This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2018.2876844, IEEE Transactions on Parallel and Distributed Systems

Dynamic Autoselection and Autotuning of Machine Learning Models for Cloud Network Analytics

Rupesh Raj Karn, Prabhakar Kudva, and Ibrahim (Abe) M. Elfadel

Abstract—

Cloud network monitoring data is dynamic and distributed. Signals to monitor the cloud can appear, disappear or change their importance and clarity over time. Machine learning (ML) models tuned to a given data set can therefore quickly become inadequate. A model might be highly accurate at one point in time but may lose its accuracy at a later time due to changes in input data and their features. Distributed learning with dynamic model selection is therefore often required. Under such selection, poorly performing models (although aggressively tuned for the prior data) are retired or put on standby while new or standby models are brought in. The well-known method of Ensemble ML (EML) may potentially be applied to improve the overall accuracy of a family of ML models. Unfortunately, EML has several disadvantages, including the need for continuous training, excessive computational resources, requirement for large training datasets, high risks of overfitting, and a time-consuming model-building process. In this paper, we propose a novel cloud methodology for automatic ML model selection and tuning that automates the model build and selection and is competitive with existing methods. We use unsupervised learning to better explore the data space before the generation of targeted supervised learning models in an automated fashion. In particular, we create a Cloud DevOps architecture for autotuning and selection based on container orchestration and messaging between containers, and take advantage of a new autoscaling method to dynamically create and evaluate instantiations of ML algorithms. The proposed methodology and tool are demonstrated on cloud network security datasets.

Index Terms—Cloud analytics, machine learning, ensemble learning, distributed learning, clustering, classification, autoselection, autotuning, decision feedback, cloud DevOps, containers, Docker, Kafka

1 Introduction

Cloud platforms are empowering new, complex business models and currently coordinating more global integration networks than many researchers and analysts have predicted [6], [13]. Cloud network monitoring environments produce and store huge amounts of data along with their telemetry signals for infrastructure management. Such network data, when properly processed, should provide important insights into cloud network behavior at both the administrative and user level, especially in terms of its interaction with application performance, security, and resilience [29]. With virtualization, network configurations, traffic patterns, and connectivity are subject to change far more often than in the past with the result being a dynamic networking environment in which the telemetry signals are distributed and exhibit several transients with rather small time constants. Pattern recognition and machine learning have been used to build predictive models relating telemetry signals to application performance and throughout. Their usage on the cloud has been well established in the context of cloud operational analytics [5]. The success of such models very much depend on data feature selection, but given the frequent variations in cloud environments, some of the data features might change their importance and clarity over a given period of time. As a result, some prediction models might be highly accurate in one dataspace but not so on a relatively different dataspace. Even comprehensive models may quickly become erroneous in the case where the training and testing data vary significantly over time due to changes in workloads, configurations, network topologies, etc. As a result, a single, static machine learning model is often insufficient to produce accurate results for a long period of time. Distributed learning with dynamic model selection and time-dependent model parameter tuning is one possible solution to this problem. More specifically, a dynamic strategy for auto-selection and auto-tuning of machine learning models is required that shows significant variation over time.

An alternative solution is Ensemble Machine Learning (EML) where different data subsets are drawn from the training set and each training subset is used to train a different classifier. Statically-tuned EML is known to provide better predictive performance than a single learning model [16]. Most EML methods use a single base-learning algorithm to produce homogeneous base learners, i.e., learners of the same type that use different sets of parameters to create different learning models over the training subset, thus leading to homogeneous learning ensembles. But in order for the prediction methods to be more accurate than any of its individual members, the base learners have to be as diverse as possible with each base learner as accurate as needed. So in this paper, we use heterogeneous base learn-

- *Rupesh Raj Karn is with the Center on Cyber-Physical Systems, Khalifa University, Abu Dhabi, UAE. Email: rupesh.karn@ku.ac.ae*
- *Prabhakar Kudva is with IBM Research Yorktown Heights, NY, USA. Email: kudva@us.ibm.com*
- *Ibrahim (Abe) M. Elfadel is with the Center on Cyber-Physical Systems, Khalifa University, Abu Dhabi, UAE. Email: ibrahim.elfadel@ku.ac.ae*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2018.2876844, IEEE Transactions on Parallel and Distributed Systems

ers, i.e., learners of different machine learning types, and weighted voting to generate a single prediction result from the output of individual learners. Deep neural network, also called deep learning is another alternative solution. But, the deep learning models require relatively larger dataset and longer time than traditional machine learnings to train the model, so they are not usually considered in dynamic system where quick response is required.

Automated methods for machine learning, which include automated ensemble generation, hyperparameter tuning and feature selection have been widely studied to derive improved machine learning models [22]. The comparison among different machine learning models and selection of the best one have also been reported, and [1] gives a representative view of the state of the art as well as ongoing research in this area.

While these approaches optimize learning models, the dynamic selection among such tuned models, and automating the tuning and selection together in a Cloud DevOps environment continues to be an area of practical interest. While prior work has been successful in finding optimal machine learning algorithms, the goal of this paper is to provide a dynamic procedure and context for model selection and tuning in environments where signal quality and data content change frequently and where tuned models need to be updated in a DevOps framework.

More specifically, a "Reflexive DevOps Architecture" for machine learning (ML) models is proposed. The term 'reflexive' denotes that the action (training and testing of the model) has to be performed spontaneously in response to the event. In this architecture, a dynamic strategy based on Cloud DevOps and autoscaling for auto-selection and auto-tuning of machine learning models is implemented, and tested on a cloud network security dataset that exhibits significant variation over time. In the remainder of the paper, we call this architecture as "Reflexive DevMLOps". In our proposed approach, models that become outdated or inaccurate over time are retired or put on standby while new or standby models are brought in. A number of different classification (supervised learning) algorithms are implemented on the data spaces that are generated from the network security dataset by clustering (unsupervised learning) in order to segment the data set for better classification accuracy. Later, the EML and deep learning approaches are also implemented on the same dataset to compare the results with the proposed Reflexive DevMLOps setup.

The contributions of this paper can therefore be summarized as follows:

1) Matching a single optimized model to a given context in a dynamic environment.
2) Creating and building multiple models and selecting the best for a given context.
3) Closed loop, auto-selection mechanism in the cloud DevOps environment.
4) Using unsupervised clustering to segment the dataset ahead of supervised classification.
5) End-to-end comparison with Ensemble Machine Learning.
6) Deep learning implementation and some of its hyperparameters tuning.

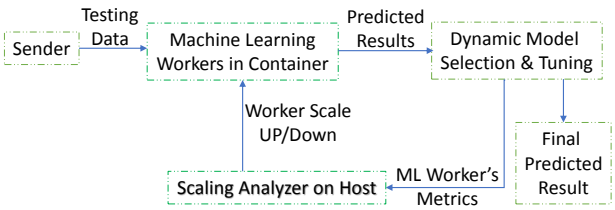


Fig. 1: Cloud DevOps Architecture for Machine Learning

Of course,the proposed Reflexive DevMLOps procedures can be applied to several areas of the cloud or even possibly outside cloud computing.

This paper is organized as follows. Section 2 covers the Cloud DevOps setup and its operational protocols. Section 3 describes the baseline analysis using static multi-class supervised learning models and seamless integration with unsupervised learning. The Reflexive DevMLOps architecture for automatic model selection and their parameter tuning methods in a dynamic environment are demonstrated in Sections 4 and 5, respectively. Several cases for scaling machine learning models are explained in Section 6. Section 7 explains the significance of dynamic feature selection for the models to improve the accuracy. The comparison of dynamic model selection and parameter tuning with statically-tuned EML is shown in Section 8. In Section 9, we present preliminary results on the use of deep learning and compare them with the more traditional ML of the preceding sections. The time comparison between our technique and ensemble learning is shown in Section 10. Representative prior publications most related to our own work are surveyed in Section 11. The conclusions are given in Section 12.

2 Cloud DevOps Architecture and Methodology

Container orchestration and scaling is used for defining and running multi-container microservice applications [31]. It is also used for autoscaling applications based on demand. In our work, this autoscaling feature has been modified for evaluating and selecting the models and parameters of machine learning algorithms in the containers. A messaging client [11] has been used to transfer the messages among containers. The machine learning algorithms are run inside containers [4].

Figure 1 shows the Cloud DevOps architecture for automodel selection and tuning. Figure 2 shows its detailed block diagram. This setup simulates the cloud environment where each of the docker containers can be located at different network/geographical locations or within the same network but on the different physical machines. The selection of models is based on cloud microservice instantiation and deletion. Containers that implement new models are brought online, while stale and outdated ones are taken off-line. We use Kafka [23] to transmit messages among the containers. The use of Docker-compose [18] enables individual containers (and therefore the machine learning workers in them) to be brought online or taken off-line in DevOps fashion. Inside each worker container, a different machine learning (ML) algorithm is modeled. In online learning,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2018.2876844, IEEE Transactions on Parallel and Distributed Systems

these machine learning containers would be enabled with some learning off-line until their accuracy increases, while some actively predict. The sender container acts as a streaming data generator that continuously streams the data. For the purposes of this paper, the UNSW-NB15 data set [2], [20] is used and treated as a streaming networking data source. The ML-worker containers accept these messages from Kafka to update their models, perform a prediction which is sent to a model selector again via Kafka. Using the auto-selection auto-tuning methods, a final result is calculated. The workers are continuously tracked by the scaling analyzer, which decides whether to scale up or down and machine learning parameters. Initially the models development is based on labeled data, but as the learning progresses, these are determined by a method of analytical consensus among the models. The scaling analyzer turns on or off a set of ML-workers using docker-compose. Using the streaming service this setup can be used to predict the labels for real-time or online data samples fed by the users or applications.

3 Baseline Machine Learning Implementation

Prior to developing the new approaches, a baseline machine learning implementation was evaluated for future comparison. Seven supervised (also called classifiers) and three unsupervised (clustering) ML algorithms are chosen randomly. The supervised algorithms include K-Nearest Neighbors (KNN), Naive Bayes, Random Forest, Multi-Layer Perceptron (MLP), Gradient Boosting, Decision Tree, and Stochastic Gradient Descent (SGD). The unsupervised algorithms include K-Mean clustering, Birch clustering, and the Gaussian mixture model. Machine learning algorithms are written with Python. Well-known python packages such as Numpy, Scipy, Scikit learn, Pandas, Matplotlib etc. have been used.

3.1 Dataset Label Encoding and Normalization

The "UNSW-NB15" dataset [2], [20], [21] has been selected to train and test all our ML implementations. It is treated and used as online data for our analysis. The dataset is used for evaluation of network anomaly and intrusion detection systems. It contains three categorical variables: 'proto', 'service' and 'state'. These variables are provided as text values which represent various traits. 'Proto' describes transaction protocol used TCP, UDP, $CRTP$, GGP, HMP etc. 'Service' describes the connection shell as SSH, SSL, $FTP-Data$, DNS, $DHCP$ etc. 'State' describes the dependent protocols used as INT, REQ, CON, RST etc. The data set has the features of both normal and contemporary synthesized attack activities of the network traffic having $82,332$ records for training and $175,341$ for testing. Each record consists of attributes of different data types (e.g., binary, float, nominal and integer). The feature set includes port numbers, service name, protocols, IP addresses, packets transmission statistics etc. and the 10 labels: 9 different types of attack name and a normal ("NO-ATTACK"). Traditional datasets such as KDD98, KDDCUP99 band NSLKDD, provide a limited number of attacks and the information of packets which are outdated. Moreover, UNSW dataset has been generated using some of the well-known tools like IXIA traffic generator, Argus, Bro-IDS etc [20]. More details on the features of this dataset can be found at [2]. Many machine learning algorithms cannot support text based categorical values, and as a result, their conversion into numeric form is required. The Python packages, Pandas and Scikit-learn, provide various techniques to transform the categorical data into numeric values. Label Encoding [14], [33] is one such technique where N different text values are picked up randomly and mapped on the $\{0, 1, 2, \ldots, N-1\}$. This encoding has been selected in this work. Other techniques like "One hot encoding", "Custom binary encoding" [24] are also available that convert each categorical value into a new column and assign a 1 or 0 (True/False) value to the column. This type of encoding has the disadvantage of adding many more columns to the data set. Furthermore, the number of model variables grows significantly, especially if the dataset has a large number of categorical values. A more sophisticated algorithm called "Backward Difference Encoding" [24] has been tested prior to choosing "Label Encoding". This encoding has generated too many negative numbers, and as a result, Python Scikit-learn run into exceptions while clustering (when unsupervised is added along with supervised learning) the dataset. Hence label encoding is chosen for converting the textual categorical values into numerical values. After encoding, the dataset is normalized to adjust large values measured on different scales to a notionally common scale.

3.2 Baseline Supervised Learning

Some baseline tests have been performed over the dataset indicated in Section 3.1 using known algorithms such as multi-class classification. As listed in the first paragraph of Section 3, all the classification algorithms are applied to a training dataset having $82,332$ records after label encoding for multiclass or multinomial classification. The dataset feature "attack_name" is used as the label. The testing data set has $175,341$ records, and the predicted label is compared with the natural label to calculate the accuracy score. The result is shown in Figure 3. With this direct approach, the highest accuracy was 61.3%. This is far from sufficient for a cybersecurity application. Some of the parameters of the machine learning are tuned to check the effect on accuracy. For example, in KNN classifier, the number of neighbors is chosen from the set $\{1, 3, 5, 7, 9, ..., 27, 29\}$, while in the Random Forest classifier, the number of estimators is chosen from the set $\{10, 15, 20, 25, ..., 90, 95, 100\}$. The 10 fold cross validation score is calculated at each parameter value, and the results are shown in Figure 4, which clearly illustrates that tuning the ML parameters increases accuracy, but the increase is limited to the $5-10\%$ range for these two ML models. Hence a different approach of modeling has been proposed in next Section for accuracy more than 95%.

3.3 Supervised Learning with Label Categorization

As explained in Section 2, the dataset has features named $attack\_name$ with a total of 10 different categorical values:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2018.2876844, IEEE Transactions on Parallel and Distributed Systems

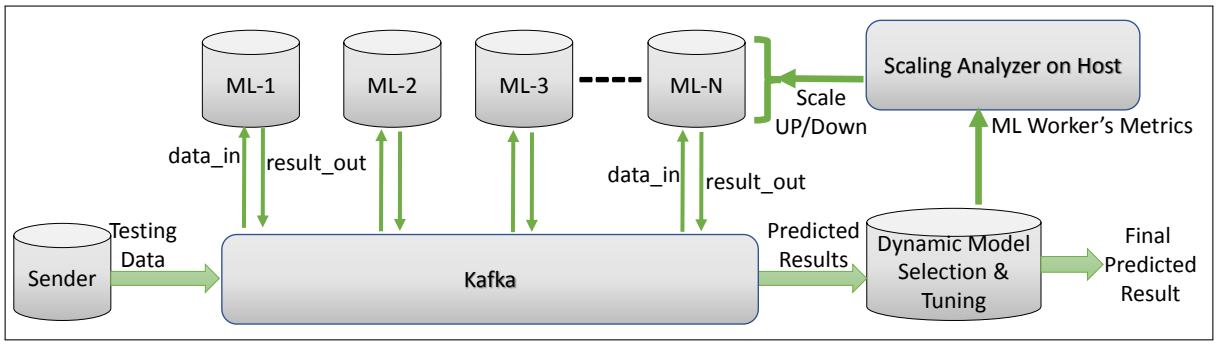


Fig. 2: Cloud DevOps Approach for Model Selection and Auto-Tuning

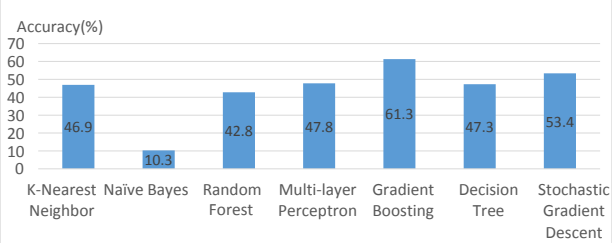


Fig. 3: Supervised Learning Accuracy

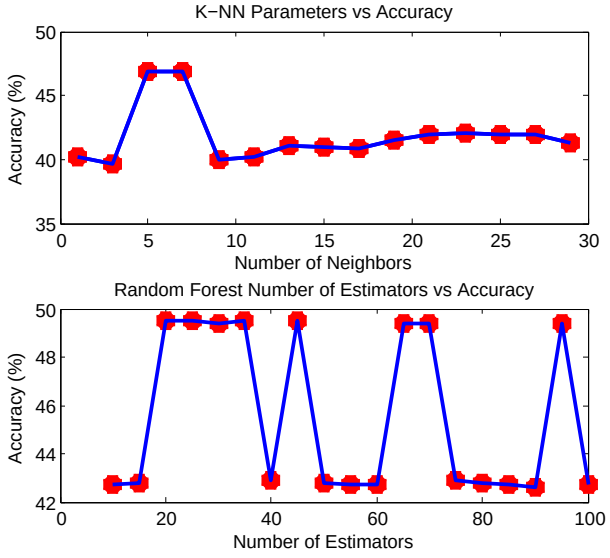


Fig. 4: Machine Learning Parameters vs Accuracy

9 for labeling 9 different types of attack and one "NO-ATTACK" for normal behavior. In the previous section, this feature has been used as the label for a multi-class classifier. To increase classification accuracy, the multi-class classifier has been converted into a multiplicity of binary classifiers, with each one of them predicting one of the attack labels: $attack\_name$. For that, 10 new labels are created from $attack\_name$ where 1 or 0 is assigned to each of data samples according to presence or absence of the particular attack type. Each classifier is run for 10 iterations, once for each of the categorical columns as a label, and the accuracy score is measured as in the previous section. The sizes of training and testing sets are the same as indicated in previous section. The result is shown in Figure 5. The attack label name is along the X-axis while the Y-axis shows the accuracy for each label prediction using 7 different classifiers. The prediction accuracy for 8 of the 9 cybersecurity attacks is at least 90%. For the 9th attack, Exploits, 5 of the classifiers achieve more than 80% accuracy. As is the case in security threat identification, all the classifiers are conservative in that they have a significant incidence of false positives, predicting the presence of attacks when none is actually present.

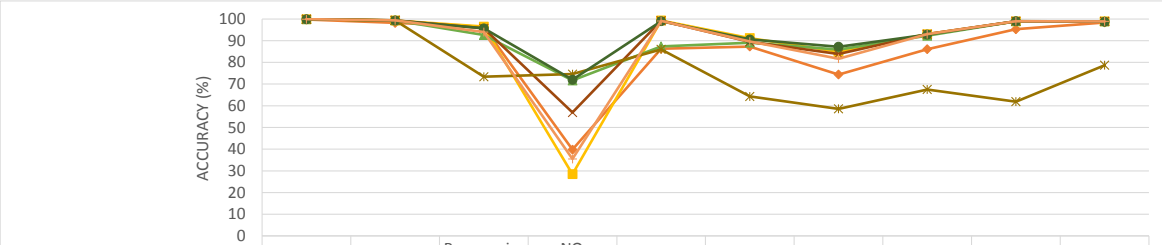
3.4 Unsupervised Learning for Clusters

From Figure 5, some of the labels like "NO-ATTACK", "Fuzzers", "Exploits" and "DoS" have prediction accuracy still below par for most of the models. In order to improve their accuracies, the unsupervised algorithms are used. These algorithms find the inherent groupings called clusters of the data samples such that the samples have high intra-cluster similarity and low inter-cluster similarity. The basic intuition is that classification models built with data points of a particular cluster have higher cross-validation accuracy score than the models of Section 3.3 because such models are created and tested over close data points within the cluster.

In support of this basic intuition, unsupervised ML is first used to partition the UNSW-NB15 into different clusters using a training dataset whose size is the same as in Section 3.2. The classifiers are subsequently applied on the top of the clustered data. K-Means clustering, Birch clustering, and the Gaussian mixture model have all been used for clustering based on the default Python Scikit-learn parameter values. The accuracy of each label prediction (as in Section 3.3) is measured for the different clusters. The comparison between accuracies of Naive Bayes, Decision Tree, and Multi-layer Perceptron classifiers for label "NO-ATTACK" and $cluster\_number = \{2, 3, 4, 5, 6\}$ is shown in Figure 6. Even though the number of clusters increases, the classifier accuracies do not always increase. For some clusters, the accuracy increases but for some others, it decreases. All the labels of the dataset have shown similar behavior.

Applying unsupervised ML is only meaningful for those labels whose prediction accuracy is low. For example, the labels "Worm", "Shellcode" and "Analysis" have accuracy more than 95% for all the selected supervised MLs and therefore they would not benefit from an unsupervised ML stage.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2018.2876844, IEEE Transactions on Parallel and Distributed Systems



	Worm	Shellcode	Reconnaissance	NO-ATTACK	Generic	Fuzzers	Exploits	DoS	Backdoor	Analysis
DecisonTree	99.8	98.2	96.4	39.8	86.3	87.3	74.4	86.1	95.3	98.4
GradientBoosting	99.9	99.3	96.5	28.5	99.3	91.2	84.9	93	98.9	98.9
knn	99.9	99.4	92.6	71.7	87.4	89.1	85.9	92.3	99	98.9
MultiLayerPerceptron	99.9	99.4	95.7	56.9	99.2	89.5	83.9	93	99	98.9
NaiveBayes	99.9	99.4	73.4	74.6	86	64.3	58.6	67.5	61.9	78.7
RandomForest	99.9	99.3	95.7	72	99	90.6	87.2	92.8	99	98.8
StochasticGradientDescent	99.9	99.4	94	35.5	99.1	89.6	81.7	93	99	98.9

Fig. 5: ML's Accuracy with Label Categorization

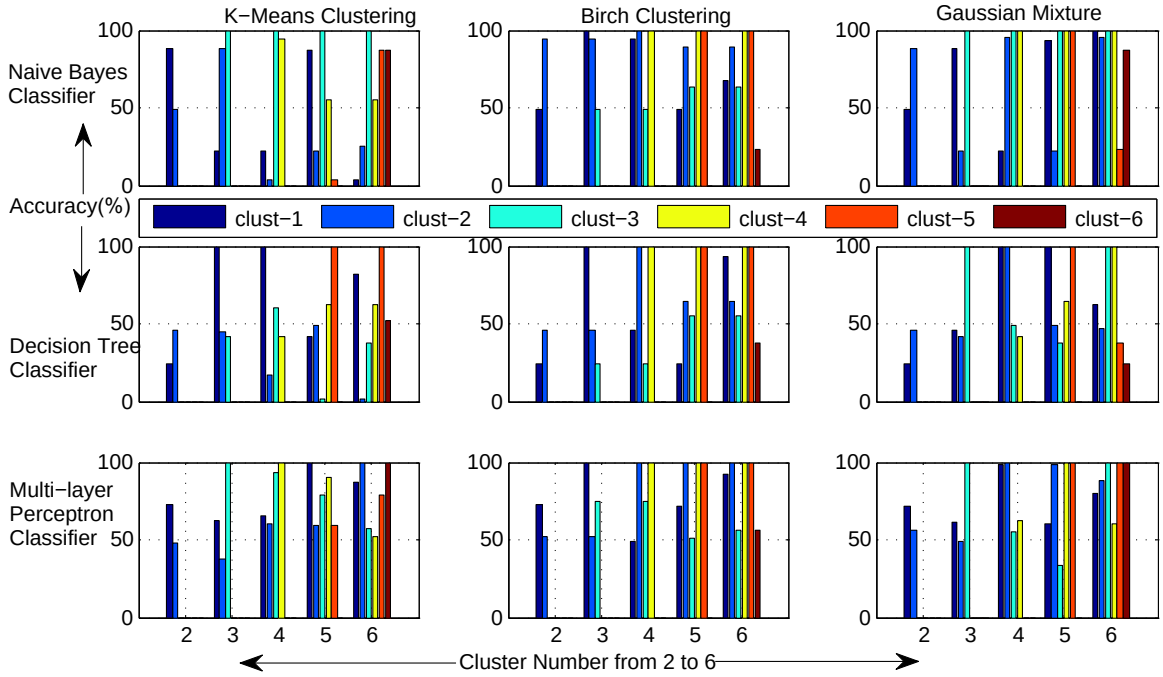


Fig. 6: Accuracy with Clustering

4 AUTOSELECTION OF THE MODELS

An automatic model selection scheme is needed that chooses the accurate model dynamically based on the real-time data. For example in the previous section, the multi-class classification is converted into a multiplicity of binary classifications to improve the cross validation accuracy. In such case, with 7 supervised learning algorithms for each label and 10 different labels in the dataset, there are 70 possible models. Adding clustering over the dataset will further increase this number and the large number of possible models further motivates the need of automatic model selection. Such selection is enabled by the DevOps framework given in Figure 1.

One technique for automatic model selection is the voting based on the weighted majority rule. The voting weights are determined by the prediction accuracies of the various models. The classifiers with higher accuracy score in Figure 5 achieve more weights than the less accurate ones. Table 1 is generated from Figure 5 and shows the ranking

TABLE 1: ML Ranking per Label

Label Name	Machine Learning Ranking 1^{st}	2^{nd}	3^{rd}	4^{th}	5^{th}	6^{th}	7^{th}
Worm	1	2	3	4	5	7	6
Shellcode	1	2	4	7	3	5	6
Reconnaissance	5	6	3	4	7	1	2
NO-ATTACK	2	3	1	4	6	7	5
Generic	5	4	7	3	1	6	2
Fuzzers	5	3	7	4	1	6	2
Exploits	3	1	5	4	7	6	2
DoS	4	5	7	3	1	6	2
Backdoor	1	3	4	7	5	6	2
Analysis	1	4	5	7	3	6	2

1 - K-Nearest Neighbor 2 - Naive Bayess 3 - Random Forest
4 - Multi-layer perceptron 5 - Gradient Boosting 6 - Decision Tree
7 - Stochastic Gradient Descent
Note : This legend is followed in other plots as well.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2018.2876844, IEEE Transactions on Parallel and Distributed Systems

of the classification models that are used to decide voting weights for predicting each label. The Reflexive DevMLOps setup for model auto-selection using weighted majority voting is shown in Figure 7. The dataset is trained as described in Section 3.3 and 3.4. During prediction, unsupervised ML identifies the cluster number for the testing data sample. Then all the classification models act on the this data sample and predict its label. The collection of these predicted labels is sent to the "Dynamic Model Selection & Tuning" module. It has a submodule called "voter" which has the ML-rankings, as shown in Table 1. Based on the rankings, the voting weights are applied to such label collections and a single value is generated as the final predicted label for the testing data sample.

Algorithm 1: ML Model Autoselection

```
Result: final_result = label of the testing data sample
1  classifiers = {ml_rank_1,ml_rank_2,....,ml_rank_n};
2  number_of_classifiers = count(classifiers);
3  voting_set = classifiers(0:m);
4  voting_set_counter = count(voting_set);
5  predicted_label = majority(voting_set);
6  if num(predicted_label) == 1 then
7      final_result = predicted_label;
8  else
9      while voting_set_counter <= number_of_classifiers
         do
10         voting_set.append(classifiers(voting_set_counter));
11         predicted_label = majority(voting_set);
12         if predicted_label == ml_rank1 or predicted_label
             == ml_rank2 then
13             final_result = predicted_label;
14         end
15         if voting_set_counter == number_of_classifiers
             then
16             final_result = predicted_label;
17         end
18         voting_set_counter = voting_set_counter + 1;
19     end
20 end
```

Autoselection is described in Algorithm 1. The models are selected using an incremental process according to the following sequence in the DevOps environment of Figure 1:

(A) For each label, the voting set of top m ranked models of table 1 is fetched (for illustration, $m = 4$). If their prediction output is same then any of those m models can be selected. Its output is the final prediction result. It is called majority based voting where 4 out of 7 MLs predicts the same output.

(B) If case (A) is false then the majority voting over the current voting set is performed. If the majority voting returns only 1 value then anyone out of those majority voted models is selected.

(C) If case (B) is false then the 5^{th} ranked model is added to the voting set. The majority voting over the current voting set is performed. If the majority voting result matches the predicted output of 1^{st} or 2^{nd} ranked model then anyone out of those majority voted models is selected and the voter output is the final predicted result.

(D) If case (C) is false then the 6^{th} ranked model is added to the voting set. The case (C) is re-run. If the majority voting result matches the predicted output of 1^{st} or 2^{nd} ranked model then the voted output is the final predicted result.

(E) If case (D) is false then last ranked (7^{th} for total 7 classifiers) model is added into the voting set. Any models that have same predicted output as the majority vote is selected and the majority voted output is the final predicted result.

Every model "casts its ballot" for each test sample. With the majority voting criteria, the final predicted output is the one that earns more than half of the votes. A voting set of 4 out of 7 is created in statement 5 of Algorithm 1 for that purpose. If the final prediction is unable to work out with the current voting set then predictions from lower-ranked models are added gradually until the weighted majority voting condition is satisfied. The models with rank 1 and 2 are given more weight in case (C)−(E) for voting because they are the models that predict the correct label most of the time.

To test the model selection algorithm, an online data stream that changes dynamically over time is needed. To mimic such a dynamic feature, a dataspace with $10,000$ records from the testing subset of UNSW-NB15 is created. All the classification models are run and their accuracy-based rankings are recorded at the end of each dataspace. Model rankings after each dataspace for label "NO-ATTACK" and "Generic" respectively are shown in Figure 8, which illustrates the dynamic change of ML ranking over the dataspace. Model selection using weighted majority voting for few samples of the dataspace is shown in Figure 9. The bold green font (in circles) shows the selected models for every sample. The predicted output is compared with the actual value. A match is represented by T and a mismatch is represented by F.

Algorithm 2: Model Selection with Unsupervised Learning + Supervised Learning

```
Result: final_result = label of the testing data sample
1  cluster_number = {2,3,4,5,6};
2  cluster_type={Kmeans,Birch,Gaussian};
3  selected_model_accuracy = Unclustered_accuracy;
4  for clustering in cluster_type do
5      for model in cluster_number do
6          predicted_cluster =
             model.predict(testing_sample);
           // Predicts the dataspace
7          if model_inside_cluster_accuracy
             >Unclustered_accuracy then
8              selected_model = model_inside_cluster;
9          end
10         if model_inside_cluster_accuracy
             >selected_model_accuracy then
11             selected_model = model_inside_cluster;
12         end
13     end
14 end
```

As mentioned in Section 3.4 and shown in Figure 6, clustering helps to improve the accuracy of prediction for some

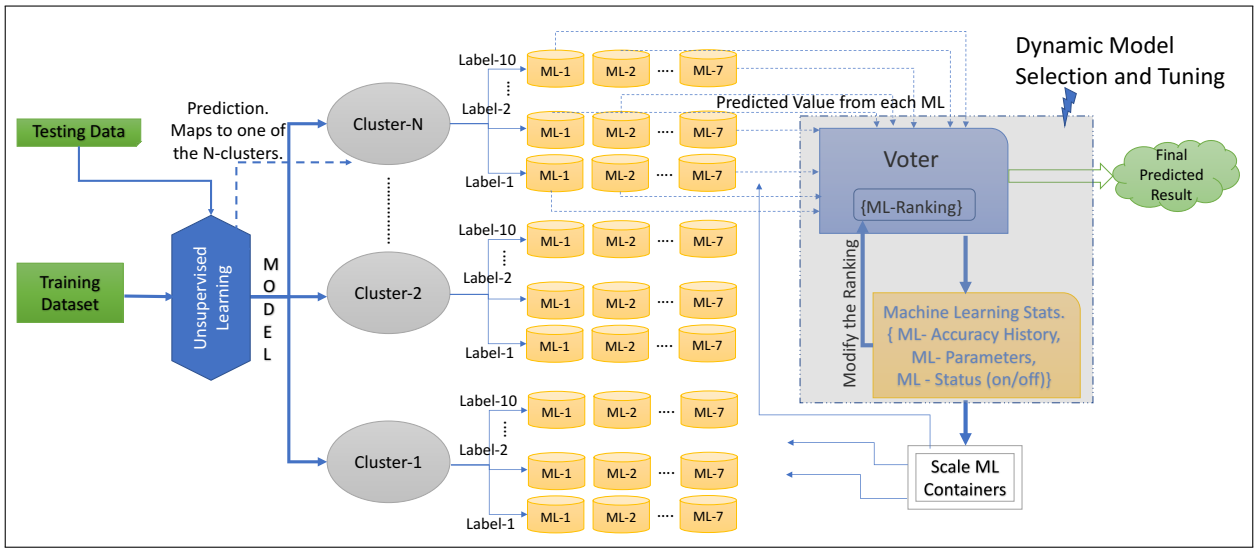


Fig. 7: Reflexive DevMLOps Architecture for Autoselection and Autotuning

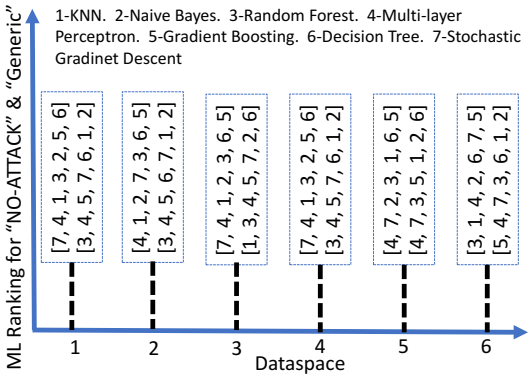


Fig. 8: ML Ranking Change Dynamically

labels such as "NO-ATTACK", "Fuzzers", "Exploits" and "DoS". Increasing the number of clusters doesn't improve classifier accuracy for all the clusters. In such cases, the accurate classifiers are selected using their cross-validation accuracy score across all the clusters. Such model selection is described in Algorithm 2.

For example, in Figure 5, the Naive Bayes has an accuracy of 74.6% for predicting "NO-ATTACK" without clustering. With clustering, Figure 6 shows that Naive Bayes has an accuracy of approximately 90% for $cluster_1$ and 50% for $cluster_2$ in $2-Mean$ clustering. If the testing data sample is predicted into $cluster_1$ (using $2-Mean$ clustering model) then Naive Bayes model of $cluster_1$ is the accurate model and hence it is selected for "NO-ATTACK" prediction. But, if the sample is predicted into $cluster_2$ then Naive Bayes model of this cluster is discarded because its accuracy (50%) is lower than the unclustered Naive Bayes accuracy (74.6%). In such case, the accurate Naive Bayes model is searched across other $k-Mean$ clustered models where, $k \in \{3,4,5,6\}$ to check if the sample is predicted into any of the clusters where the Naive Bayes classification accuracy is more than the unclustered accuracy of 74.6%. If this condition is not met then unclustered Naive Bayes model is selected. This step is repeated for Birch clustering and the Gaussian Mixture model. This procedure guarantees that for every run, the most accurate model is selected.

After model selection the collection of predicted labels are sent to the "ML Stats" submodule inside the "Dynamic Model Selection & Tuning" module, as per Figure 7, in order to improve model quality. "ML Stats" is used as a model history log for all models. In particular, the ML accuracy log stores $true$ or $false$ for the voting decisions that agree or disagree with a particular model prediction. This set is incremented at each testing sample. The parameter set stores the parameter values of each model while the status set stores the active/inactive state of the model. These "ML stats" help in scaling the number of ML models as will be explained in Section 6. After each dataspace, these sets are cleared, the ML models are recreated with the last used testing dataspace, and the ML rankings, as described in Section 3.4, are updated.

5 Autotuning of Machine Learning Parameters

ML models depend for their accuracy on parameters that can act as accuracy knobs. The impact of parameter selection on the accuracy of K-Nearest Neighbor and Random Forest was shown in Figure 4. To make the most out of a given ML model, its parameters should be allowed to change over the model life time in line with the dynamic changes of the data waveforms and their feature patterns. Before deactivating a given model completely as required in the Reflexive DevMLOps model autoselection context of Section 6, it is recommended that its parameters be adapted while tracking its accuracy and to record any significant accuracy improvement as a result of the adaptation process. In this paper, we adopt such an adaptive methodology as a necessary parameter tuning step before discarding the model during the auto-selection process. Parameter tuning is automated according to the diagram of Figure 7 and Alogrithm 3. The submodule "ML Stats" under the module "Dynamic Model Selection and Tuning" has the ML parameters stored for each model. If any model crosses a threshold on the number

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2018.2876844, IEEE Transactions on Parallel and Distributed Systems

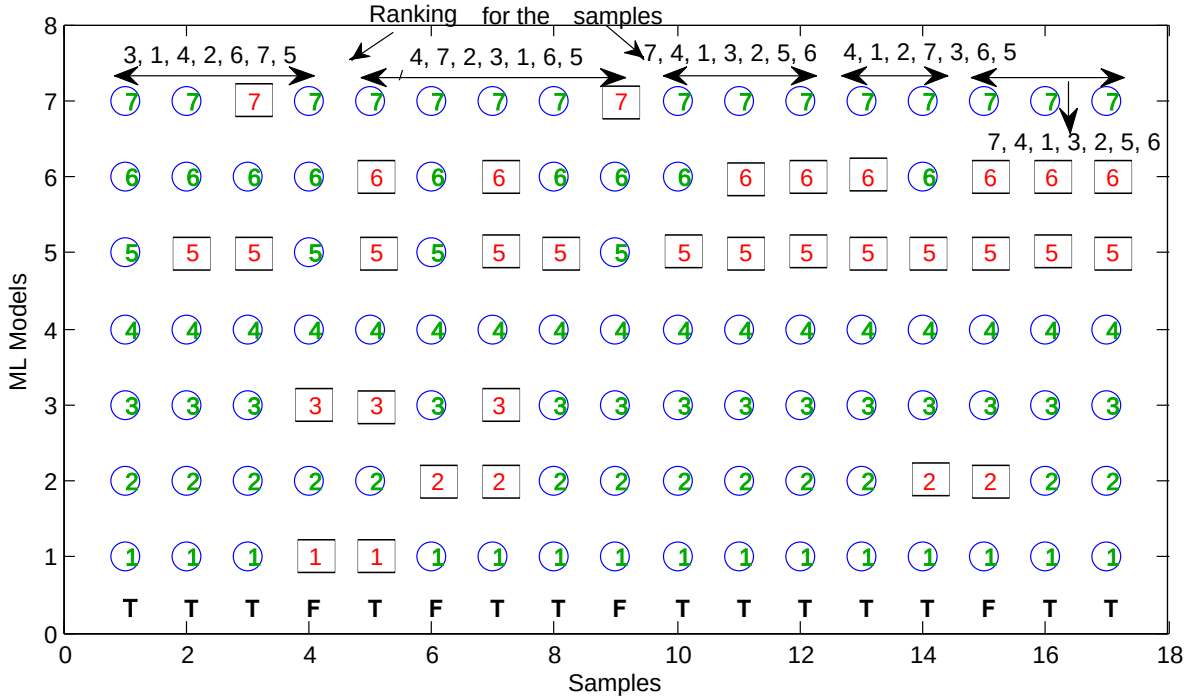


Fig. 9: Dynamic Model Selection

of consecutive erroneous predictions then its parameters are tuned UP and DOWN according to the direction of threshold crossing. The new parameters are passed to the ML docker containers via the "Scaling Analyzer" module. When all the limits are reached, the model container is deactivated. The limits are set by the user based on the dataset pattern or dataspace time window.

The auto-tuning is demonstrated here with an example. The online data stream that changes dynamically over the time is simulated by creating the dataspaces. The testing dataset is divided into 17 different *dataspace time windows*, each with $10,000$ records.

Figure 5 shows that the accuracy of "NO-ATTACK" prediction using Gradient Boosting classification over complete testing dataset is only 28.5%. On the other hand, using auto-tuning strategy, the achieved accuracy is much higher, and it dynamically changes over time as shown in Figure 10. The parameters selected for auto-tuning are $n\_estimators$, max_depth (maximum depth of the individual regression estimators which imposes a limit on the number of nodes in the tree), $min\_samples\_split$ (minimum number of samples required to split an internal node), and $max\_features$ (number of features to consider when looking for the best split) [15]. In the first experiment, Python Scikit-learn default values are used for those parameters ($n\_estimators = 100$, $max\_depth = 3$, $min\_samples\_split = 2$, $max\_features = None$) over all the dataspaces. The accuracies and parameter values are shown in blue in Figure 10. In another experiment, the parameters are tuned UP as described in Algorithm 3 at the beginning of every dataspace time window. The curve in red shows the accuracies and parameter values. The difference in accuracy between the two experiments can be seen in most of the dataspaces using parameter auto-tuning.

Algorithm 3: Parameter Tuning with Autoselection

```
Result: final_result = Parameters Tuning & Autoselection
accuracy_counter = 1;
increment_counter = 0;
decrement_counter = 0;
for accuracy_counter in ml_accuracy_history do
    if ml_accuracy_history[-accuracy_counter] == 'false' then
        accuracy_counter=accuracy_counter+1
    else
        accuracy_counter=0;
        break;
    end
    if accuracy_counter >Tuning_limit_setby_user then
        ML_parameter ← Read from Machine Learning Stats.;
        Increase the ML_parameter values;
        accuracy_counter = 0;
        increment_counter = increment_counter + 1;
    end
    if increment_counter >Tuning_UP_limit_setby_user then
        ML_parameter ← Read from Machine Learning Stats.;
        Decrease the ML_parameter values;
        accuracy_counter = 0;
        decrement_counter = decrement_counter + 1;
    end
    if counter >Deactivation_limit_setby_user or decrement_counter >Tuning_DOWN_limit_setby_user then
        increment_counter=0;
        increment_counter = 0;
        EXEC ML_Container_scale_down;
    end
end
```

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2018.2876844, IEEE Transactions on Parallel and Distributed Systems

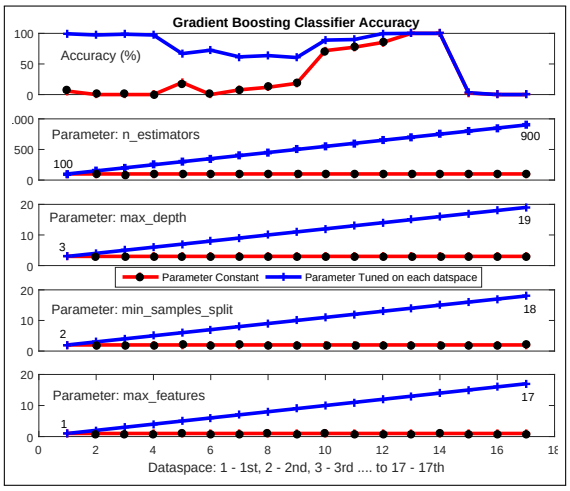


Fig. 10: Autotuning Example for Gradient Boosting Classifier on UNSW Dataset Label: 'NO-ATTACK'

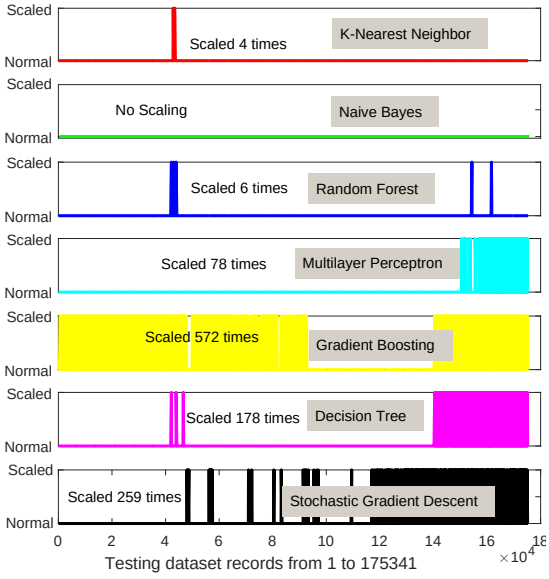


Fig. 11: Container Scaling for Label: "NO-ATTACK"

6 Scaling the ML Containers

The module "Scaling Analyzer" in Figures 1 and 2 performs the up-or-down scaling operation of ML models using metrics such as ML status, accuracy log, and other parameters stored in the "ML Stats" submodule as in Figure 7. The ML scaling rules are as follows:

(A) **Prediction Error Rule:** Any ML model that makes erroneous predictions for S consecutive samples is put in the "OFF" state. This is because it will introduce inaccuracies for the weighted majority voting. The parameter S is set by the user.

(B) **Time-out Rule:** Each ML model runs inside a docker container which might time out due to network congestion or to a completion time limit on ML algorithms. An ML docker container that times out for R consecutive samples is put in the "OFF" state and another ML docker container with the same parameter values as the discarded one is activated. The parameter R is set by the user.

(C) **Tie-breaking Rule:** Let M be the total number of ML models. When the voting results are closer to a tie between two labels for P consecutive samples and the number of active ML models is less than M, an OFF ML model is turned on for tie-breaking in the weighted majority vote. The parameter P is set by the user.

(D) **Inaction Rule:** If the number of active ML models is less than M for Q consecutive samples, the OFF ML models are activated and their predictions are included in the weighted majority vote. The parameter Q is set by the user. An ML model goes OFF due to either the Prediction Error Rule or the Time-out Rule.

The plot of MLs scaling using the Inaction Rule (A) (for scale DOWN) and Rule (D) (for scale UP) is shown in Figure 11. The numerical values for the user-defined scaling parameters are: $M = 7$, $S = 100\ samples$ and $Q = 100\ samples$. All the ML models predict the "NO-ATTACK" label for the dataset. For the entire testing set, the K-Nearest Neighbors, Naives Bayes and Random Forest are scaled less frequently than other models. This behavior is in close agreement with the ranking of ML models for the "NO-ATTACK" prediction as shown in Table 1. The feedback ML system that we have described can also be understood in the context of virtual machine autoscaling based on cloud performance metrics.

7 Accuracy with Feature Selection

Feature selection is a mechanism to choose the features automatically from the dataset that contribute most to the prediction label. Selected features can reveal the time-dependent evolution of the dataset records while unselected ones negatively impact the predictive accuracy of ML models. Other advantages of feature selection include a reduction in model overfitting and reduction in training time. The Python $sklearn.feature\_selection$ module has been used to demonstrate some of these feature-selection characteristics. All 7 supervised models of Section 3 have been used. Univariate feature selection has been adapted in this paper where the features with the strongest statistical cross-correlation with the output variable are selected. The Python Scikit-learn library provides the $SelectKBest$ class to select a specific number of features based on the cross-correlation score [15]. Figure 12 shows the classifier accuracies using the top 10, 15, and 20 features and compares with a model that uses all features. In addition, Recursive Feature Elimination (RFE) is applied to the Random Forest classifier by selecting 10 through 25 important features as shown in Figure 14. RFE works by recursively eliminating the least important attribute from the feature set and building a model based on the leftover attributes [25]. Out of the 7 classifiers, Python Scikit-learn allows only Random Forest model to use RFE. An important observation is that models based on a subset of features are not always accurate. This suggests that the dataset has large variations from one sample to another, which in turn, justifies the statistically different dataspaces that were created from the dataset to demonstrate the validity of the model autoselection and autotuning processes in Reflexive DevMLOps of Sections 4 and 5. The importance of each feature for the ML models can change over time, $e.g.$ features $\{f_1, f_2, f_3\}$ are more

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2018.2876844, IEEE Transactions on Parallel and Distributed Systems

important at time t_1 but not so important at time t_2 for the label prediction. A new model has to be created with most important features, which further motivates the need for dynamic model selection and tuning over time. To demonstrate the dynamic feature selection, the dataspaces with $10,000$ records from the training and testing dataset are again used, and new classifiers are dynamically created for each training dataspace. When such classifiers were tested using the testing dataspace, their accuracies turned out to be lower than expected. These tests are still under investigation but one possible approach to remedy the loss of accuracy is to use Q-learning. This will be undertaken in a follow-up publication.

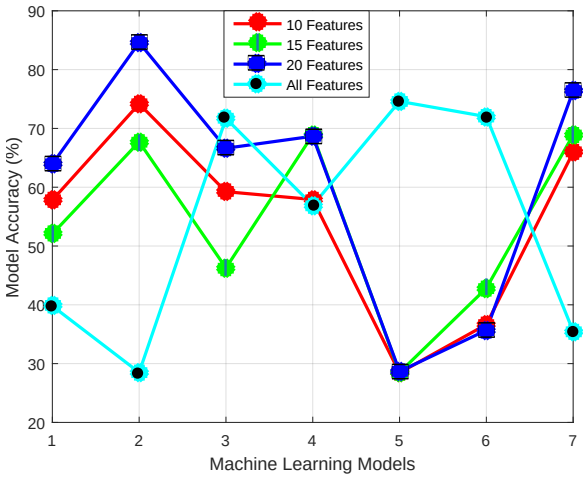


Fig. 12: Model Accuracy vs Number of Features

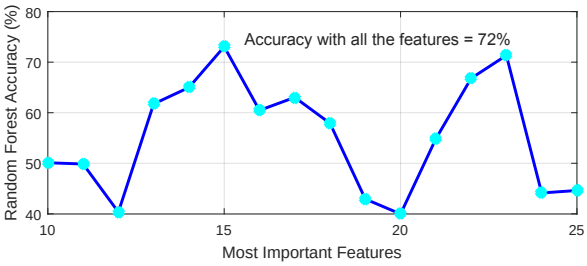


Fig. 13: Recursive Feature Elimination for Random Forest

8 Comparison with Statically-Tuned Ensemble Machine Learning

Ensemble Machine Learning (EML) has been extensively used to fulfill the need of having multiple ML models. Its goal is to combine the predictions of several individual estimators within a learning model to improve its accuracy and robustness with respect to an individual predictor. AdaBoost and XgBoost are known to be powerful EML models. The Python $sklearn.ensemble$ module provides the $AdaBoostClassifier$ and Python parent library provides independent $XgBoost$ installable EML module [34]. Both algorithms are applied over the UNSW-NB15 dataset with labels encoded as in Section 3.3. Their cross-validation accuracy for each type of label prediction is shown in Figure 14. When this is compared with the accuracy of the Reflexive DevMLOps setup (auto-selection and auto-tuning algorithms) described in Sections 4 and 5), the latter

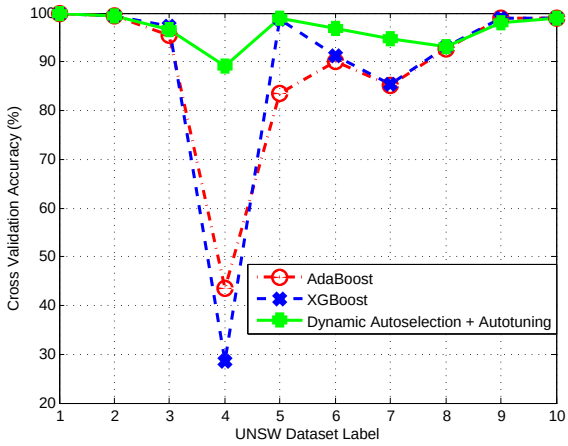


Fig. 14: Statically-Tuned Ensemble Learning Accuracy

accuracy is better than 90% for all the label predictions, especially in a dynamic environment where the datasapce changes and some of the data features lose their importance over time.

9 Classification with Deep Neural Networks

Finally we have applied deep neural networks to this dataset and compared their accuracies with the more traditional learning methods of the previous sections. The main advantage of deep neural networks, and deep learning in general, is that they provide many hyperparameters that can be tuned to build an accurate model. But as explained in section 1, a longer time is needed to build the deep learning model vs. traditional classifiers. We have performed several experiments using $Python - Keras$ and $Tensorflow$ to arrive at the right set of hyperparameters and obtain the accuracies shown in Figure 15. The hyperparameter values are as follows: {Number of Hidden Layer = 2, Number of units at input = 43, Number of units in first & second hidden layer = 60 & 30, Initialization = $Random$, Optimization = $Adam$, Activation for hidden layer = $Relu$, Activation for output = $Sigmoid$}. Different sets of hyperparameter values can be used for each type of label prediction to further improve the accuracy. Furthermore, an experiment for multi-label classification similar to section 3.2 using softmax activation at the output layer has been performed for various numbers of hidden units for each hidden layer. The result is shown in Figure 16. The hyperparameter values are as follows: {Number of Hidden Layer = 2, Number of units at input = 43, Initialization = He, Dropout probability = 0.4, Optimization= $Adam$, Activation for hidden layer = $Relu$, Activation for output= $Softmax$}. As is clear from Figures 3 and 16, the prediction accuracies can be increased to 100% by running rigorous experiments and calculating the optimal hyperparameter values but at the cost of long training times. The net conclusion is that the Reflexive DevMLOps presented in the previous sections seems to be the right design choices to improve classification accuracy without sacrificing run time.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2018.2876844, IEEE Transactions on Parallel and Distributed Systems

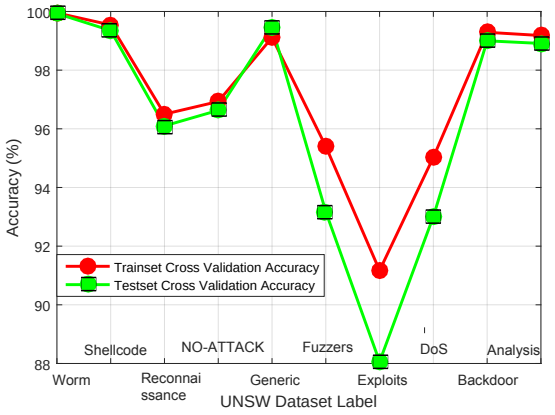


Fig. 15: Deep Learning Accuracies for each Label

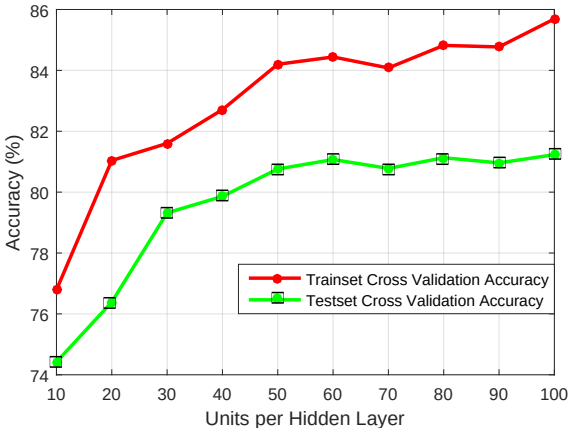


Fig. 16: Deep Learning Accuracies in Multilabel Classification

TABLE 2: ML Training and Prediction Times

Model Type	Training time(s)	Prediction time(s)
K-Means Clustering	2.198	0.007
Birch Clustering	7.696	0.017
Gaussian Mixture Model	5.583	0.034
K-nearest Neighbors (KNN)	2.001	4.032
Naive Bayess (NB)	0.074	0.0045
Random Forest (RF)	0.512	0.0054
Multi-layer Perceptron (MLP)	22.486	0.0202
Gradient Boosting (GB)	7.971	0.0077
Decision Tree (DT)	0.325	0.00097
Stochastic Gradient Descent (SGD)	0.111	0.00181
AdaBoost	3.29	0.00001
XgBoost	8.819	0.275

Voter takes $0.171sec$ for processing.

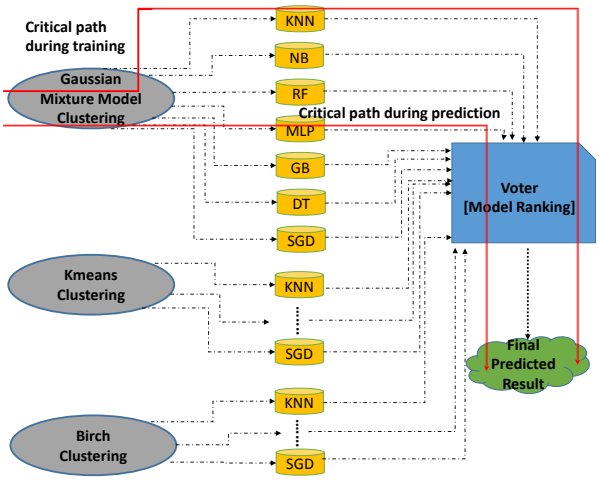


Fig. 17: Critical Path in Model Selection

10 Comparison with Ensemble Learning

As shown in Figure 2, all the ML workers execute in parallel and send the prediction result to the voter. Also, all the clustering models, shown in Figure 7, operate in parallel. As a result, the maximum time taken by the model autoselection setup is obtained by the sum of the slowest clustering model, slowest classification model and the voting stage. The training and prediction times of each ML used in the earlier sections are shown in Table 2. As expected, the multi-layer perceptron is the slowest ML model during training while the K-nearest neighbor is the slowest ML model during prediction. The presence of such MLs in the autoselection setup is shown as a critical path in Figure 17. Comparing the runtime information with that of two ensemble learning methods, namely, AdaBoost and XgBoost, the autoselection process of course runs more slowly but with the significant advantage of improved accuracy over time and dynamic learning behavior as datasets change.

11 Literature Review

For a survey on the large body of work on auto-tuning ML models, the reader is referred to [1]. In addition to approaches such as GridSearch for parameter tuning and Autoensembling for combining models, few other examples for creating statically-tuned ML models are worth pointing out. An automated tuning of the parameters of support vector machine (SVM) classifier for pattern recognition has been presented in [8]. The parameters are tuned by minimizing the estimates of the generalization error of SVM using a gradient descent algorithm over the set of parameters. This method has several advantages, including the significant reduction of runtime and the avoidance of holding back data subsets for validation. The method therefore makes full use of the training set for the optimization of parameters in contrast to cross-validation approaches. In [30] a Multi-layer Perceptron Neural Network (MLP) and a Support Vector Machine (SVM) models have used and compared for the classification of whole-sky, ground-based images of clouds. The comparison has been made based on their accuracies and standard deviations over various subsets of the image data. The results have showed they both have similar performance that is superior to the multiband thresholding algorithm popularly used for whole-sky image classification. In [19], ensembles of MLPs are used to make profit/loss decision in stock trading. A number of stock trading rules are extracted by a graph-based evolutionary algorithm. The MLP weights are used to provide similarity metrics to these rules. Results have confirmed that an ensemble of different MLPs produces better accuracy than any single model. An automated ML model search named TUPAQ has been proposed in [27] that automatically finds and trains models for a users predictive application. The quality of the resulting

ML model is similar to those found by exhaustive strategies but requires less computational effort. A comparison of five machine learning models: Naive Bayes, KNN, Random Forests, SVM, and Neural Networks has been performed in [10] using spatially constrained, remotely sensed geophysical data. The comparison results have shown Random Forests to be a good first choice in terms of computational efficiency, stability with respect to variations in the model parameter, and accuracy with respect to other ML models.

Reinforcement learning has been used for dynamic model selection in [12] where the model dynamically adapts to the environment based on expert models for each operational space to optimize resource utilization in cloud data centers. Unfortunately, reinforcement learning has a tendency to get trapped in local minima, which would restrict its ability to learn any further [28]. One possible remedy is to use reinforcement with deep learning [17].

A reinforcement learning approach for the automatic design of a deep neural network is presented in [3] where it has been applied for image recognition with CIFAR-10 and for language modeling with the Penn Treebank dataset. A feedback system is used where the controller creates multiple child models and based on each child's feedback, the controller learns to assign high/low probability to the areas where the deep learning architecture has achieved good/bad accuracy. A similar approach is the use of Evolutionary Algorithms to discover the "best" neural network for image classification [26]. The method has been tested on the CIFAR-10 and CIFAR-100 datasets with classification accuracy of more than 90% and compared with reinforcement and ensemble learning in terms of computational complexity and training cost.

An application-agnostic, ML autoscaler called MLscale has been proposed in [32]. A traditional autoscaler requires deep understanding of the application domain, the underlying cloud infrastructure, and workload dynamics in order to accurately scale resources. Such information is not always available to the system administrator, and a possible solution according to [32] is to use a neural network for online performance modeling and a regression predictor to estimate post-scaling performance and system metrics, respectively. Result have shown a resource cost reduction by 50% in comparison with the optimal static policy. A survey of self-aware, adaptive, runtime autoscaling for cloud-based applications and services is given in [9]. In particular, the major requirements for autoscaling in cloud computing are discussed and suggestions for further research in this area are given. As for feature selection methods, a survey has been made in [7] with the objective of illustrating how variable elimination can be applied to a wide array of machine learning problems.

12 Conclusions and Future Work

In this paper, several supervised and unsupervised ML models have been combined and applied on a cloud security dataset. To achieve higher accuracy, multiple models are created for predicting each type of security attack. The need of unsupervised models in addition to the supervised ones is demonstrated and shown to result in improved prediction accuracy. A Reflexive DevMLOps scheme for model selection according to dynamic data evolution is presented. New models are trained offline and brought online as needed, while older tuned models are discarded when they can no longer achieve adequate prediction accuracy. Models brought online are further automatically tuned to adapt their prediction accuracy to the dynamic data set. The requirements of dynamic feature selection in the context of distributed learning is explained. A comparison with the boosting ensemble learning is made to demonstrate the superiority of dynamic model selection over statically trained ensemble learning. In the future, we plan to add Reinforcement learning models to the portfolio of ML models that are available for autotuning and autoselction in our toolset. We also plan to exercise the expanded toolset on a variety of cloud computing datasets.

Acknowledgments

The first and third authors would like to thank IBM Research for hosting them at the IBM T. J. Watson Research Center, Yorktown Heights, NY, during the finalization of this paper. This work has been conducted under the framework of a Joint Study Agreement, No. W1463335, between IBM Research and Khalifa University, Abu Dhabi, UAE.

References

[1] Machine learning for automated algorithm design. http://www.ml4aad.org/automl/.

[2] UNSW-NB15 Dataset dataset features and size description. https://www.unsw.adfa.edu.au/australian-centre-for-cyber-security/cybersecurity/ADFA-NB15-Datasets/. Accessed: 2017-08-16.

[3] Using Machine Learning to Explore Neural Network Architecture google research blog. https://research.googleblog.com/2017/05/using-machine-learning-to-explore.html. Accessed: 2017-09-25.

[4] Acetozi, J. Docker. In *Pro Java Clustering and Scalability*. Springer, 2017, pp. 3–11.

[5] Bauer, E. Improving operational efficiency of applications via cloud computing. *IEEE Cloud Computing 5*, 1 (2018), 12–19.

[6] Botta, A., De Donato, W., Persico, V., and Pescapé, A. Integration of cloud computing and internet of things: a survey. *Future Generation Computer Systems 56* (2016), 684–700.

[7] Chandrashekar, G., and Sahin, F. A survey on feature selection methods. *Computers & Electrical Engineering 40*, 1 (2014), 16–28.

[8] Chapelle, O., Vapnik, V., Bousquet, O., and Mukherjee, S. Choosing multiple parameters for support vector machines. *Machine learning 46*, 1 (2002), 131–159.

[9] Chen, T., and Bahsoon, R. Survey and taxonomy of self-aware and self-adaptive autoscaling systems in the cloud. *arXiv preprint arXiv:1609.03590* (2016).

[10] Cracknell, M. J., and Reading, A. M. Geological mapping using remote sensing data: A comparison of five machine learning algorithms, their response to variations in the spatial distribution of training data and the use of explicit spatial information. *Computers & Geosciences 63* (2014), 22–33.

[11] Estrada, R., and Ruiz, I. The broker: Apache kafka. In *Big Data SMACK*. Springer, 2016, pp. 165–203.

[12] Farahnakian, F., Liljeberg, P., and Plosila, J. Energy-efficient virtual machines consolidation in cloud data centers using reinforcement learning. In *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on* (2014), IEEE, pp. 500–507.

[13] Ferguson, D. F., and Muñoz, V. M. Journal of grid computing, special issue of cloud computing and services science. *Journal of Grid Computing 15*, 2 (2017), 139–140.

[14] Fitkov-Norris, E., Vahid, S., and Hand, C. Evaluating the impact of categorical data encoding and scaling on neural network classification performance: the case of repeat consumption of identical cultural goods. In *International Conference on Engineering Applications of Neural Networks* (2012), Springer, pp. 343–352.

[15] GARRETA, R., AND MONCECCHI, G. *Learning scikit-learn: machine learning in python*. Packt Publishing Ltd, 2013.

[16] GOMES, H. M., BARDDAL, J. P., ENEMBRECK, F., AND BIFET, A. A survey on ensemble learning for data stream classification. *ACM Computing Surveys (CSUR) 50*, 2 (2017), 23.

[17] LI, P., LI, J., HUANG, Z., LI, T., GAO, C.-Z., YIU, S.-M., AND CHEN, K. Multi-key privacy-preserving deep learning in cloud computing. *Future Generation Computer Systems 74* (2017), 76–85.

[18] LIST, M. Using docker compose for the simple deployment of an integrated drug target screening platform. *Journal of integrative bioinformatics 14*, 2 (2017).

[19] MABU, S., OBAYASHI, M., AND KUREMOTO, T. Ensemble learning of rule-based evolutionary algorithm using multi-layer perceptron for supporting decisions in stock trading problems. *Applied soft computing 36* (2015), 357–367.

[20] MOUSTAFA, N., AND SLAY, J. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *Military Communications and Information Systems Conference (MilCIS), 2015* (2015), IEEE, pp. 1–6.

[21] MOUSTAFA, N., AND SLAY, J. The evaluation of network anomaly detection systems: Statistical analysis of the unsw-nb15 data set and the comparison with the kdd99 data set. *Information Security Journal: A Global Perspective 25*, 1-3 (2016), 18–31.

[22] ORLENKO, A., MOORE, J. H., ORZECHOWSKI, P., OLSON, R. S., CAIRNS, J., CARABALLO, P. J., WEINSHILBOUM, R. M., WANG, L., AND BREITENSTEIN, M. K. Considerations for automated machine learning in clinical metabolic profiling: Altered homocysteine plasma concentration associated with metformin exposure. In *Pac Symp Biocomput* (2017), vol. 23, World Scientific.

[23] PARK, J., LEE, S., LEE, H., AND LEE, B. Implementation of big data analysis system for ems. *Journal of Platform Technology 3*, 4 (2015), 29–42.

[24] POTDAR, K., PARDAWALA, T. S., AND PAI, C. D. A comparative study of categorical variable encoding techniques for neural network classifiers. *International Journal of Computer Applications 175*, 4 (2017), 7–9.

[25] RAVISHANKAR, H., MADHAVAN, R., MULLICK, R., SHETTY, T., MARINELLI, L., AND JOEL, S. E. Recursive feature elimination for biomarker discovery in resting-state functional connectivity. In *Engineering in Medicine and Biology Society (EMBC), 2016 IEEE 38th Annual International Conference of the* (2016), IEEE, pp. 4071–4074.

[26] REAL, E., MOORE, S., SELLE, A., SAXENA, S., SUEMATSU, Y. L., LE, Q., AND KURAKIN, A. Large-scale evolution of image classifiers. *arXiv preprint arXiv:1703.01041* (2017).

[27] SPARKS, E. R., TALWALKAR, A., HAAS, D., FRANKLIN, M. J., JORDAN, M. I., AND KRASKA, T. Automating model search for large scale machine learning. In *Proceedings of the Sixth ACM Symposium on Cloud Computing* (2015), ACM, pp. 368–380.

[28] SUTTON, R. S., AND BARTO, A. G. *Reinforcement learning: An introduction*, vol. 1. MIT press Cambridge, 1998.

[29] SYED, H. J., GANI, A., AHMAD, R. W., KHAN, M. K., AND AHMED, A. I. A. Cloud monitoring: A review, taxonomy, and open research issues. *Journal of Network and Computer Applications* (2017).

[30] TARAVAT, A., DEL FRATE, F., CORNARO, C., AND VERGARI, S. Neural networks and support vector machine algorithms for automatic cloud classification of whole-sky ground-based images. *IEEE Geoscience and remote sensing letters 12*, 3 (2015), 666–670.

[31] TURNBULL, J. *The Docker Book: Containerization is the new virtualization*. James Turnbull, 2014.

[32] WAJAHAT, M., GANDHI, A., KARVE, A., AND KOCHUT, A. Using machine learning for black-box autoscaling. In *Green and Sustainable Computing Conference (IGSC0¡ 2016 Seventh International* (2016), IEEE, pp. 1–8.

[33] ZHANG, M.-L., AND ZHOU, Z.-H. A review on multi-label learning algorithms. *IEEE transactions on knowledge and data engineering 26*, 8 (2014), 1819–1837.

[34] ZHANG, Q. *Modern Models for Learning Large-Scale Highly Skewed Online Advertising Data*. University of California, Los Angeles, 2015.

Rupesh Raj Karn is currently a PhD student in Electrical and Computer Engineering at Khalifa University of Science and Technology, Abu Dhabi, UAE, working in the area of Automated Management of Cloud Clusters and Applications using Machine Learning Techniques. He was an intern with the IBM T. J. Watson Research Center in the Summers of 2016, 2017, and 2018, working on various aspects of cloud monitoring, benchmarking, and management. He was granted a Best Paper Award from the UAE Graduate Student Conference in April 2018. He received his MSc in Microsystems Engineering from the Masdar Institute of Science and Technology, Abu Dhabi, UAE, in 2014, and his BSc in Electronics Engineering from Sardar Vallabhbhai National Institute of Technology (SVNIT), Surat, India, in 2011.

Prabhakar Kudva is a Research Staff Member at the IBM T. J. Watson Research, Yorktown Heights, NY, where he currently leads several projects in the areas of Enterprise Data Centers, Cloud Computing for business intelligence and analytics, PaaS, and CaaS. He has received several IBM awards for High-Value Patents and Outstanding Technical Achievements as well as the IEEE Region 1 Award for Outstanding Contributions to the Design Automation of Resilient Chips and Systems. Dr. Kudva was on the adjunct faculty of Yale University and Columbia University. He received his PhD in Computer Science from the University of Utah in 1995.

Ibrahim (Abe) M. Elfadel is Professor of Electrical and Computer Engineering at the Khalifa University of Science and Technology, Abu Dhabi, UAE. Since May 2013, he has been the founding co-director of the Abu Dhabi Center of Excellence on Energy-Efficient Electronic Systems (ACE^{4}S), and since May 2014, he has been the Program Manager of TwinLab MEMS, a joint collaboration with GLOBALFOUNDRIES and the Singapore Institute of Microelectronics on microelectromechanical systems. Between November 2012 and October 2015, he was the founding co-director of Mubadala's TwinLab 3DSC, a joint research center on 3D integrated circuits with the Technical University of Dresden, Germany. He also headed the Masdar Institute Center for Microsystems (iMicro) from November 2013 until March 2016. Between 1996 and 2010, he was with the corporate CAD organizations at IBM Research and the IBM Systems and Technology Group, Yorktown Heights, NY, where he was involved in the research, development, and deployment of CAD tools and methodologies for IBM's high-end microprocessors. His current research interests include IoT platform prototyping; energy-efficient edge and cloud computing; IoT communications; power and thermal management of multi-core processors; low-power, embedded digital-signal processing; 3D integration; and CAD for VLSI, MEMS, and Silicon Photonics. Dr. Elfadel is the recipient of six Invention Achievement Awards, one Outstanding Technical Achievement Award and one Research Division Award, all from IBM, for his contributions in the area of VLSI CAD. In 2014, he was the co-recipient of the D. O. Pederson Best Paper Award from the IEEE Transactions on Computer-Aided Design Automation for Integrated Circuits and Systems. He is also the co-editor (with Gerhard Fettweis) of "3D Stacked Chips: From Emerging Processes to Heterogeneous Systems," Springer, 2016, and the co-editor (with Mohammed Ismail) of the upcoming book: "The IoT Physical Layer: Design and Implementation," Springer, 2018. Between 2009 and 2013, Dr. Elfadel served as an Associate Editor of the IEEE Transactions on Computer-Aided Design. He is currently serving as Associate Editor of the IEEE Transactions on VLSI Systems and on the Editorial Board of the Microelectronics Journal (Elsevier). Dr. Elfadel has also served on the Technical Program Committees of several leading conferences, including DAC, ICCAD, ASPDAC, DATE, ICCD, ICECS, and MWSCAS. Most recently, he was the General Co-chair of the IFIP/IEEE 25^th International Conference on Very Large Scale Integration (VLSI-SoC 2017), Abu Dhabi, UAE, October 23- 25, 2017. He received his PhD from MIT in 1993.