Leveraging the serverless paradigm for realizing machine learning pipelines across the edge-cloud continuum

Efterpi Paraskevoulakou
Department of Digital Systems
University of Piraeus
Piraeus, Greece
e.paraskevoulakou@unipi.gr

Dimosthenis Kyriazis
Department of Digital Systems
University of Piraeus
Piraeus, Greece
dimos@unipi.gr

2021 24th Conference on Innovation in Clouds, Internet and Networks (ICIN) | 978-1-7281-7705-2/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICIN51074.2021.9385525

***Abstract*— The exceedingly exponential-growing data rate highlighted numerous requirements and several approaches have been released to maximize the added-value of cloud and edge resources. Whereas data scientists utilize algorithmic models in order to transform datasets and extract actionable knowledge, a key challenge is oriented towards abstracting the underline layers: the ones enabling the management of infrastructure resources and the ones responsible to provide frameworks and components as services. In this sense, the serverless approach features as the novel paradigm of new cloud-related technology, enabling the agile implementation of applications and services. The concept of Function as a Service (FaaS) is introduced as a revolutionary model that offers the means to exploit serverless offerings. Developers have the potential to design their applications with the necessary scalability in the form of nanoservices without addressing themselves the way the infrastructure resources should be deployed and managed. By abstracting away the underlying hardware allocations, the data scientist concentrates on the business logic and critical problems of Machine Learning (ML) algorithms. This paper introduces an approach to realize the provision of ML Functions as a Service (i.e., ML-FaaS), by exploiting the Apache OpenWhisk event-driven, distributed serverless platform. The presented approach tackles also composite services that consist of single ones i.e., workflows of ML tasks including processes such as aggregation, cleaning, feature extraction, and analytics; thus, reflecting the complete data path. We also illustrate the operation of the approach mentioned above and assess its performance and effectiveness exploiting a holistic, end-to-end anti-fraud detection machine learning pipeline.**

Keywords—machine learning, artificial intelligence, serverless, function as a service, cloud computing, edge computing

I. INTRODUCTION

The current century is verifiably, the revolution of the big data era since data have expanded within a short period of time [1]. The necessity for comprehension, formulation, and manipulation of heterogeneous datasets forced data scientists to explore, adopt, and invent various methods and techniques to exploit the valuable, actionable information that arises from them. 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 and exploits the cloud and edge infrastructure resources.

However, in the scope of managing the underlying infrastructure resources for the execution of the ML and AI algorithms, the scale, and the complexity to orchestrate the resources, require additional efforts from the data analysts, which inhibits both their productivity and decisiveness. The management of resources constitutes the necessity for full-stack development and deployment since the analyst has to specify and manage the infrastructure (e.g. memory, resources, allocation, number of CPUs, physical topology, etc).

Following the evolution of cloud computing environments, the need for agility, productivity, effectiveness, and scalability led to the emergence of the concept of microservices [2]. Nonetheless, an even higher level of abstraction is required to adder ss the application developers and the data scientists, by enabling them to focus mostly on the business logic and the AI/ML algorithms, rather than dealing with the allocation and the runtime adaptation of resources (e.g., virtual machines). This higher level of abstraction is facilitated by the so-called serverless architecture, which has emerged as a novel paradigm for the deployment of applications and services [3]. It is often referred as an evolution of cloud infrastructures, and as a result it is an "extra-driver" towards the even wider of cloud technologies providing benefits that include a much more comfortable development pipeline with codebases abstracted away from architectural complexities [4]. The industry around serverless architectures has increased markedly by exposing an omen that it will be the trend to continue beyond traditional applications [5]. Serverless evolution will cause the degradation of Hybrid cloud on- premises applications, though some deployments might persist due to regulatory constraints and data governance rules [6]. In the scope of serverless offerings, Amazon delivers lambda functions [7], providing scalable stateless event-driven functions. In order to gain a competitive advantage to the digital market, Google [8] and IBM [9] provide their own platforms, offering a means to realize serverless offerings. From the prospect of an end-user (e.g., data scientist, application developer, etc.), serverless platforms offer a simplified programming model, which abstracts several operational matters, by enabling the deployment of rapidly small bits of cloud-native event handlers (i.e., small fine- grained snippets of code containing the application work- flow). For a provider, serverless computing allows the reduction of operational costs through the effective optimization and administration of infrastructure resources; therefore, the developer/user has zero administrative control from his perspective. In spite of the fact that the serverless concept is reflecting as a supreme and agile cloud computing model, it is accompanied by an assortment of limitations [10] - concerning mainly and most importantly the runtime resource requirements of serverless code, the number of concurrent requests, and the maximum memory and CPU resources available for a function invocation. Resource constraints may be overhauled when users' requirements are altered (e.g., limits on concurrent requests), while other requirements are linked to the deployed platforms, the resource characteristics (e.g., maximum CPU or memory size), and the weakness supporting external storage.

By converging to the core of the serverless model, Function as a Service (FaaS) has emerged. Given the wide

978-1-7281-7705-2/21/$31.00 ©2021 IEEE

utilization of microservices and containers in enterprise applications architectures, FaaS is considered to be the best next step for the deployment of applications. FaaS enables developers to implement their applications as a set of "nanoservices", supporting massive scalability without any concern for resource administration (i.e., handling immense workloads and maintaining the entire lifecycle of the application). Nanoservices take place as an innovative application model that consists of single fine- grain functions or chain of connected functions, incorporating mechanisms for avoiding over-provisioning and excluding the maintenance of the state at a level of a sole function. The foremost vital part of the serverless ecosystem is the function entity, and the entire application's logic workflow is integrated through a sequence of such entities -i.e., a sequence of functions. Therefore, FaaS facilitates a holistic end-to-end application deployment in different ways: (i) single function deployment, (ii) several functions (of an application) deployment, and (iii) sequence of functions deployment. The most important point of this innovation is the function's stateless mode, meaning that, following the execution of each function, the life-process of the specific task concludes. In contrast, stateful functions- have higher performance costs because they need to attach external storage for retaining the data and also restrict the scalability; most importantly, due to the fact that the stateless mode dictates the necessity to synchronize the state between function invocations, when the load increases, the "state-keeping" infrastructure limits the ability for growth [10]. If a function is stateless, it is possible to increase the number of servers because we do not have the obligation of keeping the state in sync among the servers, which is complex and has limits.

Leveraging the added-value of the serverless model in cloud and edge environments, in this paper, we introduce an extendable approach that enables the realization of both ML tasks as functions, and of ML pipelines / chains as a set of functions by exposing a custom-made workflow of actions. The specific case addresses several ML tasks / models, including neural networks (in the specific case a Multilayer perceptron -MLP).

The overall offering is realized through an integrated custom- made workflow pipeline of functions - with the support of the open-source distributed event-driven serverless platform Apache OpenWhisk [11]. We assess the effectiveness of the proposed approach, from the beginning till the end, both in terms of traditional ML measures and in terms of infrastructure metrics (i.e., related to the FaaS performance).

The paper is organized as follows. The next section provides relevant approaches and research outcomes. Section 3 introduces the proposed approach to build a serverless data analytics pipeline. Section 4 provides details regarding the experimental setup and the corresponding evaluation outcomes, with a discussion of findings and limitations. Section 5 concludes the paper with a discussion on next steps.

II. RELATED WORK

Various solutions emerge as a means to enable the provision of FaaS solutions. Apache OpenWhisk constitutes a framework that allows the realization of FaaS offerings, by executing stateless functions (fx) – called actions - in response to events at any scale. The above suggests that the entire schema is initialized as the constructed functions are fed by data obtained from multiple sources (e.g., web sources, databases, message queues, IoT devices, etc.). Its entire internal development is based on containerized components that communicate as a microservice architecture; it is built onto famous open-source projects providing massive scalability and availability such as Nginx, Kafka, and Couch DB in corporation with its custom-made native components (i.e., Controller, Load Balancer, Invoker). Furthermore, the platform supports several programming languages to exploit action runtimes such as Python, Java, Node.js, Go, PHP, etc. As it concerns the languages that the platform does not support (i.e, C and C++), these can be integrated as custom dockerized runtimes [11]. Finally, the platform grants a Command Line Interface (CLI) called "wsk" to interact within its environment to create, execute and manage its entities effortlessly that can be installed in any operating system; in this way, developers are able to implement and communicate with the platform [12].

Serverless platforms come with certain constraints that prevent the main target of the developer. The main constraints and limitations refer to the dimensions of time and space (timeout, memory, frequency), and even though some of them are agile to be configured through respective parameters (e.g., execution time, concurrency, parameter size, code size, logs size, minute rate), they are limited in terms of values. A significant action limitation is concerning the physical code-base deployment size package [4]; which in terms of Apache OpenWhisk, the developer must provide his entire source code along with its necessary dependencies (i.e., libraries, frameworks) without exceeding the size of 48 MB [13]. Furthermore, OpenWhisk specifies the number of concurrencies and the minute rate for action execution; whereas facilitating the connection of functions with external Docker containers, which fulfill particular tasks in order to overcome the code size limitation constraint. Beyond solutions such as OpenWhisk, the cloud/ infrastructure providers focus on computing services and ready-to-use solutions addressing also the data analytics space.

Both serverless computing and Machine Learning - including DL (Deep Learning)- 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, reveal 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 ML knowledge. Researchers like D. Damkevala et. al. 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 on IBM Cloud due to the benefit that the developer must only be concerned with the data analysis/ processing/ evaluation and the service usage [14]. Nonetheless, serverless model fits efficiently in big data processing procedures (i.e., map reduce jobs). Since the authors Gimnez-Alventosa V. et. al., exploit serverless architecture for executing MapReduce jobs using AWS Lambda along with Amazon S3 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, but it exhibits an inhomogeneous performance behavior that may jeopardize adoption for tightly coupled computing jobs [15]. Furthermore, research community insisted in serverless development frameworks for handling ML workloads, since serverless suggests an entirely different interoperability to achieve scalability, without scheduling or monitoring the resources needed for processing workloads. PyWren [16] constitutes a map-reduce framework specialized for serverless architectures; it serializes the implemented python source code, sends it to AWS Lambda functions for massively parallel execution using Amazon S3 as an intermediate level and returns the results to the user. This proposal overcomes the serverless mainly in ML problems (i.e., retention of session state management), like the training phase, because PyWren is accompanied by a mannerism that uses external storage in a functional manner for intermediate computation results, overcoming delicately the stateless mode. Currently, most machine learning training jobs are being executed in parallel, since a large number of training samples needs 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. According to his related work and experimental profession, Ishakian V. et al. proved that by deploying Deep Learning models like SqueezeNet, Resnet, ResNet-18 with support of MXNet deep learning framework, while the inferencing latency can be within an acceptable range, longer delays, due to cold start serverless spots can skew latency distribution, causing consequently risk violation to strictly SLAs [17]. Carriera J. et al. analyzed the resource management problem in specific ML workloads, by exploring a case of machine learning frame- work, specialized both serverless architecture and Machine Learning procedures, and by arguing that, either of those in isolation, is insufficient [18]. Wang H. et al., proposed SIREN [19], a framework that 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 frame- work, 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 SIREN can reduce ML model training's phase time by up to 44%, as compared to traditional machine learning benchmarks, implemented on AWS at the same cost. More specifically, serverless lambda functions support a wide range of applications and use cases. Baldini et al., used serverless framework as a backend specifically for mobile applications [3]. Semantic research constitutes in Ripple description, a data storage architecture based on serverless architecture [20]. Further related works have emerged, such as Bila et al., who proposed a security analytics service based on Apache OpenWhisk and Kubernetes [21]. Nevertheless, research teams are attempting to find solutions and directions to adjust and optimize the resources coming from datacenters in order to favor the cloud vendors, minimize the serverless vulnerabilities, security issues and constraints, and serve high- quality artificial intelligence applications to enterprise companies, by reducing the cost and the administration of allocation resources. 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 data centers provide, serverless has also emerged in edge computing paths by leveraging the development of edge AI workflows. The authors of [22], 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 in order to illustrate the challenges in building and operating AI applications in an edge cloud scenario since by elevating concepts from AI lifecycle management into the established serverless model, the development of edge AI workflow functions constitutes a smoother procedure.

The aforementioned related works in the scope of serverless environments, distributed machine learning, and deep learning models, highlight the need for new frameworks and functionalities that can be consolidated to leverage and extend the data science and artificial intelligence services. The latter would contribute towards empowering scalability and parallelism, optimizing matters concerning the resources management, minimizing costs for the software developers, and annihilating latency issues.

Serverless, allows transmitting the business logic into an environment fully equipped with novel functionalities solely, by composing small snippets of code, providing further flexible extensions to produce entire static and dynamic workflows, overcoming typical limitations that serverless model serves, providing extensions to link containerized mechanisms, without affecting the size of code and overcoming potential problems concerning principally required libraries and their dependencies. However, all these works focus on specific (i.e. atomic) solutions for performing and managing analytics tasks through serverless offerings.

Our approach goes beyond the typical single isolated function to a chain of functions that have dependencies, exploiting serverless utilities by forming external containerized mechanisms, realizing that functions can reciprocate to any ML task, not at a level of a single function, but in several chained functions.

III. PROPOSED APPROACH

The projected approach is based on high of data analytics and machine learning techniques, including the Apache OpenWhisk serverless platform. The overall objective is the realization of a custom-made chain (custom in terms of being generalized and reflecting the needs of different analytics tasks) of functions in OpenWhisk to build a following-era serverless application; each of the functions addresses a specific machine learning assignment. To this conclusion, the functions incorporate diverse parts of a data analytics pipeline, ranging from the data pre-processing to a pre-trained MLP machine learning model.

The outcome confers a custom-made pipeline of OpenWhisk functions in which users can transfer through a REST API endpoint (which is activated and served through OpenWhisk) their data and receive a result solely in few milliseconds, according to the implementation. The proposed approach's overall conceptual view is cited in the following figure that presents the realization of a pipeline of functions in a serverless ecosystem, through two -2- phases.

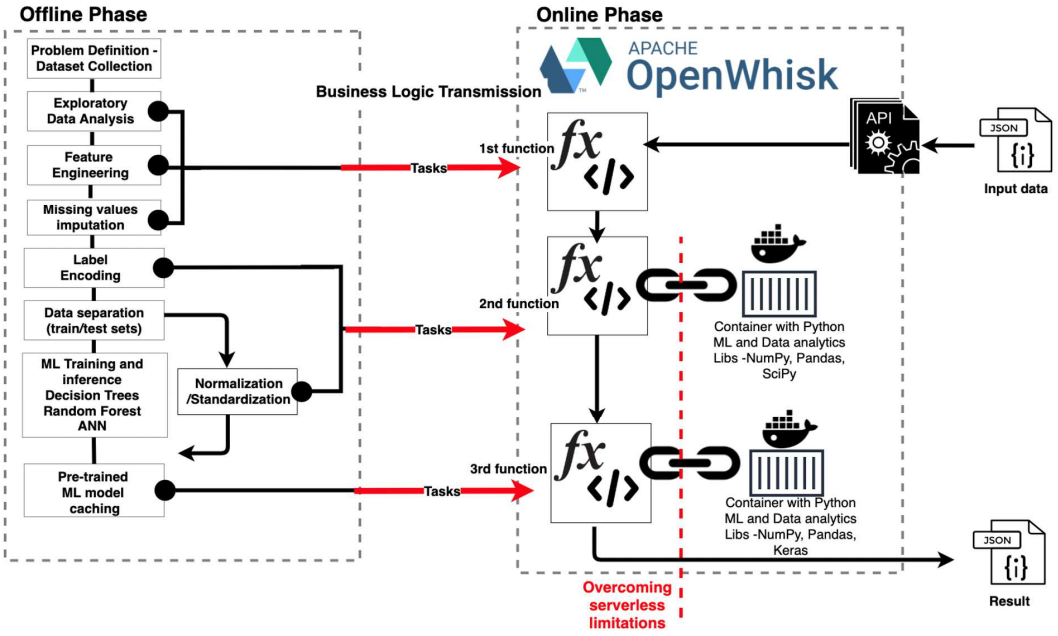


Fig. 1. Two-phase proposed hybrid ML-FaaS approach

A. Offline phase

The offline phase centers on a holistic data analytics and ML/DL procedure related to the respective data scientist's work items. The complete process incorporates the dataset's manipulation, beginning from the appropriate dataset's collection, clarification, and a review of its values to identify hidden patterns, information, and insights. With respect to the critical part of pre-processing, it consists of imputation of missing values, elimination of the non-important features by applying dimensionality reduction techniques, creation of new meaningful features using feature engineering means, formulation of label encoding in categorical data, and processes for scaling and normalization. Essential role forms the enrichment of the dataset in order to improve its performance, as it includes imbalance labeled observations (the vast majority comprises non-fraudulent transactions). Following the dataset's modification, various machine learning algorithms can be applied and evaluated (i.e., Random Forest, Extremely Randomized Trees and Artificial Neural Networks), and the training procedure can be initiated with the support of different techniques such as the Stratify K-fold cross-validation model technique, pointing out that the training materializes with four different data manipulation cases (i.e., applying oversampling techniques and weight class balancing).

Finally, according to the best performance score -taking mind evaluation indicators such as Confusion Matrix, Recall- Sensitivity, Area Under Curve (AUC) -, the final machine learning model shall be selected. Based on the offline phase's main objective, specific steps are included in the overall approach:

- *Definition of the target:* The main objective is to approach a machine learning classifier as a typical ML task in many different pipelines and domains; in particular, our notion has begun with the construction of a ML classifier that is capable to predict -accurately- fraud e-commerce transactions. To this conclusion, the selection of a dataset takes place (we have obtained a dataset with historical transactional data based on e-commerce activities) in order to train the classification model. The result provides as insight the prediction if new unlabeled data forms an authorized or fraudulent e-commerce transaction.
- *Dataset selection and explanation:* In this step, the dataset needs to be obtained. We have selected to examine a specific case-study concerning a dataset acquired from the Kaggle repository which has been supplied by VESTA Corporation and IEEE Computational Intelligence Society (IEEE-CIS), presenting real annotated fraud and legitimate e-commerce transactions [23]; due to the fact that online card fraud reported cases constitute a crucial problem not only in the Financial Sector, but also in Law Enforcement Authorities, as well as the consumer society. The provided dataset is structured with a comma-separated values format and includes four hundred and thirty-five anonymized unique features, both numerical and categorical.
- *Imbalanced dataset problem:* Since the selected data have been pre-processed with various typical techniques, including Exploratory Data Analysis, feature engineering, imputation of missing values, label encoding, normalization/standardization - the chosen ML models are prepared to be provided. One of the actions that can be performed to address the case of imbalanced labeled observations before implementing and utilizing ML/DL algorithms, is to apply various techniques to enrich the dataset with minority class samples. Principally, in this work, we have developed three diverse approaches to address this challenge and assess the performance of the chosen ML models to be used afterwards. Data scientists in this step implement methods for dataset's enrichment, which are widely used from the machine learning community: a) the Oversampling technique, b) the weight assignation to the minority class, and c) the SMOTE technique along with TOMEK LINK [24].

- *Dataset separation and selected CV method:* This step refers to the identification of a separation ratio. Several approaches can be followed, and in the case of the ML pipeline exploited in this paper, we have chosen as the training set to be the 80% of the given dataset and the remaining ones to be the test set (20% of the given dataset to be the evaluation dataset). As it concerns the training set, in order to achieve the generalization, data scientists could apply different methods such as the stratified k-fold cross-validation [25, 26].
- *Implemented ML/DL:* As a final step of the offline phase, the ML/DL approach needs to be implemented. Going beyond a typical simple case, this paper presents an approach for the implementation of neural network that will be later exploited through the FaaS paradigm. Thus, a Multilayer perceptron -MPL- has been implemented by exploiting Keras Deep Learning API [27]. Specifically, we have implemented a feedforward artificial neural network class, including three layers of nodes and one output label that defines the final result. As it concerns the initial three layers: i) we have utilized five hundred units for the surface layer and ii) two hundred and fifty-six units in the remaining following two layers. It is important to note that the algorithm's capacity increases when the number of nodes in each layer is increased. Moreover, we have exploited the Rectified Linear unit (ReLu) function [28] to be the activation function for the first three layers for taking into account the model's nonlinear relationships. We have also integrated the dropout method, which is merely used to prevent a model from the overfitting phenomenon. Following with the output layer: we have utilized the Sigmoid activation function [29] since it fits in the investigated binary classification problem; after a model had been created, the Adam algorithm has been used in order to optimize the learning rate. Regarding determining the probabilistic loss function, and since we had to deal with a binary classification problem and the labels correspond to binary values, we used the binary cross-entropy loss function. Once the above functions and their parameters have been defined, the model can be trained. The input features and the target classes are inserted, as well as the number of epochs to train (in this case: ninety).

B. Online phase

The main objectives in the Online phase involves the designing, formulation, and development of the ML/DL approaches (using various programming languages, libraries, and frameworks – in this case, we have exploited the Python programming language), their "transformation" to functions, and their chaining (i.e., connected actions) in a framework (e.g., OpenWhisk). The chaining includes the usage and configuration of an orchestrator, such as the Docker Compose Orchestrator [30]. The respective OpenWhisk internal components (e.g., NGIX, controller, invoker, Kafka, etc.) are orchestrated as Docker containers by Docker Compose. Regarding the integration of the business logic into OpenWhisk Functions, the implementation of the FaaS services needs to be performed – in this case three connected OpenWhisk actions (a sequence of nanoservices) have been developed. Each function receives -as input- a dictionary structure (JSON format); the parameters are passing into the primary action, the execution runtime is implemented, and an output is produced and returns a result, likewise the input's structure. Concerning the sequence of actions schema, the first action's output will be the input for the second function and so forward until the last action's execution delivers the ultimate result. In our case-study, all of the developed functions are nourished with new unseen data. The assignments of the developed functions are briefly summarized as follows:

- The *first function* refers to data processing procedures, which are typical in a data analytics pipeline scenario. It receives the parameters (in JSON format) and transforms their structure. Additionally, the function performs the appropriate imputation if missing values are appeared, according to the strategy taken during the Offline phase. Furthermore, it attaches new meaningful features according to the feature engineering procedure, which has also been introduced in the Offline phase and forms capitalization to categorical data in order to preserve a consistent data format.
- The *second function* is mainly responsible for the normalization procedure as a next step in a data analytics pipeline. In particular, the function receives the transformed data from the implemented first function's computations, and its main task is to perform encoding to the categorical values and normalization in accordance with what was used in the offline phase. The output parameters constitute the input to the third function.
- The *third function* is dedicated to the pre-trained ML/DL model, and in this case, the neural network. The pre-trained model from the Offline phase is utilized and integrated into the mentioned action. As the runtime is executed, it produces the final prediction (i.e., if the given data/parameters indicate a fraudulent or authorized transaction). The last action's output produces a message (e.g., whether the transaction is fraudulent or not).

C. Overcoming serverless limitations

A key challenge within the Online phase refers to addressing the constraints and limitations of the underlying platforms (in this case OpenWhisk) that enable the FaaS provisioning. To address this challenge, we have utilized external Docker containers mechanisms to include the required ML/DL libraries needed to complete each ML/DL function/action task. This approach has been introduced, since several third- party packages and libraries (e.g. python packages) that improve data performance, surpasses the function's source code limitation (48MB). As for the overall pipeline, several ML/DL libraries are required (e.g., Pandas, NumPy, sklearn, and Keras), and one action that needs to be performed is to evaluate whether these exceed the maximum limit of the serverless framework. It is well-known that almost all the ML/DL libraries are accompanied by numerous shared libraries and compile native dependencies for performance, thus producing hundreds of megabytes of dependencies. To overcome this challenge, we have

developed custom runtime containers; thus, the developer can build a custom container and associate it with the functions without concern the default size limitations.

However, the formation of a suitable Docker container for FaaS utilization must follow a specific process; custom runtime images must execute the function/action interfaces, which is the platform's protocol to pass invocation requests to the runtime containers. Containers are expected to expose a HTTP server (running on port 8080) with /init (initialized) and /run endpoints [31]. Hence, to overcome this limit custom runtime images have been utilized. The runtime pre-installs additional python ML/DL libraries amid the build process and exposes them during invocations. Utilizing custom runtimes with private source files allows developers to execute larger applications on the platform as FaaS. Using this method, any runtime, library, and tool can be utilized. Concerning the investigated case, we have utilized and configured prefixed Docker images, including python libraries, proper for data analytics, and machine learning assignments to form and run our custom OpenWhisk actions.

The overall approach of the online phase is presented in Figure 2. As new data are the input of the pipeline, a HTTP request is initialized towards the core components (i.e. NGIX, controller, CouchDB, load balancer, action container). The user practically makes a HTTP request (by providing the data) through the REST API endpoint - developed similarly within the OpenWhisk environment, which passes through the functions' sequence/pipeline, implements the procedure mentioned above to satisfy the request and afterwards, to produce the result.

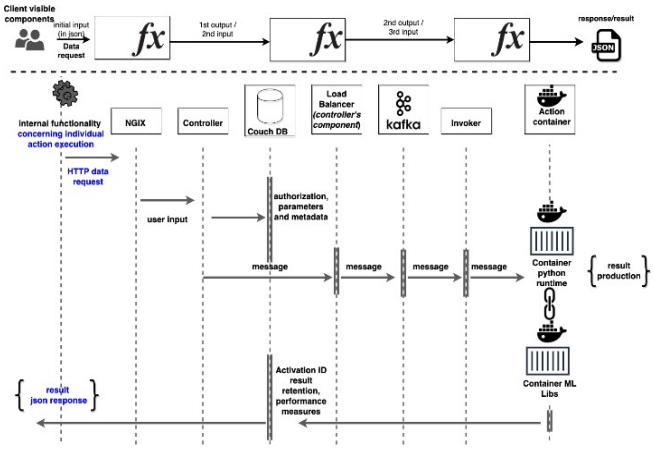


Fig. 2. ML/DL pipelines as serverless functions

IV. EVALUATION

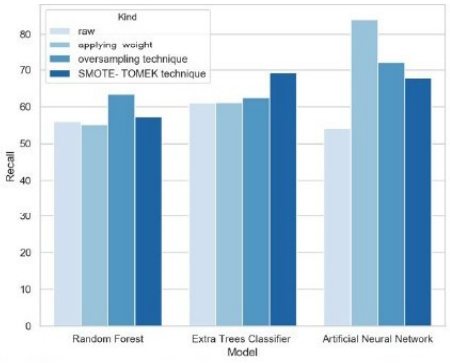
In this section, we are presenting the outcomes of the evaluation regarding the ML/DL models and the FaaS approach that has been followed – e.g. the response time of the implemented pipeline when a request is initialized.

With respect to the ML/DL-related evaluation, the foremost common metric to evaluate the constructed model's classification performance is the accuracy. Nevertheless, this is not a unique way to outline a supervised model's efficiency, performing on a given dataset. Practically, the usage of the accuracy evaluation measure is not always suitable for the designed application; thus, it is mandatory to determine the suitable measure when we are selecting within models and adjusted parameters. By setting an evaluation indicator, we should always consider the ML/DL's objective: going beyond accurate predictions to exploiting these predictions as part of an overall decision-making process. Concerning our case study, and since we have dealt with the imbalanced dataset problem, we have employed measures like sensitivity (recall), AUC, and Confusion Matrix to assess the selected models. To this conclusion, the evaluation results concerning the training phase are presented in Table I. According to these, some cases revealed the overfitting phenomenon in the training phase; the Random Forest -by applying the oversampling technique, the Extremely Randomized Trees - similarly-, and the latter by using the SMOTE along with TOMEK Link method. This implies that the model has been overtrained on the training data, causing a degradation in the unseen test data's performance.

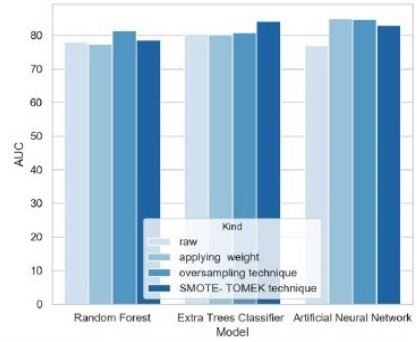
TABLE I. TABLE TYPE STYLES

Model	Recall	AUC
Random Forest *(raw data)*	56.00 %	78.00 %
Random Forest – *(weight to minority class samples)*	55.3 %	77.4 %
Random Forest – *(oversampling 40% minority class)*	63.5 %	81.4 %
Random Forest – *(SMOTE and TOMEK-LINK technique 40% minority class)*	57.3 %	78.6 %
Extremely Randomized Trees *(raw data)*	61.1 %	80.4 %
Extremely Randomized Trees – *(weight to minority class)*	61.2 %	80.2 %
Extremely Randomized Trees – *(oversampling 40% minority class)*	62.5 %	80.9 %
Extremely Randomized Trees – *(SMOTE and TOMEK-LINK technique 40% minority class)*	69.3 %	84.2 %
Keras ANN *(raw data)*	54.3 %	77%
Keras ANN – *(weight to minority class)*	84 %	85 %
Keras ANN – *(oversampling 40% minority class)*	72.1 %	84.8 %
Keras ANN – *(SMOTE and TOMEK-LINK technique 40% minority class)*	68 %	83 %

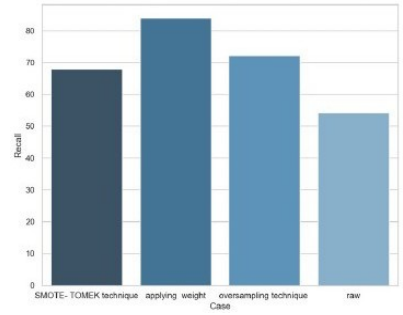
The following figure (Figure 3) provides an overview of the ML/DL performance. Especially it results in the fact that the neural network, by choosing the second data-manipulation case (weight assignation to the minority class observations), has conveyed the foremost high-grade performance by recognizing the 84% of the unseen data correctly. Regarding the FaaS assessment, we have evaluated the overall solution's performance (and the underlying platform) by testing the implemented ML/DL pipeline through several requests and assessing its performance in terms of execution time.



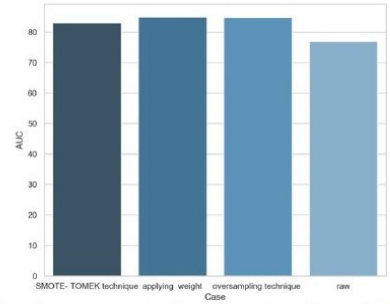
(a) Performance evaluation concerning sensitivity



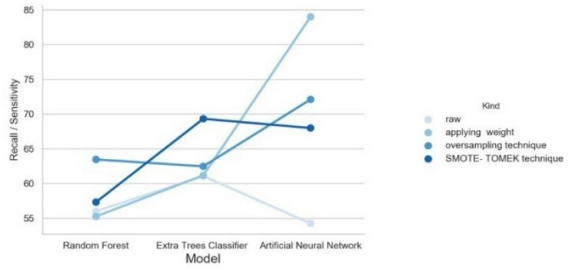
(b) Performance evaluation concerning AUC



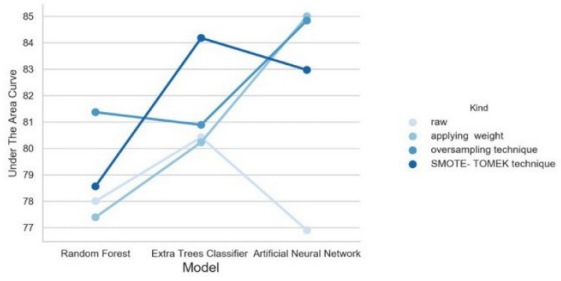
(c) Performance evaluation concerning model sensitivity



(d) Performance evaluation concerning model AUC



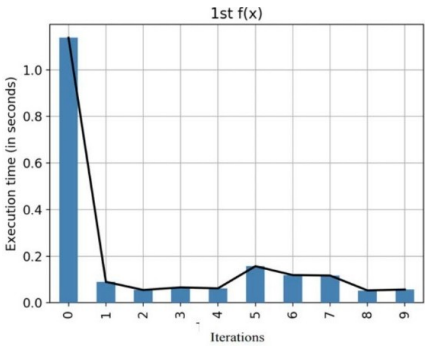
(e) Sensitivity factor plot



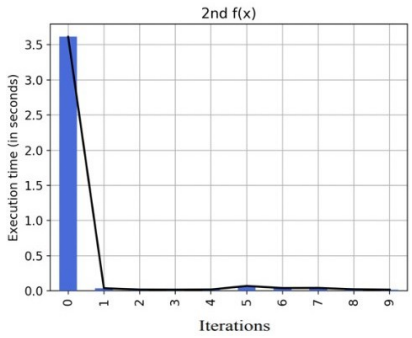
(f) AUC factor plot

Fig. 3. Evaluation outcomes of the ML/DL approach

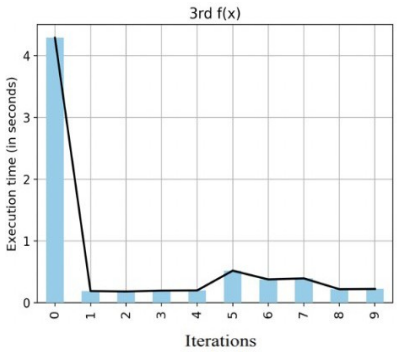
In Figure 4, we are presenting our outcomes concerning the execution performance in terms of response time. Our experiments incorporate ten executions with emphasis on each action execution that has been implemented in a time interval of seconds and milliseconds. Especially, the lowest average execution time has been 53 ms and the highest 1.40 sec for the first action. Concerning the second action implementation, the fastest average execution time has been 20 ms, and the slowest 3.60 sec. The final action, the one that invokes the pre-trained DL model, has attained the shortest execution - average- time in 180 ms, whereas the most degraded performance in 4.30 sec. Remarkably, the execution time of the pipeline has varied in milliseconds. Nevertheless, due to the fact that OpenWhisk suffers from a cold start spot -like other serverless platforms- the first request takes longer to produce the response. According to our experiments, the first request acquires approximately on average 25 sec to produce the response.



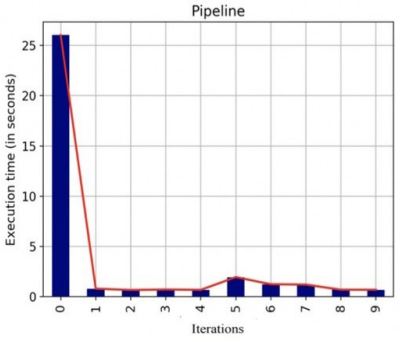
(a) Average time for the first action execution: Data transformation and imputation



(b) Average time for the second action execution: Data normalization



(c) Average time for the third action execution: Data analytics through a neural network



(d) ML/DL pipeline evaluation

Fig. 4. Average service performance through ten individual requests

V. CONCLUSIONS

Towards raising the abstraction level for developers, the computing infrastructure providers focus 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 aim also at exploiting the added-value of serverless computing. In this context, the serverless computing model proposes a novel approach to design and implement following-era artificial intelligence services by fulfilling potential Service Level Objectives (SLOs) such as execution runtime only in a few ms, while at the same time, eliminating the data scientists' necessity to manage the infrastructure level resources.

Therefore, through this paper, we have proposed a means to leverage the serverless architecture for designing and implementing ML/DL pipelines in order to exploit them as services. The proposed approach –which can be applied to any ML workflow- 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 approach 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 raw structured dataset with encrypted features, in order to showcase and highlight the requirement for applying several techniques to pre-process the dataset. Furthermore, various models have been evaluated including random forests, extremely randomized trees and neural networks, each of them in four different data manipulation cases, as the evaluation dataset had suffered from imbalanced labeled observations. In conclusion, we have selected as the best model an MLP by applying weight to minority class samples, reaching 84% recall score. We have also presented an approach for the creation of functions in a FaaS platform, Apache OpenWhisk. These functions have been realized as a pipeline, performing the respective ML/DL tasks. The overall approach has been also evaluated 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 in which several serverless platforms confront, affects the normal latency distribution by giving the response with a delay. It is within our next steps to further extend the pipelining approach with rules and triggers and connected external sources (e.g. IoT devices).

ACKNOWLEDGEMENTS

This research has been co-financed by the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH – CREATE – INNOVATE (project code: DIASTEMA - T2EDK-04612).

REFERENCES

[1] I.Yaqoob, I. A. T. Hashem, A. Gani, S. Mokhtar, E. Ahmed, N. B. Anuar, & A. V. Vasilakos, (2016). "Big data: From beginning to future." International Journal of Information Management, 36(6), 1231-1247.

[2] D. Namiot, & S. Manfred. (2014). "On micro-services architecture." International Journal of Open Information Technologies. 2. 24-27.

[3] I.Baldini, P.Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, ... & P. Suter, (2017). "Serverless computing: current trends and open problems." In Research Advances in Cloud Computing (pp. 1-20). Springer, Singapore.

[4] A.Christidis, R.Davies, & S. Moschoyiannis, (2019, November). "Serving machine learning workloads in resource constrained environments: serverless deployment example." In 2019 IEEE 12th Conference on Service- Oriented Computing and Applications (SOCA) (pp. 55-63). IEEE.

[5] R. Koller, & D.Williams, (2017, May). "Will serverless end the dominance of linux in the cloud?." In Proceedings of the 16th Workshop on Hot Topics in Operating Systems (pp. 169-173).

[6] E. Jonas, J. Schleier-Smith, V. Sreekanti , C. C.Tsai, A.Khandelwal, Q.Pu ... & J. E. Gonzalez, (2019). Cloud programming simplified: "A berkeley view on serverless computing". arXiv preprint arXiv:1902.03383.

[7] AWS — Lambda. https://aws.amazon.com/lambda/

[8] Google Cloud. https://cloud.google.com/functions/

[9] IBM Cloud. https://cloud.ibm.com/

[10] M. Sciabarrà, "Learning Apache OpenWhisk", 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, Inc., 2019. ISBN 9781492046127

[11] Apache OpenWhisk. https://openwhisk.apache.org/

[12] S. Quevedo, F. Merchán, R. Rivadeneira and F. X. Dominguez, "Evaluating Apache OpenWhisk - FaaS," 2019 IEEE Fourth Ecuador Technical Chapters Meeting (ETCM), Guayaquil, Ecuador, 2019, pp. 1-5, doi: 10.1109/ ETCM48019.2019.9014867.

[13] Apache OpenWhisk documentation https://github.com/apache/ openwhisk/blob/master/docs/reference.md

[14] D. Damkevala, R. Lunavara, M. Kosamkar and S. Jayachandran, "Behavior analysis using serverless machine learning," 2019 6th International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, 2019, pp. 1068-1072.

[15] V. Giménez-Alventosa, G. Moltó, & M.Caballer, (2019). "A framework and a performance assessment for serverless MapReduce on AWS Lambda." Future Generation Computer Systems, 97, 259-274.

[16] E. Jonas, & Q. Pu, & S.Venkataraman, & I. Stoica, & B Recht, (2017). "Occupy the cloud: distributed computing for the 99%". 445-451. 10.1145/3127479.3128601.

[17] [17]V. Ishakian, V. Muthusamy and A. Slominski, "Serving deep learning models in a serverless platform," 2018 IEEE International Conference on Cloud Engineering (IC2E), Orlando, FL, 2018, pp. 257-262, doi: 10.1109/ IC2E.2018.00052.

[18] J. Carreira, P.Fonseca, A.Tumanov, A.Zhang, & R. Katz, (2018). "A case for serverless machine learning". In Workshop on Systems for ML and Open Source Software at NeurIPS (Vol. 2018).

[19] H. Wang, D. Niu and B. Li, "Distributed machine learning with a serverless architecture," IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, Paris, France, 2019, pp. 1288-1296, doi: 10.1109/ INFOCOM.2019.8737391.

[20] R. Chard, K. Chard, J. Alt, D. Y. Parkinson, S. Tuecke, & I. Foster, (2017, June). "Ripple: Home automation for research data management." In 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW) (pp. 389-394). IEEE.

[21] N. Bila, et. al, "Leveraging the serverless architecture for securing linux Containers," 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW), Atlanta, GA, 2017, pp. 401-404, doi: 10.1109/ICDCSW.2017.66.

[22] T. Rausch, W.Hummer, V. Muthusamy, A. Rashed, & S. Dustdar, (2019). "Towards a serverless platform for edge AI". In 2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19).

[23] Kaggle, https://kaggle.com/c/ieee-fraud-detection/overview

[24] N.V. Chawla, K. W. Bowyer, L.O. Hall, & W. P. Kegelmeyer, (2002). "SMOTE: synthetic minority over-sampling technique." Journal of artificial intelligence research, 16, 321-357.

[25] S. Raschka, (2018). "Model evaluation, model selection, and algorithm selection in machine learning". arXiv preprint arXiv:1811.12808.

[26] Stratified K-Folds cross-validator, https://scikit-learn.org / stable / modules / generated /sklearn.model_selection.StratifiedKFold.html

[27] Keras the python Deep Learning API, https://keras.io/

[28] A. F. Agarap, (2018). "Deep learning using rectified linear units (relu)". arXiv preprint arXiv:1803.08375.

[29] B.Karlik, & A. V. Olgac, (2011). "Performance analysis of various activation functions in generalized MLP architectures of neural networks." International Journal of Artificial Intelligence and Expert Systems, 1(4), 111-122.

[30] Docker Compose. https://docs.docker.com/compose/

[31] Apache OpenWhisk- Creating and invoking Docker actions, https://github.com/apache/openwhisk/blob/master/docs/actions-docker.md