Machine learning-driven Scaling and Placement of Virtual Network Functions at the Network Edges

Tejas Subramanya and Roberto Riggio
Wireless and Networked Systems
FBK CREATE-NET
Via Alla Cascata 56/C, 38123, Trento, Italy;
Email: {t.subramanya,rriggio}@fbk.eu

Abstract—**Network Function Virtualization is a promising technology that proposes to decouple the network functions from their underlying hardware and transform them into software entities called Virtual Network Functions (VNFs). This approach offers network operators with more flexibility to instantiate, configure, scale, and migrate VNFs at runtime depending on the demand. On introducing these VNFs at the network edges (e.g., base stations), emerging use cases such as connected cars can be supported. However, in such an environment, efficient VNF placement and orchestration mechanisms are needed to address the challenges of continuously changing network dynamics, service latency requirements and user mobility patterns.**

The purpose of this paper is twofold. Firstly, we propose a neural-network model (i.e., a subset of machine learning) that can assist in proactive auto-scaling by predicting the number of VNF instances required as a function of the network traffic they should process. Based on the traffic traces collected over a commercial mobile network, the model achieves a prediction accuracy of 97%. Secondly, we provide an Integer Linear Programming formulation for placing these VNFs at the edge nodes with a primary objective of minimizing end-to-end latency from all users to their respective VNFs. Our results show an improvement in latency by upto 75% when VNFs are placed at the network edges.

Index Terms—**Network Function Virtualization, Machine learning, Proactive Auto-scaling, Virtual Network Function Placement, Multi-access Edge Computing**

I. Introduction

Traditionally, network functions, such as firewalls, gateways, and caches, were deployed as physical devices, where the software was tightly coupled with the proprietary hardware. Mobile Network Operators (MNOs) needed to invest a substantial amount of time to manually deploy, configure and troubleshoot physical network functions resulting in increased CAPEX and OPEX. To address these issues, MNOs have recently started moving towards virtualized and softwarized network infrastructures, generally referred to as Network Function Virtualization (NFV) [1]. In an NFV environment, network functions are implemented as software entities, called as Virtual Network Functions (VNFs), which can run on Virtual Machines (VMs) or containers within commercial-of-the-shelf servers rather than being run on dedicated hardware devices, hence providing agility, flexibility and cost efficiency.

The current LTE networks built around centralized cloud computing architecture, where IT resources (compute, storage, network, etc.) required to host VNFs reside in a few big data centers, cannot meet the demands of emerging use cases such as connected vehicles, virtual reality, and Internet of Things, which requires rapid response, uninterrupted service continuity, and higher data rates. To support these use cases, cloud computing is undergoing a radical shift from the conventional centralized architecture to a rather distributed Multi-access Edge Computing (MEC) architecture by pushing IT resources to edge nodes (e.g., base stations) to facilitate hosting of VNFs closer to end-users [2] [3]. In order to run VNFs on these resource-constrained edge devices, one can preferably use lightweight virtualization technologies (e.g., docker containers, LXC.) that has a smaller footprint compared to VMs [4]. Though placing VNFs near end-users reduces end-to-end latency and alleviates backhaul congestion, due diligence must be employed by MNOs to carefully manage the location (either at network providers data center or at edge nodes) of VNFs by considering edge node capacity constraints, service latency requirements (real-time, near real-time, and non real-time), and varying traffic dynamics [5].

In recent times, one of the main NFV research challenges has been the orchestration and placement of VNFs, mainly due to the dynamics of the emerging traffic patterns in 5G mobile networks. Differently from the centralized cloud, where traffic workload can be fairly predictable, edge clouds will encounter traffic variations based on user distributions and mobility patterns, resulting in non-uniform traffic distribution within the network. Consequently, the number of VNF instances required to handle load changes, to meet performance guarantees, is expected to fluctuate frequently. Towards this end, auto-scaling VNFs is an important mechanism for realizing the promised reductions in operational cost for MNOs. On the other hand, while most of the VNF placement solutions have focused on static VNF placement [6], the problem of placing VNFs in a distributed MEC-NFV infrastructure has not been addressed before in the literature.

Contribution and Research Outcome. In this paper, we first advocate that distributed edge VNFs has to be proactively scaled in synergy with varying network traffic dynamics to avoid service disruption. Based on the scaling decisions, we recommend that VNFs need to be dynamically placed, both at edge nodes and cloud data centers, to offer low end-to-end latency as well as to reduce latency violations. The NFV orchestrator, the VNF Manager (VNFM), and the Virtual Infrastructure Manager (VIM) functional blocks of ETSI NFV

978-1-5386-9376-6/19/$31.00 ©2019 IEEE

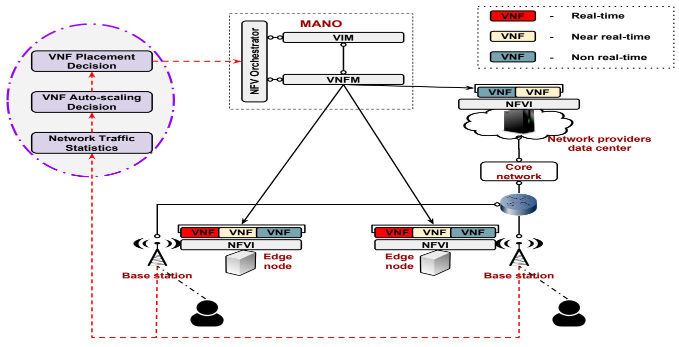


Fig. 1: A high-level distributed MEC-NFV System Architecture.

Management and Orchestration (MANO) [7] are responsible in executing the aforementioned operations, as shown in Fig. 1.

Our main contributions are: (i) we provide a neural-network-based machine learning (ML) classifier model, which estimates the required number of VNF instances at every edge node as a function of the network traffic they should process. (ii) we formulate and solve a realistic VNF placement problem that produces a latency-optimal solution for placing VNFs in distributed MEC-NFV architecture.

Our neural-network model predicts the required number of VNFs with 97% accuracy and guides the placement model to carefully place the VNFs thereby reducing the average end-to-end latency by upto 75%. To the best of our knowledge, we are the first to address the combined challenges in auto-scaling and placement of VNFs within a distributed MEC-NFV environment, based on the commercial mobile network traces collected from a private MNO.

The rest of this paper is organized as follows. Section II describes the related work. Section III describes the proposed neural-network-based ML classifier model and evaluates its performance. In Section IV, we formulate the VNF placement model and perform various experiments in order to evaluate our proposed placement solution. Finally, we conclude the paper in Section V.

II. State of the Art

ETSI NFV Industry Specification Group defines network service as a composition of one or more VNFs that are chained together. Each VNF requires a specific amount of resource to process the traffic flowing through it. To deploy a network service, the operator needs to find the right placement of VNFs complying with various resource constraints and service latency agreements. Once the hosts are selected and the VNFs deployed, resource requirements for the VNFs may vary due to traffic fluctuations. To meet these demands, a resource allocation algorithm is needed that can automatically allocate/release resources to a VNF (vertical scaling) or add/remove one or more VNF instances (horizontal scaling).

Moreover, in comparison to the centralized cloud computing approach, the edge computing framework presents with additional challenges such as dynamically varying workloads, resource-limited edge nodes, and applications with various requirements (e.g., some applications are compute-intensive while others may be network-intensive) co-located on the same edge node. Therefore, MNOs need to address these challenges by introducing proactive, dynamic and efficient NFV orchestration and placement mechanisms.

A. VNF Scaling.

Previous works on VNF auto-scaling can be divided into two categories: reactive mode and proactive mode.

In reactive mode, threshold levels can be either statically pre-defined or dynamically updated. In [8], [9], and [10], the authors propose scalability mechanisms based on static thresholds. They define two threshold levels ($scalein_{thr}$ and $scaleout_{thr}$), to determine if the load reduces below or exceeds above the respective limits, to trigger the scaling process. However, such techniques may result in an oscillating behaviour affecting the overall system performance. On the other hand, [11] and [12] propose mechanisms such as queuing theory and reinforcement learning, which allows to improve the scaling policy based on dynamic or adaptive thresholds. Although it performs better than static approaches, it remains a reactive solution with similar weaknesses.

In proactive mode, forecasting techniques (e.g., machine learning) are applied to allow the systems to automatically learn and to anticipate future needs, based on which scalability decisions are taken. For example, the authors in [13] propose a solution to forecast CPU usage based on a historical dataset using time series model. Other authors such as Mijumbi et al. [14] and Mestres et al. [15] addresses the problem of managing VNF resource fluctuations by predicting resource requirements using ML techniques and thereby enhancing the performance of the resource allocation algorithm.

In contrast to these works which targets data centers, our approach investigates the problem of proactive auto-scaling in a distributed MEC-NFV deployment. Moreover, we use real-operator traffic traces to generate training sets required for predicting auto-scaling decisions, unlike other works that are based on simulated datasets.

B. VNF Placement.

There exists a significant amount of literature on placing VNFs in the NFV infrastructure ([16], [17], [18] and [19]). VNF-P [16] can be considered as one of the most prominent works on VNF placement, where the authors present a generic model for efficient placement of VNFs. In [17], the authors determine the required number of VNFs and their optimal placement in such a way that minimizes network operational costs and maximizes network utilization. In [18], the authors present a real-world implementation of VNF placement on OpenStack [20] to optimize the overall system performance and to provision resilience. The authors in [19] formulate the VNF placement problem as a resource-constrained shortest path problem to minimize the overall latency.

In contrast to other VNF placement solutions which targets traditional VMs in cloud data centers, our work formulates a novel VNF placement model that considers distributed MEC-NFV deployment with a key objective of minimizing the end-to-end service latency for users.

III. Machine Learning-driven Proactive VNF Auto-scaling

In this section, we create an ML classifier model that can identify and exploit hidden patterns in network traffic load instances to predict VNF scaling decisions ahead of time. In particular, we illustrate on the different steps involved in creating our model and eventually evaluate it based on several performance metrics [21].

A. Problem Description

We investigate how to map traffic load statistics X to VNF scaling decisions Y using supervised learning, which involves learning from a training set of data. The traffic load statistics X include measurements from a commercial LTE mobile network. The VNF scaling decisions Y refer to the required number of VNFs in order to process incoming traffic without violating QoS Service Level Agreement (SLA). The details on the composition of X and Y are discussed in Section III-D.

The X and Y metrics evolve over time, influenced mainly by the mobile network traffic dynamics and the active number of mobile users. Consequently, the combined evolution of X and Y metrics is modeled as a time series $\{(x_t, y_t)\}$. Our goal is to determine the distribution of scaling decision metric Y constrained on knowing the traffic load metric $x \in X$.

Employing the statistical learning framework, X and Y are modeled as random variables. We assume that each sample (x_t, y_t) in the training set is obtained from the joint probability distribution of (X, Y). Further, we assume that x_t is multi-dimensional and y_t is one-dimensional (univariate). In this formalism, the inference problem consists of finding a model $F : x \text{ ->} P(Y|x)$ for $x \in X$, so as to maximize the likelihood function $L(\{P(y_t|x_t)\})$, which can be attained by minimizing the loss function $E = -log(L)$ [22].

In this work, a special neural-network called Multilayer Perceptron (MLP) is used to estimate the parameters of the model in order to predict the probability distribution $P(Y|x)$. We select neural-network in our approach for two reasons:

(i) it has proven its potential in identifying traffic patterns due to its effectiveness in predicting time-series problems, whether periodic or not [23] [24].

(ii) It can build new customized features through hidden layers and fit nonlinear activation functions when a definite mathematical definition is not available.

B. Multilayer Perceptron (MLP)

An MLP is a class of feed-forward artificial neural-network, consisting of atleast three layers of nodes (neurons): an input layer, one or more hidden layers and an output layer, as shown in Fig. 2. These nodes are fully interconnected in the form of a directed graph, starting from the input to the output. All nodes except the input nodes have an associated activation function, which is used to compute the node output based on the weighted inputs from other nodes. Usually, relu activation function is used for the hidden layer nodes and softmax activation function is used for the output layer nodes [25]. The output is a vector containing the probabilities that sample $x \in X$ belongs to each class, which is equivalent

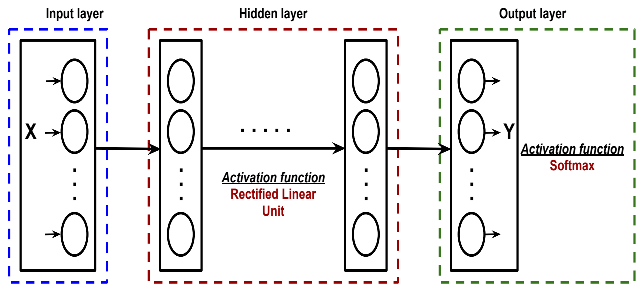


Fig. 2: Structure of the proposed MLP Classifier.

to a categorical probability distribution. The final result is the class with the highest probability.

With a categorical cross-entropy loss function, the network parameters are chosen to minimize the following:

$$E = -\sum_{l=1}^{C} b_{x,l} log(p_{x,l}) \tag{1}$$

where C is the number of classes, b is the binary indicator (0 or 1) whether class label l is the correct classification for input x, and p is the predicted probability that input x belong to class l. Here, a separate loss is calculated for each class label per input and the result is the sum of all those losses.

An MLP model is trained through a backpropagation mechanism using gradient-descent as an optimization algorithm, where the weights between the nodes are adjusted iteratively for minimizing the error function.

C. Modeling MLP in Keras

Keras is an open-source neural-network Python library capable of running on top of Theano [26] or TensorFlow [27]. It is charaterized by a clean, uniform, and streamlined high-level API, allowing users to rapidly define, train and evaluate neural-network models [28].

In Keras, the structure of the neural-network model can be defined in a modular way, as a sequence of standalone and fully configurable modules, which can be readily plugged together. Keras offers a number of predefined neural layers such as a dense layer, a recurrent layer, and a convolutional layer. A wide range of activation functions are also available including relu, sigmoid, softmax, tanh, to name a few. Similarly, a number of predefined loss functions (e.g., mean squared error, cross entropy) and regularization schemes (e.g., dropout) are supported. Also, since Keras performs backpropagation automatically, users do not need to implement it. Moreover, numerous approaches are available to partition the dataset into training, validation, and test sets.

To implement an MLP in Keras, we construct a sequential model with a number of predefined dense layers and their corresponding activation functions. We then configure the learning process of the model by chosing an optimizer, a loss function (equation 1) and a list of metrics to be reported. Lastly, the model is trained with an objective to minimize the loss function and then evaluated.

D. Collecting Data and Feature Engineering

The different steps involved in creating models using supervised learning are as follows:

1) Data Collection: The dataset utilized in this work is generated from a commercial MNO in Armenia by monitoring the mobile network traffic load on 6 LTE base stations, with each base station having 10 cells, for a period of 8 consecutive days. The traces in the dataset are in the form of a time series $\{(x_t, y_t)\}$ and we interpret this time series as a set of samples $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$. The traces are collected on an hourly timescale.

2) Feature Extraction and Class definition: We now describe the input feature sets $X_{default}$ and $X_{constructed}$, which when combined together is referred as X, as well as the output classes/predictions Y.

The $X_{default}$ feature set includes 8 numeric features that are already available in the dataset as described in Table I. In addition to these default features, we construct 9 numeric features ($X_{constructed}$) from the basic dataset, as shown in Table II, using a process called *feature transformation*. These constructed features contain information or patterns on how the traffic load evolves over time, therefore assisting in proactive VNF scaling decisions.

Furthermore, it is necessary to define the desired output classes of the proposed MLP classifier model. Here, class refers to the number of VNF instances required per cell at time t, such that the auto-scaling decision allocates enough resources to meet the traffic demand until next auto-scaling decision at time $t+1$, which is defined in equation 2,

$$No.\ of\ VNFs\ (Y) = min(vnf_{max}, max(\frac{\lambda(t)}{\gamma}, \frac{\lambda(t+1)}{\gamma})) \tag{2}$$

where $\lambda(t)$ and $\lambda(t+1)$ are the traffic load in a cell at time t and $t+1$, respectively, γ is the maximum traffic load a single VNF can handle, and vnf_{max} is the maximum number of VNFs per cell that can be hosted at the edge node. Since, there is a maximum limit on the hosting of VNFs, we model this as a classification problem rather than a regression problem.

Note, we perform auto-scaling decisions once every hour, since our traffic traces are collected on hourly time intervals. However, our model is generic enough to handle lower interval granularities.

Default features ($X_{default}$)
1. Base station ID.
2. Date.
3. Time-stamp t.
4. Average number of users between t and $t-1$ in each cell.
5. Maximum number of users between t and $t-1$ in each cell.
6. Average downlink user throughput in each cell.
7. Average uplink user throughput in each cell.
8. Traffic load measured in each cell at time t, given by $\lambda(t)$.

TABLE I: Default set of features available in the dataset.

Constructed features ($X_{constructed}$)
9. Traffic load measured in each cell at time $t-2$, given by $\lambda(t-2)$.
10. Traffic load measured in each cell at time $t-1$, given by $\lambda(t-1)$.
11. Traffic load measured in each cell at time $t+1$, given by $\lambda(t+1)$.
12. Traffic load measured in each cell at time $t+2$, given by $\lambda(t+2)$.
13. Change in traffic load in each cell from time $t-2$ to $t-1$.
14. Change in traffic load in each cell from time $t-1$ to t.
15. Change in traffic load in each cell from time t to $t+1$.
16. Change in traffic load in each cell from time $t+1$ to $t+2$.
17. Weekday or weekend.

TABLE II: Constructed set of features from the dataset.

3) Feature Subset Selection: This process eliminates the redundant features from $X_{default}$ and $X_{constructed}$ feature sets in order to reduce the dimensionality of the data and also to reduce computational overhead. Therefore, to understand the impact of different features on our ML classifier model, we use *Principal Component Analysis (PCA)* and *Recursive Feature Elimination estimator*. Based on the ranking of these features, we use only 12 features (eliminating $2, 4, 5, 6$ and 7 from Table I) that provides the best accuracy for our model.

4) Dataset Decomposition: Once data is collected and features extracted, the dataset is decomposed into training and test datasets. We use a rule-of-thumb decomposition conforming to $75\%/25\%$ between the training and test datasets, respectively. During the training phase, the MLP classifier model learns the relationship between the features and the classes.

E. Classification using neual networks

Finding the parameters of a neural-network model means searching for the best hyper-parameters of the MLP that can make best predictions on the input. We applied *grid search* and *baby-sitting* as search strategies to perform an extensive search on the space of hyper-parameters in order to find the most accurate neural-network classifier. This process included finding the number of hidden layers and nodes, the batch size, the regularization parameter, the learning rate of the optimizer, and the number of epochs. We encountered the process of finding hyper-parameters time-consuming and hard, which assures that this topic still requires significant research.

We eventually found the architecture of the neural-network that performs best on our traffic load traces and is described as follows. The structure includes one input layer with 12 nodes, three hidden layers with 12, 24 and 12 nodes, respectively, and an output layer with 10 nodes. The regularization parameter used is 0.01, the optimizer is based on stochastic gradient approach with a constant learning rate of 0.001, the batch size is fixed to 100, and the number of epochs equals 300.

F. MLP model evaluation

We consider that edge nodes in proximity to the base stations are capable of hosting VNFs on their NFV infrastructure, similar to containers in data center deployments [29]. We assume the bandwidth capacity of each edge node to be 20 Gbps and each VNF can process a maximum of 200 Mbps traffic without QoS degradation.

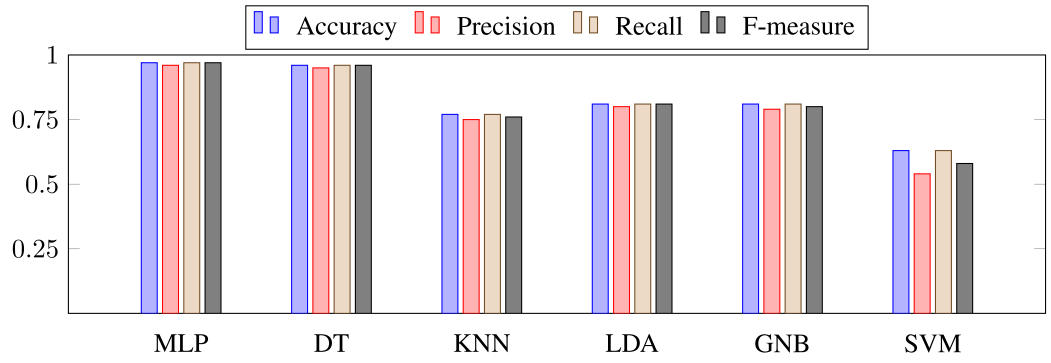


Fig. 3: Performance comparision of different classification algorithms for VNF auto-scaling.

We consider horizontal VNF auto-scaling with each edge node capable of hosting 100 $(20Gbps/200Mbps)$ VNFs and $vnf_{max} = 10$, i.e., a maximum of 10 VNFs can be hosted per cell. These assumptions are derived based on the evaluations performed by authors in [30]. If traffic load increases, additional VNF instances are deployed to meet QoS requirements, whereas if traffic load decreases, VNF instances are removed to save operational expenses.

Once the MLP model is created as discussed before, a test dataset is used to assess the performance of the model in predicting outcomes. The test outcomes can be classified into four groups: True Positive (TP) and True Negative (TN) are when the model correctly predicts actual positive and negative instances, respectively. Whereas, False Positive (FP) and False Negative (FN) are when the model makes incorrect predictions for negative and positive actual instances, respectively. Therefore, we consider four performance metrics to evaluate our MLP model: accuracy, precision, recall, and f-measure, as given by equations 3, 4, 5 and 6, respectively.

$$Accuracy = \frac{1}{|C|}\sum_{i=1}^{|C|}\frac{TP+TN}{TP+TN+FP+FN} \tag{3}$$

$$Precision = \frac{1}{|C|}\sum_{i=1}^{|C|}\frac{TP_i}{TP_i+FP_i} \tag{4}$$

$$Recall = \frac{1}{|C|}\sum_{i=1}^{|C|}\frac{TP_i}{TP_i+FN_i} \tag{5}$$

$$F_{measure} = 2.\frac{Precision.Recall}{Precision + Recall} \tag{6}$$

where C is the number of classes in the MLP model.

Accuracy is the most intuitive performance measure that gives the proportion of true predictions among the total number of predictions observed. However, accuracy is a great measure only if the datasets are completely symmetric i.e., false positives and false negatives are almost the same. Therefore, other performance metrics need to be considered when evaluating a model. *Precision* is a measure of correctly predicted positive observations to the total predicted positive observations. It is a good measure to determine when the cost of FP is high. In case of VNF auto-scaling, a high number of FPs will result in over-provisioning of resources leading to increased operational costs. On the other hand, *Recall* is a measure that calculates how many of the actual positives are captured in our model by labeling it as positive. It is a good measure to determine when the cost of FN is high. In case of VNF auto-scaling, a high number of FNs will result in under-provisioning of resources leading to QoS degradation. Finally, *F-measure* is the weighted average of precision and recall and it is used when there is an uneven class distribution.

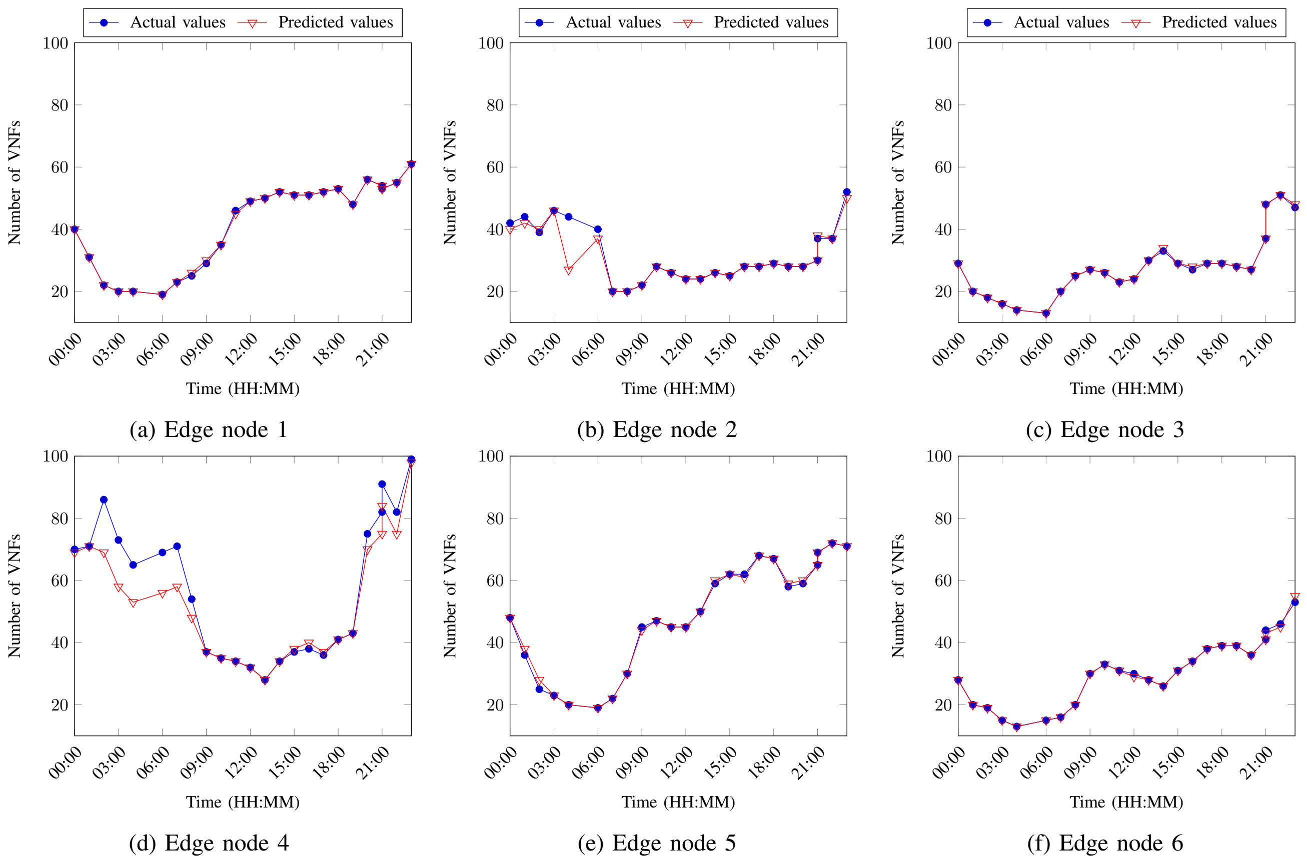
Class	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	1014	1	0	0	0	0	0	0	0	0
2	7	505	8	0	0	0	0	0	0	0
3	0	2	499	0	0	0	0	0	0	0
4	0	0	5	295	2	0	0	0	0	0
5	0	0	0	2	220	8	0	0	0	0
6	0	0	0	0	2	118	5	0	0	0
7	0	0	0	0	0	0	79	0	0	0
8	0	0	0	0	0	0	5	51	1	0
9	0	0	0	0	0	0	0	1	35	0
10	0	0	0	0	0	0	0	0	0	15

TABLE III: Confusion matrix for the proposed MLP classifier model.

Fig. 3 compares the performance of the proposed MLP classifier model implemented in Keras with other classification algorithms implemented in Scikit-learn [31], such as Decision Tree (DT), K-Nearest Neighbour (KNN), Linear Discriminant Analysis (LDA), Naive Bayes (NB), and Support Vector Machine (SVM). We use 6 days of data (i.e., 6 days * 6 base stations * 10 cells * 24 hours = 8640 samples) for training and two days of data (i.e., 2880 samples) for testing. The MLP model outperforms other models in all measures, with 97% accuracy, 96% precision, 97% recall, and 97% f-measure. The closest to MLPs performance was DT with 96% accuracy, 95% precision, 96% recall, and 96% f-measure.

Table III reports the confusion matrix for the proposed MLP classifier model with respect to the test data samples. For example, it can be seen that class 2 has 7 and 8 instances of misclassification as class 1 and class 3, respectively.

Fig. 4 shows the prediction results of VNF auto-scaling (for 1 full day) using MLP classifier model, where we display the prediction performance on all six edge nodes, aggregated over all 10 cells for each base station. In the figure, the blue line represents the actual output generated from the dataset, and



(a) Edge node 1 (b) Edge node 2 (c) Edge node 3

(d) Edge node 4 (e) Edge node 5 (f) Edge node 6

Fig. 4: Prediction results on the number of VNFs required at each edge node based on the proposed MLP model.

Time	MAE (CI, 95%)	Time	MAE (CI, 95%)	Time	MAE (CI, 95%)
00:00	0.5 (-0.16 to 1.16)	08:00	0.33 (-0.07 to 0.74)	16:00	0.16 (-0.16 to 0.49)
01:00	0.5 (-0.16 to 1.16)	09:00	0 (0)	17:00	0 (0)
02:00	3.5 (-1.87 to 8.87)	10:00	0.16 (-0.16 to 0.49)	18:00	0.16 (-0.16 to 0.49)
03:00	2.5 (-2.4 to 7.4)	11:00	0.16 (-0.16 to 0.49)	19:00	1 (-0.6 to 2.6)
04:00	3.16 (-0.95 to 7.29)	12:00	0 (0)	20:00	1.16 (-1.12 to 3.45)
05:00	2.66 (-1.49 to 6.82)	13:00	0.33 (-0.07 to 0.74)	21:00	1.5 (-0.69 to 3.69)
06:00	2.16 (-2.08 to 6.41)	14:00	0.16 (-0.16 to 0.49)	22:00	1.33 (-0.91 to 3.57)
07:00	1.16 (-0.75 to 3.08)	15:00	0.66 (0.01 to 1.32)	23:00	1 (0.28 to 1.71)

TABLE IV: Mean Absolute Error with 95% Confidence Interval for the proposed MLP model.

the red line means the predicted VNF scaling decisions. As we can observe, the MLP classifier model introduced in this study can exactly follow the pattern of actual data, which point out to the strong predicting capability of the model.

Table IV presents the mean absolute error (MAE) between the actual and predicted values of VNF scaling decisions, calculated over all six edge nodes for every hour during the entire day. We also calculate the confidence interval (CI) that determines the 95% likelihood on the range of classification errors that can be expected from our model. As shown in the table, the model can have an upper limit of 8.87 MAE (at time 02:00) and a lower limit of -2.08 MAE (at time 06:00) in predicting VNF auto-scaling decisions.

It is worth mentioning that the predicted scaling decisions from our MLP model is used as an input to evaluate the VNF placement model presented in Section IV.

IV. Latency-optimal VNF Placement Problem in a distributed MEC-NFV environment

In this section, we introduce the definition of parameters used in our system model and formulate a VNF placement problem as an Integer Linear Program (ILP), with a key objective of minimizing the end-to-end latency, by calculating the optimal location for VNFs in MEC-NFV environment.

A. System Model

With the recent advances in cloud computing, any physical server, for instance edge nodes (e.g. base stations, home routers etc) or a distant cloud, can host VNFs. To minimize the end-to-end latency, the MNOs are intending to first place the VNFs on edge devices that are closer to the end users, and once they run out of capacity falling back to host VNFs in the network providers distant cloud data center.

Table V summarizes all the parameters used in the problem formulation. We model the physical network infrastructure as an unidirected graph $G = (N, E, Z)$, where N denotes the set of physical nodes, E denotes the set of physical links connecting these nodes, and Z denotes the users within the network. We assume that each physical node $n_i \in N$ can host one or multiple VNF instances, meaning that all physical nodes have cpu, memory, network capabilities denoted by θ^i. Similarly, each physical link $e_l \in E$ has a physical bandwidth limit denoted by δ^l.

Notation	Definition
$G = (N, E, Z)$	Graph of the NFVI.
$N = \{n_1, n_2, ..., n_i\}$	Set of physical nodes (edge and distant cloud) within the network.
$E = \{e_1, e_2, ..., e_l\}$	Set of physical links in the network.
$Z = \{z_1, z_2, ...z_q\}$	Set of users associated with VNFs.
θ^i	Hardware capacity (CPU, memory, network) of the physical node $n_i \in N$.
δ^l	Capacity of the physical link $e_l \in E$.
d^l	Latency on the physical link $e_l \in E$.
$V = \{v_1^1, v_2^2, ..., v_j^q\}$	VNFs associated to users (e.g. $v_j^q \in V$ is associated to user $z_q \in Z$).
$P = \{p_1, p_2, ..., p_k\}$	All paths in the network.
ψ^j	Required capacity (CPU, memory, network) of the physical node to host VNF $v_j \in V$.
d_{max}^j	Maximum end-to-end latency threshold VNF $v_j \in V$ tolerates from its user.
X_{ijk}	Binary variable denoting if VNF $v_j \in V$ is hosted by physical node $n_i \in N$ using path $p_k \in P$.
b_{ijk}	Required bandwidth between VNF $v_j \in V$ to the user, if the VNF is hosted by physical node $n_i \in N$ using path $p_k \in P$.
d_{ijk}	Required latency between VNF $v_j \in V$ to the user, if the VNF is hosted by physical node $n_i \in N$ using path $p_k \in P$.

TABLE V: Key notations in our model.

Each VNF $v_j^q \in V$ has its own cpu, memory and network requirements denoted by ψ^j. In addition to the compute requirements, each VNF has an end-to-end delay threshold, denoted by d^j, in order to meet the service providers SLA. Similarly, each VNF also specifies a bandwidth requirement along the path p_k from their user, denoted by b_{ijk}, where VNF v_j^q associated with user z_q is hosted on physical node n_i. Another important parameter, the latency from a user to a VNF denoted by d_{ijk}, is calculated by summing up all individual link latency values (d^l) along the path p_k if VNF v_j^q associated with user z_q is hosted by node n_i. Finally, the binary variable $X_{ijk} \in \{0, 1\}$ (given by equation 7) represents the decision variable in our model.

$$X_{ijk} = \begin{cases} 1, & \text{if we assign } v_j \text{ to node } n_i \text{ using path } p_k \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

B. Problem Formulation

The problem addressed in this paper can be formulated as an ILP model that takes as input a set of users U, a set of VNFs V, a set of VNF hosts N, and a latency array d, and ouputs the optimal solution for placing all VNFs in the NFVI by minimizing the total end-to-end latency from all users to their respective vNFs. The formulated objective function is given in equation 8.

$$ILP : minimize \sum_{n_i \in N} \sum_{v_j \in V} \sum_{p_k \in P} X_{ijk}.d_{ijk} \tag{8}$$

The optimization objective is subject to following four constraints:

$$\sum_{v_j^q \in V} \sum_{p_k \in P} X_{ijk}.\psi^j < \theta^i, \forall n_i \in N \tag{9}$$

$$\sum_{n_i \in N} \sum_{p_k \in P} X_{ijk}.d_{ijk} < d_{max}^j.\forall v_j^q \in V \tag{10}$$

$$\sum_{n_i \in N} X_{ijk} = 1, \forall v_j^q \in V, \forall p_k \in P \tag{11}$$

$$\sum_{n_i \in N} X_{ijk}.b_{ijk} < \delta^l, \forall e_l \in p_k, \forall p_k \in P \tag{12}$$

Constraint (9) ensures that the amount of hardware resources allocated to VNFs adheres to the available resource capabilities on the physical node. Constraint (10) ensures that the end-to-end delay between the user and the VNF does not exceed the maximum delay specified by the service. Constraint (11) guarantees that each VNF is hosted exactly by only one physical node (either the edge node or the cloud). Constraint (12) ensures that none of the physical links get overloaded.

Generic applications	Expected latency
Real-time (e.g., Virtual Reality)	$< 5ms$
Near real-time (e.g., Video conference call)	$< 20ms$
Non real-time (e.g., Video streaming)	$< 100ms$

TABLE VI: Latency requirements for generic applications.

C. ILP Model Evaluation

The performance of the VNF placement model is evaluated based on simulation experiments. Real-operator network topology and realistic latency values are used when modeling the simulation environment to produce realistic simulation results, which can better illustrate the benefits of moving VNFs from distant cloud data centers to the network edges.

Simulation Environment: The network topology used is based on the backbone network reported by a private MNO. We introduce the edge nodes at all the base stations of the network topology, with each node capable of hosting a limited number of VNFs without degrading the QoS. This approach of deploying edge nodes to already existing base stations is recommended by ETSI MEC since it is cost-efficient and vendor-agnostic [2]. In addition to the edge nodes, we introduce one cloud data center as part of the network providers infrastructure capable of hosting numerous VNFs.

Depending on the latency tolerance levels of end-user applications [32], VNFs are divided into three categories:

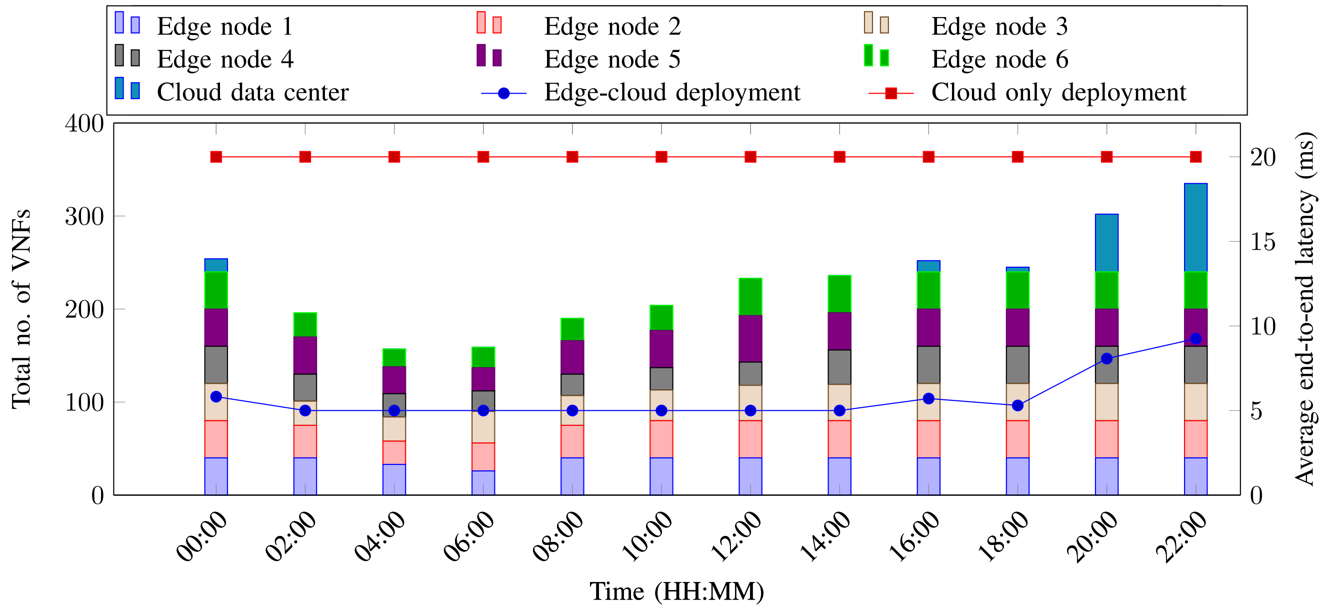


Fig. 5: Performance measure of the proposed system model.

real-time, near real-time, and non real-time, as shown in Table VI. In this experiment, we use equal number of VNFs distributed across all three categories. However, our findings below can be generalized for any categorical combination of VNFs. The latency values for network links are based on the research conducted by Choi et al. in [33], which reports usual average packet delays in wide area networks.

Simulation Results: The model presented in this section was implemented using the commercial IBM ILOG CPLEX solver [34], since no available open-source solvers outperforms CPLEX in terms of speed and capability as reported in [35]. The solver determines the latency-optimal placement for VNFs and calculates the cumulative end-to-end latency (i.e., the objective function) for our problem. We conduct two experiments and compare them to demonstrate the benefits of hosting VNFs at the edge nodes. In the first scenario, all the VNFs are assigned to the cloud data center, while in the second scenario, VNFs are first assigned to the edge nodes (every edge node has an equal but finite amount of compute capacity) and once they run out of capacity, VNFs are assigned to the cloud data center, resulting in higher end-to-end latency.

To perform these experiments, we set a fixed latency of $5ms$ from the user to the edge node. Also, we limit the number of VNFs that can be hosted on each edge node to be 40. Therefore, with 6 edge nodes/base stations, the total edge capacity of the considered network is 240 VNFs, while the rest being assigned to the cloud data center. These numbers are chosen solely to demonstrate the distribution of VNFs between edge nodes and the cloud data center according to the proposed ILP model, which minimizes the end-to-end latency.

Fig. 5, y-axis represents the total number of VNFs predicted over all 6 edge nodes from our MLP classifier model and X-axis represents the predicted time of the day, as illustrated in Section III. In the figure, the line chart compares the average end-to-end latency from all users to their respective VNFs in an edge-cloud deployment (blue line) to that of cloud-only deployment (red line). The cloud-only scenario results in an average latency of $20ms$, whereas running all VNFs at the network edge gives an average latency of $5ms$. Once the capacity of the edge nodes are exhausted, VNFs are assigned to the cloud data center, gradually increasing the average latency. On the other hand, the stacked bar chart in Fig. 5 shows the distribution of VNFs in all the nodes of the network, as scheduled by the ILP solver. We observe that once the total number of VNFs exceed 240, they are automatically assigned to the cloud data center. The ILP solver took 6.25 seconds in order to associate 335 VNFs (e.g., at time = 22:00 in Fig. 5) to the distributed edge-cloud network nodes, so as to minimize the aggregated user to VNF end-to-end latency.

V. Conclusions

The first part of the paper aims at applying machine learning techniques to optimize network management operations. Towards this end, we proposed a neural-network model to facilitate proactive auto-scaling of VNFs, based on the traffic traces obtained from the MNO. We evaluated the proposed model for its effectiveness in accurately predicting the amount of VNF instances required as a function of the network traffic it should process. Moreover, we compared the performance of the neural-network model with five other classification learning methods, and the performance metrics show that the neural-network model can achieve higher prediction accuracy (97%) than other methods. Thereby, allowing MNOs to minimize service downtime and reduce operational costs.

In the second part of the paper, we argue that in order to achieve low end-to-end latency from users to their respective VNFs, it is necessary to host VNFs at the network edges rather than hosting them on the distant cloud data center. We therefore proposed an optimal placement model that carefully chooses the location of the VNFs to reduce user to VNF latency. We evaluated the proposed model using simulations based on real-operator network topology and real-world latency values. Our results show that the average end-to-end latency reduces by 75% when all VNFs are placed at the network edges. Therefore, it is possible for MNOs to provision low-latency services for end users in their network.

Limitations and Future Work: The proposed MLP model is developed on the assumption that distributed edge nodes can send all its collected data to a centralized location, to facilitate detection, classification, and prediction of forthcoming events. However, due to the limited computational and communication resources available in the edge computing framework, it is not practical to send all the data to a centralized location. Thus, it is imperative to determine the model parameters from data scattered across numerous edge nodes, without sending the raw data to a consolidated place. This can be achieved from the federation of edge nodes through a distributed machine learning approach known as Federated Learning [36], which is the main focus of our future work.

Furthermore, in addition to finding the latency-optimal solution to place VNFs in a distributed MEC-NFV architecture, we plan to extend our ILP model with an added goal of optimizing the overall resource utilization (e.g., CPU, RAM, network) in the distributed edge nodes.

Acknowledgements

This work has been performed in the framework of the European Unions Horizon 2020 project 5G-CARMEN co-funded by the EU under grant agreement No 825012. The views expressed are those of the authors and do not necessarily represent the project. The Commission is not liable for any use that may be made of any of the information contained therein.

References

[1] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, 2016.

[2] T. Subramanya, G. Baggio and R. Riggio, "lightMEC: A Vendor-Agnostic Platform for Multi-access Edge Computing," in *Proc. of 14th International Conference on Network and Service Management*, Rome, Italy, 2018.

[3] A. Manzalini, R. Minerva, F. Callegati, W. Cerroni, and A. Campi, "Clouds of virtual machines in edge networks," *IEEE Communications Magazine*, vol. 51, no. 7, 2013.

[4] R. Riggio, S. N. Khan, T. Subramanya, I. G. R. Yahia, D. Lopez, "LightMANO: Converging NFV and SDN at the Edges of the Network," in *IEEE/IFIP Network Operations and Management Symposium (NOMS 2018)*, Taipei, Taiwan, 2018.

[5] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. Soong, and J. C. Zhang, "What will 5g be?" *IEEE Journal on selected areas in communications*, vol. 32, no. 6, 2014.

[6] M. F. Bari, S. R. Chowdhury, R. Ahmed and R. Boutaba, "On orchestrating virtual network functions," in *Proc. of 11th International Conference on Network and Service Management*, Barcelona, Spain, 2015.

[7] ETSI, "Network Functions Virtualisation (NFV); Management and Orchestration," , ETSI GS NFV-MAN 001, 2014.

[8] S. Dutta, T. Taleb, and A. Ksentini, "Qoe-aware elasticity support in cloud-native 5g systems," in *Proc. of International Conference on Communication*, Kuala Lumpur,Malaysia, 2016.

[9] G. A. Carella, M. Pauls, L. Grebe, and T. Magedanz, "An extensible autoscaling engine (ae) for software-based network functions," in *Proc. of International Conference on Network Function Virtualization and Software Defined Networks*, California,USA, 2016.

[10] M. K. Mohan Murthy, H. A. Sanjay and J. Anand, "Threshold Based Auto Scaling of Virtual Machines in Cloud Environment," in *Proc. of IFIP International Conference on Network and Parallel Computing*, Berlin, Germany, 2014.

[11] C. H. T. Arteaga, F. Rissoi, and O. M. C. Rendon, "An adaptive scaling mechanism for managing performance variations in network functions virtualization: A case study in an nfv-based epc," in *Proc. of 13th International Conference on on Network and Service Management*, Tokyo, Japan, 2017.

[12] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *Journal of Grid Computing*, vol. 12, no. 4, 2014.

[13] A. Bilal, T. Tarik, A. Vajda, and B. Miloud, "Dynamic cloud resource scheduling in virtualized 5g mobile systems," in *Proc. of IEEE GLOBECOM*, Washington, USA, 2016.

[14] R. Mijumbi, S. Hasija, S. Davy, A. Davy, B. Jennings, and R. Boutaba, "Topology-aware prediction of virtual network function resource requirements," *IEEE Transactions on Network and Service Management*, vol. 14, no. 1, 2017.

[15] A. Mestres and et. al, "Knowledge-Defined Networking," *ACM SIGCOMM Computer Communication Review*, vol. 47, no. 3, 2017.

[16] H. Moens and F. D. Turck, "VNF-P: A model for efficient placement of virtualized network functions." *Proceedigs of the 10th International Conference on Network and Service Management (CNSM)*, 2014.

[17] M. F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, O. C. Muniz and B. Duarte, "Orchestrating virtualized network functions." *IEEE Transactions on Network and Service Management*, 2016.

[18] S. Oechsner and A. Ripke, "Flexible support of VNF placement functions in OpenStack," in *Proc. of 1st IEEE Conference on Network Softwarization (NetSoft)*, London, United Kingdom, 2015.

[19] B. Martini, F. Paganelli, P. Cappanera, S. Turchi and P. Castoldi, "Latency-aware composition of virtual functions in 5g," in *Proc. of 1st IEEE Conference on Network Softwarization (NetSoft)*, London, United Kingdom, 2015.

[20] "Openstack." [Online]. Available: https://www.openstack.org/

[21] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, no. 1, p. 16, 2018.

[22] N. M. Nasrabadi, "Pattern recognition and machine learning," *Journal of electronic imaging*, vol. 16, no. 4, p. 049901, 2007.

[23] S. Basterrech, G. Rubino, and V. Snášel, "Sensitivity analysis of echo state networks for forecasting pseudo-periodic time series," in *2015 7th International Conference of Soft Computing and Pattern Recognition (SoCPaR)*. IEEE, 2015, pp. 328–333.

[24] F. A. Gers, D. Eck, and J. Schmidhuber, "Applying lstm to time series predictable through time-window approaches," in *Neural Nets WIRN Vietri-01*. Springer, 2002, pp. 193–200.

[25] A. K. Jain, J. Mao, and K. M. Mohiuddin, "Artificial neural networks: A tutorial," *Computer*, vol. 29, no. 3, 1996.

[26] "Theano." [Online]. Available: http://deeplearning.net/software/theano/

[27] "Tensorflow." [Online]. Available: https://www.tensorflow.org/

[28] "Keras 2018." [Online]. Available: https://keras.io/

[29] S. F. Piraghaj, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, "A framework and algorithm for energy efficient container consolidation in cloud data centers," in *2015 IEEE International Conference on Data Science and Data Intensive Systems*. IEEE, 2015, pp. 368–375.

[30] S. Rahman, T. Ahmed, M. Huynh, M. Tornatore, and B. Mukherjee, "Auto-scaling network resources using machine learning to improve qos and reduce cost," *arXiv preprint arXiv:1808.02975*, 2018.

[31] "Scikit-learn." [Online]. Available: https://scikit-learn.org/

[32] GSMA-Intelligence, "Understanding 5G: Perspectives on future technological advancements in mobile," *Technical Report*, 2014.

[33] B. Y. Choi, S. Moon, Z. L. Zhang, K. Papagiannaki, and C. Diot, "Analysis of point-to-point packet delay in an operational network," *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 51, no. 13, 2007.

[34] "IBM ILOG CPLEX Optimizer." [Online]. Available: https://www.ibm.com/analytics/cplex-optimizer

[35] S. N. Laboratories, "Comparison of Open-Source Linear Programming Solvers," *Technical Report*, 2013.

[36] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.