A Microservice Based Architecture Topology for Machine Learning Deployment

José Lucas Ribeiro, Mickael Figueredo, Adelson Araujo jr, Nélio Cacho, Frederico Lopes
Department of Informatics and Applied Mathematics
Federal University of Rio Grande do Norte
Natal, Brazil
zelucasr@gmail.com, mickaelfigueredo@gmail.com,
adelsondias@gmail.com, neliocacho@dimap.ufrn.br, fred@imd.ufrn.br

***Abstract*—Smart solutions that make use of machine learning and data analyses are on the rise. Big Data analysis is attracting more and more developers and researchers, and at least five requirements (Velocity, Volume, Value, Variety, and Veracity) show challenges in deploying such solutions. Across the globe, many Smart City initiatives are using Big Data Analytics as a tool for doing predictive analytics which can be helpful to human well being. This work presents a generic architecture named Machine Learning in Microservices Architecture (MLMA) that provides design patterns to transform a monolithic architecture of machine learning pipelines in microservices with separate roles. We present two case studies deployed to a Smart City initiative, where we discuss how each component of the architecture applied in specific applications that use predictions with machine learning. Among the benefits of this architecture, we argue prediction performance, scalability, code maintenance and reusability makes such transition a natural trend in Big Data and machine learning applications.**

***Index Terms*—microservices, machine learning, design patterns, recommendation systems, predictive policing.**

I. Introduction

Big data approaches have been used by many Smart City Initiatives to discover new patterns and insights that can improve public and private services. However, with the proliferation of data sources in Smart City initiatives, it is necessary to adopt improved processing approaches to make scalability and maintenance of these initiatives sustainable. At least six requirements of big data (Velocity, Volume, Value, Variety, Veracity and Complexity) bring challenges to extract meaningful information of the data and to the way technologies are being developed. The methods to analyze a big volume of data must be able to extract features we don't know yet, which brings several benefits like a better understanding of patterns, business value and an improved scientific discovery [1]. Also, with the rise of cloud computing [2] as a powerful way of deploying predictive systems that use machine learning, designing new applications requires more appropriate architectural modelling.

A recent trend in the practice of developing Smart City applications hosted in clouds is to use architectures based on microservices [3]. This type of architecture has been adopted by many big tech companies, such as Amazon, IBM and Microsoft, so prominent investments has made it reach a higher degree of maturity in recent years. It consists of dividing a system into a set of small services, highly decoupled, each one performing in its own process and communicating with light mechanisms [4]. Each service has a purpose-specific and well-defined role, focusing on doing a small task [5]. As microservices practices have been developed most in IT industrial environment, Soldani et al. [6] recently suggested that there is a gap between the academic state-of-the-art and industrial state-of-practice.

The fact that microservices are loosely coupled brings several advantages, such as:

- The developer has the freedom to develop and deploy services independently;
- The developer can implement them in different programming languages;
- The application is easily integrated and deployed using container-based and distributed technologies such as Docker or Mesos;
- The code is easier to understand and maintain because each service has isolated responsibilities;
- The application has multiple points of failures and can be implemented to handle them without a crash.

In this work, we propose a generic architecture, named *Machine Learning in Microservices Architecture* (MLMA) for implementing machine learning pipelines by means of separating similar processing steps into smaller services. With such a design, we argue that it improves the performance and code maintenance of machine learning pipelines. Also, the independence between the microservices makes them reusable in different tasks within the same workflow. We have assessed the feasibility of our proposed work by means of two case studies deployed to a Smart City Initiative. The first one is a Recommendation System applied to a Smart Tourism [7] application, in which we adapted the monolithic framework described by Figueredo et al. [8]. The second one is a Predictive Policing [9], following the framework described by Araujo jr et al. [10]. We observe different scenarios, in which incoming requests levels and business logic differ significantly, and we derive a discussion from such differences that are useful when implementing the proposed architecture.

978-1-7281-0846-9/19/$31.00 ©2019 IEEE

II. Related Works

Machine Learning (ML) continues to grow as a powerful tool for problem-solving. However, implementing ML in applications is not an easy task and requires specific knowledge from the developer. To facilitate this process, [11] proposes the technique of encapsulating ML as microservices, which he called REST ML. For this, three algorithms that were considered relevant to the area of internet of things were implemented as microservices. In the end, the author shows that the proposed approach simplifies the implementation of algorithms for ML since it allows the reuse of services and configurations.

With the use of Machine Learning in applications, the lack of computational resources become a problem. However, improved solutions with the implementation of parallelization algorithms gain performance, and this is where the work of [12] goes into detail. In this work, gains are discussed and presented by parallelizing and using services that modularize the layers of the Deep Neural Network (DNN). As a result, the work contributes a framework called SERF, which finds the best parallelization settings for DNN services.

When working with multiple services, challenges arise as to how to properly host those services and what kind of technology to use. In the work of [13], Docker containers are used to host monitoring applications. These containers run on top of a virtual machine created in an OpenStack cloud, which allows the use of resources from several computers at the same time [14]. A processing component that uses Deep Learning to monitor the environment in which the containers was developed and hosted on a separate virtual machine. This component enables the detection of security problems, as we can see in Figure 1. The solution proved promising and functional, but as explained by the author of the solution, the monitoring component needs to be improved.

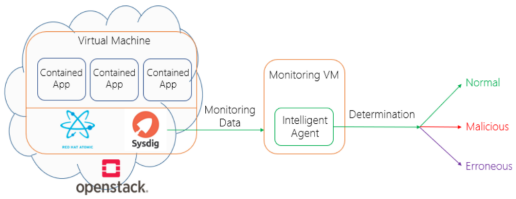

Fig. 1. Monitoring the environment of Docker containers. (Source: [13])

Still following the line of monitoring, the work of [15] monitors the path travelled by typhoons through satellite images. To perform this monitoring, Docker and Kubernetes, an administrator for Docker containers [16], host the application that processes the images and distributes computational resources.

In the proposal of [17], containers were also used to host a solution that makes use of Convolutional Neural Network to monitor the memory integrity of a system. Although the last two papers cited do not work very much on the issue of containers and distributed systems, they show that solutions that use services to classify data and use Docker to control and deploy these services are gaining ground in the literature.

III. Machine Learning in Microservices Architecture Topology

The concept of generic architecture occurs in the face of the great need for artificial intelligence models, always applied in different scales and contexts, to be made available in a simple way for the utilizing services. The generic architecture, represented in Figure 2, was designed to be implemented in any project that wants to make use of data classification, so it would only be necessary to map the classification steps for the services defined in the architecture, from data collection to the prediction stage. The Application Programming Interface (API) of this architecture works by getting data to classify and the model that would be used. In this way, it would be possible to do various types of analysis, such as texts to find out the realm of a sentence, or even real-time image analysis, such as checking if a driver is sleeping behind the wheel.

In the proposed study, resources are managed from an end-to-end perspective, from the extraction of features of the data until the moment of the binary classifications. The resources are managed so that the final task involving artificial intelligence is optimally and scalably reproduced. The generic workflow proposes the creation of microservices based on the following topology that will be described in the next sections.

A. Flow Controller Service and Post Processing Service

The Flow Controller Service and the *Post Processing Service* are the initial services of the architecture and are optional. The purpose of these services is to generate some information from the result of the data classification. In case the *Flow Controller Service* is not used, communication will be made between the **Consumer** and the *Data Collector Service*.

After receiving the result of the classification, the *Flow Controller Service* will request for possible services that are classified as *Post Processing Service*, that will transform the result of the classification into some information that will be returned to the user in the request. We can have more than one service that fits into the *Post Processing Service*, this will depend on the processing required after sorting the data, and the *Flow Controller Service* will integrate the results of these services.

B. Data Collector Service

The purpose of this service is to extract the information, data or content from a well-defined source to place it at the beginning of an analysis process. There is no type of analysis or concepts of artificial intelligence or embedded data analysis at this stage of the process. However, given the wide variety of data possible to work with a generic architecture, it can be said that *Data Collector Service* was thought to be as robust as possible to support the most different contexts.

Data used at this level may vary in type, size, structure, and availability. The first and second situations can be constantly perceived by the mass of data generated by users, such as

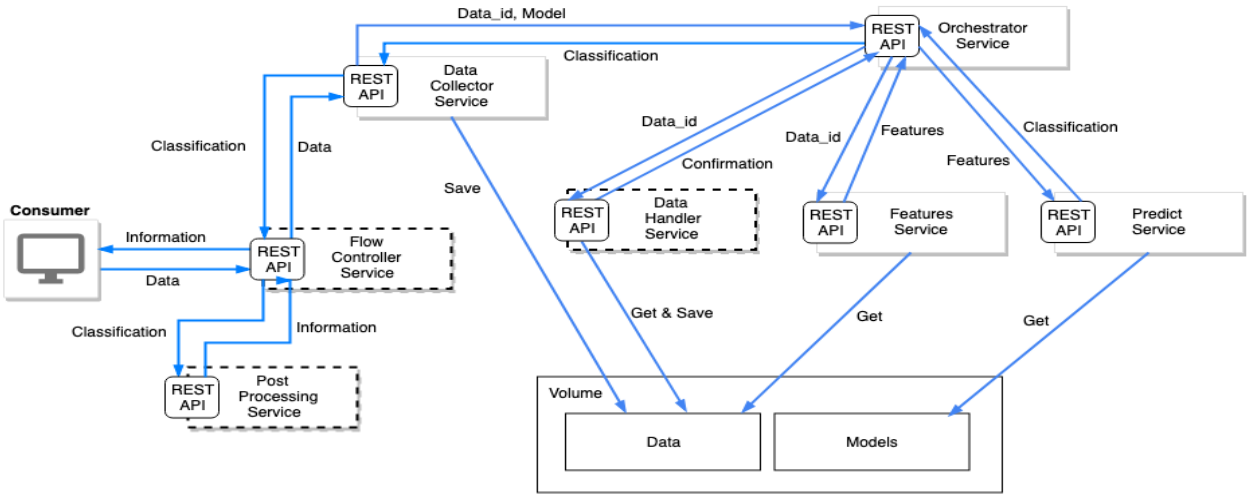

Fig. 2. Proposed Machine Learning in Microservices Architecture to Support Generic ML Pipelines

images, geospatial data, text and time series. In terms of structures, this service should be robust enough to support structured and unstructured data, widespread when the data source is social networks like Twitter.

C. Data Orchestrator Service

The *Data Orchestrator Service* has an intermediate function between data from three different services. As seen in Figure 2, data collection, feature extraction, and classification/prediction services are managed by the *Data Orchestrator Service*.

In this step, the data collected and sent by the *Data Collector Service* are passed on first to the feature extraction service. After extracting the information made by the service, the data is returned in another structure so that it can be used. Thus each data that has the features extracted are processed so that they can serve as input to the next service, which has a variance in its structure in front of the type of classifier used. After the preparation process, the *Data Analysis Service* sends the new data structure created by the classification so that at the end of the process can receive the result of the classification.

D. Data Handler Service

In some cases, it will be necessary to preprocess the data before extract the features, for which the *Data Handler Service* was intended. In this stage of the architecture, some information is not extracted from the data, but cleanup is done that can disrupt the classification or some modification to adapt the data to some specified pattern. For instance, a typical preprocessing step in text processing pipelines is stop-word filtering.

E. Features Service

The feature extraction service, named *Features Service*, is where raw or preprocessed data will be used. Unlike previous services that work with data storage or cleanup, the Feature Service needs to work directly with the data to process them properly [18].

Extracting features in the context of artificial intelligence is the act of extracting information or processing data so that it can be learned more simply by training algorithms. The generic architecture proposed here separates the process of extracting features from the classification. There is no drop in performance in terms of classification. However, in multilabel classification based on binary classification systems, the act of separating the extraction from the classification can represent a relevant classification time gain because we use the *Feature Service* only once, while the data analysis service makes several calls to the *Predict service* using the same features extracted.

F. Predict/Classification Service

The classification service will be responsible for generating the most relevant information for the users of the system at the generic stage of the process. In this case, the result of one classification, predictive or any other machine learning process is created using this service.

A high differential of the generic approach mentioned is the use of the final step of the machine learning process. In many cases, the classifier or predictors structure is maintained for all classes. In this way, what is done in the proposed architecture is the use of the structure of the classifiers to avoid redundancy. Thus, the structure loads new configurations dynamically so that it can perform the output process.

The data structured as a form of features or preprocessed, when applicable, enter this service so that it can come out as a set of classifications. The result is a vector of N values that are related to the number of outputs present in this architecture. The values present in this vector of N positions refer to the adopted approach in the classification, prediction or output process. They can have different data structures as output.

G. Volume

The volume represents the location where the data used in the application will be stored, as well as the model files that will be loaded by the *Predict Service*. Data may be stored in files or a database, and these decisions are fully open to the needs of the application.

IV. Case Studies

In order to assess the proposed architecture, we have implemented two case studies designed for the Natal Smart City Initiative[1]. Natal is a city of Northeastern Brazil that has joined the IEEE Smart City initiative as an affiliated city. This initiative aims to transform Natal into a smart city through the development of systems and applications to bolster the use of IT as means of contributing to improve the life quality of its citizens. Both case studies have already been deployed [8], [10] to Natal by using a monolithic architecture. Next sections describe how we have implemented the same case studies using our proposed microservice architecture.

A. Tourism Recommendation based on Social Media Photos

Systems related to tourism recommendation are known to use different data steps to create a preference profile and generate recommendations. Nowadays, through the advent of social networks and information propagation technologies, diversity of data is generated and can be used to model tourism preferences, and they create a personalized recommendation. However, it is essential that the model adopted to map user preferences be accurate.

The FindTrip Platform[19], [8] recommendation system can use photos of tourist trips to generate recommendations of points of interest to be visited in the city of Natal, Brazil. For this, the recommendation system identifies the types of environments where the photos were taken and models a preference profile using fuzzy logic. In this way, common problems of recommendation systems are avoided. Each step of this process has been modelled within the generic MLA to create a scalable, well-performing framework context, as shown in Figure 3 and shown in the following topics:

1) Data Collector: The first set of microservices is responsible for data extraction. The data extraction comprises two steps. The photos collector step is represented in our generic approach in the data collector step for this application. First, the user performs the log in the user interface using one account/password from one of the three most relevant social media or users devices. The web interface component uses the permissions given through the authentication process to gather the user's photos. The second step comprises starting the respective microservice (i.e., microservice one if the user authenticated using a Facebook account) to download the photo and store in a shared database. The photo collection service can be easily used by other applications that have the main data source social network images supported by the service.

[1]https://smartcities.ieee.org/municipal-partners

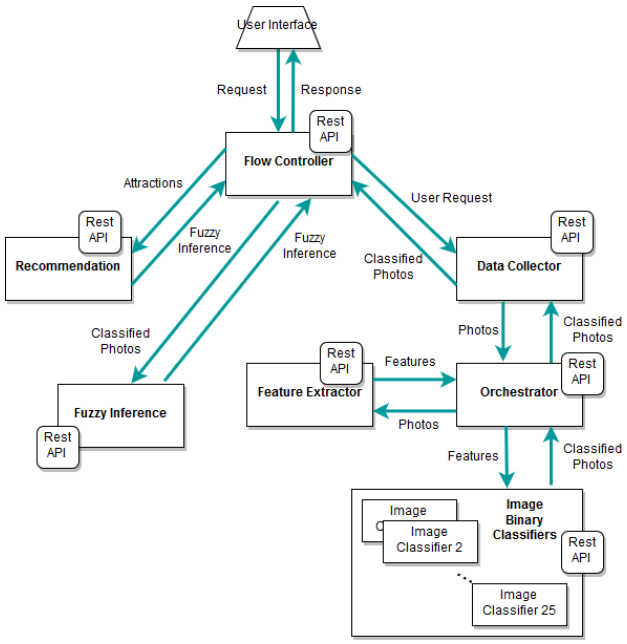

Fig. 3. Application of the generic micro services architecture to the Find Trip recommendation system.

2) Features: The scenario classification step is the first related to profile analyses and most essential to creating the user tourism preferences profile. The scenario classifier is the feature extraction step in the recommendation process described. The main goal of this step is to identify the types of environments most frequented by users. Each photo collected using the Data Collector microservice is sent individually to a set of the binary classifiers. The binary classifiers approach was used to support a scalable and multilabel classification. Instead of using a single classifier that classifies an image into one single class, the adopted approach can classify one image as belonging to several classes using a true or false output. In this way, 25 binary classifiers were created using a deep learning architecture. The great advantage of using the generic architecture at this stage is the ability to add new classifiers dynamically, making the feature extraction step (Figure 2) scalable.

3) Post Processing: A fuzzy approach was adopted to classify the tourist into the five classes through the features extracted from photos. After the user photos analyses, one vector containing 25 features is created using the Scenario Classifier. These features are the basis for the fuzzy rules. The main goal of the fuzzy inference is to infer which of the five tourism classes are relevant for the user based on the scenarios. The fuzzy approach will generate a human logic based preferences map. The idea is relating the tourism classes with the scenarios. In this case, what is crucial for us is to see the membership value for each tourism class. Tourism classes containing high values of memberships are considered relevant for us. This process also becomes scalable due to the generic architecture. For structure modification, rules and

fuzzy outputs can be added dynamically according to the platform's need.

4) Recommendation: The output from the fuzzy inference step is a vector containing five features relating the user with the five classes of tourism created by the Classification Microservice. The recommendation step will relate this output vector with the attractions. To do so, each attraction is modelled by each tourism experts in the universe of the attractions. The Flow Controller will manage the output of this service for applications that seek to consume information. In our topology, the recommendation step in works inside the *Post Processing*, such as the Fuzzy Inference Process, generating an output for the users of the *Find Trip Platform*.

5) Orchestrator: The process to collect the data from the user request, create features based on the photos and fuzzy inference and then create the recommendation are managed by the central service called in our topology of *Orchestrator*. This service is the same used in other applications. In this case, the request contains information about what services will be used in the recommendation process. Them the Orchestrator will create a machine learning workflow to fulfil the recommendation request while other requests from other application are received.

B. Predictive Policing

Allocating police resources in areas of crime concentration has been done with the help of predictive algorithms, e.g. [20], [9]. In a previous paper, Araujo jr et al. [10] described a framework for adjusting machine learning models applied to predictive policing in the context of hotspot detection. The goal is to produce spatial predictions related to crime incidence values for a time interval in the future. In this case, we observe an asynchronous batch prediction structure, given that with each new time interval, new predictions are generated. We map the steps described in this framework to a microservice architecture following the modelling discussed in this paper using the following components.

1) Data Collector: The loading of data sources specific to a given city is done through this service. It has the constraint of having to be implemented individually since each city can store a different data model.

2) Data Handler: The creation of a space grid and the extraction of time series are performed in this service. We divide this service processing step of the data loading process so that an application with another database can reuse the same implementation of this microservice, assuming that primary attributes are present, such as geographic coordinates of the occurrence, its type and corresponding time. One of the configurable aspects of this service is the sampling frequency of the series (which will correspond to the frequency of the prediction).

3) Features: The translation of the space-time points of the occurrences into entries for the machine learning algorithms is done in this service. Araujo jr et al. [10] uses the expression feature ingestion to describe this process, which may include other auxiliary processes such as feature selection and other transformations. Like the Data Handler, the implementation of this microservice is also reusable for applications involving the same task of machine learning in another city.

4) Predict: This service will be responsible for loading a trained model and making the inferences for the next time interval, according to the same sampling frequency considered previously. The result of what has been extracted of features for this next period will be the trigger to generate the predictions in this service.

5) Orchestrator: To manage the sequence of data loading, time series extraction of the spatial grid, feature ingestion and prediction, we have this central service. In addition to being reusable between analogous applications, its implementation must be generic to consider a data preparation service, one of the features and one of prediction. Thus, it can be used in machine learning applications from different domains, such as the recommendation system described above.

6) Post Processing: As described by Perry et al. [9], predictive policing brings more information to a real police operation. In this case, a post processing step described in the framework [10] is the formulation of an optimized patrol program that covers the hotspot identified in the predictions. This program should consider vehicle restrictions and other operating costs in an optimization model to map routes that cover dangerous areas of the city. Thus, this service operates with the inputs constraints and predictions, producing as output the given routes in a separate microservice.

V. Evaluation and Results

The architecture proposed in this work divides the steps of the ML process into microservices and proposes optional components for possible additional treatments in the data. Following the model of components in microservices, we can take advantage of the result of the extraction of features for different classifications, as in the first example of the case study. Thus, one can easily add new classification models for use in Predict Service.

When using multiple microservices, we have some problems like a more complex application deploy and more costs to keep the project active. To solve these problems, solutions such as AWS Lambda and Azure Functions are promising [21], since the environment automatically controls the entire application load balancing and the charge is based on the number of requests or time of resources used.

Table 1 shows the processing times in seconds for the recommendation system steps, where time was calculated as the average of 10 runs. Two approaches were used, the monolithic and the version based on the MLMA architecture in a scalable environment. Clearly, we have a significant gain in the Scene Classifier, which abstracts the steps of extraction of features and classification of the data, this mainly occurs by the use of the result of the extraction of features by several classification services that execute in parallel. The tests were performed on a Dell 7567 computer.

In the case of the application of predictive policing we don't have a runtime comparative analysis yet, but using the MLMA

TABLE I
PROCESSING TIMES FOR EACH STEP OF THE RECOMMENDATION APPLICATION.

Step	Monolitc	MLMA
Scene Classifier	29,863 s	9,848 s
Fuzzyfication	0,241 s	0,247 s
Recommendation	0,079 s	0,071 s

architecture, the ability to process multiple prediction requests simultaneously for possible different users and locations is obtained, for this, it is necessary to label the data used for each specific prediction and maintain the application in a scalable environment like the AWS Lambda. Also,

Upgrading components or adding new optional components to an MLMA-based application are easy activities as we are working with microservices. Project maintainability, in general, is a strong attribute, since each component is independent and well-defined, making its code simple, easy to understand and change.

VI. CONCLUSION

The proposed work aims to present a topology able to improve the deployment of Machine Learning applications in a smart city production context. We presented two cases studies to illustrate the applicability of our approach. First, in the Recommendation System, the topology was essential to improve the feature engineering based in photos classification. The microservices represented the parallelism of the multilabel classification approach used in the system. Besides, a significant gain in system maintenance was noticed through the ability to add new classifiers and steps into the system without much complexity.

On the other hand, we analyzed the application of the topology in the Predictive Policing system from a different perspective. A limited set of police units will interact with the results, unlike a high-demand request platform. Therefore, the processing performance gains in the microservice architecture for this study case are not representative. The main advantages in shifting from the monolithic architecture to microservices, in this case, are related to reusability of code in other cities and the greater ease of separate maintenance of the services that make up the application.

In this way, it is possible to infer that the proposed topology can have different impacts based on the application. This leads us to believe that topology has its advantages but should be evaluated before its actual application. Improvements can be noticed depending on the amount of incoming client requests and complex code workflow.

VII. ACKNOWLEDGEMENTS

This work is supported by SmartMetropolis Project[2]. Nelio Cacho is supported in part by CAPES - Brazil (88881.119424/2016-01).

[2]http://smartmetropolis.imd.ufrn.br

REFERENCES

[1] R. Elshawi, S. Sakr, D. Talia, and P. Trunfio, "Big data systems meet machine learning challenges: Towards big data science as a service," *Big data research*, vol. 14, pp. 1–11, 2018.
[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1721654.1721672
[3] A. Krylovskiy, M. Jahn, and E. Patti, "Designing a smart city internet of things platform with microservice architecture," in *2015 3rd International Conference on Future Internet of Things and Cloud*, Aug 2015, pp. 25–30.
[4] N. Alshuqayran, N. Ali, and R. Evans, "A systematic mapping study in microservice architecture," in *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, Nov 2016, pp. 44–51.
[5] S. Newman, *Building Microservices: Designing Fine-Grained Systems*, 1st ed. O'Reilly Media, February 2015.
[6] J. Soldani, D. A. Tamburri, and W.-J. Van Den Heuvel, "The pains and gains of microservices: A systematic grey literature review," *Journal of Systems and Software*, vol. 146, pp. 215–232, 2018.
[7] Z. Xiang and D. R. Fesenmaier, *Analytics in Tourism Design*. Cham: Springer International Publishing, 2017, pp. 1–10.
[8] M. Figueredo, J. Ribeiro, N. Cacho, A. Thome, A. Cacho, F. Lopes, and V. Araujo, "From photos to travel itinerary: A tourism recommender system for smart tourism destination," in *2018 IEEE Fourth International Conference on Big Data Computing Service and Applications (BigDataService)*, March 2018, pp. 85–92.
[9] W. L. Perry, *Predictive policing: The role of crime forecasting in law enforcement operations*. Rand Corporation, 2013.
[10] A. Araujo, N. Cacho, L. Bezerra, C. Vieira, and J. Borges, "Towards a crime hotspot detection framework for patrol planning," in *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 2018, pp. 1256–1263.
[11] M.-O. Pahl and M. Loipfinger, "Machine learning as a reusable microservice," in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2018, pp. 1–7.
[12] F. Yan, Y. He, O. Ruwase, and E. Smirni, "Efficient deep neural network serving: Fast and furious," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 112–126, 2018.
[13] R. Anderson, "From bare metal to private cloud: Introducing devsecops and cloud technologies to naval systems," 2018.
[14] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "Openstack: toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 3, pp. 38–42, 2012.
[15] M. Lee, S. Shin, and S.-K. Song, "Design on distributed deep learning platform with big data," 2017.
[16] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, no. 3, pp. 81–84, 2014.
[17] S.-H. Choi, "Usability of docker-based container system health monitoring by memory dump visualization," 2017.
[18] S. S. Haykin, S. S. Haykin, S. S. Haykin, and S. S. Haykin, *Neural networks and learning machines*. Pearson Upper Saddle River, 2009, vol. 3.
[19] A. Cacho, D. Estaregue, M. Figueredo, J. Lucas, M. Aurélio, H. Farias, L. Karen, P. Câmara, N. Cacho, F. Lopes, L. M. Filho, and C. Alves, "A smart destination initiative: The case of a 2014 fifa world cup host city," in *2015 IEEE First International Smart Cities Conference (ISC2)*, Oct 2015, pp. 1–6.
[20] G. O. Mohler, M. B. Short, S. Malinowski, M. Johnson, G. E. Tita, A. L. Bertozzi, and P. J. Brantingham, "Randomized controlled field trials of predictive policing," *Journal of the American Statistical Association*, vol. 110, no. 512, pp. 1399–1411, 2015.
[21] M. Malawski, K. Figiela, A. Gajek, and A. Zima, "Benchmarking heterogeneous cloud functions," in *European Conference on Parallel Processing*. Springer, 2017, pp. 415–426.