ML-FaaS: Toward Exploiting the Serverless Paradigm to Facilitate Machine Learning Functions as a Service

Efterpi Paraskevoulakou and Dimosthenis Kyriazis

***Abstract*—Serverless computing has emerged as a revolutionary model that enables the deployment of applications and services by raising the level of abstraction from the underline resources. Its main functionality is enlightened by the notion of Function-as-a-Service (FaaS) as the core means to realize efficient serverless offerings. Following the shift from traditional architectures to microservices - by attaining flexibility, productivity, portability, and performance in industrial-scale IT projects, the serverless model introduces even more fine- grained services, named "nanoservices", which facilitate required scalability by abstracting the deployment and management of the infrastructure resources. On the application space, advances in big data analysis contribute towards extracting actionable knowledge in various application domains. In this context, approaches for big data analysis aim at exploiting the added value of serverless architectures. To this end, we are presenting an extendable and generalized approach for facilitating the provision of Machine Learning Functions-as-a-Service (MLFaaS). The proposed approach outstrips the classical atomic and standard isolated services by facilitating composite services, i.e., workflows/pipelines of ML tasks, thus enabling the realization of the complete data path functions as required by data scientists. We demonstrate the operation of the proposed approach by modeling a real-world analytics scenario as an ML workflow pipeline and evaluate its performance in terms of performance. Furthermore, we address the challenge of utilizing a function oriented service template recommendation system, by expanding the serverless functional boundaries towards a holistic Quality-of-Service (QoS)-aware service function selection approach based on Artificial Intelligence techniques. These techniques propose the optimal number of functions to be implemented in a pipeline by exploiting the importance of response time as the primary key of the application's performance.**

***Index Terms*—Artificial intelligence, information management, function as a service, machine learning, serverless computing, service management.**

I. Introduction

THE 21ST Century is the reflection of the big data era since the available data have been exponentially increased within a short time frame [1]. The necessity for comprehension, formulation and manipulation of heterogeneous datasets forced data scientists to explore, adapt, and propose various methods and techniques to exploit the valuable, actionable information that arises from the data. To this end, based on the nature and volume of data, the analysts apply various machine learning (ML) and artificial intelligence (AI) algorithms towards the creation of a production-oriented model that fulfills their goals. However, in the scope of managing the underlying infrastructure resources for the execution of ML and AI algorithms, the scale and complexity of managing the resources require additional efforts from the data analysts, which inhibits both their productivity and decisiveness. Furthermore, the management of infrastructure resources (e.g., servers, memory, allocation, number of CPUs). Therefore, an even higher level of abstraction is required, which would allow application developers and data scientists to focus mainly on the business logic and the AI/ ML algorithms, rather than dealing with the allocation and the runtime adaptation of resources. This higher level of abstraction is facilitated by the so-called serverless architectures, which have emerged as a new paradigm for deploying applications and services [2]. It represents an evolution of cloud programming models, abstractions, and platforms and is a testament to the maturity and wide ado tion of cloud technologies providing benefits that include a much more comfortable development pipeline with codebases abstracted away from architectural complexities [3]. One of the most important benefits of the serverless model is that the runtime allocates resources as events arrive, avoiding the need for costly pre-allocated or dedicated hardware [4], which in contrast with traditional architectures and cloud infrastructure discrete products, reduces the cost for the user/developer.

By converging to the core of the serverless model, function-as- a-service (FaaS) showcases the compelling paradigm for the deployment of applications, primarily due to the recent shift of enterprise application architectures to containers and microservices. FaaS allows developers to implement their applications as "Functions" - a set of "nanoservices"; Function is the primitive component of the serverless computing model since the entire business logic of an application is stored within it (i.e., stand-alone function or even a chain of functions). Furthermore, it has stateless nature since the "state-keeping" mode dictates the necessity to synchronize the state between function invocations and when the load increases, the "state-maintenance" infrastructure limits the ability for growth [5]. However, the serverless functionality poses specific challenges and limitations: the runtime resource requirements

Manuscript received 29 September 2021; revised 16 March 2022, 19 July 2022, and 4 November 2022; accepted 13 January 2023. Date of publication 25 January 2023; date of current version 9 October 2023. This work was supported in part by the University of Piraeus Research Center. The associate editor coordinating the review of this article and approving it for publication was B. Martini. *(Corresponding author: Efterpi Paraskevoulakou.)*

The authors are with the Department of Digital Systems, University of Piraeus, 18534 Piraeus, Greece (e-mail: e.paraskevoulakou@unipi.gr).

Digital Object Identifier 10.1109/TNSM.2023.3239672

of the serverless code, the number of concurrent requests, the maximum memory and CPU resources available to a function invocation, and the maximum memory size. These challenges and limitations are further highlighted in the case of AI/ML services given the respective data sizes, the data flows across functions in function chains, and the codebase size (through the utilization of various libraries for analytics).

This paper enables the design and realization of a chain/pipeline of ML functions by introducing an approach for extending functions through attached containers. This approach overcomes the serverless environments constraints and enables the provision of ML services through FaaS offerings. The current manuscript introduces an end-to-end approach for building, serving, and optimizing a ML/DL serverless pipeline (i.e., chain of functions).

The approach proposes an extended container-based functionality in order to deal with challenges and restrictions posed by serverless architectures and environments, ensuring that the serverless model is well-suited for data analytics and ML/DL tasks - even if not initially designed for heavy workload tasks as the ML/DL ones.

Additionally, the deployment of a data analytics and ML/DL oriented application as a pipeline of serverless functions, has highlighted additional performance-related challenges in the scope of latency and overall execution time. To address these performance-related challenges, the proposed approach enables the automated analysis of a serverless application in terms of its composition of functions. The latter is feasible through an AI-based technique that recommends the optimal number of functions that compose the entire application, with the objective of end-to-end performance improvement.

The remainder of this manuscript is organized as follows. Section II provides an overview of related work. Section III introduces the proposed approach towards the realization of a performant serverless data analytics pipeline, while Section IV presents the functionality of our prototype named "Function partition Regulator" an approach for recommending the proper number of functions for creating the developer's service achieving the optimal response time. Section V includes the experimental setup and the respective evaluation outcomes with a discussion of findings and limitations presents. Section VI concludes the manuscript including a discussion regarding the future research directions and next steps.

II. Related Work

Both serverless computing and machine learning/deep learning (DL) gained a colossal reputation and became a trend. The cloud vendors in order to maintain their competitive position in the global market and address the datacenter management resource problem -since by the hosting cloud applications consume huge amounts of electrical energy, leading to high operational costs [6]– have invented a new state of the art models and researchers -by taking advantage from each innovation's benefits – develop novel solutions to combine evolution cloud computing services and models with machine learning knowledge. Thus, various solutions emerge to enable the provision of FaaS solutions to exploit the complete data science path. Researchers [7] have exposed machine learning as a service focusing on the user behavior analysis over varied demographics in a serverless manner with the support of Watson Machine Learning API [8] on IBM Cloud due to the benefit that the developer must only be concerned with the data analysis/processing/evaluation and the service usage. Nonetheless, the serverless model -due to its benefits- fits efficiently in laborious data-intensive applications (i.e., map-reduce jobs). While authors [9] exploit serverless architecture for executing MapReduce jobs using AWS Lambda along with Amazon S3 [10] as the storage backend, reflecting their usage and suitability of Lambda functions as a platform for the execution of high throughput computing jobs.

According to the results, they have recognized that AWS Lambda provides a convenient computing platform for general purpose applications that fit within the constraints of the service. However, it exhibits an in homogeneous performance behavior that may jeopardize adoption for tightly coupled computing jobs. Furthermore, the research community insisted on serverless development frameworks for handling ML workloads since serverless suggests entirely different interoperability to achieve scalability without scheduling or monitoring the resources needed for processing workloads. PyWren [11] constitutes a map-reduce framework specialized for serverless architectures; it serializes the implemented python source code and sends it to AWS Lambda functions for massively parallel execution using Amazon S3 as an intermediate level returns the results to the user. This proposal overcomes the serverless mainly in machine learning problems, like the training phase, because PyWren is accompanied by a mannerism that functionally uses external storage for intermediate computation results, overcoming the stateless mode delicately. Same notion follows the researchers in [12] by using the serverless technology for achieving the distributed ML training avoiding operational costs. The aforementioned work approaches a Federated Learning-oriented (FL) manner by using concrete functions as distributed workers and a major function as the FL server, moreover due to the lack of functions stateless nature, the authors-researchers are using external storage for result retention and aggregation. Currently, most machine learning training jobs are being executed parallelly since many training samples need to be processed by different workers in parallel. In combination with cloud storage, data parallelism joins among the serverless architecture naturally, in which we can launch several stateless functions, each accessing a different batch of data, without managing and maintaining any servers. Researchers explore, analyze, and benchmark the serverless model along with ML frameworks. Authors in [13] present an identical approach by proposing SIREN. This framework constitutes an asynchronous distributed machine learning framework based on the emerging serverless architecture, with which stateless functions that can be executed in the cloud avoid the complexity of building and maintaining virtual machine infrastructures. SIREN framework can achieve a higher level of parallelism and elasticity by using a swarm of stateless AWS Lambda functions, each of them working on a different batch of data, while significantly reducing system configuration expenses; also, according to their

extensive experimental results, they proved that their prototype could reduce ML model training's phase time by up to 44%, as compared to traditional machine learning benchmarks, at the same cost.

Authors of [14] intending to approach the resource management decision problem efficiently concerning DL services that count on dynamic workloads, are proposing a framework based on the serverless model, which hosts containerized, pretrained deep learning models for predictive analytics in the cloud environment while decreasing hosting costs. According to their proposal, their work constitutes an efficient, data-driven resource allocator, which estimates the resource requirements ahead of time by exploiting the incoming requests' variable patterns and a forecast-aware scheduling mechanism, which improves resource utilization while preventing physical resource exhaustion.

The challenge regarding the execution of complex and costly ML workflows in conjunction with the resource cost reduction has triggered the Authors of [15] to design a novel serverless framework that is specialized in the agile deployment and scheduling of data analytics processes, live-streaming applications, ML-as-a-service for inference tasks, and visualization tools proving that the serverless architecture is consolidating as a promising and innovative tool within the cloud- edge environments. The proposed platform is accompanied by its native domain-specific modeling language (DSML) and provides a user interface with higher-level abstractions for direct interaction with the ML developers and deployers. Also, the developer/ML practitioner, by using the DSML, can create and evaluate the constructed ML models using existing well-known ML libraries and frameworks. Finally, yet importantly, due to the fact that the data are growing exponentially within seconds and milliseconds, and because of the limited space that traditional cloud datacenters provide, serverless has also emerged in edge computing paths by leveraging the development of edge AI workflows. Authors of [16], are going beyond edge-AI applications by proposing a serverless platform for building and operating edge AI applications. Moreover, they analyzed edge AI use cases to illustrate the challenges in building and operating AI applications in an edge cloud scenario. By elevating concepts from AI lifecycle management into the established serverless model, the development of edge AI workflow functions constitutes a smoother procedure. Towards Edge AI, another framework has been proposed by authors [17], called STOIC, a system for executing distributed machine learning applications in IoT (edge and cloud) settings based on serverless architecture. Unquestionably, serverless computing draws the attention of time-sensitive smart applications since its impact provides a flexible solution to monitor the valuable computing resources for edge computing. Even though serverless came to change the application deployment fashion completely by reducing operational costs, we can still observe the realization of ML-data analytics service composition with traditional manners; in [18] the authors have illustrated a real-world scenario in which reflects automated service composition using an -out of serverless bounds-algorithm named MLS-PLAN based in hierarchical service planning, which is evaluated by a black-box objective function. Nevertheless, the serverless paradigm emerges as a promising solution for implementing the intensive Internet of Medical Things (IoMT). Healthcare services are using sensors to produce data for consumption; thus, a VM-based service approach is less efficient because it causes wasting of computational resources.

Therefore, the authors of [19] present a serverless function as a service-based system charged only when the applications are utilized instead of renting costs for an entire execution system. According to their simulation results, their proposed system minimizes the IoMT application cost and confers improved resource utilization. Indeed, the serverless model presents a competitive candidate compared to traditional architectures in even if it is not designed to manage scenarios that include heavy AI workloads, stream analysis, or scenarios performing speedup look up of large volumes of data located in stateful candidate compared to traditional architectures in even if it is not designed to manage scenarios that include heavy AI databases. The capabilities and strengths of the serverless model, such as the effectiveness in terms of monetary cost, in terms of system and monitoring maintenance, exaggerates the model's weakness and drives research to address the challenges related to various use-cases and to propose agile deployment solutions even for heavy and complex tasks. The authors of [20] designed a set of serverless code optimization techniques that can be used to transform production AI workloads on big data so that they can be deployed in a serverless architecture. Specifically, they designed a suite that deals with slimming and packing important AI libraries-frameworks used in the codebase by isolating sections of the necessary library into each individual function's environment, supporting interoperability procedures for training and inference ML/DL models, loading – dynamically- pre-trained Machine Learning Models from cloud storage into local temporary storage and improving the speed related to the large volume data-lookup in order to deal with the function's lifetime. The authors of [21] addressed the same AI-workload and data preprocessing flow challenge for serverless models, which introduces an open-source platform to support serverless model for scientific data-processing workflow-based applications to process data captured at the edge. Specifically, this open-source platform supports the execution of workload-based data- processing applications packaged as Docker containers that can elastically provision resources from on-premises clouds and perform automated bursting into a public cloud using an event-driven serverless approach. Although serverless, the approach can even handle such cases as complex data- preprocessing work-flow procedures, the issue concerning the retention and handling large Deep Learning models within serverless architecture is addressed by the authors of [22] since the deployment of machine and deep learning- based applications for both training models and or inference on serverless architectures confers significant challenges. Notably, using vendor lock-in services (AWS Lambda, AWS Elastic File System - EFL-), the researchers were able to deploy applications based on large AI models and formed a performance comparison of FaaS with on-premises deployment of their case study AI application.

TABLE I
DATA ANALYTICS/ML-DL STEPS AS PART OF THE ONLINE PHASE

FaaS pipeline	Data cleaning-imputation	Data transformation	Label Encoding	Data Normalization/ Standardization	Pretrained ML-DL model adaptation and prediction
2 functions	$1^{st}\ f_x$	$1^{st}\ f_x$	$1^{st}\ f_x$	$1^{st}\ f_x$	$2^{nd}\ f_x$
3 functions	$1^{st}\ f_x$	$1^{st}\ f_x$	$2^{nd}\ f_x$	$2^{nd}\ f_x$	$3^{rd}\ f_x$
4 functions	$1^{st}\ f_x$	$2^{nd}\ f_x$	$3^{rd}\ f_x$	$3^{rd}\ f_x$	$4^{th}\ f_x$
5 functions	$1^{st}\ f_x$	$2^{nd}\ f_x$	$3^{rd}\ f_x$	$4^{th}\ f_x$	$5^{th}\ f_x$

Nevertheless, all the aforementioned fruitful works introduce flexible solutions for performing and managing independent tasks and analytics processes by exploiting the serverless offerings as concrete functions and eliminating the architecture's stateless nature by using external persistence storage for storing both intermediate results and output data. Our approach moves beyond the typical single isolated function towards implementing flows of interconnected functions with dependencies exploiting serverless utilities and establishing that stateless functions can reciprocate to any ML task, not at an individual function level but in several connected functions [23].

III. ML-FAAS PROPOSED APPROACH

This section introduces an ML-FaaS application that highlights the challenges of potential heavy workloads due to the data processing tasks demands as well as the ML/DL training and inference. Our work tackles these challenges by bisecting the data and ML/DL procedures across the offline and online phases.

The offline phase includes the data preparation steps (i.e., data quality assessment, data cleaning, transformation, and elimination) and the ML/DL training according to the preprocessed data. The ML/DL inference procedure is supported by the online phase towards delivering the predictions against the new unseen data that the user provides through the serverless pipeline. More specifically, the designed workflow follows the ML/DL lifecycle in terms of the corresponding steps/tasks to be performed: (i) cleaning, (ii) transformation (feature elimination), (iii) label encoding, (iv) normalization/transformation, (v) ML/DL training and inference.

Based on these distinct steps, the respective outcomes have been stored into the corresponding functions in order to be tested in new unseen data. It should be noted that this workflow composition is flexible and can be updated or extended based on the steps in different ML/DL pipelines (e.g., in case of introducing data filtering in the overall workflow through a filtering function). Thus, the presented approach tackles both the offline and the online phase, since several of its core components (e.g., feature elimination, normalization, ML/DL training, etc) are realized as part of the offline phase, while ML/DL inference is realized as part of the online phase.

Our work is utilizing and extending the Apache OpenWhisk Serverless event-driven serverless platform [24], a flexible platform providing various functionalities for service management across cloud/fog/edge environments. We have selected to exploit the capabilities and functionalities of Apache OpenWhisk because it provides all the necessary components in order to support event-driven serverless and scalable functions, flexibility in terms of agile deployment and efficient configuration without the necessity of additional CSP products such as the API Gateway, the cloud-specified Container Registry, or the topology-based workflow configuration files.

Commercial solutions that deliver such serverless functionality require additional efforts (i.e., AWS-step-functions demand con- figuration with Amazon State Machine, AWS API Gateway, AWS ECR) from the developer's perspective that exceeds the developing code boundaries and minimizes the main scope and concept of the serverless paradigm, i.e., raising the abstraction level. Considering that OpenWhisk provides the functionality to establish a chain of functions, the proposed approach is oriented in realizing a custom-made chain of services (custom functions in terms of being generalized and reflecting the needs of different analytics tasks) that in each one of them is injecting a specific assignment for accomplishment (according to developer's business logic).

More than that, it should be noted that even though it is feasible to compose workflows of functions across different CSPs, it will not be efficient in terms of higher-than-expected costs, overall application performance and data transfer across the cloud providers. Figure 1 depicts the realization of the pipeline of functions in a serverless ecosystem.

A. Offline Phase

The offline phase showcases a real-world data analytics and machine learning scenario. It relates to the development of an ML classifier, trained with historical data and aiming at identifying fraud e-Commerce transactions. The selected dataset has been acquired from VESTA Corporation and IEEE Computational Intelligence Society (IEEE-CIS-) [25], presenting real labeled fraud and authorized e-commerce transactions. It consists of 600.000 records and includes 435 anonymized unique features. Since we use a binary classification problem as an example case, we exploit the series of typical steps related to data analytics, including exploratory data analysis, missing values imputation, feature engineering, and data standardization. Based on the overall aim of the offline phase, Figure 2 presents the typical steps included in the overall use-case approach. Escalating the necessity to extract dataset insights, the data exploitation task is essential for a data scientist to overview the selected dataset. The Exploratory Data Analysis (EDA) method can be applied as it can serve a brief overview of the dataset's information and discover hidden patterns. More specifically, meaningful insights derived from EDA implementation are presented in Figure 3, which depicts the abnormal distribution of the original dataset; most of the entire dataset's transactions are non-fraud, and in particular, only 3, 5% of the entire observations constitute fraud transactions. The visualization highlights that the usage of the dataframe -as it is- would raise many errors for the predictive models and analysis due to a potential overfit since the models would "assume" that most of the transactions are authorized. Figure 4 reveals that the most fraudulent transactions occurred

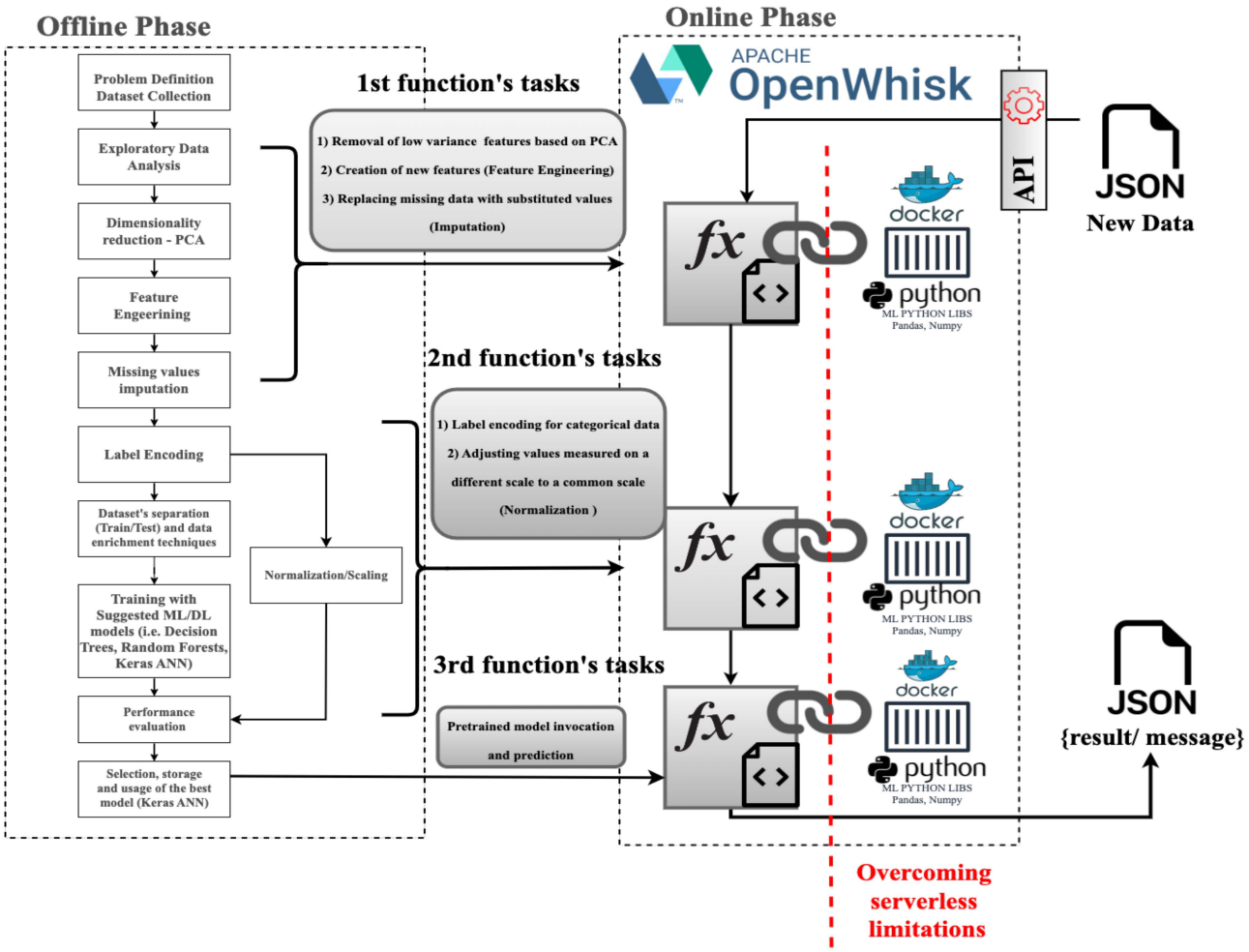


Fig. 1. Two-phase proposed hybrid ML-FaaS approach.

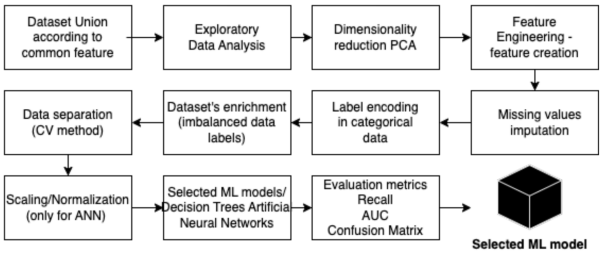


Fig. 2. Proposed data analytics exploitation path.

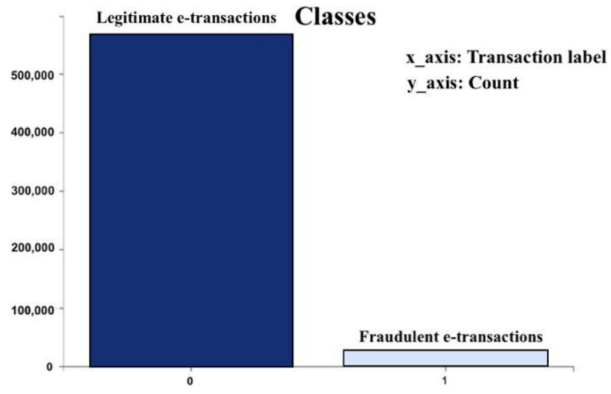


Fig. 3. Imbalanced dataset's observations.

on Friday, Saturday, and Sunday. This pattern is reasonable due to the fact that the perpetrators/digital thieves usually commit fraud at weekends because it is more difficult for the bank sector and the law enforcement agencies to react fast and adequately.

Naturally, data visualization can also provide insights about the potential existence of missing values within a dataset to inform the data scientist about the phenomenon mentioned above and to help build more effective data models and pipelines. Figure 5 depicts a nullity matrix for visualizing the patterns in data completion [26]. Specifically, the matrix provides a color fill for each column, when data is present the plot is shaded in grey, and when it is missing the plot is displayed in white. Therefore, we conclude that the selected dataset includes a high percentage of missing values, which influences the training of ML/DL models, and specific data mining techniques related to missing values' imputation should be applied in order to tackle the target.

1) Data Mining Advances: With the advance of the technology incorporated in various domains, a data ocean has been revealed, thus data mining can be used in many different domains to both predict and discover trends and hidden

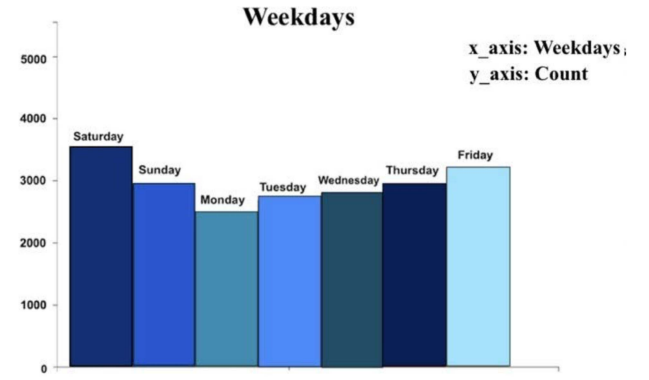


Fig. 4. Feature engineering for revealing hidden patterns.

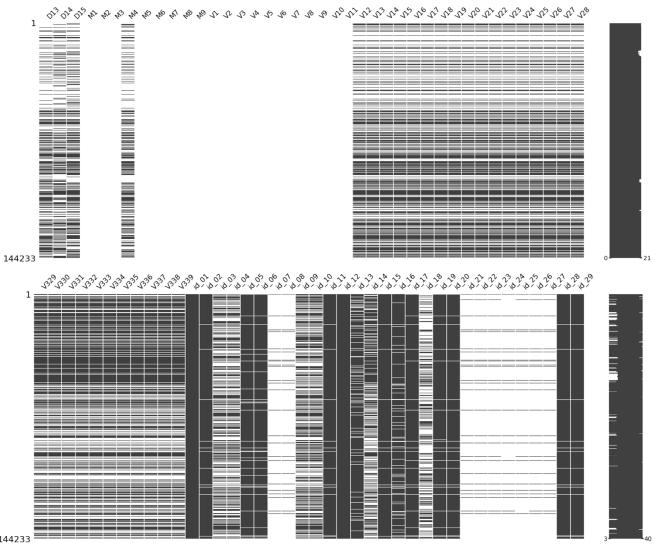


Fig. 5. Matrix of missing values for a sample of dataset's features.

patterns. In our case study, we apply various data mining techniques such as imputation of data missing values, dimensionality reduction, feature engineering, data encoding, and data augmentation.

In the context of missing values, it is essential to understand and manage data successfully, using the most suitable data imputation strategy to produce the most efficient data pattern. Researchers have proposed a plethora of strategies [27], [28], [29] that allow the analyst/developer to experiment and examine which of them effectively suits the selected dataset to provide the best performance for the selected ML model. Other than that, in several data-analysis cases, the abundant quantity of features ascribes a high dimensional dataset, which makes it challenging to plot it effectively, more than that, a high dimensional dataset may contain correlated features adding extra and useless information for the dataset, and finally and most importantly it raises complexity to the ML models. Regarding our case, the necessity for feature extraction is emerging, and therefore, the PCA algorithm [30] is applied, producing the result that 53 components retain the most significant percentage of variance (importance); thus, these constitute the essential features to operate with.

Nevertheless, and even operating with the optimal subsample of features, the lack of anonymity -due to bank privacy settings- forces us to implement feature engineering to create several meaningful features appending additional information that will contribute to the analysis. Thus, in this case study we have additionally applied the One-hot encoding technique [31].

One major challenge that arises when developing ML/DL models for classification problems is the class imbalance. Datasets indicating real-world examples such as human disease diagnosis, online fraud cases, or transportation accident occurrence are hard to be resolved because they do not have an adequate portion of samples [32]. The weakness to predict rare events constituting the minority class detracts from the predictive built models because the algorithm learns from the majority class at the most, making it "natural" to have a greater tendency towards it, and it prone to overfitting the majority class. Hence, by predicting the majority class, models score high on their loss-functions. Since our dataset suffers from imbalanced data classes, three different data enrichment techniques are applied: a) the random oversampling technique, b) the assignation of weight to the minority class, and c) the SMOTE technique along with TOMEK LINK [33].

2) ML/DL Materials: Going beyond a simple case, this research proposes the implementation of a neural network that will be later exploited through the FaaS paradigm. Thus, a multilayer perceptron is implemented by exploiting Keras [34], consisting of 3 core layers and 1 output label, which defines the result. Furthermore, the Rectified Linear unit (ReLu) function has been exploited to consider nonlinear relationships for the model, and the dropout method is adopted, which is used to prevent a model from overfitting. Concerning the output layer, the Sigmoid activation function has been applied since it is more suitable for our binary classification problem; after the model's construction, the Adam algorithm forms to optimize the learning rate. Concerning the loss function, the binary cross-entropy is applied to calculate the loss due to the fact that we are dealing with a binary classification problem, and the labels correspond to binary values. Regarding the training procedure, the CV-stratified k-fold cross-validation [35], has been applied to the training set in order to deliver robust model prediction. Lastly, MLP has been chosen to be the predominant model, a data normalization procedure must be conducted because without this, training the neural networks would have been very slow and undependable [36].

B. Online Phase

The main challenge for effective service provisioning in the case of a serverless application is the design and integration of the developer's business logic in fine-grain functions, called nanoservices. Nanoservices evolve as an architecture pattern with properties such as isolation and technological freedom [37], designed mostly to overcome complexities related to microservices. It can be considered as an evolutionary step to implement more fine-grained application components (functions) for maintenance efficiency and therefore, a "term" parallelism has been created since both serverless functions and nanoservices adopt the same features [38], [39], [40]. The serverless computing model allows the developer to specify the processes inside functions without supervising the allocation

resources and handling dynamic workloads, either for the entire management of the infrastructure.

Thus, a pipeline of functions is proposed in order to execute data analytics and ML tasks by utilizing Apache OpenWhisk. Each function is implemented in python and executed -similarly- to python runtime, the task of each function derives from tasks in the offline phase and in detail is presented:

The **first function** receives the parameters (in JSON format) and transforms their structure in order to increase the efficiency of data analysis. It also traces if there are missing values and performs the required imputation according to the strategy of the offline phase. Furthermore, it adds the external features according to the feature engineering procedure of the offline phase and forms capitalization to categorical data to maintain a consistent format for the given data.

The **second function** receives as input the transformed data from the first function; its main task is to perform label encoding to these values that constitute categorical variables and performs the normalization according to the one that applied in the offline phase. The modified parameters from the second function pass as input in the third and final function. In the **third function**, the pre-trained model (the custom-made MLP) from the offline phase is acquired, becomes part of the mentioned function, and it is used as an inference for final prediction - in particularly if the given data (parameters) constitutes a fraudulent or authorized transaction. The last function's output produces a message that refers to if the transaction is fraud or legitimate.

C. Overcoming Serverless Constraints and Limitations

Serverless computing and the respective platforms that realize it, have specific constraints and limitations. One of them refers to the restricted size of code for the functions. We propose an approach to overcome it by extending the OpenWhisk's feature for the integration of external Docker container mechanisms without violating the restriction. Thus, developers can utilize external Docker containers to include the required ML libraries needed to complete each ML function task, which can be injected dynamically by the platform during initialization (with the flag -docker accompanied by the custom container image). The external container mechanisms that support the necessary libraries for the task execution, are not stateful. According to the OpenWhisk architecture principles, the functionality that the external docker container provides will be pre-installed into the runtime during the function's creation process and not in invocation, thus the holistic process is stateless reflecting once again the serverless offerings. Creating custom runtime images constitutes a solution to address the difficulty of having external application dependencies too large to be deployed, due to the function size limit, by merely composing a Dockerfile with the necessary contents that must have in order to support the OpenWhisk's constructed function. For the selected machine learning libraries, a requirements.txt file has been composed, including all the preferable libraries and their dependencies to perform each function's assignment. This solution has been fully adopted in the presented approach since most of the third-party python packages exceed the limitation of the function's source code

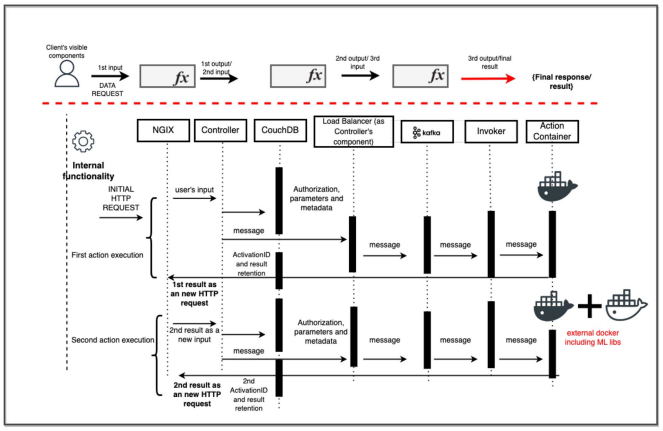


Fig. 6. ML/DL pipelines as serverless functions.

(48MB) [41]. Especially for the overall custom-made pipeline, several ML libraries were required (e.g., Pandas, NumPy, scikit-learn, and Keras), carrying numerous shared libraries and compiled native dependencies for performance, leading to hundreds of megabytes of dependencies. Thereby, we have exploited the above fundamental benefit that OpenWhisk provides by building and configuring custom-made docker containers and injecting them into OpenWhisk's environment to be linked with the constructed function without affecting the limitation of the size (as it is shown in Figure 1). As a principal limitation of OpenWhisk is underlined that the construction of a suitable docker container for OpenWhisk usage follows a specific process; custom runtime images must implement the function interface since this is the protocol used by the platform to pass invocation requests to the runtime containers. Additionally, containers are expected to expose an HTTP server (running on port 8080) with /init (initialize) and /run endpoints [42].

An additional challenge/limitation refers to the custom-made container, which must be available and validated through the official Docker repository (i.e., Dockerhub) since it is the only registry currently supported. To overcome this limitation, custom runtime images with the appropriate specifications have been exploited. The applied container runtimes pre-installs major ML Libraries such as pandas, Keras, NumPy, scikit-learn, and SciPy during the build process and expose them during invocations in order to execute the data analytics and machine learning tasks that the pipeline addresses. The overall approach of the online phase is presented in Figure 6. As the new data are inserted -as input- in the pipeline, an HTTP request is initialized towards all the core components (i.e., NGIX Server, Controller, CouchDB, LoadBalancer, and Function-Container). The user practically makes an HTTP request (by providing the given data) through the REST API endpoint-constructed similarly in the OpenWhisk environment-, which passes through the pipeline of functions and follows the procedure mentioned above to fulfill the request and produce a result. It is crucial to mention that open- source platforms like OpenFaaS, offer the ability to overcome the code-size constraints, by creating and deploying serverless functions -solely- as containers.

Apache OpenWhisk provides both the ability to create functions as standalone docker containers and thus bypass the

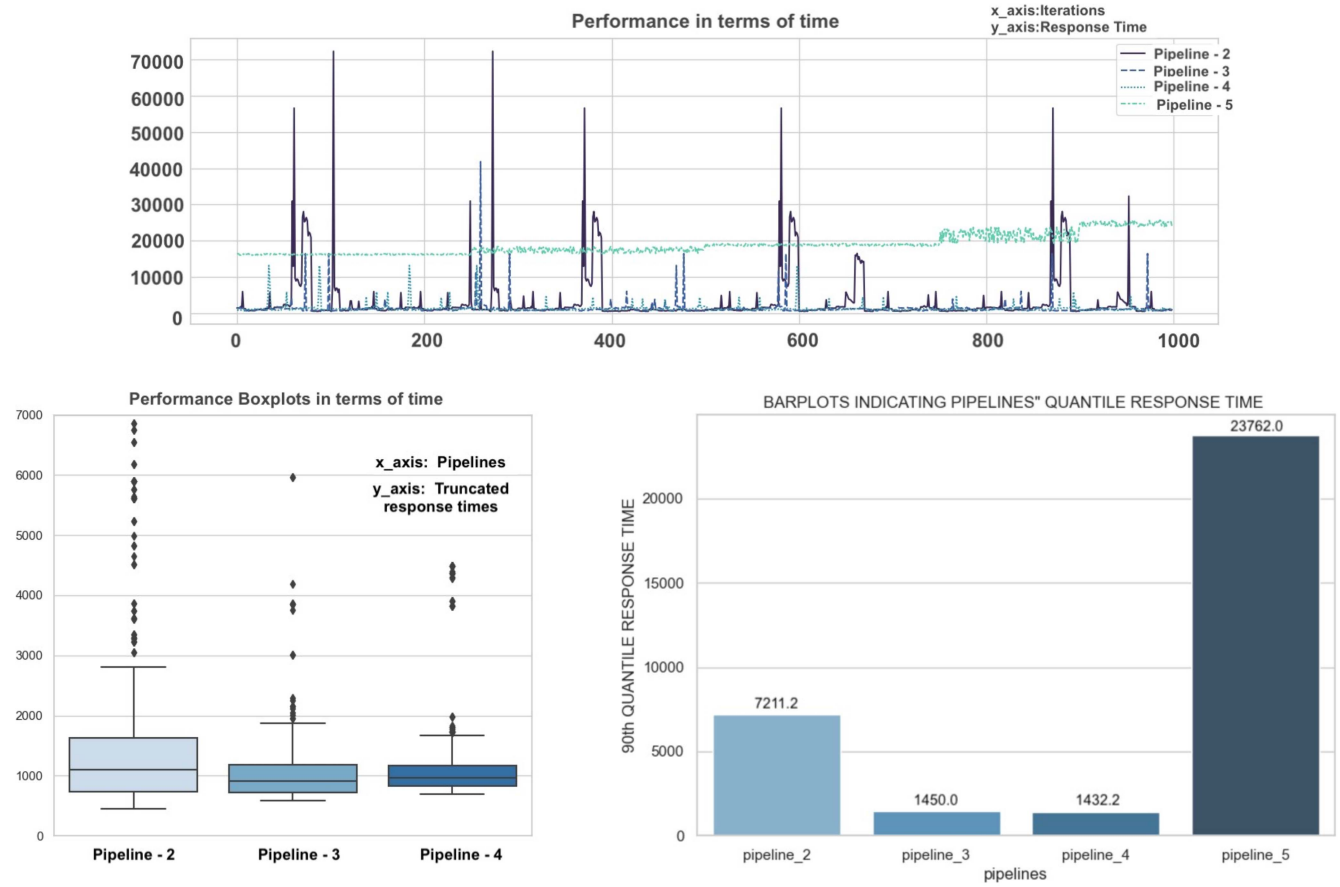


Fig. 7. Visualization concerning response time (in ms).

code size limit, and also as hybrid components including the specific function runtime along with the proper support container. We have chosen this hybrid (fine-grained) functionality since it increases modularity, making applications easier to be developed, tested, deployed, and, more importantly, updated and maintained. Since both Openwhisk and OpenFaaS can avoid the code-size limitation, we concentrate on their cloud-based offerings. OpenFaaS is not a multi-tenant product, in contrast with OpenWhisk, which may have restrictions related to code (in its hybrid functionality), but it is a multi-tenant, and it can power several public cloud functions offerings.

IV. AI-Based Function Regulator

A. Problem Formulation

Even though cloud providers support template engines through their marketplaces for building functions given specific programming languages, the lack of providing templates and guidelines regarding the optimal number of functions for designing the desirable service workflow still needs to be uncovered. Although efforts have been made by researchers [43] who point to learning the relationships between cost/run-time and unseen configurations related to a set of the developer's predefined number of serverless functions in order to select the best configuration that minimizes the cost but also meet the user-specified performance criteria (i.e., response time), and efforts [44] related to automating the memory size optimization for serverless functions (resource sizing). Our prototype moves beyond the standard intelligent resource adaptation and recommends the optimal number of serverless function blocks providing the optimal response time, therefore we examine the hypothesis's acceptance or rejection that the smaller number of composed pipelines of functions produces the minimum application's response time. According to the hypothesis mentioned above, we have performed our experiments by forming our implemented ML-FaaS application as 4 different discrete pipelines composed of 2, 3, 4, 5 chained functions. The experimentation focuses on 4 composition scenarios following the established data preprocessing lifecycle in terms of the corresponding five key steps (i.e., cleaning, imputation, transformation, label encoding, normalization-standardization) [45], [46], [47]. Subsequently, we have triggered 1000 requests in each of these to perform a comparison along the produced response times. The below figure (Figure 7) - explicitly-depict the response times concerning each of the constructed pipelines as mentioned above.

By observing Figure 7, except the response time data that concerns the 5-pipeline function execution which has been eliminated due to the divergence in terms of response time, we can conclude by rejecting the hypothesis, which constitutes that the smaller the number of functions pipeline is, the minimum response time the service will provide (i.e., the application pipeline composed with 3 functions results in shorter response time compared to the pipeline composed with 2 functions). Thus, the number of functions composing

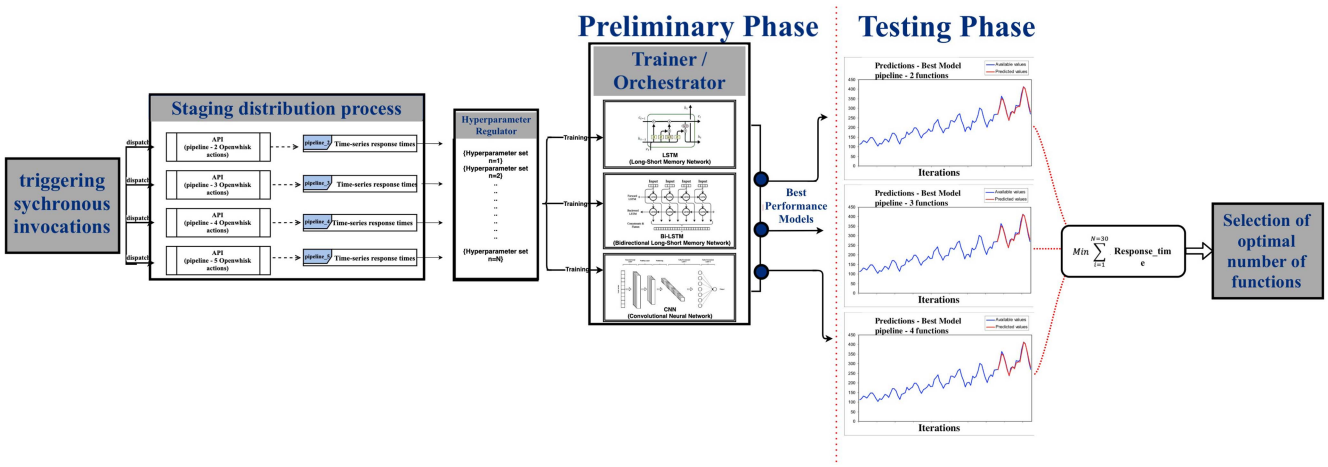


Fig. 8. AI-based Function Regulator conceptual architecture.

the pipeline is not correlated with the produced response time. Furthermore, we have acquired data concerning the CPU usage during the functions' runtime, and we have discovered that it has homogenous nature therefore, it is not a valuable attribute to attach in our proposed approach.

Based on Figure 8, we exploit the dynamics of specific Deep Neural Networks architectures to implement forecasts for the future response times related to each testing pipeline serverless application; eventually, a prefixed number of testing iterations is selected, and finally, we maintain the number of functions that produce the minimum forecasted response time cumulatively. The n-functions pipeline that produces the number of specific forecasts reflecting the service's minimum response time is the selected template for our application realization. Our AI-based approach is composed of the following main components:

B. Overall Approach

By validating the assumption that the number of functions doesn't depend on the serverless pipelines' response times, we propose a holistic AI-based approach named "AI-based Function Regulator" to recommend the number of functions towards the optimal service composition, by predicting the function composition runtime execution. The developer provides as input the implemented source code and, as an output, receives a concrete number that validates the optimal number of functions that will produce the minimum response time, ensuring the high quality of the user's experience. Figure 8 depicts the conceptual architecture of our implemented "AI-based Function Regulator", and the fine-grained description of its core components is cited below.

1) Staging Distribution Process: The initial component accepts as input the source and creates different pipelines with a different function-partition process. For our case study, according to our experiments, we have selected to form our aforementioned ML-FaaS use case as a pipeline of a) 2 functions, b) 3 functions, and c) 4 functions, since the pipeline with 5 functions has produced incredibly high values in terms of response time, and thus it has been eliminated. Moreover, this component triggers 1000 requests (in successive order over some period) in each of our implemented pipelines and gathers response times in the form of time-series. Nevertheless, and since serverless suffers from the so-called "cold-start phenomenon," which causes data spike points related to the response time due to the initialization of the runtime containers, we performed data pruning by adopting an outlier detection method named Hampel filter [48]. The primary purpose of the Hampel filter is to identify and replace the outliers in each of the series by using sliding windows over the time-series sequences.

2) Hyperparameter Regulator: Since we intend to perform forecasts based on specific Neural Network architectures, we utilized the hyperparameter Regulator component, an automated optimization method to evade the trial-and-error process as it concerns the hyperparameter tuning. The component generates "an artificial landscape" of all the possible combinations of the Neural Networks parameters and tests the effectiveness and robustness of the selected architectures. This functionality encourages the data scientist to obtain a means for comparing the performance of various algorithms in terms of Root-mean-square-error (RMSE).

3) Trainer/Orchestrator: is the following and most important component supporting the training procedure. Following the process flow of Figure 8, the pre-processed response times for each of the implemented sequences constitute the inputs in three selected ANN architectures; a) The Long-Short term memory (LSTM) Neural Network) [49] a specific recurrent neural network (RNN) architecture that was designed to model temporal sequences and achieves high effectiveness in the field of time-series forecasting, b) Bidirectional LSTM networks (BLSTM), as an alternative to the standard architecture that operates on the input sequence in both directions to make a decision, and c) Convolutional Neural Networks (CNN) which constitutes the most common architectures in image processing and computer vision [50] although they have emerged for solving data sequence problems. Therefore, the best performance models -based on the lowest RMSE values-are selected, and after a predefined number of iterations (in the testing phase), we collect the time-series forecasted values for each of the pipelines. By finalizing the process, we retain the lowest value related to each of the pipelines' sum of the forecasted response

TABLE II
BENCHMARKING DIFFERENT GROUPINGS OF FUNCTIONS DURING THE PARTITION PROCEDURE

Combinations	Task [1] Data cleaning - imputation	Task [2] Data transformation	Task [3] Label encoding	Task [4] Data Normalization	Task [5] Pre-trained ML/DL model adaptation and prediction	Avg. response times *(30 executions)*
2 functions: [[1, 2, 3, 4], [5]]	1st f(x)	1st f(x)	1st f(x)	1st f(x)	2nd f(x)	**1.42 sec**
2 functions: [[1, 2, 3], [4, 5]]	1st f(x)	1st f(x)	1st f(x)	2nd f(x)	2nd f(x)	1.44 sec
2 functions: [[1], [2, 3, 4, 5]]	1st f(x)	2nd f(x)	2nd f(x)	2nd f(x)	2nd f(x)	1.70 sec
3 functions: [[1,2,3], [4], [5]]	1st f(x)	1st f(x)	1st f(x)	2nd f(x)	3rd f(x)	**0.87 sec**
3 functions: [[1], [2,3,4], [5]]	1st f(x)	2nd f(x)	2nd f(x)	2nd f(x)	3rd f(x)	0.89 sec
3 functions: [[1], [2], [3,4,5]]	1st f(x)	2nd f(x)	3rd f(x)	3rd f(x)	3rd f(x)	0.93 sec
3 functions: [[1], [2, 3], [4, 5]]	1st f(x)	2nd f(x)	2nd f(x)	3rd f(x)	3rd f(x)	0.92 sec
> 4 functions: [[1,2], [3], [4], [5]]	1st f(x)	1st f(x)	2nd f(x)	3rd f(x)	4th f(x)	**0.96 sec**
4 functions: [[1], [2], [3], [4,5]]	1st f(x)	2nd f(x)	3rd f(x)	4th f(x)	4th f(x)	1.1 sec
4 functions: [[1], [2,3], [4], [5]]	1st f(x)	2nd f(x)	2nd f(x)	3rd f(x)	4th f(x)	0.98 sec
5 functions: [[1], [2], [3], [4], [5]]	1st f(x)	2nd f(x)	3rd f(x)	4th f(x)	5th f(x)	**18.45 sec**

TABLE III
EVALUATION RESULTS REGARDING THE OFFLINE PHASE

Model	Recall	AUC
Random forest	56%	78%
Random forest - applying weight to minority class samples	55.26%	77.39%
Random Forest - with oversampling 40% minority class	63.46%	81.37%
Random Forest - applying SMOTE & TOMEK - Link technique	57.32%	78.56%
Extra Trees Classifier	61.07%	80.42%
Extra Trees Classifier - applying weight to minority class samples	61.97%	80.23%
Extra Trees Classifier - with oversampling 40% minority class	62.47%	80.89%
Extra Trees Classifier applying SMOTE & TOMEK -Link technique	69.32%	84.18%
Keras ANN	54.25%	76.9%
Keras ANN - applying weight to minority class samples	**84%**	**85%**
Keras ANN - with oversampling 40% minority class	72.1%	84.84%
Keras ANN - applying SMOTE & TOMEK -Link technique	67.99%	82.97%

times, indicating the number of functions that the application should be composed. As shown in the Table II, different groupings of functions, and thus different partitions, result to different end-to-end response times (average for 30 executions). The selected partitioning is marked with bold and underline fonts per number of functions and is the partitioning with the optimum end-to-end performance. It should be noted that this information and the respective decision is taken in an automated way by the AI-based approach that has been implemented, i.e., the AI-based Function Regulator. The different response times that appear in the mentioned table are due to the fact that different groupings result to data transfer between functions. This data transfer introduces latency to the overall process. Thus, the optimized grouping minimizes latency emerging from data transfers and contributes to efficient utilization of functions in service chains.

V. EVALUATION

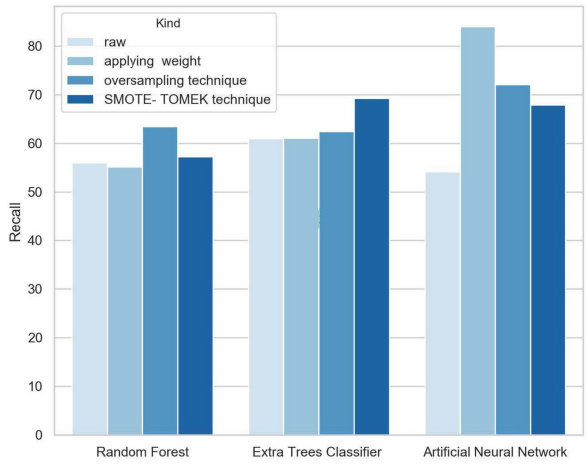
Regarding the ML/DL-related evaluation, the most common metric to assess the classification performance of the constructed model is the accuracy metric. However, this is not the only way to summarize the efficiency of a supervised model performing on a given dataset. Realistically, this accuracy might not be the appropriate measure for the constructed application since it demonstrates a good accuracy rate on the majority class but poor on the minority class [51]; thus, it is essential to choose the ideal metric when selecting between models and adjusting parameters.

Due to the imbalanced dataset problem, we have used measures like recall, AUC, and Confusion Matrix to evaluate the chosen models. To this end, initially, the results from the training phase are presented.

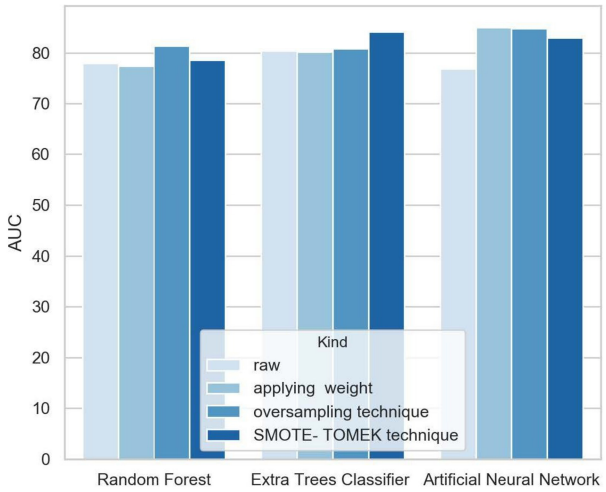
According to the Table III, there are some cases which, in the training phase, revealed the overfitting phenomenon; the random forest -by applying the technique oversampling, the extra trees classifier -similarly-, and the latter by using SMOTE and TOMEK Link method. This practically indicates that the models had been over-trained with the training dataset but failed to predict the new unseen data giving low rates on the collected measures.

Figure 9 (all the sub-figures) provides an overview of the ML/DL performance. Especially it highlights the fact that the neural network, by applying weight in minority class, has achieved the most high-grade performance by correctly identifying the 84% of the test observations.

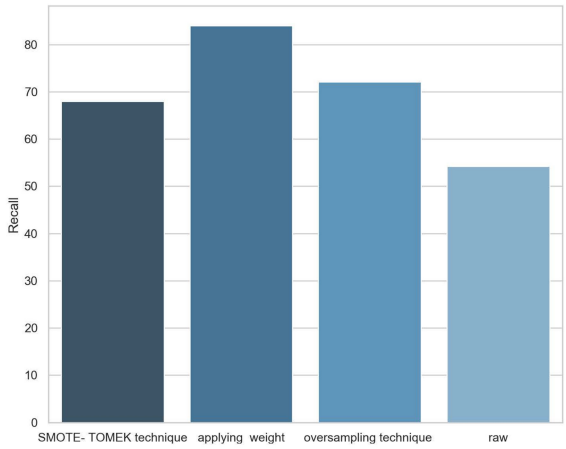
Additionally, we evaluate the added value from the business perspective regarding the holistic implementation of an ML-FaaS application by using well-known models for estimating the development effort in computer programming, the ConstructiveCostModel-COCOMO [52] and COCOMO2.0 [53]. COCOMO is a regression-based model estimating the efforts and schedule of a software product expressed in person-months (PM) by using the source lines of the project's code (*SLOC*). In addition, it considers several aspects named "cost drivers" such as developers' capacity, experience, ability, product complexity. The COCOMO model is based on a specific formula $E = a^*(KLOC)^*b$, where a is the project development mode, *KLOC* is the size of the code measured in thousands of *SLOC*, and b is a scaling factor considering economies of scale. More than that, projects are categorized into three types based on COCOMO principles: i) Organic, ii) Semi-detached, and iii) Embedded. We use the organic development mode for our case study, which is suitable for small projects, giving us the constants mentioned above as: $a = 3.2$ and $b = 1.05$. Therefore, for our implemented ML-FaaS application, we measured 352 lines of code for writing the holistic approach (i.e., data acquisition, the entire data preprocessing procedure, Exploratory Data Analysis for data visualization, training with different ML/DL algorithms, hyperparameter tuning, functions' code related to the inference part). Thus, the nominal effort for a



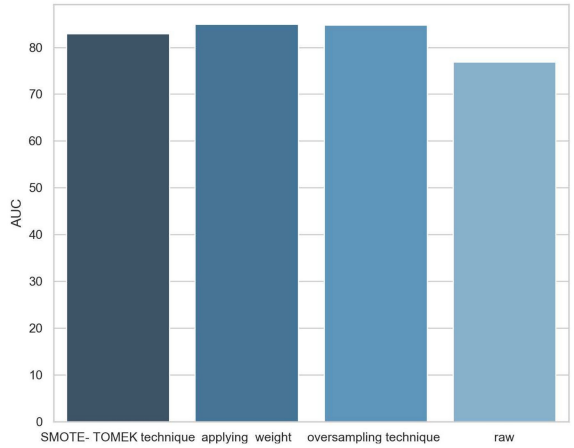
Sub-figure 9.a Visualization concerning response time (in ms)



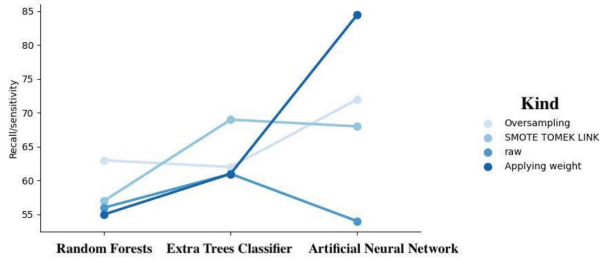
Sub-figure 9.b Evaluation performance regarding Area Under Curve



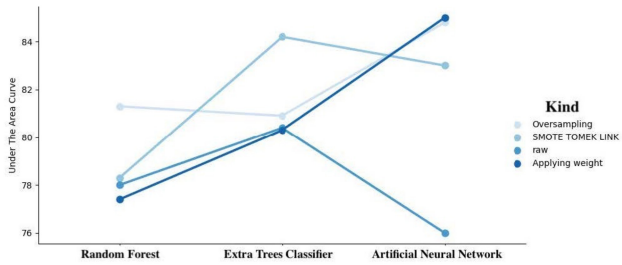
Sub-figure 9.c Evaluation performance (MLP) regarding sensitivity



Sub-figure 9.d Evaluation performance (MLP) regarding Area Under Curve



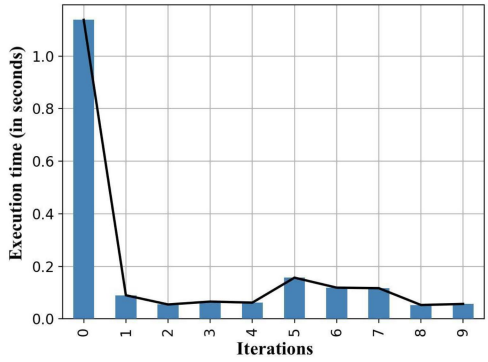
Sub-figure 9.e Factorplot for sensitivity



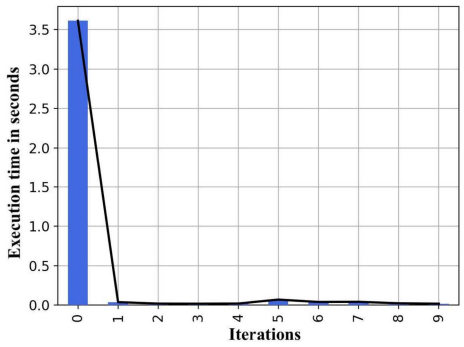
Sub-figure 9.f Factorplot for Area Under Curve

Fig. 9. a. Visualization concerning response time (in ms). b. Evaluation performance regarding Area Under Curve. c. Evaluation performance (MLP) regarding sensitivity. d. Evaluation performance (MLP) regarding Area Under Curve. e. Factorplot for sensitivity. f. Factorplot for Area Under Curve.

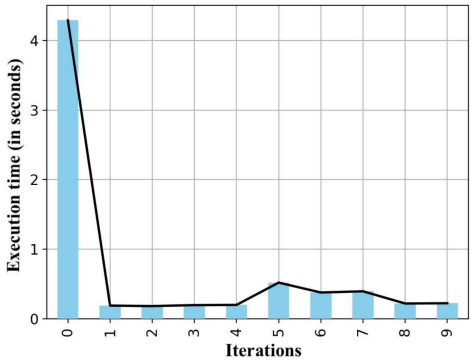
Data Scientist/ML Practitioner to write the application's code is 3.2 * (0.352) * 1.05 = 1.02*PM*. Nevertheless, the nominal effort estimation computed above has to be corrected by effort multipliers which are the above-mentioned cost drivers. COCOMO has 15 cost drivers, and COCOMO2 has 17, which improves the estimation's accuracy. For our case studied, we considered the nominal multiplier values (1.0) for all the cost drivers, except for Product Complexity (*CPLX*), which presents the complexity to serve the final application. Thus, since the serverless model alleviates the developer from operational costs regarding the maintenance, scaling, and monitoring, we only take into account the installation of Apache OpenWhisk and the deployment of functions. According to the relevant documentation this is a straightforward procedure and therefore, we estimate the *CPLX* = 0.70 (very low complexity). Finally, we estimate 0.71 PM as the effort for building, deploying, and serving serverless ML applications. On the other side, in the case of cloud infrastructure deployments,



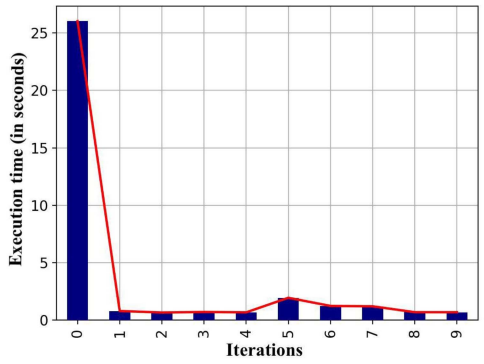
Sub-figure 10.a On average response time for the first OpenWhisk action



Sub-figure 10.b On average response time for the second OpenWhisk action



Sub-figure 10.c On average response time for the third OpenWhisk action



Sub-figure 10.d On average response time for the ML-FaaS pipeline

Fig. 10. a. On average response time for the first OpenWhisk action. b. On average response time for the second OpenWhisk action. c. On average response time for the third OpenWhisk action. d. On average response time for the ML-FaaS pipeline.

configuration needs to be performed. This refers to deployment of virtual instances, configuration of load balancers for scaling, API gateways for user interaction with the service, and a messaging-queuing system for serving the concurrent requests. Thus, we estimate the $CPLX = 1.30$ (very high complexity) since this procedure is extremely complicated creating a burden for developers. Therefore, for the second case, we obtain $1.33PM$ as the estimated effort for creating and deploying a typical ML/DL application in the cloud, almost double effort for implementing and serving such an application. Regarding the FaaS evaluation, we have evaluated the performance of the overall solution (and the underlying platform) by testing the implemented ML/DL pipeline through several requests and evaluating its performance in terms of execution time. As cited in Figure 10 (all the sub-figures) and according to our experiments that include 10 re-executions, each action individually -on average-is executed in a time interval of seconds and milliseconds. For the first action, the lowest average execution time has been 53 ms and the highest 1.40 sec. For the second action, the fastest average execution time has been 20 ms, and the slowest is 3.60 sec. The third function, which is the one that invokes the DL model, has attained the best execution-on average-time in 180 ms and the most delayed performance in 4.30 sec. The execution time of the pipeline has varied in milliseconds. Nevertheless, because OpenWhisk has a "cold start spot", like other serverless platforms, the first request consumes a longer time to produce a response.

It is worth mentioning that, according to our experiments, our findings indicate that the response time of a service is not significantly affected by the allocated resources, thus additional resources would not lead to improved performance. The Table III demonstrates this case for two different flavors and resource configurations, indicating that the response time is independent to the allocated resources.

Regarding the results of the AI-based Function Regulator training procedure, as an essential measure of forecast performance has been selected the Root-mean-square-error (RMSE). The Table IV presents the results based on RMSE value for each of the selected ANN architectures and the optimal combinations of applied hyperparameters.

VI. Conclusion and Future Work

Towards raising the abstraction level for developers, the computing infrastructure providers are focusing on the provision of serverless offerings. Beyond the typical applications (which consist of several services that can be offered as functions in a FaaS paradigm), data analytics and artificial intelligence techniques also aim at exploiting the added value of serverless computing. In this context, the serverless computing model would provide an approach to building innovative artificial intelligent services, executed only in a few milliseconds while eliminating the need for data scientists to manage the infrastructure level resources. To this end, this paper proposes a means to leverage the serverless architecture

TABLE IV
ML-FAAS EVALUATION PERFORMANCE TESTED WITH DIFFERENT ALLOCATED RESOURCES

Performance results	On average response time			
	Ubuntu 16.04 - 4 processor cores - 4 RAM		Ubuntu 16.04 - 6 processor cores - 8 RAM	
	Low-level	High-level	Low-level	High-level
1st f(x)	53ms	1.40 sec.	51ms	1.41 sec
2nd f(x)	20ms	3.60sec	22ms	3.51 sec
3rd f(x)	180ms	4.30 sec	180ms	4.11sec
f(x)s pipeline	3.8 sec	25.4 sec	3.4sec	25.6 sec

TABLE V
EVALUATION RESULTS REGARDING AI-BASED FUNCTION REGULATOR

Model	Pipeline-2		Pipeline-3		Pipeline-4	
	RMSE results train/test	Hyperparameter	RMSE results train/test	Hyperparameter	RMSE results train/test	Hyperparameter
LSTM	180.71 **183.19**	units=**8**, kernel_size= **2**, epochs=**200**, lag=**8**	79.01 **80.42**	units=6, kernel_size= **2**, epochs=**90**, lag=**7**	60.93 / **75.98**	filters=**6**, **kernel_size=3**, epochs=**200**, lag=**8**
Bi-LSTM	192.94 199.17	units=7, kernel_size= 2, epochs=90, lag=8	78.55 87.15	units=7, kernel_size= 3, epochs=200, lag=7	70.85 / 77.44	units=4, kernel_size= 2, epochs=200, lag=8
CNN	452.24 502.33	filters=64, kernel_size=2, epochs=90, dense = 50, pool = 2,lag=7	290.11 330.72	filters=64, kernel_size=2, epochs=200, dense = 50, pool = 2, lag=8	320.56 / 410.88	filters=64, kernel_size=2, epochs=90, dense = 50, pool = 2, lag=6

for designing and implementing ML/DL pipelines in order to exploit them as services. The proposed approach constitutes a functional, lightweight solution for the developer/analyst with zero administration to deploy his entire business logic in a next-generation application. The evaluation of the proposed technique has been performed through a representative use case that reflects an ML pipeline. In our use case, we have approached a binary classification problem and a structured dataset with encrypted features, to showcase and highlight the requirement for applying several techniques to pre-process the dataset. Furthermore, various models have been evaluated including forests, and neural networks, each of them in four different data manipulation cases, as the evaluation dataset had suffered from imbalanced labelled observations. The best model (an MLP) has been selected by applying weight to minority class samples, reaching an 84% recall score. Additionally, this paper presents an approach for the creation of functions in a FaaS environment, Apache OpenWhisk. These functions have been realized as a pipeline, performing the respective ML/DL tasks. The overall solution has also been in terms of performance. The results indicate that warm serverless actions executions are within an acceptable latency range. However, we have observed that the cold-start phenomenon that several serverless platforms confront affects the normal latency distribution by giving the response with delay. Finally, we have proposed an AI-based framework for recommending the optimal function composition related to a serverless pipeline, optimizing the user's quality of experience in terms of the application's response time. Since the regulator enables a trade-off between modularity and performance efficiency (i.e., custom-made aggregating functions possibly would achieve better performance in terms of time, whereas it will decrease the level of re-usability), we intend to extend the functionalities of the regulator by allowing the analysis and consideration of the correlations between different partitions and the response time values in a holistic way. Our future actions include further experimenting with the pipelining approach applying rules and triggers and connected external sources (e.g., IoT devices). Also, it is within our future actions to exploit how the compute, I/O, and data intensity of each step of a workflow affects the composition of an application in different functions and how both each one of these parameters but also as a combination can be optimized in end-to-end scenarios in order to expand the functionality of the AI-based Regulator.

REFERENCES

[1] I. Yaqoob et al., "Big data: From beginning to future," *Int. J. Inf. Manag.*, vol. 36, no. 6, pp. 1231–1247, 2016.

[2] I. Baldini et al., "Serverless computing: Current trends and open problems," in *Research Advances in Cloud Computing*. Singapore: Springer. 2017, pp. 1–20.

[3] A. Christidis, R. Davies, and S. Moschoyiannis, "Serving machine learning workloads in resource constrained environments: A serverless deployment example," in *Proc. IEEE 12th Conf. Serv. Orient. Comput. Appl. (SOCA)* Nov. 2019, pp. 55–63.

[4] I. Baldini et al., "The serverless trilemma: Function composition for serverless computing," in *Proc. ACM SIGPLAN Int. Symp. New Ideas New Paradigms Reflections Program. Softw.*, Oct. 2017, pp. 89–103.

[5] M. Sciabarrà, *Learning Apache OpenWhisk: Developing Open Serverless Solutions*. Sebastopol, CA, USA: O'Reilly Media, 2019.

[6] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Gener. Comput. Syst.*, vol. 28, no. 5, pp. 755–768, 2012.

[7] D. Damkevala, R. Lunavara, M. Kosamkar, and S. Jayachandran, "Behavior analysis using serverless machine learning," in *Proc. 6th Int. Conf. Comput. Sustain. Global Develop. (INDIACom)*, Mar. 2019, pp. 1068–1072.

[8] "IBM Watson machine learning API." Accessed: Oct. 15, 2020. [Online]. Available: https://watson-ml-api.mybluemix.n

[9] V. Giménez-Alventosa, G. Moltó, and M. Caballer, "A framework and a performance assessment for serverless MapReduce on AWS Lambda," *Future Gener. Comput. Syst.*, vol. 97, pp. 259–274, Aug. 2019.

[10] "AWS cloud object storage—Store and retrieve data anywhere—Amazon simple storage service (S3)." Accessed: Oct. 15, 2020. [Online]. Available: https://aws.amazon.com/s3

[11] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht, "Occupy the cloud: Distributed computing for the 99%," in *Proc. Symp. Cloud Comput.*, Sep. 2017, pp. 445–451.

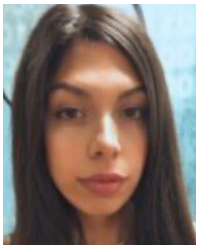
[12] J. Jiang et al., "Towards demystifying serverless machine learning training," in *Proc. Int. Conf. Manag. Data*, Jun. 2021, pp. 857–871.

[13] H. Wang, D. Niu, and B. Li, "Distributed machine learning with a serverless architecture," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2019, pp. 1288–1296.

[14] A. Bhattacharjee, A. D. Chhokra, Z. Kang, H. Sun, A. Gokhale, and G. Karsai, "BARISTA: Efficient and scalable serverless serving system for deep learning prediction services," in *Proc. IEEE Int. Conf. Cloud Eng. (IC2E)*, Jun. 2019, pp. 23–33.

[15] A. Bhattacharjee, Y. Barve, S. Khare, S. Bao, A. Gokhale, and T. Damiano, "Stratum: A serverless framework for the lifecycle management of machine learning-based data analytics tasks," in *Proc. USENIX Conf. Oper. Mach. Learn. (OpML)*, 2019, pp. 59–61.

[16] T. Rausch, W. Hummer, V. Muthusamy, A. Rashed, and S. Dustdar, "Towards a serverless platform for edge AI," in *Proc. 2nd USENIX Workshop Hot Topics Edge Comput. (HotEdge)*, 2019, pp. 1–7.
[17] M. Zhang, C. Krintz, and R. Wolski, "STOIC: Serverless teleoperable hybrid cloud for machine learning applications on edge device," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops (PerCom Workshops)*, Mar. 2020, pp. 1–6.
[18] F. Mohr, M. Wever, E. Hüllermeier, and A. Faez, "(WIP) towards the automated composition of machine learning services," in *Proc. IEEE Int. Conf. Serv. Comput. (SCC)*, Jul. 2018, pp. 241–244.
[19] A. Lakhan, F. H. Khoso, A. A. Arain, and K. Kanwar, "Serverless based functions aware framework for healthcare application," *Int. J. Emerg. Trends Eng. Res.*, vol. 9, no. 4, pp. 446–450, 2021.
[20] A. Christidis, S. Moschoyiannis, C.-H. Hsu, and R. Davies, "Enabling serverless deployment of large-scale AI workloads," *IEEE Access*, vol. 8, pp. 70150–70161, 2020.
[21] S. Risco, G. Moltó, D. M. Naranjo, and I. Blanquer "Serverless workflows for containerised applications in the cloud continuum," *J. Grid Comput.*, vol. 19, no. 3, pp. 1–18, 2021.
[22] D. Chahal, R. Ojha, M. Ramesh, and R. Singhal, "Migrating large deep learning models to serverless architecture," in *Proc. IEEE Int. Symp. Softw. Rel. Eng. Workshops (ISSREW)*, Oct. 2020, pp. 111–116.
[23] E. Paraskevoulakou and D. Kyriazis, "Leveraging the serverless paradigm for realizing machine learning pipelines across the edge-cloud continuum," in *Proc. 24th Conf. Innov. Clouds Internet Netw. Workshops (ICIN)* Mar. 2021, pp. 110–117.
[24] "Apache OpenWhisk open source serverless cloud platform." Accessed: Jul. 19, 2019. [Online]. Available: https://openwhisk.apache.org
[25] "IEEE-CIS fraud detection- kaggle." Accessed: Feb. 9, 2019. [Online]. Available: https://www.kaggle.com/competitions/ieee-fraud-detection
[26] A. Bilogur, "Missingno: A missing data visualization suite," *J. Open-Sour. Softw.*, vol. 3, no. 22, p. 547, 2018.
[27] M. J. Azur, E. A. Stuart, C. Frangakis, and P. J. Leaf, "Multiple imputation by chained equations: What is it and how does it work?" *Int. J. Methods Psychiatric Res.*, vol. 20, no. 1, pp. 40–49, 2011.
[28] R. J. Little and D. B. Rubin, *Statistical Analysis With Missing Data*, vol. 793. Hoboken, NJ, USA: Wiley, 2019.
[29] J. R. Cheema, "Some general guidelines for choosing missing data handling methods in educational research," *J. Modern Appl. Stat. Methods*, vol. 13, no. 2, p. 3, 2014.
[30] I. T. Jolliffe and J. Cadima, "Principal component analysis: A review and recent developments," *Philos. Trans. Roy. Soc. A, Math. Phys. Eng. Sci.*, vol. 374, no. 2065, 2016, Art. no. 20150202.
[31] P. Rodríguez, M. A. Bautista, J. Gonzàlez, and S. Escalera, "Beyond one-hot encoding: Lower dimensional target embedding," *Image Vis. Comput.*, vol. 75, pp. 21–31, Jul. 2018.
[32] A. Liu, J. Ghosh, and C. Martin, "Generative oversampling for mining imbalanced datasets," in *Proc. DMIN*, Jun. 2007, pp. 66–72.
[33] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *J. Artif. Intell. Res.*, vol. 16, no. 1, pp. 321–357, 2022.
[34] "Keras: The Python deep learning API." Accessed: Sep. 15, 2019. [Online]. Available: https://keras.io
[35] "StratifiedKFold." Accessed: Sep. 5, 2019. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html
[36] B. Karlik and A. V. Olgac, "Performance analysis of various activation functions in generalized MLP architectures of neural networks," *Int. J. Artif. Intell. Expert Syst.*, vol. 1, no. 4, pp. 111–122, 2011.
[37] E. Wolff, *Microservices: Flexible Software Architecture*. Sydney, NSW, Australia: Addison-Wesley Prof., 2016.
[38] I. Morenets and A. Shabinskiy, "Serverless event-driven applications development tools and techniques," *NaUKMA Res. Papers Comput. Sci.*, vol. 3, pp. 36–41, Dec. 2020.
[39] S. Banez, M. Shahbaz, and N. McKeown, "The case for a network fast path to the CPU," in *Proc. 18th ACM Workshop Hot Topics Netw.*, Nov. 2019, pp. 52–59.
[40] K. Kritikos and P. Skrzypek, "A review of serverless frameworks," in *Proc. IEEE/ACM Int. Conf. Utility Cloud Comput. Companion (UCC Companion)* Dec. 2018, pp. 161–168.
[41] "Apache OpenWhisk system details and limits." 2022. [Online]. Available: https://github.com/ibm-cloud-docs/openwhisk/blob/master/limits.md
[42] "Apache OpenWhisk- creating and invoking actions." 2021. [Online]. Available: https://github.com/apache/openwhisk/blob/master/docs/actions-new.md
[43] N. Akhtar, A. Raza, V. Ishakian, and I. Matta, "COSE: Configuring serverless functions using statistical learning," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Jul. 2020, pp. 129–138.
[44] S. Eismann, L. Bui, J. Grohmann, C. Abad, N. Herbst, and S. Kounev, "Sizeless: Predicting the optimal size of serverless functions," in *Proc. 22nd Int. Middlew. Conf.*, Dec. 2021, pp. 248–259.
[45] V. Agarwal, "Research on data preprocessing and categorization technique for smartphone review analysis," *Int. J. Comput. Appl.*, vol. 131, no. 4, pp. 30–36, 2015.
[46] H. Jamshed, M. S. A. Khan, M. Khurram, S. Inayatullah, and S. Athar, "Data preprocessing: A preliminary step for Web data mining," *3C Tecnología Glosas de Innovación Aplicadas a la Pyme*, vol. 8, no. 1, pp. 206–221, 2019.
[47] B. Cannas, A. Fanni, L. See, and G. Sias, "Data preprocessing for river flow forecasting using neural networks: Wavelet transforms and data partitioning," *Phys. Chem. Earth Parts A/B/C*, vol. 31, no. 18, pp. 1164–1171, 2006.
[48] R. K. Pearson, "Outliers in process modeling and identification," *IEEE Trans. Control Syst. Technol.*, vol. 10, no. 1, pp. 55–63, Jan. 2002.
[49] B. Lindemann, T. Müller, H. Vietz, N. Jazdi, and M. Weyrich, "A survey on long short-term memory networks for time series prediction," *Procedia CIRP*, vol. 99, pp. 650–655, May 2021.
[50] L. Sadouk, "CNN approaches for time series classification," in *Time Series Analysis-Data, Methods, and Applications*, vol. 5, InTech, 2019.
[51] Q. Zou, S. Xie, Z. Lin, M. Wu, and Y. Ju, "Finding the best classification threshold in imbalanced classification," *Big Data Res.*, vol. 5, pp. 2–8, Sep. 2016.
[52] B. W. Boehm, "Software engineering economics," *IEEE Trans. Softw. Eng.*, vol. SE-10, no. 1, pp. 4–21, Jan. 1984.
[53] B. Boehm, B. Clark, E. Horowitz, C. Westland, R. Madachy, and R. Selby, "Cost models for future software life cycle processes: COCOMO 2.0," *Ann. Softw. Eng.*, vol. 1, no. 1, pp. 57–94, 1995.



Efterpi Paraskevoulakou received the bachelor's degree from the Department of Economics, University of Piraeus in 2017, and the Postgraduate Diploma degree in big data and analytics from the Department of Digital Systems, University of Piraeus in 2020. She is currently pursuing the Ph.D. degree with the University of Piraeus and a member of the Data and Cloud laboratory team under the supervision of Associate Professor Dr. D. Kyriazis, where she is a Research Assistant. Her research interests are oriented toward the orchestration and optimal management of the cloud-edge continuum applications by exploiting artificial intelligence, machine learning, and data analytics.

Dimosthenis Kyriazis received the diploma degree from the School of Electrical and Computer Engineering, National Technical University of Athens (NTUA) in 2001, and the M.Sc. degree in techno-economics in 2004 (co-organized by the Electrical and Computer Engineering Dept—NTUA, Economic Sciences Dept—National Kapodistrian University of Athens, Industrial Management Dept—University of Piraeus). He is currently pursuing the Ph.D. degree in the area of service oriented architectures with a focus on quality aspects and workflow management with the School of Electrical and Computer Engineering Department, NTUA. He is an Associate Professor with the Department of Digital Systems, University of Piraeus. His expertise lies with service-based, distributed and heterogeneous systems, software engineering and data management. He has participated in several EU and National funded projects (e.g., BigDataStack, CrowdHEALTH, MATILDA, 5GTANGO, CYBELE, ORBIT, VISION Cloud, IRMOS, and 4CaaSt) leading research for addressing issues related to quality of service provisioning, fault tolerance, workflow management, performance modeling in service oriented environments and application domains, such as multimedia, post-production, virtual reality, finance, and e-health. He is currently focusing on data management, virtualization technologies for high-availability in cloud and edge environments, as well as socially-enhanced techniques for IoT management—coordinating EU funded projects that target these areas, while also analyzing topics related to big data management and content syndication. He has also participated is several EU working groups (such as Future Internet Architecture and Cloud QoS&SLAs).