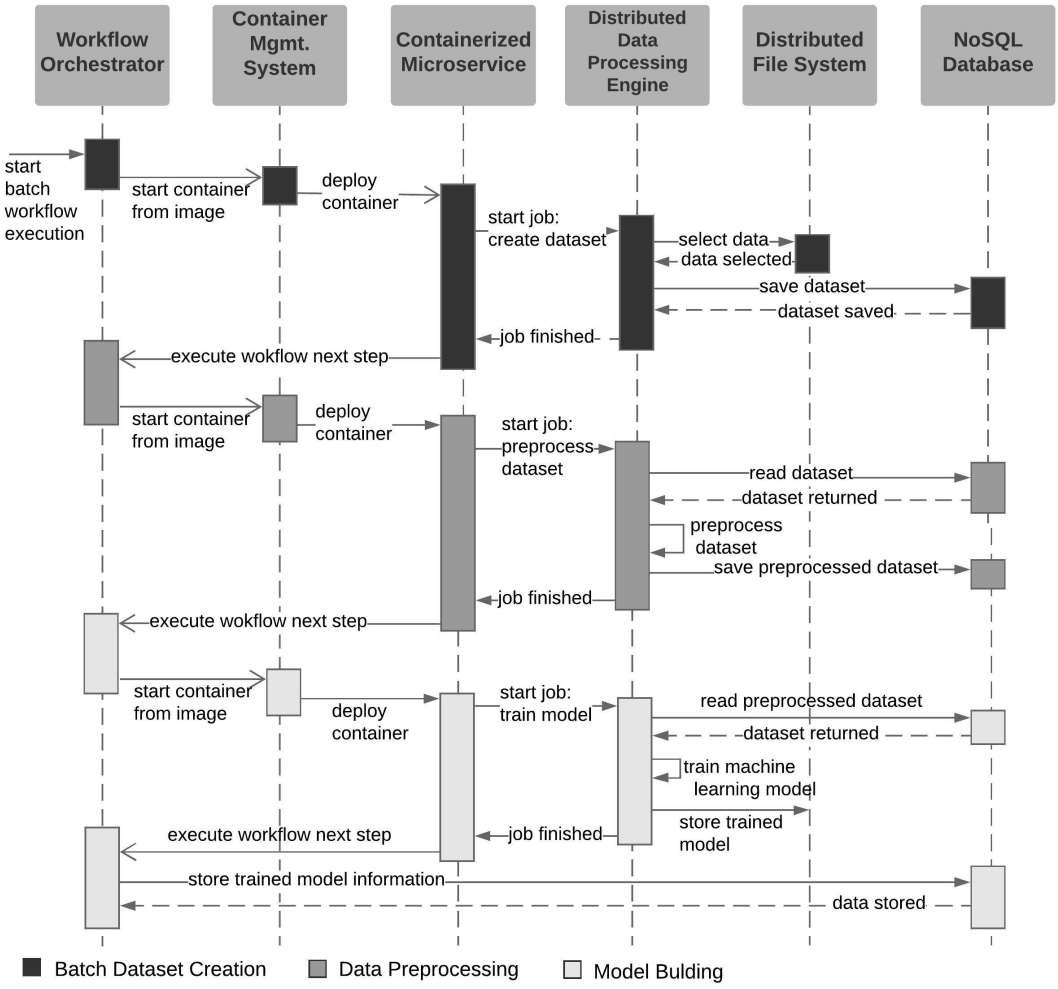

Figure 3.9: Sequence diagram: execution of batch ML workflows.

Also, the execution of the steps shown in Algorithm 1 involves the interactions among the containerized microservices created dynamically and other components of the framework. The

sequence diagram depicted in Figure 3.9 shows an example of these interactions. The diagram has three main parts highlighted by the different shades of grey in the activation bars, which indicates the execution of tasks related to the three steps outlined in Algorithm 1.

The details of the diagram are discussed below.

- **Batch dataset creation**is the first step performed in the execution of batch ML workflows. In the diagram, 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 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, the diagram 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.

- **Model Building** is the step where tasks are executed to train and test ML models according to the parameters defined in batch ML workflows. The diagram 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 execution is stored in the *Distributed File System* and can be used in online ML workflows.

3.3.2 Online ML Workflow Orchestration

The pseudo-code used in the *Workflow Orchestrator* to orchestrate the execution of online ML workflow is outlined in Algorithm 2 and highlights the execution of three steps: online dataset creation, data preprocessing, and model inference. Some instructions of the Algorithm

2 contains high-level abstractions (e.g., *batchWorkflow, dataset*) that are based on entities described in the conceptual data model shown in 3.5.

Algorithm 2: Orchestration of Online ML Workflows

```
    /* Online Dataset Creation [line 1 to 3] */
 1: procedure StartOnlineWorkflowExecution(onlineWorkflow)
 2:     for each onlineModel in onlineWorkflow do
 3:         createOnlineDataset(onlineDataset)

 4: procedure ExecuteOnlineWorkflowNextStep(onlineWorkflow, onlineModel, lastStep)
 5:     workflowNextStep ← getWorkflowNextStep(onlineWorkflow, lastStep)

    /* Data Preprocessing [line 6 to 8] */
 6:     if workflowNextStep is DataPreprocessing then
 7:         preprocessingTask ← getNextPreprocessingtask(onlineWorkflow, onlineModel, lastStep)
 8:         onlineDataset ← getOnlineDataset(onlineWorkflow, onlineModel)
 9:         applyPreprocessingTask(onlineDataset, preprocessingTask, onlineWorkflow, onlineModel)

    /* Model Inference [line 9 to 11] */
10:     else if workflowNextStep is ModelInference then
11:         inferModel(onlineDataset, onlineModel, onlineWorkflow )
```

In Algorithm 2, lines (1-3) execute the creation of online datasets defined in the online ML workflows. The objective of online ML workflows is to deploy trained ML models to infer new information using online IoT data. For this reason, this type of workflow is executed continuously, and the containerized microservice created by the function in line (3) is not destroyed after its execution. Contrary to the batch ML workflows, the containerized microservices created in the first step (online dataset creation) repeat their execution at intervals defined in the online ML workflows. When a containerized microservice finishes the creation of an online dataset, it sends a REST request to execute the procedure *ExecuteOnlineWorkflowNextStep* (line 4). Then, depending on the workflow definitions, data preprocessing (lines 6-8) or the model inference (lines 9-11) steps are executed.

The orchestration of online ML workflows also involves the interactions among the containerized microservices created dynamically and other components of the framework. The sequence diagram depicted in Figure 3.10 shows an example of these interactions.

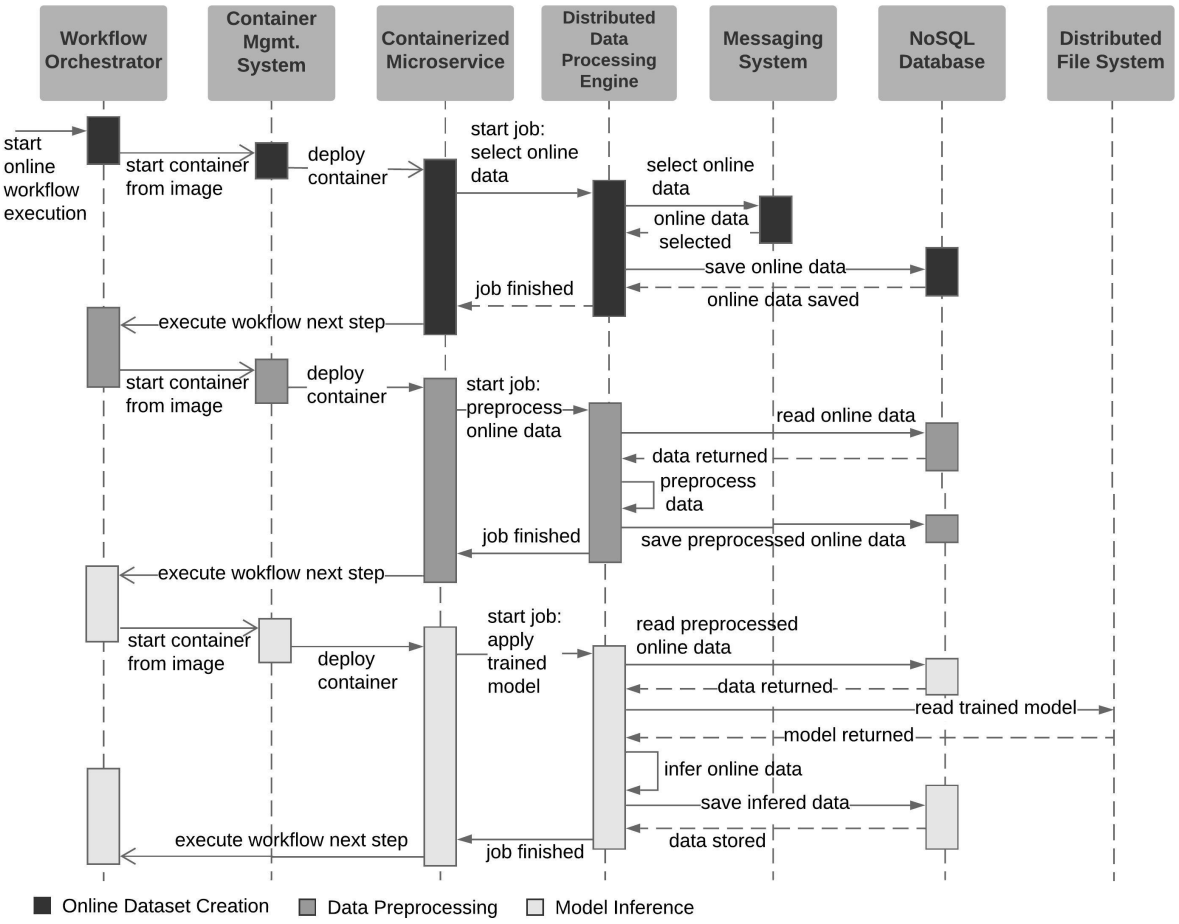

Figure 3.10: Sequence diagram: execution of online ML workflows.

The diagram has three main parts highlighted by the different shades of grey in the activation bars, which indicates the execution of tasks related to the three steps outlined in Algorithm 2. The details of the diagram are discussed below.

- **Online Dataset Creation** is the step where data processing tasks are executed to create an online dataset from the online IoT data that is ingested in the *Messaging System*. The diagram shows that this step starts with the creation of a containerized service that uses the *Distributed Data Processing Engine* to select from the *Messaging System* an online IoT dataset. Then, the selected online dataset is stored in the *NoSQL* database.

- **Data Preprocessing** is the step where data preprocessing tasks are applied to the online datasets created previously. These tasks are the same which were used to train the ML

model that is being deployed in the workflow. The sequence of tasks demonstrated in the diagram for the data preprocessing step is repeated for each preprocessing task associated with the trained ML model. For example, if the removal of null values and the elimination of repeated values were applied to the batch dataset used to train the ML model, the same tasks are applied to the online dataset.

- **Model Inference** is the step where tasks are executed to use trained ML models to infer new information according to the parameters defined in the workflow. In the diagram, a containerized microservice is used to read a trained ML model from the *Distributed File System*, then the model is employed to infer new values using a preprocessed online dataset. The predictions produced by the trained ML model are stored in the *NoSQL* Database.

3.4 Summary

This chapter has introduced the *ML4IoT*, presented its components and sub-components, and discussed the orchestration of ML workflows.

Chapter 4

Evaluation

This chapter presents an evaluation of the *ML4IoT* focused on three main aspects: machine learning orchestration, elasticity, and performance. It starts by introducing the implementation details of a prototype system built to perform the experiments in this section. Next, it details the IoT data used in the experiments. Following that, an experimental setup section outlines the infrastructure used to run the experiments. Finally, it describes the tests conducted to evaluate the framework. The first experiment investigated the feasibility of the framework by assessing its ability to orchestrate ML workflows on different IoT datasets. The second experiment examined the elasticity of the framework by studying its ability to scale to support the execution of multiple batch ML workflows in parallel. The performance of the framework was evaluated in the third experiment by analyzing the latency of the execution of online ML workflows.

4.1 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 described below.

- ***ML4IoT User Interface.*** This component is a web application implemented as a microservice in AngularJS. The *ML4IoT User Interface* component communicates with the

others component of the framework using REST APIs.

- ***ML4IoT Core.*** The three microservices, *Workflow Designer*, *Workflow Orchestrator*, and *Workflow Scheduler*, are implemented in Java and Spring Boot. The *NoSQL* database that stores the ML workflows is MongoDB. Table 4.1 shows the container images that were implemented to support the orchestration of ML workflows. The *Container Management System* (CMS) and the *Distributed Data Processing Engine* used in this evaluation were Docker Swarm and Apache Spark Framework, respectively.

Workflow Type	**Orchestration Step**	**Image Name**	**Description**
Batch	Batch Dataset Creation	data-selection	Creates historical datatsets
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 Dataset 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

Table 4.1: Containers images and services implemented in each image.

- ***ML4IoT Data Management.*** The *Messaging System* component is implemented using Apache Kafka. The *Distributed File System* uses the Apache Hadoop File System, and the *NoSQL* database is implemented using MongoDB.

4.2 IoT Data

In the experiments presented in this section, two distinct sets of real-world IoT data were used. The details of the data are given below.

4.2.1 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. Listing 4.1 shows an example of the raw data returned by the API.

```
{
  "success": 1,
  "error": "",
  "time": "2019-03-06T01:05:54.457Z",
  "result": {
    "tz_abbr": "EST",
    "colID": [1528556549],
    "colLabel": ["10:02:29"],
    "rowData": [[35.2]],
    "rowID": ["4708"],
    "rowLabel": ["MLDP-1"],
    "seriesUnits": [ "Amps"]
  }
}
```

Listing 4.1: Sample of IoT energy raw data.

The JSON file returned by the API does not specify which feature is being provided. The feature retrieved in each response of the API is determined in each request sent to the API. For each sensor, 60 distinct features were collected. These features are composed of the ten distinct energy domain measures shown in Table 4.2 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.

Features	Description
Apparent Power	The power supplied to the electric circuit
Reactive Power	The energy generated or 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	How fast an electric current flows in the electric circuit
Volts	The energy potential 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 sensors location

Table 4.2: Features and description of the energy data.

4.2.2 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 [103], where the data were refreshed every 5 minutes. The data provided by the API are refreshed every 5 minutes, and they were returned in an XML format shown in Listing 4.2.

```
<pms>
  <fecha_hora>30/12/2018 16:35:08</fecha_hora>
  <pm>
    <idelem>3409</idelem>
    <descripcion>SEPLVEDA ENTRADA CRUCE  N-S</descripcion>
    <accesoAsociado>240102</accesoAsociado>
    <intensidad>85</intensidad>
    <ocupacion>0</ocupacion>
    <carga>3</carga>
    <nivelServicio>0</nivelServicio>
    <intensidadSat>3000</intensidadSat>
    <error>N</error>
    <subarea>1718</subarea>
    <st_x>436008,175534995</st_x>
    <st_y>4472593,78531503</st_y>
  </pm>
</pms>
```

Listing 4.2: Sample of IoT traffic raw data.

The features available in the traffic data are described in Table 4.3.

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

Table 4.3: Features and description of the traffic data.

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.

4.3 Experimental Setup

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 4.4: Hardware environment used in the experiments.

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 data tools used to implement the prototype system were configured in clusters running on Linux containers. Table 4.5 describes the initial cluster's configuration of each distributed application.

The energy and traffic IoT data were ingested into Apache Kafka at an average rate of 525.81 records/sec. Apache Kafka was configured to retain the IoT data for one month, which was equivalent to approximately 275 GB of data.

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

Table 4.5: Clusters configuration of big data tools used in the prototye system.

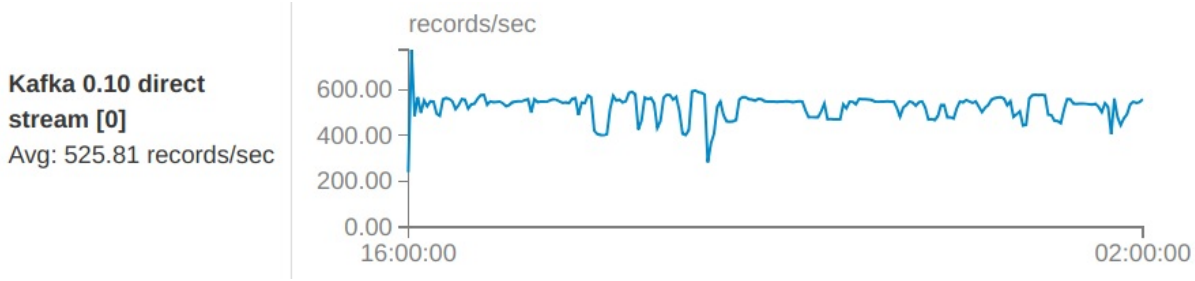

Figure 4.1: Average rate of IoT data ingestion.

The HDFS stored 4,988,326,606 sensor readings in a compressed format (ORC) that reduced the data size from 1100 GB to 44 GB.

4.4 Machine Learning Orchestration Evaluation

The goal of this evaluation was to validate the ability of the framework of providing orchestration services to automate the execution of ML workflows to train and infer ML models on IoT data. The experiment performed in this evaluation is described below.

4.4.1 Experiment

To demonstrate how *ML4IoT* can automate ML workflow orchestration in IoT data, batch ML workflows were created to train ML models to predict short-term energy consumption and traffic flow. Online ML workflows were also used to deploy the previously trained models and make predictions using online IoT data.

Two types of IoT data were used in this experiment, energy consumption and traffic flow.

Energy consumption prediction is crucial for improved decision-making to reduce energy consumption and CO_2 emissions. Predicting traffic flow can also help reduce air pollution and provide more secure traffic conditions. The ML workflows were defined using the REST APIs provided by *ML4IoT*. The tasks defined in the ML workflows were executed automatically while being orchestrated by backend services provided by *ML4IoT*. Details of the batch and online ML workflows created in this experiment are described below.

- **Batch ML workflows.** As described in Table 4.6, four batch ML workflows were built to train the ML models. In these workflows, the short-term prediction horizons varied from 15 to 60 minutes. Each batch ML workflow was defined to execute with both datasets (energy and traffic) at the same time.

Batch Workflow	Minutes Ahead
1	15
2	30
3	45
4	60

Table 4.6: Minutes ahead defined in each batch ML workflow.

  - **Batch Data Selection**. The energy dataset contained data from one sensor, and nine features were used as input to predict a target, which was the power feature. The traffic dataset also contained data from one sensor, the target was the velocity feature, and four features were used as input. Both datasets contained two months of historical data. Table 4.7 shows details of the energy and traffic datasets used in this experiment.

Dataset	Inputs	Target	Readings
Energy	ApparentPower-last, ReactivePower-last, PF-last, Amps-last, Volts-last, Frequency-last, Z-last, Humidiy-last, Temp-last	Power-last	252.447.882
Traffic	Occupation, Service Level, Load, Intensity	Velocity	12.828

Table 4.7: Description of the energy and traffic datasets.

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, RF and LSTM. 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.

Algorithm 3: Training of Machine Learning Models

Input: *Historical Dataset (D)*, *Model (M)*, *Training Parameters (TP)*
Output: *List of trained models(TM)*

1: $D_{train}, D_{test} \leftarrow$ splitDataset(D)
2: **if** output from $Model(M)$ is multiple **then**
3: $trainedModel \leftarrow$modelBulding(D_{train}, D_{test},$model$, TP)
4: append $trainedModel$ to TM

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

The steps performed to train the ML models are described in Algorithm 3, which contains an if statement to check the 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 execution of batch ML workflows, the trained ML models were used to infer online IoT data using online ML workflows, which are described below.

- **Online ML workflows.** Also, four online ML workflows were created to apply the previously built model 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.

Algorithm 4: Inference of Machine Learning Models

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

```
1:  for T interactions do
2:      if traineModelOutputType is Multiple then
3:          nextPredictedStep_list ←modelPrediction(onlineData, trainedModel)
4:          append nextPredictedStep_list to onlineData
5:          append nextPredictedStep to predictedOutput_list

6:      else if traineModelOutputType is single then
7:          for each attribute in onlineData do
8:              nextPredictedStep ←modelPrediction(onlineData,trainedModel)
9:              append nextPredictedStep to nextPredictedStep_list
10:         append nextPredictedStep_list to onlineData
11:         append nextPredictedStep_list to predictedOutput_list
```

  - **Trained Model Selection**. 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 4 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.

– **Trained Model Deployment**. The prediction intervals were configured according to the number of steps ahead for which each model was trained. For example, the ML models trained to predict 15 steps ahead were configured to predict the data in intervals of 15 minutes.

4.4.2 Results and Discussion

The prediction performance of the ML models was evaluated using the mean absolute error (MAE) and the mean square error (MSE) metrics defined in Eqs. 4.1 and 4.2, where y_k and $\hat{y_k}$ are the actual and predicted values respectively. The MAE is an average of the absolute prediction errors, and the MSE is the average of the squares of the prediction errors. MAE and MSE are two of the most common metrics used to measure accuracy for continuous variables, and they are negatively oriented scores, meaning that lower values are better. MAE and MSE are also scale-dependent, which provides a straightforward way to quantify the prediction error:

$$MAE = \frac{\sum_{k=1}^{n} |y_k - \hat{y_k}|}{n} \tag{4.1}$$

$$MSE = \frac{\sum_{k=1}^{n} (y_k - \hat{y_k})^2}{n} \tag{4.2}$$

Table 4.8 shows the results for the prediction accuracy of the ML models deployed on the online ML workflows.

Minutes Ahead	Traffic				Energy			
	MAE		MSE		MAE		MSE	
	RF	LSTM	RF	LSTM	RF	LSTM	RF	LSTM
15	0.0910	0.0985	0.0305	0.0176	0.0746	0.1062	0.0170	0.0236
30	**0.0863**	0.0916	**0.0273**	0.0155	0.0357	0.0572	0.0046	0.0079
45	0.1210	0.0849	0.0523	0.0134	0.0496	**0.0483**	0.0053	0.0066
60	0.0941	**0.0769**	0.0309	**0.0117**	**0.0306**	0.0503	**0.0040**	**0.0063**

Table 4.8: Prediction accuracy of ML models rendering prediction using online IoT data.

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 MAE and MSE than models trained with the RF algorithm on traffic data. On the other hand, with the energy data, the best results were obtained by models trained with the LSTM algorithm. However, the plots depicted in Figs. 4.2, 4.3, 4.4, and 4.5 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 on online IoT data because of several factors such as the number of online samples available and time and processing constraints. Tuning and retraining the ML models can also help increase overall prediction quality.

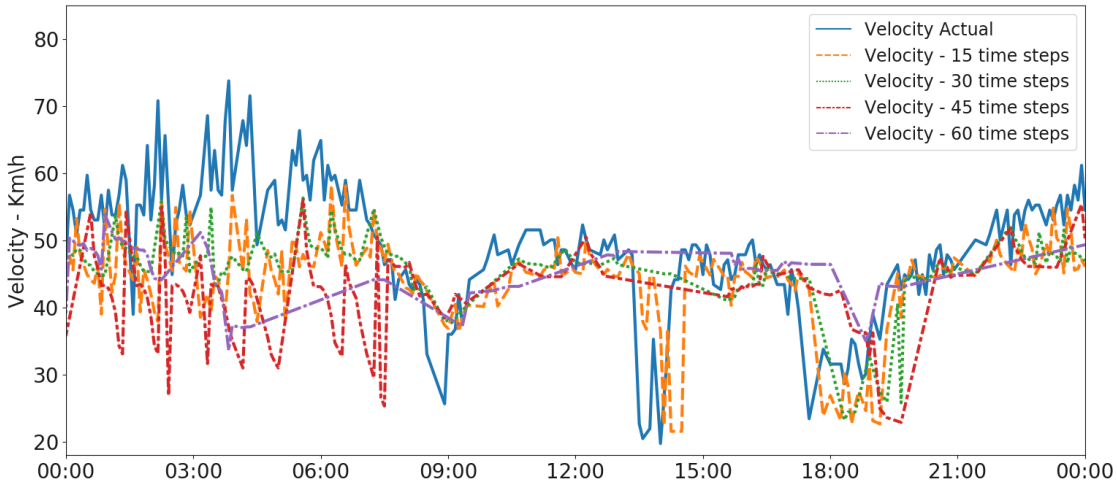

Figure 4.2: Results of RF models predicting traffic data.

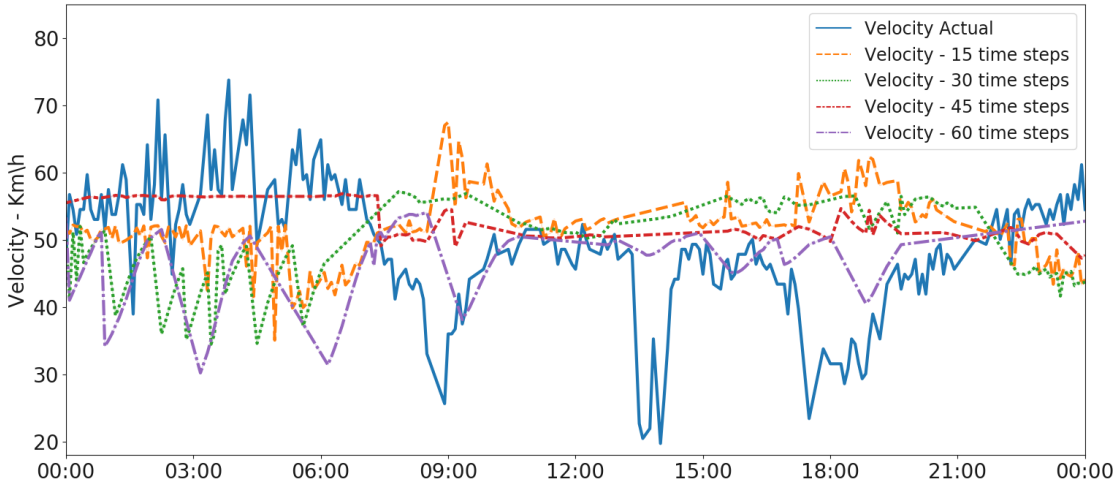

Figure 4.3: Results of LSTM models predicting traffic data.

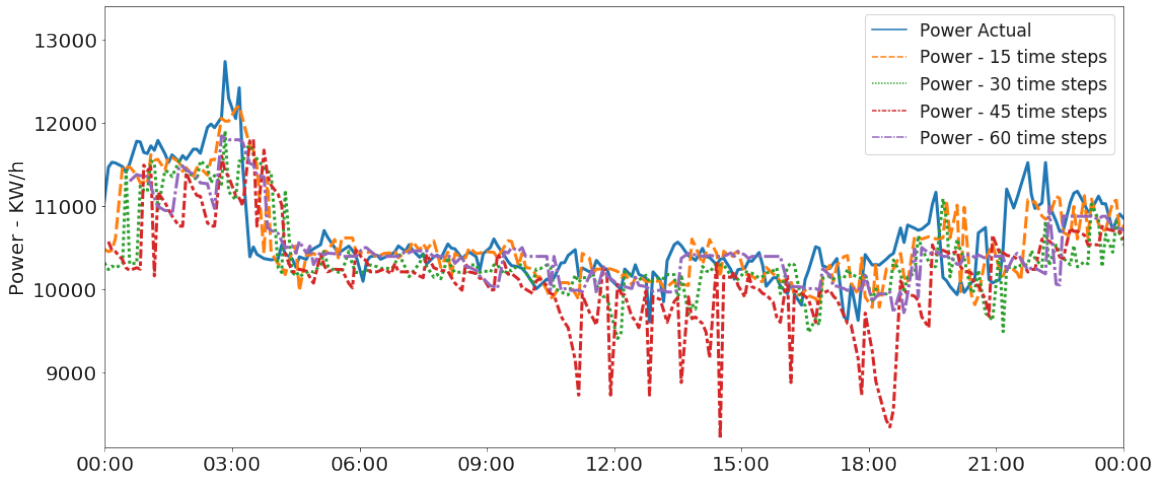

Figure 4.4: Results of RF models predicting energy data.

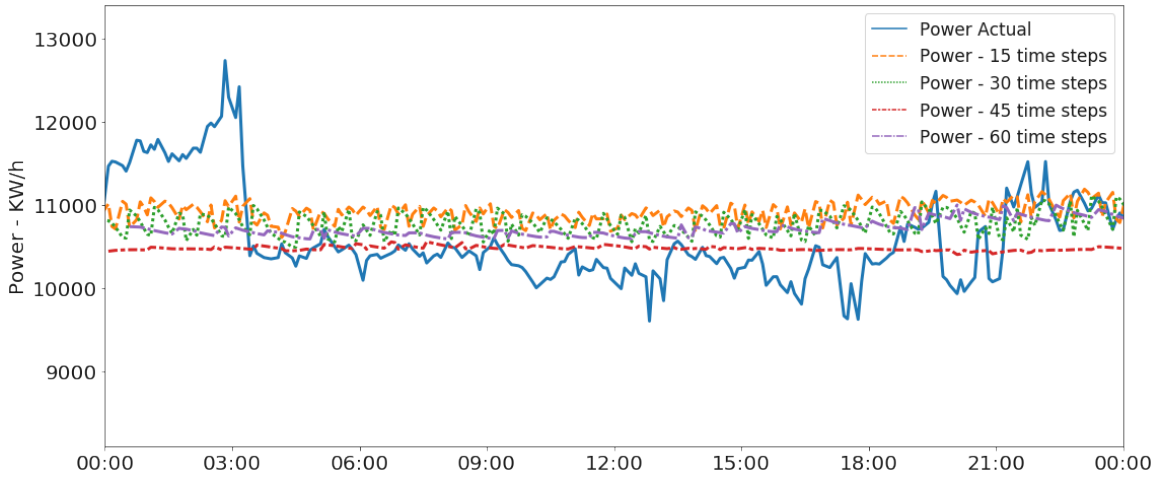

Figure 4.5: Results of LSTM models predicting energy data.

In the *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 retrained ML models that are deployed in the online ML workflows are automatically updated to the new model version.

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.

4.5 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.

- **Batch workflows**. Each workflow was composed of the two datasets described in Table 4.7. 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.

- **Workloads**. Table 4.9 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.

4.5.1 Results and Discussion

Figures 4.6, 4.7, 4.8, and 4.9 show the results of each workload execution.

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

Table 4.9: Description of workloads used in the elasticity evaluation.

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 utilizations returned to their original levels, releasing server resources.

The results in terms of CPU and memory utilization also demonstrated that the number of ML workflows running in parallel is limited by the number of resources available. For example, in Figure 4.9, the CPU allocation achieved more than 90% when 96 containers were executing in parallel. In the current design, the *ML4IoT* do not provide methods to queue the execution of ML workflows, when there are no resources available in the servers. This lack of resources management can affect the execution time of the ML workflows, when the demand for resources is higher than the amount of resources available.

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 provide an elastic solution that can support execution of multiple ML workflows in parallel. The obtained results also suggested that the proposed framework can manage real-world IoT data, by providing elasticity to execute 32 ML workflows in parallel, which were used to train 128 ML models simultaneously.

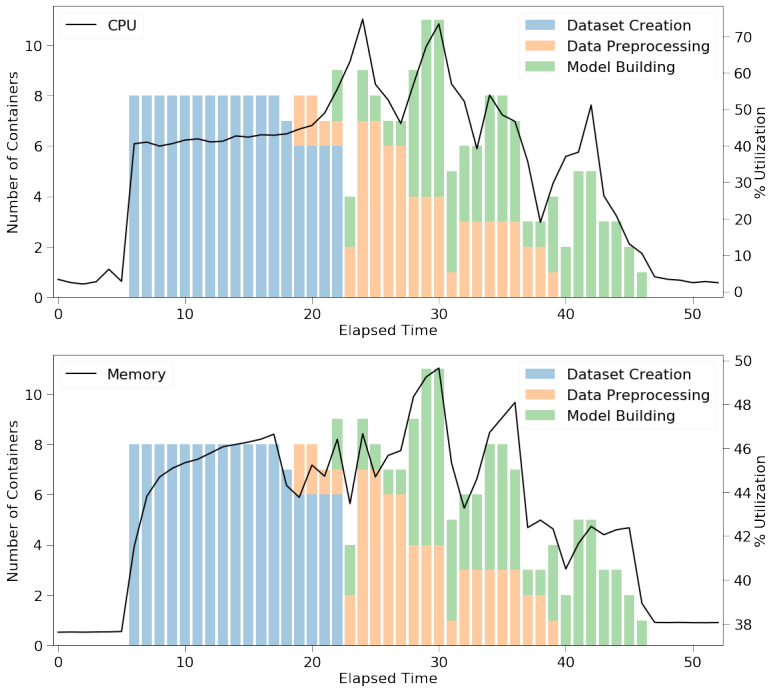

Figure 4.6: Workload 1 - Containers, CPU and memory allocation during parallel training of 16 ML models.

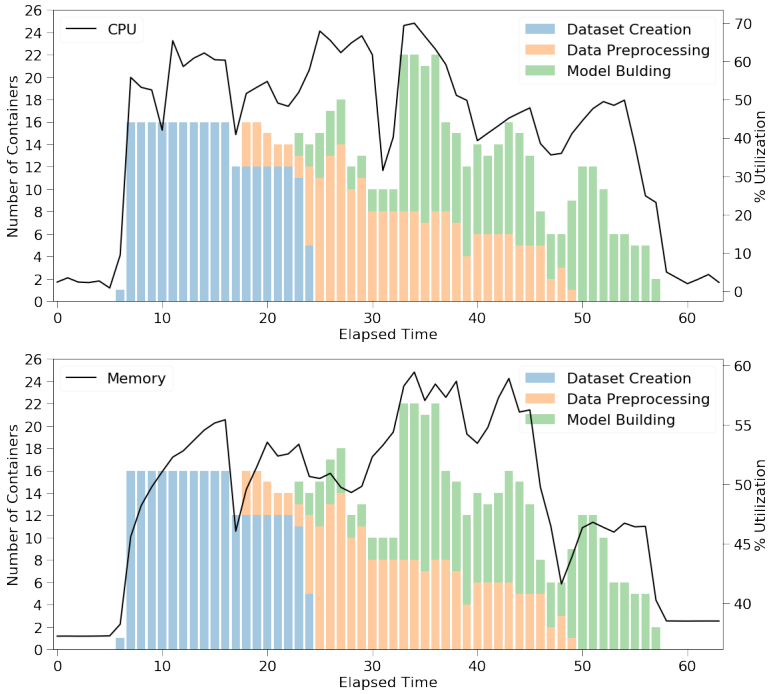

Figure 4.7: Workload 2 - Containers, CPU and memory allocation during parallel training of 32 ML models.

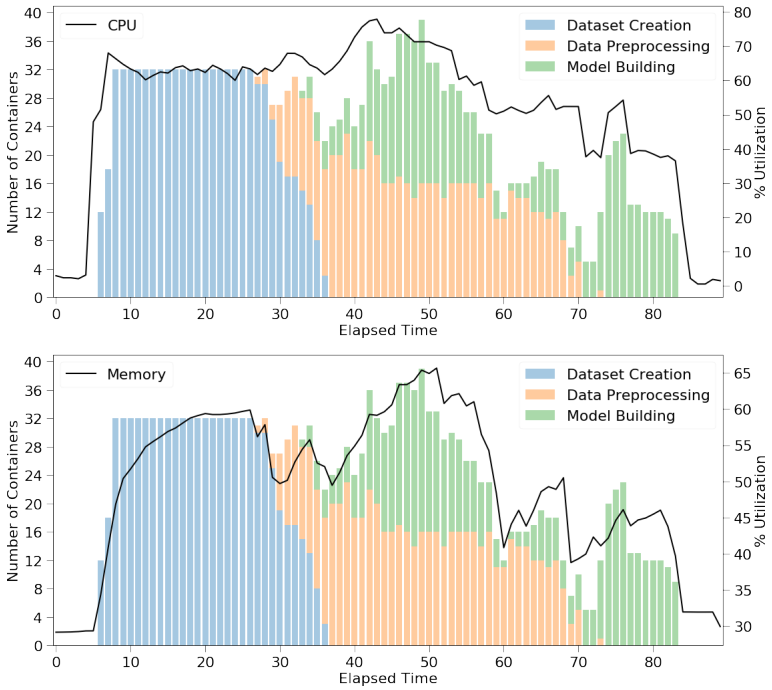

Figure 4.8: Workload 3 - Containers, CPU and memory allocation during parallel training of 64 ML models.

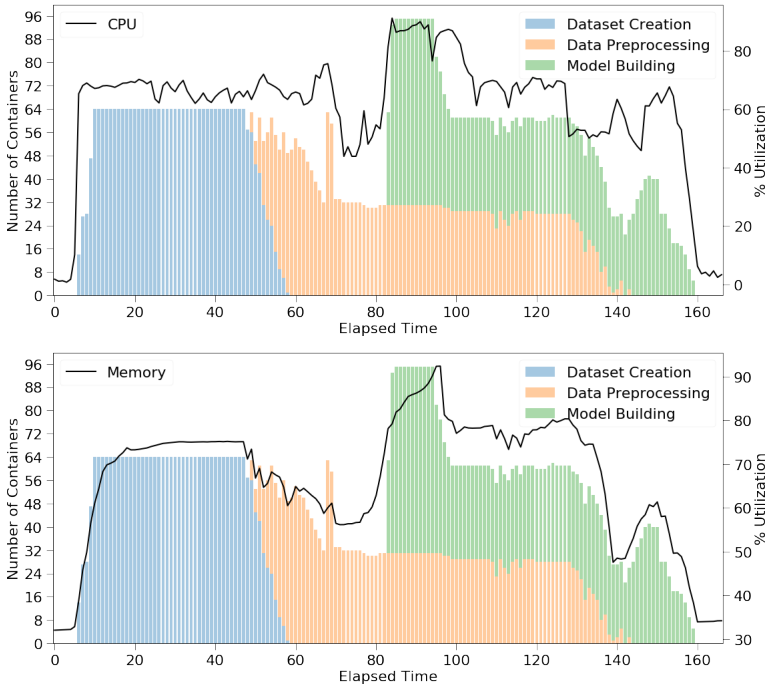

Figure 4.9: Workload 4 - Containers, CPU and memory allocation during parallel training of 128 ML models.

4.6 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, performance was evaluated by measuring the latency of rendering online predictions as 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 4.10 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.

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

Table 4.10: Description of workloads used in the performance evaluation.

4.6.1 Results and Discussion

The boxplot presented presented in Figure 4.10 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 selection of the online data, application of preprocessing tasks, and prediction of new values.

The median of the 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 4.10 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 4.10 and 4.11, the scatter points represent the latency measures in the 99^{th} percentile.

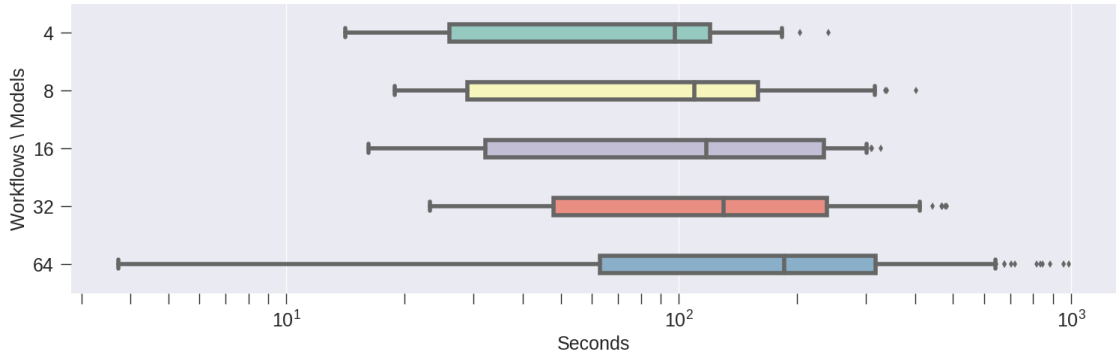

Figure 4.10: Latency of online ML workflows in different workloads.

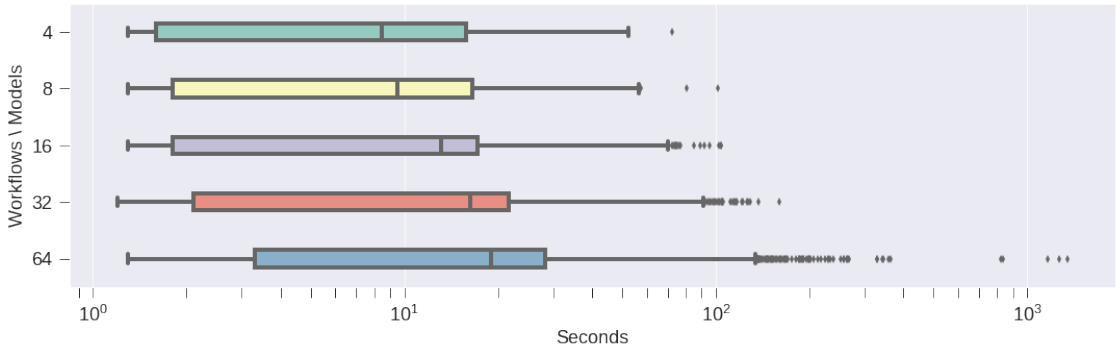

Figure 4.11: Latency time of model inference step during the execution of online ML workflows.

Figure 4.11 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 greater the number of online ML workflows running in parallel, the longer the model inference step takes, and the greater 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 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 (99^{th} percentile). Overall, results demonstrated that the performance of rendering online predictions is not affected when 64 models are deployed in parallel to infer new information using online IoT data. Also, the experiment has shown that the framework can manage the execution of multiple online ML workflows in parallel, and it can be

used to deploy trained ML models to render predictions using online IoT data.

4.7 Summary

In this chapter, the evaluation of the framework described in Chapters 3 was presented. Moreover, the implementation details of the experiments as well as the results were discussed. The orchestration capabilities of the framework were assessed in the first evaluation, which demonstrated that the framework could define ML workflows and automate their execution to train ML models, and use these models to infer online IoT data. In the elasticity and performance evaluations, experiments were conducted to demonstrate how the framework could deal with the execution of multiple ML workflows in parallel.

Chapter 5

Conclusions and Future Work

This chapter presents a concluding summary based on the contributions of the proposed Machine Learning Framework for IoT data (*ML4IoT*). Also, a description of possible future research involving *ML4IoT* and its components is provided.

5.1 Conclusions

This research proposed the Machine Learning Framework for IoT data ((*ML4IoT*)) to solve the challenges involved in the integration of big data enabling tools and ML frameworks to provide a unified platform to execute end-to-end ML workflows in IoT data. Its main goal is to provide orchestration services for the training and the inference of ML models on IoT data, which allows the automated execution of ML workflows on top of various big data tools and ML frameworks. The main contributions of this work are presented below:

- The proposed framework enables the definition and execution of end-to-end ML workflows using REST APIs. The definition of ML workflows using high-level APIs abstracts from users and developers the complexities of creating complex and repetitive code to integrate the numerous tools required to apply ML to IoT data. Two types of ML workflows can be created in this framework: batch and online ML workflows. These workflows

are composed of a set of configurations that define the sequential tasks and parameters required to train and deploy ML models to infer online IoT data.

- The *ML4IoT* was designed to use container-based components to provide a convenient mechanism for horizontally and independently scaling the *ML4IoT* to execute multiple ML workflows in parallel. The automated execution of the ML workflows is orchestrated by backend services provided by the *ML4IoT*, which uses containerized microservices to execute the tasks defined in the workflows. 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.

- To address the common production issues faced during the development of ML applications, the proposed framework used microservices architecture to bring flexibility, reusability, and extensibility to the framework. In the *ML4IoT*, ML frameworks and libraries used to build ML models can be added, replaced, or updated without affecting the other components of the framework. Also, splitting the design of ML workflows into small and specialized microservices facilitates the reuse of these software components. Lastly, it contributes to the extensibility of the ML4IoT, because when the design of ML workflows is divided into well-defined components, it creates natural points of extension for new functionalities.

To demonstrate the applicability of *ML4IoT* framework, a prototype system was built to perform three evaluations using two real-world IoT data. Experimental results have shown that the *ML4IoT* can simplify the definition and automate the execution of ML workflows that run on top of heterogeneous big data tools and ML frameworks. Also, results demonstrated the *ML4IoT* can manage the orchestration of ML workflows by providing scalability and elasticity to allow the execution of multiple ML workflows in parallel.

5.2 Future Work

This section presents several areas of future work that can be explored:

- **Automated machine learning.** Future work will consider extending the proposed framework to automate the selection and the tuning of ML models, which is a method known as Automatic ML (AutoML) [104]. Given the ever-increasing number of ML algorithms being developed, selecting the appropriate algorithm is an essential factor to achieve optimal performance during the training of ML models. Also, ML models often depend on hyper-parameters that require extensive fine-tuning. In this way, the framework can be extended to provide model customization by automatically fine-tuning ML models and choosing the ones that perform best. Automation of tuning and selection of ML models will allow the creation of high-performing ML workflows in IoT data.

- **Online Machine learning.** Most of IoT data are produced in real-time. Also, in some cases, there are no historical IoT data to train ML models using past data. Future work will explore the creation of ML models using just online IoT data. The online training of ML models also includes the adoption of methods to provide faster retrain of the models, which can help the model to keep our models accurate as the IoT data changes. This approach will expand the applicability of the framework and can help improve the overall efficiency of ML models on IoT data.

- ***ML-as-a-Service.*** Another interesting future work would be to explore the use of this framework as part of cloud computing services. The deployment of ML applications always needs to deal with conflicting priorities such as speed, uptime, and costs. In this way, the *ML4IoT* can be implemented in cloud services as *ML-as-a-Service* platform, which can bring benefits such as preventing downtime, optimizing data center costs, and reducing response latency. Also, the implementation of the *ML4IoT* can be explored in multi-cloud services to avoid *vendor lock-in* and take advantage of cloud providers price

competition.

- **Data Validation.** A future project is to implement a layer for data validation, which would provide automated methods for exploring and preprocessing IoT data. Because of big data characteristics of IoT data, in some situations, it is impossible to inspect and choose the best techniques to improve the data quality manually. Automating the data exploration and the selection of data preprocessing tasks can amplify the productivity of the process of using ML to IoT data.

- **Resources Management.** Future work will consider other ways to manage the use of computational resources. In the current design, the allocation of computational resources is configured programmatically for each task executed during the orchestration of the ML frameworks. One approach is to introduce the use of advanced algorithms or even ML to define the allocation of computation resources, which could provide better scalability and elasticity capabilities to the *ML4IoT*.

- **Extensibility.** In this study, the proposed framework was evaluated in two types of IoT data by using two different ML frameworks and one type of real-world problem (time-series prediction). Future work will validate the framework with a broad range of IoT data, new ML frameworks, and different real-world problems such as classification, anomaly detection, and so on.

Bibliography

[1] C. Nguyen, Y. Wang, and H. N. Nguyen, "Random forest classifier combined with feature selection for breast cancer diagnosis and prognostic," *Journal of Biomedical Science and Engineering*, vol. 6, no. 05, p. 551, 2013.

[2] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pp. 6645–6649, IEEE, 2013.

[3] S. B. Taieb, G. Bontempi, A. F. Atiya, and A. Sorjamaa, "A review and comparison of strategies for multi-step ahead time series forecasting based on the nn5 forecasting competition," *Expert systems with applications*, vol. 39, no. 8, pp. 7067–7083, 2012.

[4] C. Pahl, "Containerization and the paas cloud," *IEEE Cloud Computing*, vol. 2, no. 3, pp. 24–31, 2015.

[5] M. Hung, "Leading the iot [ONLINE]." `http://www.gartner.com/imagesrv/books/iot/iotEbook_digital.pdf`. Accessed: 2019-01-17.

[6] H. Arasteh, V. Hosseinnezhad, V. Loia, A. Tommasetti, O. Troisi, M. Shafie-Khah, and P. Siano, "Iot-based smart cities: a survey," in *Environment and Electrical Engineering (EEEIC), 2016 IEEE 16th International Conference on*, pp. 1–6, IEEE, 2016.

[7] B. L. R. Stojkoska and K. V. Trivodaliev, "A review of internet of things for smart home: Challenges and solutions," *Journal of Cleaner Production*, vol. 140, pp. 1454–1464, 2017.

[8] L. Li, K. Ota, and M. Dong, "When weather matters: Iot-based electrical load forecasting for smart grid," *IEEE Communications Magazine*, vol. 55, no. 10, pp. 46–51, 2017.

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

[10] D. E. O'Leary, "Big data, the internet of thingsand the internet of signs," *Intelligent Systems in Accounting, Finance and Management*, vol. 20, no. 1, pp. 53–65, 2013.

[11] G. M. D'silva, A. Khan, S. Bari, *et al.*, "Real-time processing of iot events with historic data using apache kafka and apache spark with dashing framework," in *Recent Trends in Electronics, Information & Communication Technology (RTEICT), 2017 2nd IEEE International Conference on*, pp. 1804–1809, IEEE, 2017.

[12] L. M. Pham, "A big data analytics framework for iot applications in the cloud," *VNU Journal of Science: Computer Science and Communication Engineering*, vol. 31, no. 2, 2016.

[13] A. Vera-Baquero and R. Colomo-Palacios, "Big-data analysis of process performance: A case study of smart cities," in *Big Data in Engineering Applications*, pp. 41–63, Springer, 2018.

[14] A. Pal and M. Kumar, "Pattern generation from event oriented sensor data using distributed sensor transaction model," in *Proceedings of the 4th Multidisciplinary International Social Networks Conference on ZZZ*, p. 36, ACM, 2017.

[15] P. M. Kumar and U. D. Gandhi, "A novel three-tier internet of things architecture with machine learning algorithm for early detection of heart diseases," *Computers & Electrical Engineering*, vol. 65, pp. 222–235, 2018.

[16] 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 Journal*, vol. 16, no. 2, pp. 485–497, 2016.

[17] S. Wu, L. Bao, Z. Zhu, F. Yi, and W. Chen, "Storage and retrieval of massive heterogeneous iot data based on hybrid storage," in *2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, pp. 2982–2987, IEEE, 2017.

[18] A. Mahgoub, S. Ganesh, F. Meyer, A. Grama, and S. Chaterji, "Suitability of nosql systemscassandra and scylladbfor iot workloads," in *Communication Systems and Networks (COMSNETS), 2017 9th International Conference on*, pp. 476–479, IEEE, 2017.

[19] S. Dharur and K. Swaminathan, "Efficient surveillance and monitoring using the elk stack for iot powered smart buildings," in *2018 2nd International Conference on Inventive Systems and Control (ICISC)*, IEEE, 2018.

[20] K. Grolinger, W. A. Higashino, A. Tiwari, and M. A. Capretz, "Data management in cloud environments: Nosql and newsql data stores," *Journal of Cloud Computing: advances, systems and applications*, vol. 2, no. 1, p. 22, 2013.

[21] M. M. Rathore, A. Ahmad, and A. Paul, "Iot-based smart city development using big data analytical approach," in *Automatica (ICA-ACCA), IEEE International Conference on*, pp. 1–8, IEEE, 2016.

[22] Y. N. Malek, A. Kharbouch, H. El Khoukhi, M. Bakhouya, V. De Florio, D. El Ouadghiri, S. Latre, and C. Blondia, "On the use of iot and big data technologies for real-time

monitoring and data processing," *Procedia Computer Science*, vol. 113, pp. 429–434, 2017.

[23] M. Falkenthal, U. Breitenbücher, K. Képes, F. Leymann, M. Zimmermann, M. Christ, J. Neuffer, N. Braun, and A. W. Kempa-Liehr, "Opentosca for the 4th industrial revolution: automating the provisioning of analytics tools based on apache flink," in *Proceedings of the 6th International Conference on the Internet of Things*, pp. 179–180, ACM, 2016.

[24] M. El Moulat, O. Debauche, S. Mahmoudi, L. A. Brahim, P. Manneback, and F. Lebeau, "Monitoring system using internet of things for potential landslides," *Procedia Computer Science*, vol. 134, pp. 26–34, 2018.

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

[26] J. Kwok and Y. Sun, "A smart iot-based irrigation system with automated plant recognition using deep learning," in *Proceedings of the 10th International Conference on Computer Modeling and Simulation*, pp. 87–91, ACM, 2018.

[27] M. Sewak and S. Singh, "Iot and distributed machine learning powered optimal state recommender solution," in *Internet of Things and Applications (IOTA), International Conference on*, pp. 101–106, IEEE, 2016.

[28] C. Ruiz, S. Pan, A. Sadde, H. Y. Noh, and P. Zhang, "Posepair: pairing iot devices through visual human pose analysis: demo abstract," in *Proceedings of the 17th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pp. 144–145, IEEE Press, 2018.

[29] J. A. C. Soto, M. Jentsch, D. Preuveneers, and E. Ilie-Zudor, "Ceml: Mixing and moving complex event processing and machine learning to the edge of the network for iot

applications," in *Proceedings of the 6th International Conference on the Internet of Things*, pp. 103–110, ACM, 2016.

[30] M. Mohammadi, A. Al-Fuqaha, M. Guizani, and J.-S. Oh, "Semisupervised deep reinforcement learning in support of iot and smart city services," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 624–635, 2018.

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

[32] Y. Cheng, F. C. Liu, S. Jing, W. Xu, and D. H. Chau, "Building big data processing and visualization pipeline through apache zeppelin," in *Proceedings of the Practice and Experience on Advanced Research Computing*, p. 57, ACM, 2018.

[33] 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 *Advances in neural information processing systems*, pp. 2503–2511, 2015.

[34] "Uber Michelangelo [ONLINE]." `https://eng.uber.com/michelangelo`. Accessed: 2019-01-17.

[35] "Airbnb Bighead [ONLINE]." `https://databricks.com/session/bighead-airbnbs-end-to-end-machine-learning-platform`. Accessed: 2019-01-17.

[36] "Netflix Meson [ONLINE]." `https://medium.com/netflix-techblog/meson-workflow-orchestration-for-netflix-recommendations-fc932625c1d9`. Accessed: 2019-01-17.

[37] D. Baylor, E. Breck, H.-T. Cheng, N. Fiedel, C. Y. Foo, Z. Haque, S. Haykal, M. Ispir, V. Jain, L. Koc, *et al.*, "Tfx: A tensorflow-based production-scale machine learning plat-

form," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1387–1395, ACM, 2017.

[38] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, *et al.*, "Applied machine learning at facebook: A datacenter infrastructure perspective," in *High Performance Computer Architecture (HPCA), 2018 IEEE International Symposium on*, pp. 620–629, IEEE, 2018.

[39] "Amazon AWS IoT Analytics [ONLINE]." `https://aws.amazon.com/iot-analytics`. Accessed: 2019-01-17.

[40] "Google Cloud IoT Core [ONLINE]." `https://cloud.google.com/iot-core`. Accessed: 2019-01-17.

[41] "Microsoft Azure IoT Edge [ONLINE]." `https://azure.microsoft.com/en-ca/services/iot-edge`. Accessed: 2019-01-17.

[42] N. Gondchawar and R. Kawitkar, "Iot based smart agriculture," *International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE)*, vol. 5, no. 6, pp. 177–181, 2016.

[43] J. Meehan, C. Aslantas, S. Zdonik, N. Tatbul, and J. Du, "Data ingestion for the connected world.," in *CIDR*, 2017.

[44] H. Cai, B. Xu, L. Jiang, and A. V. Vasilakos, "Iot-based big data storage systems in cloud computing: Perspectives and challenges," *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 75–87, 2017.

[45] S. Yang, "Iot stream processing and analytics in the fog," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 21–27, 2017.

[46] M. R. Bashir and A. Q. Gill, "Towards an iot big data analytics framework: Smart buildings systems," in *High Performance Computing and Communications; IEEE 14th*

International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2016 IEEE 18th International Conference on, pp. 1325–1332, IEEE, 2016.

[47] R. Young, S. Fallon, and P. Jacob, "Dynamic collaboration of centralized & edge processing for coordinated data management in an iot paradigm," in *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*, pp. 694–701, IEEE, 2018.

[48] J. Wan, S. Tang, D. Li, S. Wang, C. Liu, H. Abbas, and A. V. Vasilakos, "A manufacturing big data solution for active preventive maintenance," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 4, pp. 2039–2047, 2017.

[49] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of machine learning*. MIT press, 2018.

[50] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.

[51] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[52] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[53] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[54] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," in *Advances in neural information processing systems*, pp. 802–810, 2015.

[55] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang, *et al.*, "Ese: Efficient speech recognition engine with sparse lstm on fpga," in *Proceedings of*

the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp. 75–84, ACM, 2017.

[56] C. Finn and S. Levine, "Deep visual foresight for planning robot motion," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 2786–2793, IEEE, 2017.

[57] J. Lin and A. Kolcz, "Large-scale machine learning at twitter," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pp. 793–804, ACM, 2012.

[58] D. Georgakopoulos, M. Hornick, and A. Sheth, "An overview of workflow management: From process modeling to workflow automation infrastructure," *Distributed and parallel Databases*, vol. 3, no. 2, pp. 119–153, 1995.

[59] G. C. Tiao and R. S. Tsay, "Some advances in non-linear and adaptive modelling in time-series," *Journal of forecasting*, vol. 13, no. 2, pp. 109–131, 1994.

[60] A. Sorjamaa, J. Hao, N. Reyhani, Y. Ji, and A. Lendasse, "Methodology for long-term prediction of time series," *Neurocomputing*, vol. 70, no. 16-18, pp. 2861–2869, 2007.

[61] S. Newman, *Building microservices: designing fine-grained systems*. " O'Reilly Media, Inc.", 2015.

[62] R. Morabito, J. Kjällman, and M. Komu, "Hypervisors vs. lightweight virtualization: a performance comparison," in *Cloud Engineering (IC2E), 2015 IEEE International Conference on*, pp. 386–393, IEEE, 2015.

[63] M. Fazio, A. Celesti, R. Ranjan, C. Liu, L. Chen, and M. Villari, "Open issues in scheduling microservices in the cloud," *IEEE Cloud Computing*, vol. 3, no. 5, pp. 81–88, 2016.

[64] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. De Rose, "Performance evaluation of container-based virtualization for high performance computing environments," in *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*, pp. 233–240, IEEE, 2013.

[65] A. Sill, "The design and architecture of microservices," *IEEE Cloud Computing*, vol. 3, no. 5, pp. 76–80, 2016.

[66] 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," *Proceedings of the VLDB Endowment*, vol. 12, no. 2, pp. 128–140, 2018.

[67] 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 *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pp. 611–626, 2018.

[68] "ML.NET [ONLINE]." `https://dotnet.microsoft.com/apps/machinelearning-ai/ml-dotnet`. Accessed: 2019-01-17.

[69] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A low-latency online prediction serving system.," in *NSDI*, pp. 613–627, 2017.

[70] S. Zhao, M. Talasila, G. Jacobson, C. Borcea, S. A. Aftab, and J. F. Murray, "Packaging and sharing machine learning models via the acumos ai open platform," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 841–846, IEEE, 2018.

[71] "Apache Prediction I/O." `https://predictionio.apache.org/`. Accessed: 2019-01-17.

[72] "IBM Watson IoT Platform [ONLINE]." https://www.ibm.com/internet-of-things/spotlight/watson-iot-platform. Accessed: 2019-01-17.

[73] G. D. F. Morales and A. Bifet, "Samoa: scalable advanced massive online analysis.," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 149–153, 2015.

[74] S. Hido, S. Tokui, and S. Oda, "Jubatus: An open source platform for distributed online machine learning," in *NIPS 2013 Workshop on Big Learning, Lake Tahoe*, 2013.

[75] "Apache Spark." https://spark.apache.org. Accessed: 2019-01-17.

[76] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[77] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, *et al.*, "Apache spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[78] 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 Generation Computer Systems*, vol. 92, pp. 225–240, 2019.

[79] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[80] S.-F. Chou, H.-W. Yen, and A.-C. Pang, "A rem-enabled diagnostic framework in cellular-based iot networks," *IEEE Internet of Things Journal*, 2019.

[81] M. Shen, X. Tang, L. Zhu, X. Du, and M. Guizani, "Privacy-preserving support vector machine training over blockchain-based encrypted iot data in smart cities," *IEEE Internet of Things Journal*, 2019.

[82] P. Sun, J. Li, M. Z. A. Bhuiyan, L. Wang, and B. Li, "Modeling and clustering attacker activities in iot through machine learning techniques," *Information Sciences*, 2018.

[83] L. Spitzner, *Honeypots: tracking hackers*, vol. 1. Addison-Wesley Reading, 2003.

[84] H. Haskamp, M. Meyer, R. Möllmann, F. Orth, and A. W. Colombo, "Benchmarking of existing opc ua implementations for industrie 4.0-compliant digitalization solutions," in *Proc. of the 15th IEEE Intern. Conf. on Ind. Info.(INDIN), Emden, Germany*, pp. 589–594, 2017.

[85] A. Javed, H. Larijani, and A. Wixted, "Improving energy consumption of a commercial building with iot and machine learning," *IT Professional*, vol. 20, no. 5, pp. 30–38, 2018.

[86] 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*, 2019.

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

[88] N. Marz and J. Warren, *Big Data: Principles and best practices of scalable realtime data systems*. Manning Publications Co., 2015.

[89] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, *et al.*, "Storm@ twitter," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pp. 147–156, ACM, 2014.

[90] M. Strohbach, H. Ziekow, V. Gazis, and N. Akiva, "Towards a big data analytics framework for iot and smart city applications," in *Modeling and processing for next-generation big-data technologies*, pp. 257–282, Springer, 2015.

[91] A. Lheureux, K. Grolinger, H. F. Elyamany, and M. A. Capretz, "Machine learning with big data: Challenges and approaches," *IEEE Access*, vol. 5, no. 5, pp. 777–797, 2017.

[92] N. Mishra, C.-C. Lin, and H.-T. Chang, "A cognitive adopted framework for iot big-data management and knowledge discovery prospective," *International Journal of Distributed Sensor Networks*, vol. 11, no. 10, p. 718390, 2015.

[93] O. B. Sezer, E. Dogdu, M. Ozbayoglu, and A. Onal, "An extended iot framework with semantics, big data, and analytics," in *Big Data (Big Data), 2016 IEEE International Conference on*, pp. 1849–1856, IEEE, 2016.

[94] G. Wu, S. Talwar, K. Johnsson, N. Himayat, and K. D. Johnson, "M2m: From mobile to embedded internet," *IEEE Communications Magazine*, vol. 49, no. 4, 2011.

[95] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: a system for large-scale machine learning.," in *OSDI*, vol. 16, pp. 265–283, 2016.

[96] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, *et al.*, "Mllib: Machine learning in apache spark," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235–1241, 2016.

[97] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, 2011.

[98] Y. Shichkina, M. Kupriyanov, and S. Moldachev, "Application of docker swarm cluster for testing programs, developed for system of devices within paradigm of internet of things," in *Journal of Physics: Conference Series*, vol. 1015, p. 032129, IOP Publishing, 2018.

[99] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[100] J. Kreps, N. Narkhede, J. Rao, *et al.*, "Kafka: A distributed messaging system for log processing," in *Proceedings of the NetDB*, pp. 1–7, 2011.

[101] "Apache hadoop [ONLINE]." `https://hadoop.apache.org/`. Accessed: 2019-01-17.

[102] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "Devops," *IEEE Software*, vol. 33, no. 3, pp. 94–100, 2016.

[103] "Madrid Council Traffic Data." `http://informo.munimadrid.es/informo/tmadrid/pm.xml`. Accessed: 2019-01-17.

[104] C. Wong, N. Houlsby, Y. Lu, and A. Gesmundo, "Transfer learning with neural automl," in *Advances in Neural Information Processing Systems*, pp. 8366–8375, 2018.

Curriculum Vitae

Name:	José Miguel Alves
Post-Secondary Education and Degrees	Western University London, ON - Canada MESc in Software Engineering 2017 - 2019
	University of São Paulo São Carlos, SP - Brazil BSc in Information Systems 2005 - 2010
Related Work Experience:	Teaching Assistant Western University 2017 - 2019
	Data Engineer and Data Scientist Ícaro Technologies (Brazil) 2013 - 2017
	Software Engineer Ícaro Technologies (Brazil) 2010 - 2013