Exploring ML methods for Dynamic Scaling of beyond 5G Cloud-Native RANs

Akrit Mudvari, Nikos Makris, Leandros Tassiulas
Dept. of Electrical Engineering, Yale University,
New Haven, CT, USA
Email: akrit.mudvari@yale.edu, nikolaos.makris@yale.edu, leandros.tassiulas@yale.edu

ICC 2022 - IEEE International Conference on Communications | 978-1-5386-8347-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICC45855.2022.9838562

***Abstract*—As the containerization of network services is expanding towards the Radio Access Network (RAN), the operators seek to benefit from the paradigm of cloud-native services through a wide ecosystem of practices that are applicable to such resources. Such practices include the dynamic scaling of the services in response to the demand, with the network service being assigned more/less resources, or replicated, for accommodating the incoming demand. In such cloud-native environments, proactive decisions can be accomplished through Machine Learning models, which are efficiently trained for specific metrics that reflect the network demand. In this work, we use a real cloud-native telecommunications network and real traffic patterns, and evaluate four different Machine Learning methods for predicting the incoming demand. The decisions made based on the predictions regard the scaling of the base station (gNB/eNB) and the core network entities that deal with the User-Plane traffic (UPF/SPGW-U). Our results show that higher accuracy for such predictions can be accomplished using the tree-based methods over the Neural Network-based solutions, when each method is used for making accurate pro-active decisions for scaling of the under-study network functions.**

***Index Terms*—5G network, cloud-native, auto-scaling, Machine Learning, Kubernetes, OpenAirInterface**

I. Introduction

5G networks integrate several technologies for easing the deployment, rolling out of updates, and decision management for network operators. Such functionality is enabled through the adoption of softwarized entities across the entire stack, initially for the Core Network (CN) but slowly expanding towards the Radio Access Network (RAN). The adoption of such software network functions, enabled through the deployments of edge infrastructure, drives the instantiation of the telecommunication network through cloud-native micro-services. As a matter of fact, the 5G CN has been designed to support all its functions as cloud-native, through the realization of the Service Based Architecture (SBA) [1].

As network functions move to a cloud-native setup, several cloud-driven tools can be used for adapting them according to the demand at a given time period. Such dynamic adaptation of the networking resources can prove to be very beneficial for the overall network efficiency, while the network adapts according to the user load that it receives. For example, in the case of the 5G network, the User Plane Function (UPF) that is in charge of transmitting user data to the respective Data Network (DN) can dynamically scale out during high traffic loads, offering better response and latency times for the users connected to the network. Similarly, a 5G cell running as a software function (e.g. a Cloud-RAN instantiation) can dynamically use more resources as the transmitted network traffic increases, and thus some intensive tasks like signal encoding can become bottlenecks for the network operation.

A key factor to decide on such allocations in the network is the prediction of the demand; Machine Learning (ML) solutions come to the forefront to provide accurate predictions based on recorded metrics. The selection of such metrics is not apparent though and not the same for all the functions. Different metrics at the base station level might not reflect the exact conditions at the RAN, and usually need to be combined together with others in order to truly reflect the pressure under which the network is placed. Similarly, the selection of the ML method to predict the different metrics is not apparent. Although off-the-shelf solutions like Long Short Term Memory (LSTM) Recurrent Neural Networks (RNNs) can provide accurate predictions for some metrics, the metric volatility can produce significant errors in the predictions.

In this work, we focus on a cloud-native cellular network, with the CN and the RAN realized using micro-services with the OpenAirInterface [2] platform. By exploiting the Kubernetes framework, we use an experimentally driven approach with the following contributions:

- To determine meaningful metrics, specific to the Telecom network, that reveal the actual network utilization for the RAN or the CN.
- To evaluate different Machine Learning algorithms on their accuracy of prediction for the different under-study metrics.
- To jointly define a prediction approach for the different components in the end-to-end network.
- To exploit the predictions for defining a proactive scaling mechanism for the RAN and the CN functions, that meets the network demand while achieving efficiency of resource utilization.

Towards accomplishing the last goal, we use the Horizontal and Vertical Pod Autoscalers (HPA/VPA) supported in the Kubernetes framework. The VPA is used for determining the scaling decisions for the base station, while we employ the HPA for the CN case.

II. Related Work

The application of Machine Learning (ML) for network optimization and reconfiguration based on the predicted demand has been identified as key for the progress beyond 5G networks. Several works have emerged in this context, such

This work was supported by the National Science Foundation under Grant CNS 2112562 and the Naval Research Laboratory under Grant N00173-21-1-G006.

as in [3], where the authors employ ML for the allocation of resources in the cellular network, in a vehicular environment. Industry efforts are also steering towards the adoption of a ML driven RAN, through the specification of interfaces around the Open-RAN ecosystem, and the definition of the Radio Intelligent Controller (RIC) platform that allows for near-realtime decisions for the RAN operation [4]. The identification of the suitable ML model and approach depends on the values that need to be predicted. In [3], authors use a Convolutional Neural Network (CNN), a Deep Neural Network (DNN) and a Long-Short Term Memory (LSTM) model, to predict traffic and establish flows in an SDN controller managing the network, and evaluate it in terms of training cost and prediction accuracy. In [5], authors employ three learning models (supervised, unsupervised and reinforcement learning) for efficiently allocating slices over cellular network infrastructure. In [6] the fundamental concepts of supervised, unsupervised, and reinforcement learning are presented, and how ML can contribute to supporting each target key performance indicator per each slice in the 5G network is discussed. Authors in [7] present a conceptual model for 6G networks and show the use and role of ML techniques in each layer of the model. Supervised and unsupervised learning, Reinforcement Learning (RL), Deep Learning (DL) and Federated Learning (FL) are examined. Similarly, authors in [8] propose employing ML for supporting specific functions of applications for 5G networks, including cognitive radios, massive MIMO and heterogeneous networks.

Although the previous works highlight the application on ML for different tasks, they approach the problem from a holistic view, instead of optimizing specific services, and in many cases fail to address the decisions that should be taken post the predictions. In [9], authors use state-of-the-art ML methods to infer location, social behaviour, and traffic demand through a cloud-edge computing framework. Using this knowledge, they focus on exploiting and integrating the demand predictions for proactive optimizations, including load balancing, mobile edge caching, and interference management. In [10], authors focus on a scaling algorithm based on Control Theory for the Access and Mobility Function (AMF) of the 5G Core Network (5G-CN), optimizing the Control Plane (CP) operation of the cellular network. They present a scaling algorithm for the AMF function of 5G-CN, balancing traffic among the replicas for achieving higher user admissions to the network under high load. In [11] their solution is extended with a ML approach using a LSTM model for predicting peaks in traffic, and proactively scaling the AMF for absorbing future incoming traffic. Both solutions are evaluated using the 4G equivalent of the AMF, the Mobility Management Entity (MME). Authors in [12] evaluate three ML models (linear regression, LSTM, RNN) for studying behaviour information estimation (e.g., anomalies in the network traffic) and network load prediction. For the prediction of network load, three different models are used to minimize the mean absolute error, which is calculated by subtracting the actual generated data from the model prediction value.

In this work, we focus on the User Plane (UP) part of the 5G network, with respect to the functions of UPF (User Plane Function) and the actual cell for RAN access. Given the observed traffic patterns, we evaluate the efficiency of four different ML models for predicting the load under which the network functions are placed. Based on this load, we determine a proactive scaling mechanism for the functions, allowing the network to operate smoothly even under high demand. The scaling decisions for the CN are horizontal, enabling the replication of the service to multiple replicas that serve concurrently the attached UEs. Moving beyond existing state-of-the-art, and based on the fact that the base station is running in a virtualized environment, we propose scaling the base station as well in a vertical manner, meaning that more computational resources are added to the VNF, as under load several bottlenecks in scheduling/decoding exist [13].

III. System Model

In this section we present the system architecture and the ML approaches that are further evaluated with respect to the accuracy of predictions for different metrics, used for determining the scaling decisions for the network.

A. System Setup

Our target system for optimizing the scaling decisions is a Telecom network, provided through the OpenAirInterface (OAI) platform [2], and the respective micro-service implementation of the RAN and CN [14]. OAI is an all-software based implementation of the 4G and 5G RAN and CN, realizing the network over commodity equipment using a Software Defined Radio (SDR) front-end. Since the 5G-CN implementation is at the time of writing not mature yet, we focus on the LTE implementation. The CN is realized using the Control and User Plane Separation (CUPS) architecture, allowing each function to run as a separate micro-service. The OAI implementation of the CUPS architecture is providing the Home Subscriber Service (HSS), the Mobility Management Entity (MME), a Service/Packet GW for CP (SPGW-C), a Service/Packet GW for UP (SPGW-U), and a Cassandra-based database. We focus on scaling the RAN eNB function, and the SPGW-U for UP operations. The scaling cases are equivalent to the 5G implementation, without changing the monitored metrics as follows: instead of the eNB we scale the gNB, and instead of the SPGW-U we scale the UPF. Therefore, the solution is directly plug-able to any 5G system, just by changing the deployed functions.

Components of the network are containerized (as docker microservices) and this includes the RAN function (eNB) which only requires access to an SDR-based front-end (i.e. network-based or USB-based). The network is instantiated through Kubernetes (K8s), used for orchestrating and scaling the OAI services in our system. We employ horizontal scaling, realized by the Horizontal Pod Autoscaler (HPA) which spawns replicas of the service and redirects traffic to the pool of replicas, or vertical, realized by the Vertical Pod Autoscaler (VPA) which assigns more/less resources to a container.

We deploy the Prometheus Operator [15] within our K8s cluster, for the collection of metrics in real-time. Prometheus is a monitoring stack registering with the custom metrics API

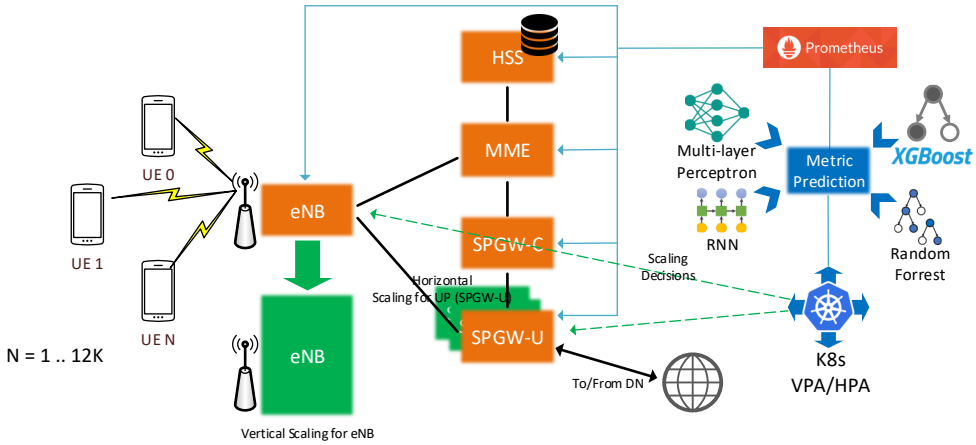


Fig. 1: System architecture for the under-study cloud native network: telemetry for the collected metrics is provided by Prometheus, and data is managed with different ML implementations; decisions on the number of replicas (in green color) are applied through the Kubernetes HPA and VPA.

of K8s, enabling the K8s services such as the HPA/VPA to take advantage of extended metrics from the deployed services. To accurately measure the metrics pertaining to the scheduling decisions and the load of the cell on the radio side, we deploy and integrate in Prometheus the FlexRAN [16] platform. FlexRAN provides real-time information and statistics from the eNB/gNB, through a REST API. The API provides existing configuration settings, and lower level statistics such as bytes sent through the different layers of the stack, retransmissions, etc.. Figure 1 our system-level cloud-native RAN architecture.

B. Machine Learning approaches

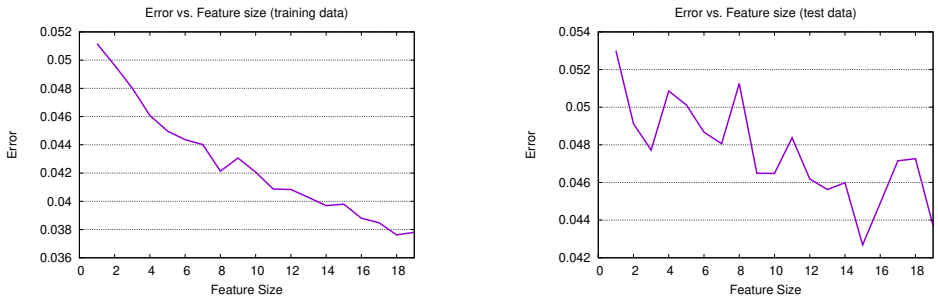
The input metrics to the scaling algorithm are discrete time-series data scrapped by Prometheus over the past time steps, and the output are the accurate predictions of the number of connection instances in the present time. Towards making this decision via a ML-based approach, there are some fundamental questions that need to be tackled, including *'which ML method is the best-suited to accomplish the learning task?'* and *'what is the importance of each of the input data point used to train and implement the scaling task?'*.

The relationship between the cellular network traffic, across control and user plane, and the demand for new connections served over the network should have certain basic features that could be intuitively predicted. We can expect higher user activity and hence higher demand to result in higher traffic in the network in certain cases. Similarly, the fluctuating nature of the network traffic could be indicative of changing demands. For instance, if we observe a decreasing trend in data plane traffic, it might be a result of user activity decreasing and a subsequent reduction in demands in a near future. In other words, there is a temporal aspect that a predictive mechanism can exploit. Such a relationship between the indicative variables and the required predictions, in a data-driven paradigm, would suggest we also explore methods that have shown results in solving problems using temporally related data. These intuitions and survey or literature led us to choose four different methods for analysis: Multi-layer perceptrons [17], LSTMs [18], Random Forest [19], and XGBoost [20]. The choice of LSTM and XGBoost were based on their recent success in various fields in predicting similar data to the ones we are working with, while the other methods are added to see if simpler approaches could give us equally good or better predictions. In each of the cases, the input is a vector of relevant network traffic volumes collected over a certain time, while the output is the predicted number of connections in the future. Specifics of the relevant network traffic statistics we decided to use in each case are discussed in Section IV, for each of the two cases (SPGW-U and eNB) that we investigate. The ML algorithms are executed to generate output (predicted number of connections) at each time point, i.e. for the SPGW_U for each time step t the output is the predicted number of connections accessing the SPGW_U at the time t, while the input is the volume of traffic generated in the multiple past time steps (i.e, $t-3, \ldots.., t-12$). Therefore, the input is a vector $([x_{t-12}, x_{t-11}, \ldots.., x_{t-3}])$ of a certain size (i.e. 10 in this example), containing the relevant network traffic volumes x at a time $t-n$ where $n \in \{3, ..., 12\}$ (in the example above with 10 past time steps). During the supervised training steps, the predicted number of connections was measured against the actual connection number.

1) Neural Network-based Approaches: The recent advancements in the field of Artificial Neural Networks (ANNs), alongside the developments in the field of parallel computing, have pushed ANN learning methods to the forefront. In our work, we investigate two variants: the Multi-layer Perceptron and LSTM.

Multi-layer Perceptron: One of the methods implemented in our work is the multi-layered perceptron, a widely used variant of ANNs. Here, the different traffic metrics scrapped from the network are collected and served as the input to this Deep Neural Network (DNN) architecture and the neural networks are trained to predict the future traffic demands.

LSTM: As discussed earlier, heuristically, the nature of the traffic data being scraped from our system is recognized as having a temporal correlation with the traffic data that are being collected in the past and the present being indicative of the future demands. Such a dataset is well-handled by RNN [21], a NN implementation that captures the relation between sequentially provided data towards making a data-driven prediction. LSTMs have become a highly effective and scalable



(a) Error for training data

(b) Error for testing data

Fig. 2: Error as a function of feature size for the XGBoost implementation

deep learning method for tackling various problems with data that are sequential in nature. LSTMs have been developed as an improvement on RNNs, with benefits including avoidance of vanishing gradient and improved accuracy for various tasks [22]. In our method, the input (time sequence of traffic), is the input for the LSTM with the earliest data being the first input and the other values provided sequentially.

2) Tree-based approaches: In this subsection we discuss our methods that leverage decision trees for predicting the connection demands: Random Forest and XGBoost.

Random Forest: Random Forest, an ensemble learning method of aggregating decision trees, was used in [19] random to address the problem of then-contemporary methods, where uneven distribution of data led to poor performance [23]. Since then, decision trees have become a popular machine learning tool for numerous regression and classification problems faced in a wide array of fields. In our work, the traffic volume at different times is used as input for creating the trees, and the predicted output is the number of connections.

XGBoost: XGBoost is a decision tree-based system introduced in [20], which has been used in several applications to achieve state-of-the-art results. In particular, it is a gradient boosting mechanism wherein a prediction model is formed as an ensemble of weak prediction models. As in gradient boosting methods, XGBoost uses the gradient descent approach towards loss minimization, and creates subsequent decision trees to improve the model predictability. Some features the method uses for enhancement include regularization for preferring a less complex model and preventing over-fitting, and accounting for sparsity in data by approximating the missing feature values. In our case, the traffic volumes at discrete time steps are the features used to predict the number of connections.

IV. Evaluation

For the evaluation of the different ML approaches, we use a real deployment in an indoor testbed located in the Yale University premises. We use an extended version of the OAI platform, able to run in a cloud-native manner, using Kubernetes and docker, as detailed in [24]. We use a single node setup, equipped with a USRP B210 SDR, where the core K8s framework is configured and the cloud-native RAN is instantiated. A second node is used with a LTE dongle as UE, that attaches to the deployed RAN. On the node that is attaching to the network, we use the open source Telecom Italia dataset [25], which contains patterns of users requesting service from the network, spanning an entire week. The dataset was used to feed new connections requesting data service from the UE side over the network. Each incoming connection is allocated to one of three traffic classes defined randomly, as low/medium/high. The traffic is reflected to the data flow requested on the DL channel (CN to UE path). The three classes equal to 70/25/5 % of the traffic connections for low/medium/high traffic load categories respectively, as also detailed in our prior work [24]. Each UE connection starts a DL UDP stream from a container deployed alongside the SPGW-U. The sum of the traffic going over the network is configured to reach the maximum capacity of the eNB cell at any time, equal to 35Mbps, based on the configuration used (10 MHz bandwidth, FDD cell in band 7 using SISO).

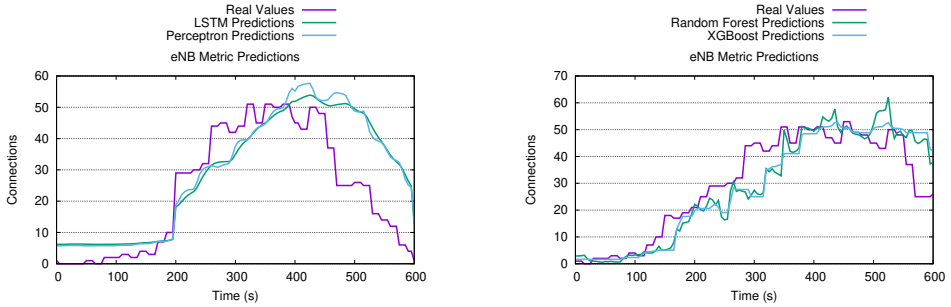
The metrics used for predicting the demand were observed based on the evolution of the different metrics monitored by Prometheus, when traffic was generated over our network. These were determined as follows:

- For the eNB, the UP bytes sent over the PDCP protocol.
- For the SPGW-U, the UP traffic transmitted over the S1-U interface

Each of the different models were used to predict the metric evolution. Based on the predictions, we use a threshold to proactively determine when the network should be scaled for accommodating the demand.

A. Feature Selection

In each of the deployed methods, it was predictable heuristically that adding more features would be beneficial; though, it might not be a good idea to use a large number of features as it could lead to an increase in training and inference time, as well as the computational cost. Here, we specifically talk about the number of instances of past traffic volumes collected while discussing the number of features. In our approach, we begin by considering traffic data, for a demand prediction at time t, from $t-3\Delta t$ to $t-23\Delta t$, where Δ is the time intervals at which the consecutive traffic (and connection) statistics are collected in real time. Selecting the most useful subset of the collected metrics would entail analyzing the Mean Absolute Error (MAE) between the real value and the predicted value, generated when different number of aforementioned features are selected. For each of the different methods, the errors were



(a) Data predictions for NN methods

(b) Data predictions for tree-based methods

Fig. 3: eNodeB data predictions (predicting the number of connections that request service over the network)

calculated for different numbers of features considered. For the case of the XGBoost model, results are shown in Figures 2a and 2b, illustrating in 2a the error when different feature lengths are used for training data and in 2b for test data. In this example, selecting the right feature length could improve the accuracy by 19.5% during the test case.

	Multi-layer Perceptron	LSTM	XGBoost	Random Forest
SPGW-U	4.9	4.8	3.2	3.6
eNodeB	7.9	6.5	3.4	3.9

TABLE I: MAE Calculation for the different ML methods

B. Hyperparameter selection:

In each of the cases, the training and testing data were separated with 80% data used for training and 20% for testing. **Multi-layer Perceptron** For SPGW-U, a deep perceptron network with two hidden layers, with the first hidden layer having a 2 times more nodes than the number of input features, and second layer with 4 times the number of input features, was chosen, with the output passing through a ReLu. An L1 regularization of 0.0002 was used, with a learning rate of 0.0012 and Adam as the optimizer. A batch size of 40 was used while training. For eNodeB data, the difference was that the learning rate was set to 0.0008.

LSTM In case of SPGW-U data, an LSTM with 2 layers and 20 hidden layers was used. Here, we found the effective learning rate to be 0.001, using the Adam optimizer, and L1 regularization to have a value of 0.0002; the training batch size was 40. In case of the eNodeB, an LSTM with 2 layers and 15 hidden layers was used, and other choices included learning rate of 0.001, optimizer as Adam, L1 regularization value of 0.00002, with the training batch size of 40.

Random Forest For the SPGW-U case, the number of trees was 70, and the maximum allowed depth was 90. For the eNodeB case, the number of trees was 60 and maximum allowed depth was 60. For both cases, the minimum number of samples required for split was 2, and the squared error regression method was used.

XGBoost The following hyperparameters were chosen for XGBoost. The regression method in all the cases was squared error. In case of SPGW-U data, the number of trees allowed was 100, and the maximum depth was left to be 100, the learning rate was 0.1, the fractions of columns sampled for each tree was 0.3, and this faction was 1 for each split, the L1 regularization value was set to 0.01. For the eNodeB data, the number of trees allowed was 100, the maximum depth was left to be 100, the learning rate was 0.1, the fractions of columns sampled for each tree was 0.7, and this fraction was 1 for each split, the L1 regularization value was set to 0.01.

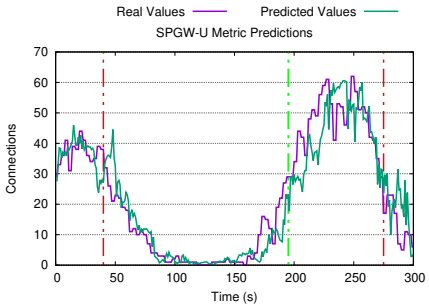


Fig. 4: SPGW-U Metric Predictions using XGBoost; green lines denote the points of horizontal scaling-out and red lines vertical scaling-in.

C. Results

To measure the efficiency of the different methods, **Mean Absolute Error (MAE)** was used, where the difference between the actual number of connections and the predicted number of connections were compared. The results for the comparison are shown in Table I.

Our observation was that in either of the two cases, SPGW-U and eNodeB, the values were found to be generally closer for the decision-tree based methods, XGBoost and Random Forest, and similarly so for neural network-based implementations, LSTMs and multilayer perceptron. XGBoost performed better than other implementations, with Random Forest having high accuracy as well. To further interpret the MAE measurements, it can be considered the difference between real and predicted demands at a point in time, so a difference between worst and best methods of 1.7 in case of SPGW-U results in 1224 more instances recognized more accurately. Similarly, a difference of 4.5 in case of eNodeB results in 3240 instances recognized more efficiently. Between XGBoost and Random Forest, first and second-best performers, the values drop to 144 for SPGW-U and 360 for eNodeB data respectively. In order to demonstrate the differences in predictions from the different methods, Figure 3 illustrates the predictions that the four methods can achieve, classified with the method that they

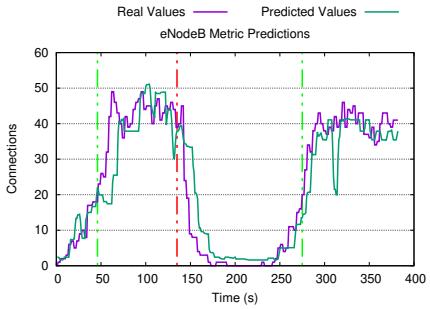


Fig. 5: eNodeB Metric Predictions using XGBoost; green lines denote the points of vertical scaling-up and red lines vertical scaling down.

implement (NN or tree-based). As it can be observed, the NN methods for the specific data are less accurate than the tree-based methods, of which the XGBoost performs better. (Figures 3a and 3b are illustrations based on real connections, but the overall evaluation uses the exact same set of data for training and testing in case of each investigated method.) A sample result showing difference in real and predicted demands over time is illustrated in Figure 4 for SPGW-U and in Figure 5 for eNodeB. Based on the configured thresholds for the scaling mechanisms, the Figures also illustrate the points when the scaling process shall happen, in order to accommodate the incoming demand. The framework is able to identify the number of connections that are transferred over the network, based on replaying the connections with random traffic generation, as detailed in the beginning of Section IV.

A faster inference time for the methods implies that the method is suitable for making proactive decisions that lead to a more efficient utilization of the resources. For instance, the number of instances could be updated/scaled every minute or so based on predictive decision making, leading to more efficient utilization of the computational resources. Hence the other factors to consider included the inference time, which was around 5ms for the random forest, 1ms for XGBoost, 1ms for LSTM and less than 1ms for the multilayer perceptron. The implication here is that each method takes a short time for the computations compared to the frequency with which the data is collected. It was realized that there is no need to see either of the methods as being particularly beneficial above others.

V. Conclusion

In this work, we focused on the evaluation of different ML methods for predicting the demand in the cellular network, and appropriately making scaling decisions for the network functions. We used a real cloud-native RAN deployment, monitored its operation, and executed each part of the network using micro-services. The results showed clearly that tree-based methods can predict with higher accuracy the data streams for the under-study network, compared to traditional NN approaches. The predictions can be used in order to proactively decide on the scale of each network function, and hence operate a more efficient network, while providing seamless service to end-users under high loads. In the future, we foresee to extend our scheme towards proactively allocating resources under different slices, based on the hosted applications.

References

[1] A. Ghosh, A. Maeder, M. Baker, and D. Chandramouli, "5G evolution: A view on 5G cellular technology beyond 3GPP release 15," *IEEE Access*, vol. 7, pp. 127 639–127 651, 2019.

[2] N. Nikaein, M. K. Marina, S. Manickam, A. Dawson, R. Knopp, and C. Bonnet, "OpenAirInterface: A flexible platform for 5G research," *ACM SIGCOMM Computer Communication Review*, vol. 44, 2014.

[3] S. Khan Tayyaba, H. A. Khattak, A. Almogren, M. A. Shah, I. Ud Din, I. Alkhalifa, and M. Guizani, "5G Vehicular Network Resource Management for Improving Radio Access Through Machine Learning," *IEEE Access*, vol. 8, pp. 6792–6800, 2020.

[4] H. Lee, J. Cha, D. Kwon, M. Jeong, and I. Park, "Hosting AI/ML Workflows on O-RAN RIC Platform," in *2020 IEEE Globecom Workshops*. IEEE, 2020, pp. 1–6.

[5] V. P. Kafle, Y. Fukushima, P. Martinez-Julia, and T. Miyazawa, "Consideration on automation of 5G network slicing with machine learning," in *2018 ITU Kaleidoscope: Machine Learning for a 5G Future (ITU K)*. IEEE, 2018, pp. 1–8.

[6] S. Natarajan and S. Mohan, "A Supervised Learning Approach for Reducing Latency during Context Switchover in 5G MEC," in *2021 IEEE 18th CCNC*, 2021, pp. 1–2.

[7] J. Kaur, M. A. Khan, M. Iftikhar, M. Imran, and Q. Emad Ul Haq, "Machine Learning Techniques for 5G and Beyond," *IEEE Access*, vol. 9, pp. 23 472–23 488, 2021.

[8] C. Jiang, H. Zhang, Y. Ren, Z. Han, K.-C. Chen, and L. Hanzo, "Machine Learning Paradigms for Next-Generation Wireless Networks," *IEEE Wireless Communications*, vol. 24, no. 2, pp. 98–105, 2017.

[9] B. Ma, W. Guo, and J. Zhang, "A Survey of Online Data-Driven Proactive 5G Network Optimisation Using Machine Learning," *IEEE Access*, vol. 8, pp. 35 606–35 637, 2020.

[10] I. Alawe, Y. Hadjadj-Aoul, A. Ksentini, P. Bertin, and D. Darche, "On the scalability of 5G core network: The AMF case," in *2018 15th IEEE CCNC*, 2018, pp. 1–6.

[11] I. Alawe, A. Ksentini, Y. Hadjadj-Aoul, and P. Bertin, "Improving Traffic Forecasting for 5G Core Network Scalability: A Machine Learning Approach," *IEEE Network*, vol. 32, no. 6, pp. 42–49, 2018.

[12] S. Sevgican, M. Turan, K. Gökarslan, H. B. Yilmaz, and T. Tugcu, "Intelligent network data analytics function in 5G cellular networks using machine learning," *Journal of Communications and Networks*, vol. 22, no. 3, pp. 269–280, 2020.

[13] N. Budhdev, M. C. Chan, and T. Mitra, "Poster: IsoRAN: Isolation and Scaling for 5G RAN via User-Level Data Plane Virtualization," in *2020 IFIP Networking Conference*. IEEE, 2020, pp. 634–636.

[14] "OpenAirK8s," https://github.com/OPENAIRINTERFACE/openair-k8s.

[15] J. Turnbull, *Monitoring with Prometheus*. Turnbull Press, 2018.

[16] X. Foukas, N. Nikaein, M. M. Kassem, M. K. Marina, and K. Kontovasilis, "FlexRAN: A flexible and programmable platform for software-defined radio access networks," in *Proceedings of the 12th CoNEXT*, 2016, pp. 427–441.

[17] S. K. Pal and S. Mitra, "Multilayer perceptron, fuzzy sets, classifiaction," 1992.

[18] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2016.

[19] L. Breiman, "Random forests," in *Machine Learning*, 2001, p. 45(1).

[20] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[21] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Tech. rep. ICS 8504. San Diego, California: Institute for Cognitive Science, University of California*, Sept, 1985.

[22] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[23] V. Svetnik, A. Liaw, C. Tong, J. C. Culberson, R. P. Sheridan, and B. P. Feuston, "Random Forest: A Classification and Regression Tool for Compound Classification and QSAR Modeling," *Journal of Chemical Information and Computer Sciences*, vol. 43, no. 6, 2003.

[24] A. Mudvari, N. Makris, and L. Tassiulas, "ML-driven scaling of 5G Cloud-Native RANs," in *2021 IEEE Global Communications Conference (GLOBECOM)*, 2021, pp. 1–6.

[25] G. Barlacchi, M. De Nadai, R. Larcher, A. Casella, C. Chitic, G. Torrisi, F. Antonelli, A. Vespignani, A. Pentland, and B. Lepri, "A multi-source dataset of urban life in the city of Milan and the Province of Trentino," *Scientific data*, vol. 2, no. 1, pp. 1–15, 2015.