Predictive Analytics for Fog Computing using Machine Learning and GENI

Jon Patman, Meshal Alfarhood, Soliman Islam, Mauro Lemus, Prasad Calyam, Kannappan Palaniappan
Dept. of Electrical Engineering & Computer Science, University of Missouri, Columbia, Missouri USA
{jpxrc, may82, miq44, lemusm}@mail.missouri.edu, {calyamp, pal}@missouri.edu

***Abstract*—Fog computing is a rapidly emerging paradigm concerned with providing energy- and latency-aware solutions to users by moving computing and storage capabilities closer to end users via fog networks. A major challenge associated with such a goal is ensuring that forecasts about network quality are not only accurate but also have small operational overhead. Machine Learning is a popular approach that has been used to model network parameters of interest. However, due to the small amount of public datasets and testbeds available, designing reproducible models becomes cumbersome and more likely to under-perform during deployment. For these reasons, we seek to design an exploratory testbed for benchmarking the forecasting strength of a suite of supervised learning models aimed at inferring network quality estimates. To create a realistic fog computing sandbox, we deployed an image processing ensemble of services in the GENI infrastructure. The nodes in GENI have varying hardware specifications for the purpose of generating compute-intensive workloads on heterogeneous systems. Our experimental results suggest that machine learning can be used to accurately model important network quality parameters and outperforms traditional techniques. Moreover, our results indicate that the training and prediction times for each model is suitable for deployment in latency-sensitive environments.**

I. Introduction

Fog computing has recently been proposed to alleviate the ever-increasing demands of devices and applications belonging to the Internet of Things (IoT) paradigm [1]. The goal of fog computing is to provide cloud services at the edge of the network where the data could be generated by a variety of devices such as industrial appliances, smart vehicles, and emergency responders [2]. Fog computing not only helps reduce the backbone traffic sent to the cloud, it also improves latency for delay-sensitive IoT applications by reserving the need to exchange user data with a centralized cloud [3]. The data deluge produced by IoT devices will generate large demand for responsive, platform-agnostic computing services that need to be efficiently managed using cloud and fog infrastructures for user and providers alike.

Software-Defined Networking (SDN) is another enabling technology that has transformed cloud computing by allowing resources and network functionality to be virtualized and orchestrated in order to provide new services to users [4]. SDN is remarkably flexible; it can operate with different kinds of switches and at different protocol levels, thereby interconnecting cloud and fog resources. SDN is also an attractive choice when considering the deployment of analytics platforms in cloud-fog environments because an SDN-operated controller can collect information about packets in the network and can enact routing policies by updating the flow tables of connected OpenFlow switches [5][6].

The recent successes of Deep Learning (DL) has also ushered in a new realm of possibility with regards to designing intelligent systems that can learn to make inferences from complex data. One of the most popular applications of DL is in the field of computer vision which deals with how computers can be used to gain a high-level understanding of digital images or video. Such advances typically outperform traditional methods in the areas of general object recognition [7], scene understanding, automatic language translation from images, and generating sound from silent video. It is expected that DL-based models will be the underlying technology for many cutting-edge visual processing systems for the foreseeable future [8]. Due to these advancements, we believe it is important to investigate the viability of deploying deep learning-based services in a cloud that is coupled with network edge resources.

Figure 1 shows how a DL-based image processing pipeline can be distributed in a fog network in order to provide more responsive, specialized services to end users. The pipeline begins by taking a raw image from a variety of modalities (e.g. mobile phone, security camera, drone, industrial device, etc.) and scanning the image for any detected objects that belong to a defined class (e.g. person, vehicle, etc.). Once a target object is detected, the image is then further processed for contextual features such as identifying the face region or the license plate of a car. Lastly, those features can be processed using facial or action recognition for enhanced contextual understanding.

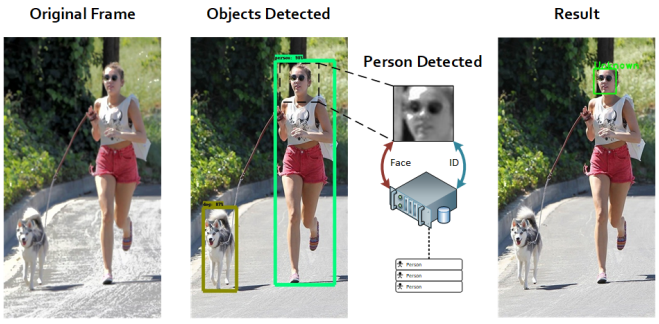

Fig. 1. Overview of a distributed DL-based image processing service. As raw images are fed in, the system scans the image for an object of interest (e.g. person, car), and depending on the result with further drill-down analyzing in order to identify any contextual information such as a known face or legible number.

978-1-5386-5979-3/18/$31.00 ©2018 IEEE

In this paper, we present two main contributions: The first is a discussion on the design and implementation considerations when deploying image processing services on SDN-operated hardware in the GENI infrastructure [9]. The second contribution is the presentation of an exploratory testbed for investigating the prediction accuracy for a variety of popular Machine Learning (ML) models in order to demonstrate the feasibility of deploying big data analytics in the network-edge i.e., fog. We use the proposed DL-based image processing service for representing a compute-intensive fog service. We then investigate the accuracy of our ML models for predicting end-to-end delay (e.g. the total time spent processing and transmitting data) as well as link utilization for various image processing workloads. We also show that deploying DL-based applications is a viable option for bringing responsive and state-of-the-art visual processing services to the network-edge.

The rest of the paper is organized as follows: Section II discusses the benefits of using SDN for fog computing. Section III defines the network model as well as the parameters of interest for prediction. The results from the experiments are discussed in Section IV. Lastly, Section V concludes the paper.

II. SDN-enabled Networks for Fog Computing

The nature of providing cloud-like services closer to the edge requires a centralized control mechanism that can orchestrate the distributed environment. As previously discussed, SDN is an attractive choice for coupling core cloud and fog services. In order to anticipate the range of user requests and device platforms, the components need to be homogenized in such a way so that a centralized control strategy can be implemented. SDN can help achieve this by hiding the internal complexities of the system from the users, application developers, and service providers. Since SDN concentrates network intelligence at a central software-based controller, IoT devices don't need to spend extra time and resources performing networking related tasks such as service discovery.

Figure 2 shows an example of the role that SDN plays in orchestrating an OpenFlow-enabled network. Data requests come from users of IoT devices and enter the network through a gateway or access point (typically via a terminal or wireless node). The data is then forward by an OpenFlow-enabled switch that can route packets to different nodes depending on factors such as availability, sequence, or reliability. Naturally, any type of analytics platform would reside on the same machine or at least remain in contact directly with a central SDN controller. The interplay between the analytics, management, and computing platforms allows for the possibility of even high-level intelligence by using information collected from the environment before and after network updates.

III. Predicting Network Quality Parameters

Popular monitoring tools such as ping, which measure Round-Trip Time (RTT) generally do not perform well at accurately predicting end-to-end delay for on-demand processing services, mainly because it uses the ICMP protocol which is reserved for diagnostic or control purposes, and differs

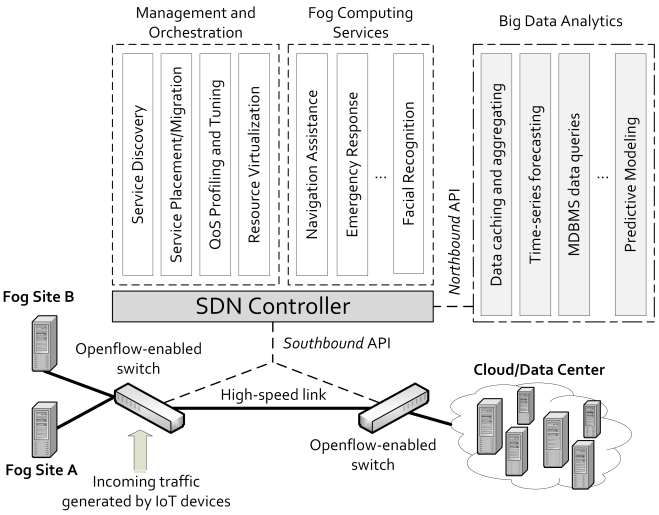

Fig. 2. Layout showing the centralized role that an SDN controller has when orchestrating fog networks. IoT devices generate data that is forward by a gateway device to an openflow-enabled switch which is controlled via the southbound API of an SDN controller. The SDN controller handles a variety of tasks in order to relieve the IoT device of any extra computing or networking duties.

from real communication protocols such as TCP. Intelligent mechanisms must be developed in order to provide more reliable assessments of network quality, particularly in those fog networks most crucial for low-latency, energy-aware applications. Due to the nature of most fog computing problems being compute-bound, our goal is to measure the performance characteristics for a variety of machine learning algorithms that try to predict relevant network quality estimates (e.g. one-way-delay, average link utilization, etc.) to enable SDN controller actions to effectively leverage available cloud and fog resources.

A. Network Delay Components Model

When discussing networking terms such as latency, throughput, and quality, we do so in the context of a packet-switching network where *end-to-end delay*, refers to the total elapsed time for data of length L bits to propagate from a source to destination node. The end-to-end delay parameter ψ is defined in Equation 1:

$$\psi = \sum_{i=1}^{N} d_{proc} + d_{trans} + d_{prop} + d_{local} \tag{1}$$

as the summation of the inter-operational delays such as the processing and queuing delay for each router as well as the local processing time on the source and destination nodes [10]. N is the number of routers or hops with delay components: d_{proc}, d_{trans}, d_{prop}, which represent the processing, transmission, and propagation delays, respectively. The time it takes for local execution is represented as d_{local}. The local processing time metric d_{local} is network agnostic and is instead influenced by hardware capabilities and concurrent relative workloads.

Another important parameter of interest to traffic engineering is *link utilization* which is useful for assessing Quality

of Service (QoS) violations that can then be proactively avoided [11]. The link utilization parameter, β, in Equation 2:

$$\beta = \frac{1}{1 + 2a + (d_{ack} + \psi)/\Gamma} \tag{2}$$

where ψ is the end-to-end delay, d_{ack} is the delay assuming a 48-bit acknowledgment-frame size, and Γ is the standard formula for calculating transmission delay described in Equation 3. The variable a is the ratio of d_{prop} and d_{trans} [10].

And lastly, in order to provide calculations performed by a baseline model, we can use the standard formula for calculating transmission delay, Γ, which is defined in Equation 3 as:

$$\Gamma = \frac{L}{R_T} \tag{3}$$

where L is the number of bits of each file and R_T is transmission rate of the data link measured in bits per seconds. As was the case with our network model, we acknowledge this metric is simplistic and only serves to provide a baseline for comparison of our methods. More advanced experiments and protocols will need to be evaluated in the future.

Calculating the processing time for a given file size on machines with varying hardware specs proves to be more difficult. Without using historical data, this type of measurement would require knowing how many instructions are executed for each program (which can vary drastically in image processing applications). For this reason, we will use the average time spent processing a file of L bits on machines with different hardware specifications in order to represent a baseline for benchmarking the prediction of d_{local}.

We also make the distinction here that in order to calculate link utilization, an accurate value of ψ is needed which can be difficult to obtain a priori. To overcome this limitation, we use machine learning to model ψ which will also be required for calculating the link utilization. Popular software tools such as iperf [12] and Traceroute [13] provide more accurate estimates of network quality but do so at the expense of increased network overhead and usually provide only a discrete measurement. Instead, we propose the use of predictive models that rely on global and historical system information for predicting quality assessment.

B. Supervised Learning Models

The three main branches of machine learning are supervised, unsupervised, and reinforcement learning. Of the types of supervised learning algorithms, the class that produce continuous value outputs are known as *regressors*. Conversely, the class of algorithms concerned with mapping inputs to a given categorical label are known as *classifiers*. We have chosen to focus on regressors in this framework for several reasons, the main one being, that most of the previous research involves measuring the performance of models using real-valued metrics such as the transmission time or processing time, measured in seconds, or the energy consumption, measured in watts. Using common metrics allows for comparison against methods that use different approaches to predict the same target variables. All algorithms discussed were implemented using the Python libraries Scikit-learn and Tensorflow.

1) Kernel-Ridge Regression: Naturally the first type of model to evaluate for regression tasks would be linear regression. However, some features that are collected in a fog computing environment may be highly correlated or have collinearity, which can result in the model being more susceptible to over-fitting. One way to remedy this problem is to use Kernel-Ridge Regression (KRR). KRR is much more suitable for fitting independent variables that have collinearity to a linear model by penalizing coefficients that have collinearity [14]. KRR combines the popular ridge regression model (linear least squares using l2-norm regularization) with the kernel trick [15] in order to learn a linear function in the space induced by the kernel and the data.

2) Support Vector Regression: Support Vector Regression (SVR) is similar to KRR in that it uses the kernel trick, but has some fundamental differences [16]. First, they use a different loss function: KRR uses squared error loss, whereas SVR uses ε-insensitive loss. Because SVR is part of the support vector machine family, it inherits similar learning characteristics. For instance, SVR only uses a subset of the training data for inferences, because the cost function for building the model ignores any training data not close to the model prediction [17]. Empirical results have shown that SVR tends to take longer to train than KRR but is faster at making predictions due to learning a sparse model. This may be of practical importance when deploying ML for decision making in real-time systems.

3) Gaussian Process Regression: Gaussian processes (GP) compute their outputs probabilistically by using a Gaussian distribution for approximating a set of functions (processes) in a high-dimensional space [18]. The main argument for using GP is the fundamental assumption that the mapping of independent to dependent variables cannot be sufficiently captured by a single Gaussian process. GP also uses lazy learning, which delays generalization about training data until after a query has been made. The main advantage of using a lazy learning method, such as instance-based learning, is that the target function is approximated locally for each query. This makes lazy-learning most useful for large datasets with few attributes which reflects the limited number of features available in an offloading environment.

4) Random Forest Regression: Random forests are a popular ensemble model belonging to the decision-tree class of ML algorithms. Random Forest Regression (RFR) works by fitting a number of decision trees (5 were used in our experiments), to various subsamples of the dataset that are then averaged to improve accuracy and reduce over-fitting in a popular process known as bagging. Random forests also have a variety of useful properties such as the advantage of being able to use the same model for both regression and classification tasks. Random forests also have the ability to learn features that are most important from a set of features from the training set [19].

5) Multi-Layer Perceptron: Lastly, we implement a popular artificial neural network, the Multi-Layer Perceptron (MLP), which has three hidden layers. Each hidden layer has a rectified linear unit (ReLU) activation function. ReLU is the most popular activation function used in deep learning due its simplicity, faster training, and reduction of the vanishing gradient problem. Finally, the inputs of the MLP model, such as the data size and the image height, are concatenated into feature vector layer. The feature vector layer is used to feed the model with inputs.

$$\hat{x_i} = \frac{x_i - min(x)}{max(x) - min(x)} \tag{4}$$

$$x_i = \hat{x_i} * (max(x) - min(x)) + min(x) \tag{5}$$

Feature normalization shown in Equation 4 can be used to make convergence quicker and can limit the influence of small or large values in the training set. After preliminary analysis, we found that normalizing the input features led to a greater increase is accuracy over non-normalized features. The final output of the MLP is produced by a sigmoid function that generates continuous values between 0 and 1. Therefore, de-normalizing the output value to the original range using Equation 5 is required for meaningful comparison because our target variables take on values outside of the $\{0,1\}$ range.

C. Data Management Strategies for Large-Scale Analytics

In order to perform analytics for aggregating multiple service requests efficiently, we consider the use of a Multidimensional Database Management System (MDBMS). An MDBMS allows for performing queries in order to find geometric relationships between nodes that can in turn be used to make policy decisions (i.e. offloading services to nodes who are occupied in regions of interest in the database). The major benefit of using an MDBMS data structure is that it is highly optimized for large search spaces which increases the scalability of our analytics methods. We previously presented a similar analytics method that used a generic feature space called a *hyperprofile*, to compute geometric relationships [20]. We believe such a technique can greatly be improved by adopting a MDBMS strategy.

Predictions from the machine learning models can be collected and used as inputs to build the dimensions of the database. The current network and device information about the nodes can also be used as inputs. Another exciting consequence of using a data structure such as a MDBMS, is that we can introduce a weight factor that be used to parameterize the dimensions of the database thus allowing for fine-tuning of each input feature's influence on the distance calculation used for selecting nodes. This capability allows for the mapping of features to policies that take into account certain desirable properties such as a preference for energy-aware or low-latency policies.

Each record in the database includes the node ID, the IP address, calculated distance of the node with respect to some origin of interest, and any other information that would be required by the SDN controller. Once the metadata is loaded into the MDBMS, the distance for each server node, in relation to the access nodes, can be calculated using the Euclidean or more general, Minkowski distance.

We conducted some preliminary experiments using an MDBMS with sample data and found that the execution time of the query was less than a millisecond which supports our hypothesis that an MDBMS is an ideal candidate for managing large-scale fog computing requests.

IV. Experimental Results

We conducted two sets of experiments for benchmarking the accuracy and response time for our suite of ML models. The first experiment consists of a simple client-server topology where we predict the end-to-end delay parameter ψ by combining the transmission and processing delays encountered for a series of 100 images on several device profiles and networking conditions. The data sizes of the 100 images used ranged between 135 KB and 3.4 MB.

The second experiment is concerned with evaluating the accuracy of predicting processing and transmission delay for each function in an image processing pipeline. Both set of experiments use Openflow-enabled switches configured with an Ethernet bridge and connected to an SDN-controller. The SDN controller was programmed to function as a simple L2 Learning Switch using the Ryu SDN controller. In the first experiment we make use of the TensorFlow Object Detection API for demonstrating the ease with which deep learning-based functions can be deployed. The second experiment makes use of Tensorflow and the OpenCV library in order to construct a more elaborate image processing pipeline that performs various functions on images in a distributed manner.

To evaluate our predictive models, we use the popular K-fold cross validation technique. Each dataset is split into K folds, such that the current model is trained with K-1 folds and tested with the remaining fold. We set K to be equal to 20 in both experiments. This technique allows the model to generalize better and gives a more accurate evaluation of the model's capabilities. We repeat this process K times such that all folds are tested individually. And lastly, in order to measure the error with regards to how close our model was to the real value, we use standard metrics for evaluation: Mean Absolute Error (MAE) in Equation 6 and Root Mean Square Error (RMSE) in Equation 7 described below:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{6}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{7}$$

A. GENI-Tensorflow computing study

Due to the ψ parameter being a linear combination of several contributing factors, we conduct the experiments for

delay type separately in order to better understand the influence of each term. We refer to each subset of data as the *transmission* and *processing* datasets. The transmission dataset was generated by measuring the propagation delay of transferring images under varying link bandwidths (in kbps): 128, 256, 512, 768, 1000, 2000, and 3000. The transmission dataset contains 700 instances and 4 features: image data size, image height, image width, and transmission bandwidth. Using an OpenFlow-enabled switch we can vary the input and output rate of the switch to effectively throttle the link to a particular bandwidth. As we did not experience any dropped packets, it is important to note that a sufficiently high buffer size is needed so as not to loose any packets in the event of congestion.

The processing dataset has 1600 instances and contains 5 features: image data size, image height, image width, CPU load, and the node's hardware type. In order to simulate fog resources that were being used by concurrent services, we varied the baseline CPU load to 0-, 25-, 50-, and 75% using the Linux stress package. Lastly, in order to represent a heterogeneous computing environment, four different ExoGENI machines were used: XOSmall, XOMedium, XOLarge, and XOXLarge. Hardware details of the corresponding machine specifications are shown in Table I.

TABLE I
PROVISIONED EXOGENI HARDWARE USED FOR BENCHMARKING.

Spec Type	Clock Speed (Ghz)	No. of CPUs	Cores per CPU	RAM (GiB)
XOSmall	2.2	1	1	1
XOMedium	2.2	1	1	3
XOLarge	2.2	2	1	6
XOXLarge	2.2	1	1	12

The results from Table II show the MAE and RMSE for each model from the first experiment involving the prediction of the ψ and β parameters. The Γ-method performs the worst as expected because it is the most naive approach and has inaccurate model assumptions. We can also see that the RFR model performed the best for predicting end-to-end delay and only performed slightly better in some regards to the MLP model for predicting link utilization. The RFR model most likely performed best due to the capabilities of decision trees and other ensemble methods being able to learn non-linear mappings more effectively. MLPs are also known for having the ability to map non-linearities of features, but in this case the number of features chosen may not have been representative enough or more training instances are needed.

TABLE II
EXPERIMENTAL RESULTS OF PREDICTION ACCURACY.

Model	End-to-End Delay ψ		Link Utilization β	
	MAE	RMSE	MAE	RMSE
Γ-method (baseline)	2.874	5.366	N/A	N/A
Kernel-Ridge Regression	0.232	0.322	21.736	29.332
Support Vector Regression	0.216	0.346	18.651	34.521
Gaussian Process Regression	0.230	0.322	21.954	29.424
Random Forest Regression	0.082	0.143	1.714	4.369
Multi-Layer Perceptron	0.090	0.168	2.007	3.690

To further investigate the capabilities of using an Artificial Neural Network such as an MLP, we evaluate the MLP model accuracy as a function of the number of hidden layers to examine if a deeper neural network would perform better by learning to generalize the features more effectively. We test the MLP with 3, 4, and 5 hidden layers. The results in Table III show that increasing the number of hidden layers actually degrades MLP performance in both datasets. This may be due to the limited number of features or training instances used and agrees with our observations from Table II.

TABLE III
IMPACT OF HIDDEN LAYER ON MLP PREDICTION PERFORMANCE.

	MSE	
No. of Hidden Layers	Transmission Time	Processing Time
3	16.40	0.025
4	18.075	0.027
5	26.157	0.028

Lastly, in order to evaluate the viability of using ML-based analytics for fog computing, we need to evaluate the response time for each model to ensure that they are within the operating constraints of fog computing networks. Each model was benchmarked by measuring the total time spent training and making predictions for the transmission and processing datasets. Table IV shows the response times with 95% confidence intervals. The results are interesting in that they show that SVR tends to train and make predictions the fastest most likely due to characteristics of SVR discussed in Section III. The KRR model is omitted in this study because the time to fit a linear model with data as simple as was used in our dataset, is negligible.

TABLE IV
TRAINING AND RESPONSE TIME EVALUATION

	Training Time $(s \times 10^{-3})$		Prediction Time $(s \times 10^{-3})$	
	Transmission	Processing	Transmission	Processing
SVR	17.69 ± 3.38	13.60 ± 3.38	2.69 ± 3.38	1.23 ± 1.78
GPR	5467.40 ± 221.44	3278.9 ± 7.52	217.40 ± 21.44	192.45 ± 14.70
RFR	157.78 ± 5.64	178.82 ± 4.65	3.63 ± 3.81	3.21 ± 2.87
MLP	121.23 ± 7.84	145.78 ± 5.75	12.12 ± 2.45	17.54 ± 2.85

B. Distributed image processing pipeline study

The visual data processing pipeline depicted in Figure 3 was deployed in GENI using a star topology with a centralized OpenFlow switch operated by an SDN controller. There were a total of 5 nodes, 4 of which were computing nodes (2 XOSmall, 1 XOMedium, and 1 XOLarge) and the last node being the OpenFlow switch. One XOSmall node serves as the client or gateway node most closest to the edge, followed by a node dedicated to each image processing function of the pipeline in Figure 3. 100 images were transferred and processed through all of the steps and the corresponding times were collected during each execution. The features for this dataset are the data size in bytes, the bandwidth (kept constant at 1Mbps), image height, image width, and spec type. The

target features are preprocessing time, object detection time, face detection time and total end-to-end delay.

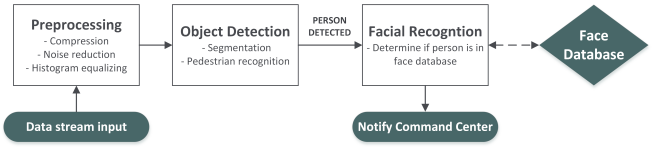

Fig. 3. Proposed image processing pipeline. Each computational service (preprocessing, object detection, and facial recognition) is deployed on GENI nodes in order to collect data on the processing and transmission characteristics of nodes in a distributed fog computing environment.

The results for the distributed image processing experiment are shown in Table V. What is interesting to note is that some models perform better than others with regards to making predictions in various computing environments. This could be related to the fact that some functions particularly in image processing, may terminate early depending on the image content. For example, two images of the same size but with different content (e.g. one with a person and one without) will take different times to process, which complicates the ability of making generalized predictions for that image size with a single model for the entire pipeline.

TABLE V
PREDICTION RESULTS FROM THE DISTRIBUTED IMAGE PROCESSING PIPELINE STUDY.

Model	Preprocesing	Object Detection	Face Recognition	End-to-End
	RMSE	RMSE	RMSE	RMSE
KRR	0.176	0.111	0.179	0.797
SVR	0.179	0.109	0.184	0.842
GPR	0.204	0.115	0.172	0.883
RFR	0.187	0.112	0.168	0.899
MLP	0.192	0.080	0.151	0.813

V. CONCLUSION

In this paper, we provided an analytics method for realizing a distributed network protocol with machine learning models in a fog computing context. Our method relies on benchmarking several popular supervised learning algorithms applied to the problem of predicting network quality parameters for a fog computing service with SDN control to orchestrate available cloud and fog resources. The need for proactive measurement mechanisms is crucial for low-latency and energy-aware scheduling in resource-constrained environments particularly those envisioned for fog computing. Our results indicate that machine learning not only performs well at accurately predicting end-to-end delay and link utilization but does so more accurately than the discrete calculation approaches used traditionally. Moreover, our results also provide insight into how these intelligent mechanisms can be deployed in real-time within real cloud and fog computing infrastructures. Future work could involve more extensive studies in order to identify the limitations of such data-driven approaches in terms of feature selection, parameter prediction, and training data size.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Award Number: CNS-1647182. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.
[2] R. Gargees, B. Morago, R. Pelapur, D. Chemodanov, P. Calyam, Z. Oraibi, Y. Duan, G. Seetharaman, and K. Palaniappan, "Incident-supporting visual cloud computing utilizing software-defined networking," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 1, pp. 182–197, 2017.
[3] H. Trinh, D. Chemodanov, S. Yao, Q. Lei, B. Zhang, F. Gao, P. Calyam, and K. Palaniappan, "Energy-aware mobile edge computing for low-latency visual data processing," in *IEEE International Conference on Future Internet of Things and Cloud (FiCloud)*, 2017.
[4] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.
[5] A. Mestres, A. Rodriguez-Natal, J. Carner, P. Barlet-Ros, E. Alarcón, M. Solé, V. Muntés-Mulero, D. Meyer, S. Barkai, M. J. Hibbett *et al.*, "Knowledge-defined networking," *ACM SIGCOMM Computer Communication Review*, vol. 47, no. 3, pp. 2–10, 2017.
[6] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and openflow: From concept to implementation," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014.
[7] C. Szegedy, A. Toshev, and D. Erhan, "Deep neural networks for object detection," in *Advances in neural information processing systems*, 2013, pp. 2553–2561.
[8] L. Wang and D. Sng, "Deep learning algorithms with applications to video analytics for a smart city: A survey," *arXiv preprint arXiv:1512.03131*, 2015.
[9] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar, "Geni: A federated testbed for innovative network experiments," *Computer Networks*, vol. 61, pp. 5–23, 2014.
[10] J. F. Kurose and K. W. Ross, "Computer networking: a top-down approach," *Addison Wesley*, vol. 4, p. 8, 2007.
[11] K. Kolomvatsos, C. Anagnostopoulos, A. K. Marnerides, Q. Ni, S. Hadjiefthymiades, and D. P. Pezaros, "Uncertainty-driven ensemble forecasting of qos in software defined networks," in *Computers and Communications (ISCC), 2017 IEEE Symposium on*. IEEE, 2017, pp. 1284–1289.
[12] A. Shriram, M. Murray, Y. Hyun, N. Brownlee, A. Broido, M. Fomenkov *et al.*, "Comparison of public end-to-end bandwidth estimation tools on high-speed links," in *International Workshop on Passive and Active Network Measurement*. Springer, 2005, pp. 306–320.
[13] Z. M. Mao, J. Rexford, J. Wang, and R. H. Katz, "Towards an accurate as-level traceroute tool," in *Proc. of conference on Applications, technologies, architectures, and protocols for computer communications*. ACM, 2003, pp. 365–378.
[14] V. Vovk, "Kernel ridge regression," in *Empirical inference*. Springer, 2013, pp. 105–116.
[15] S. Theodoridis and K. Koutroumbas, "Pattern recognition and neural networks," *Machine Learning and Its Applications*, pp. 169–195, 2001.
[16] D. Basak, S. Pal, and D. C. Patranabis, "Support vector regression," *Neural Information Processing-Letters and Reviews*, vol. 11, no. 10, pp. 203–224, 2007.
[17] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and computing*, vol. 14, no. 3, pp. 199–222, 2004.
[18] C. E. Rasmussen and C. K. Williams, *Gaussian processes for machine learning*. MIT press Cambridge, 2006, vol. 1.
[19] M. R. Segal, "Machine learning benchmarks and random forest regression," *Center for Bioinformatics & Molecular Biostatistics*, 2004.
[20] A. Crutcher, C. Koch, K. Coleman, J. Patman, F. Esposito, and P. Calyam, "Hyperprofile-based computation offloading for mobile edge networks," in *Proc. of IEEE MASS)*, 2017, pp. 525–529.