Machine Learning for Dynamic Resource Allocation in Network Function Virtualization

Stefan Schneider*, Narayanan Puthenpurayil Satheeschandran†, Manuel Peuster*, and Holger Karl*
*Paderborn University, Germany: {stefan.schneider, manuel.peuster, holger.karl}@upb.de
†Amrita School of Engineering, Amritapuri, India: thepsnarayanan@gmail.com

***Abstract*—Network function virtualization (NFV) proposes to replace physical middleboxes with more flexible virtual network functions (VNFs). To dynamically adjust to ever-changing traffic demands, VNFs have to be instantiated and their allocated resources have to be adjusted on demand. Deciding the amount of allocated resources is non-trivial. Existing optimization approaches often assume fixed resource requirements for each VNF instance. However, this can easily lead to either waste of resources or bad service quality if too many or too few resources are allocated.**

To solve this problem, we train machine learning models on real VNF data, containing measurements of performance and resource requirements. For each VNF, the trained models can then accurately predict the required resources to handle a certain traffic load. We integrate these machine learning models into an algorithm for joint VNF scaling and placement and evaluate their impact on resulting VNF placements. Our evaluation based on real-world data shows that using suitable machine learning models effectively avoids over- and under-allocation of resources, leading to up to 12 times lower resource consumption and better service quality with up to 4.5 times lower total delay than using standard fixed resource allocation.

I. Introduction

In network function virtualization (NFV), network services consist of multiple interconnected virtual network functions (VNFs), which can be scaled and placed at available points of presence (PoPs) on demand. Dynamically scaling and placing such network services and VNFs is non-trivial since it depends on various characteristics of the network service, the underlying substrate network, as well as the current service demand. Due to this complexity and because service demands are constantly changing, significant research effort focuses on designing algorithms and optimization approaches to automate such dynamic scaling and placement of network services according to the current demand. In addition to the number and location of VNF instances, these algorithms need to decide how many resources to allocate to each instance.

Conventionally, each VNF is instantiated with a fixed amount of resources that is pre-defined by the VNF developer and specified in the VNF descriptor. However, VNFs typically do not require constant resources but need more or less resources depending on the traffic load. Thus, allocating a fixed amount of resources easily leads to either under- or over-allocation of resources. In the case of under-allocation, VNFs lack resources to process all packets, leading to packet drops and reducing the service quality or even completely destroying the service functionality. Thus, operators typically try to avoid under-allocation by allocating fixed, high amounts of resources to each VNF instance. This, in turn, leads to over-allocation, wasting resources, blocking other VNFs from using these resources, and resulting in unnecessary high costs.

To mitigate these issues and take appropriate resource allocation decisions, an understanding of the relationship between the load a VNF is supposed to handle, the resources dedicated to it, and the resulting performance is necessary – expressed in a so-called *performance profile*. VNF performance profiling is an approach to measure such performance with varying resource configurations [1]–[3]. The relationship between VNF performance and resource requirements is often non-linear [4] and hard to extract from raw measurement data, which can contain millions of data points [5].

These properties make performance profiles ideal candidates for machine learning, automatically deriving suitable, accurate, and compact models from measurement data. We compare machine learning approaches from different families of regression algorithms (e.g., linear, tree-based, kernel-based) as well as neural networks. For each approach, we evaluate the prediction accuracy and ability to generalize for predictions that lie between measured data points. Rather than proposing a completely new VNF placement algorithm, we focus on how to model and decide the amount of required and allocated resources per VNF instance. The derived models can then be used inside existing or future VNF placement algorithms to accurately predict resource requirements for each VNF instance and allocate resources accordingly. This machine learning-based dynamic resource allocation helps to ensure good service quality while avoiding over- and under-allocation. Overall, our contributions are three-fold:

- We design a modular workflow for deriving trained machine learning models from raw performance measurements and for using them in VNF placement algorithms to accurately predict VNF resource requirements.
- We use six different machine learning algorithms on real-world VNF data to train and compare resulting models for predicting VNF resource requirements.
- We integrate these models for resource prediction and allocation into an existing VNF placement algorithm [6]. We evaluate their impact on resulting VNF placement quality and investigate potential trade-offs regarding computational runtime.

After discussing related work in Sec. II, we provide a

formal problem statement and motivate the importance of accurate, dynamic resource allocation using an example (Sec. III). In Sec. IV, we present our proposed workflow leveraging machine learning for dynamic resource allocation in VNF placement algorithms and evaluate it using real-world data in Sec. V.

II. Related work

Significant research interest has been drawn to designing novel algorithms and approaches that optimize automatic scaling, placement, and resource allocation in NFV [7] – both with and without machine learning.

A. Approaches without Machine Learning

Most existing VNF placement algorithms considers pre-defined, fixed resource requirements per VNF instance [8]–[12]. While these algorithms may use horizontal scaling, i.e., adding or removing VNF instances, to adjust to changing demand, they completely neglect vertical scaling, i.e., dynamically adjusting the allocated resources per VNF instance. As mentioned before, pre-defined, fixed resource allocation can easily lead to poor service quality or wasted resources.

Recent approaches [6], [13] improve this simplistic model of fixed resource requirements by considering a linear relationship between the VNF performance and required resources. While this increases the flexibility of the model, it requires proper parametrization, which is non-trivial, and it may not be very realistic, since VNFs often do not have a strictly linear relationship between allocated resources and VNF performance [4].

To this end, Dräxler and Karl [4] propose the use of piece-wise linear functions instead. A major drawback, especially for highly non-linear functions, is that the approximation with piece-wise linear functions either becomes increasingly inaccurate or increasingly cumbersome as the function has to be split in smaller and smaller pieces. Similarly, Eramo et al. [14] use a quite complex model considering packet length, flow data rate, and the traffic state (assuming cycle-stationary traffic) to compute a realistic model for VNF performance and resource requirements. Due to complex required a priori knowledge (e.g., about the traffic states), such a model is challenging to use.

Overall, most models for resource allocation that are used in existing VNF placement algorithms are not very realistic. In contrast, our machine learning approach learns accurate models automatically and directly from real VNF performance measurements, building on existing work on VNF performance profiling [1]–[3]. Our work is complementary to most research in VNF placement as the resulting trained machine learning models can be integrated into existing approaches with little overhead.

B. Approaches with Machine Learning

As one of the open challenges for machine learning in networking, Ayoubi et al. [15] mention the mapping from high-level requirements to low-level configurations. Our work is addressing this challenge by automatically mapping desired VNF performance to required resource configurations.

Similarly, other machine learning-based approaches have been used successfully in network management to predict VNF processing delays [16] and traffic demands [17], [18]. Our approach is complementary and could be combined with this related work.

Similar to our work, other authors have proposed to use machine learning to predict VNF resource requirements [19], [20]. Sembiring et al. [19] only rely on historical data but do not take traffic load into account. Mijumbi et al. [20] use graph neural networks to derive a model for VNF resource requirements from the topology of sub-components within a VNF. In contrast to our approach, theirs requires insight into internals of each VNF. It also disregards that different instances of the same VNF may process different traffic loads and thus may have very different resource requirements, limiting the accuracy of their model.

Most similar to our work, van Rossem et al. [21] use VNF performance profiling and propose machine learning to learn from VNF performance measurements. Compared to our work, van Rossem et al. focus more on the process of VNF performance profiling but also mention dynamic resource allocation as potential, high-level use case. Complementing and going beyond their work, we apply machine learning on available VNF performance measurements and particularly focus on integrating the trained models into VNF placement algorithms for dynamic resource allocation. Additionally, we evaluate the impact of different models on resulting VNF placements.

III. Problem Statement

Our goal is to use machine learning to build useful performance profiles, which accurately model and predict VNF resource requirements. We then use these machine learning models inside existing VNF placement algorithms for improved dynamic resource allocation. These algorithms – irrespective of which model for resource requirements they use – dynamically scale, place, and allocate resources to VNFs, solving the *VNF placement problem* defined in Sec. III-A. Sec. III-B illustrates why it is crucial to allocate VNF resource accurately and dynamically according to the current traffic load rather than allocating pre-defined, fixed amounts of resources. In the end, we intend to show that using machine learning-based performance profiles results in superior performance compared to using simpler models.

A. VNF Placement Problem

The main objective of VNF placement is to provide network services to users by embedding them in the substrate network according to the current user demand. In doing so, VNF placement algorithms have to consider VNF resource requirements, capacity limitations of the network, and should minimize the resulting end-to-end delay by deploying VNF instances close to their users. For completeness, we define this problem more formally in the following meta-discourse, largely in line with existing definitions of this problem [7].

The substrate network $G = (V, L)$ is the underlying compute network, in which the given network services

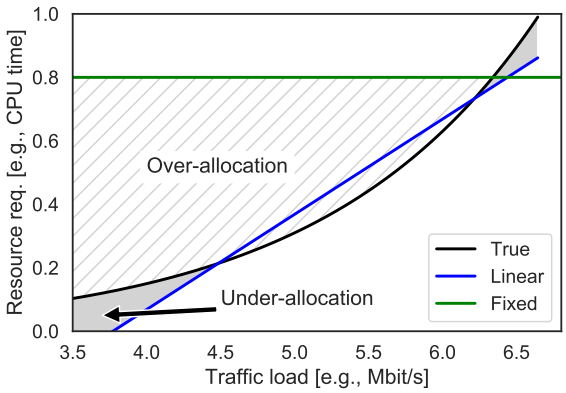

Fig. 1: Example of true VNF resource requirements in comparison to fixed and linear resource allocation.

should be deployed. Each node $v \in V$ has a capacity of $r_v \in \mathbb{R}_{\geq 0}$, which may be zero. Here, we consider an abstract resource type, which may correspond to available CPU or memory.[1] Similarly, each link $l \in L$ has a capacity $r_l \in \mathbb{R}_{\geq 0}$ (e.g., corresponding to the maximum data rate) and a delay $d_l \in \mathbb{R}_{\geq 0}$. We assume unlimited capacity and zero delay for traffic sent between VNF instances on the same node. Users (or other sources of traffic) are distributed across the network where each user $u \in U$ is located at a network node v and requests a network service S with a data rate λ_u.

A network service $S = (C, A)$ consists of multiple components $c \in C$ (i.e., VNFs), which are interconnected by arcs (or often called virtual links) $a \in A$. VNFs can be instantiated one or more times, where each instance of VNF c is allocated a certain amount of resources $r_c \in \mathbb{R}_{\geq 0}$ according to its resource requirements. In existing literature (Sec. II), r_c is typically assumed to be a pre-defined, fixed value or a linear function $r_c(\lambda)$ of the traffic load λ (e.g., in byte/s). In Sec. III-B, we show that both assumptions are problematic as they easily lead to over- or under-allocation. In Sec. IV, we therefore propose to model and predict $r_c(\lambda)$ using machine learning.

B. Why Accurate VNF Resource Allocation Matters

Instead of providing a new VNF placement algorithm, we focus on how to model $r_c(\lambda)$, deciding the amount of required and allocated resources per VNF instance. As detailed in Sec. II, the most common approach is to assign a pre-defined, fixed amount of resources to all instances of a VNF. However, this neglects the fact that most VNFs require more resources to process higher rates of traffic. Fig. 1 illustrates the problem using example data, which is similar to the Nginx dataset in our evaluation (Sec. V-A2). Here, the true resource requirements of the VNF increase superlinearly with increasing traffic load. When assigning fixed resources based on the maximum expected traffic (here, $r_c = 0.8$), this leads to severe over-allocation, i.e., far more resources being

[1]It is straight forward to extend this model to any number n of resource types per node by using a vector $r_v \in \mathbb{R}^n_{\geq 0}$ instead.

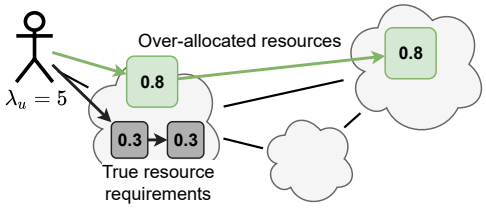

Fig. 2: Two example placement options depending on VNF resource allocation. Severe over-allocation leads to placement of VNFs at separate nodes and additional link delay.

allocated than are really required, which cannot be used by other VNF instances.

Fig. 2 shows an example with two placement options of two chained VNFs, depending on the resource allocation model of Fig. 1. The true required resources $r_c(5) = 0.3$ (gray) would allow placing both VNF instances close to the user with low delay. However, when allocating fixed resources $r_c = 0.8$ (green) to each instance, they have to be distributed over different nodes, leading to unnecessary link delay and reduced service quality.

To avoid such severe over-allocation, some VNF placement algorithms approximate VNF resource requirements as a linear function of the traffic load that each VNF instance has to process. In doing so, each VNF instance is assigned an amount of resources that is directly proportional to its traffic load. Fig. 1 shows a linear function that was determined using linear regression on the true VNF data in this example. The figure illustrates that such linear approximation can effectively reduce the risk of over-allocation. However, for traffic loads between 4.5 and 6.2, the linear function still allocates slightly more resources than are really required.

Additionally, depending on which linear function is chosen, the problem of under-allocation arises. In this example, the linear function predicts less resources than are really required for traffic loads below 4.5 and above 6.2. Consequently, an algorithm using such a linear approximation for resource dimensioning may allocate less resources to VNF instances than they actually need. In the following, we investigate how machine learning may help to accurately predict VNF resource requirements by using more powerful regression approaches that can be trained on real VNF data.

IV. Machine Learning for Dynamic VNF Resource Allocation

To ensure that the allocated resources closely match the real requirements of each VNF instance, we propose to use performance profiles derived from measurement data using a machine learning-based approach. We provide an overview of the approach in Sec. IV-A and describe its details in Sec. IV-B and IV-C.

A. Overview

Fig. 3 illustrates the workflow of our proposed overall approach, which consists of four steps. It starts with profiling a given VNF, testing it systematically with different resource

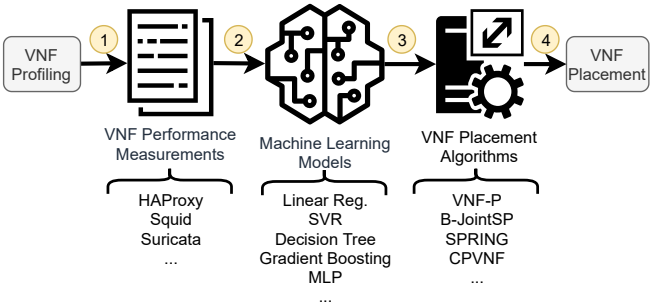

Fig. 3: Modular workflow from raw VNF performance measurements to machine learning-based VNF resource allocation and placement.

configurations and measuring its corresponding performance (step 1). In step 2, the resulting raw VNF performance measurements are used to train machine learning models. In doing so, these machine learning models learn to predict the amount of resources a specific VNF requires to successfully handle a given traffic load. The trained models generalize the measured data such that predictions can be made for traffic loads that lie between measured data points. The accuracy of these predictions depends on how well the model can generalize from the training data. In step 3, the machine learning models are integrated into a VNF placement algorithm, where they are used to predict the resource requirements $r_c(\lambda)$ for an instance of a VNF c that needs to process traffic load λ. Hence, in step 4, these VNF placement algorithms can leverage the machine learning models for accurate resource allocation, mitigating the problem of over- and under-allocation (Sec. III-B).

The whole process from step 1 to 3 can be performed offline in advance, e.g., directly after developing and publishing a new VNF. While the process is time-consuming, it only has to be performed once per VNF since trained machine learning models can be saved and reused later. In step 4, the VNF placement algorithm loads the trained machine learning models for all requested VNFs and uses them to dynamically decide the allocated resources of each placed VNF instance. In contrast to training machine learning models, predicting VNF resource requirements is very fast (less than 1 ms in our evaluation) and can be done online.

The described process is modular in that all of the components are freely exchangeable. In particular, it is not limited to a specific kind of VNF, machine learning model, or VNF placement algorithm (Fig. 3). Instead, VNF performance profiling can automatically measure the performance of any available VNF and the measurement results can be used for training using any machine learning regression algorithm. We suggest and compare candidate machine learning algorithms in Sec. IV-B and V. Finally, the trained machine learning models can be integrated into existing VNF placement algorithms with relatively little overhead.

To facilitate reproducibility, our trained machine learning models, their integration into an existing VNF placement algorithm, and all other code and data are publicly available as open-source project [22].

B. Predicting VNF Resource Requirements

1) VNF Performance Profiling: VNF performance profiling is the process of systematically testing a given VNF under different resource (and other) configurations while measuring its performance, e.g., in terms of max. sustainable traffic load, when sending as much traffic as possible through the VNF. Several authors have proposed how to automate this process [1]–[3] and perform it effectively within limited time [23]. There is also a trend to open datasets of community-created performance profiles [5].

Performance profiling leads to massive amounts of raw measurements per VNF (typically millions of data points [5]), reflecting the real behavior and dynamics of a VNF. While these recorded datasets usually contain many, potentially VNF-specific configuration parameters and measurement metrics, for the purpose of VNF resource allocation, we are only interested in metrics related to VNF resource requirements and the corresponding performance. Typically, various configurations of CPU time (in percent) and memory (e.g., in MB) are tested per VNF, recording performance in terms of max. sustainable traffic load (e.g., in Mbit/s). In addition, other parameters possibly affecting VNF performance (e.g., configurable VNF thread count) should be considered.

2) Training Machine Learning Models: For training machine learning models on VNF performance measurement data, the dataset needs to be split into a feature set containing the performance measurements (e.g., max. sustainable traffic load) and a set of target values containing the resource requirements. Due to possible errors in the performance profiling process, some values may be missing in the dataset. Since most machine learning models cannot deal with missing values, a common practice is to set these values to the median of the corresponding feature. Additionally, it is important to normalize or scale all features to values between 0 and 1, which helps ensure similar impact of features with different value ranges and can significantly improve convergence speed when training with gradient descent [24]. Finally, machine learning models can be trained minimizing their prediction error, e.g., using stochastic gradient descent [25].

The trained machine learning models can be saved and reused later without re-training. Similarly, the scaler used for preprocessing the data can be saved and reused to ensure consistent processing of any new data. We suggest to bundle VNFs together with their trained machine learning models and scalers in the VNF package or upload the trained models in an online repository. In the VNF descriptor, additional (optional) fields may reference the saved model and scaler of each VNF such that the models can easily be retrieved and used by a VNF placement algorithm.

3) Suitable Machine Learning Algorithms: Any machine learning regression algorithm can be used for training models on VNF performance measurements. Here, we present different families of candidate regression algorithms and asses their suitability for performance profiles.

a) Linear Regression: Linear regression is the simplest regression technique and can only model linear dependencies between features and target values (here, VNF performance and resource requirements). There are numerous variations of linear regression, e.g., ridge regression, which is a regularized version of linear regression that works particularly well with multiple correlated features (e.g., traffic load in bytes/s and packets/s). Due to their simplicity, linear models are easy and fast to train with few hyperparameters to tune. At the same time, linear regression cannot approximate common non-linear VNF data well.

b) Support Vector Regression (SVR): To avoid overfitting, SVR uses regularization to find the simplest, most general function that, as far as possible, fits all training data within a tolerance ϵ. Regularization strength C and tolerance ϵ are hyperparameters that can be set according to domain knowledge or tuned automatically (see Sec. IV-B4). If configured appropriately, SVR's regularization leads to simple but not too simple models that generalize well to new inputs. However, training time complexity is more than quadratic with the number of samples, making it problematic for very large VNF datasets.

c) Decision Trees: Decision trees perform regression by breaking the dataset into smaller and smaller subsets based on learned characteristics of the data. Decision trees work well even on non-scaled data and prediction results are easy to interpret since the decision trees can be visualized. A major drawback of decision trees is that they easily overfit and thus not generalize well to new data.

d) Ensemble Learning: Ensemble learning combines multiple machine learning models and combines their predictions. This allows leveraging the benefits of different models, controlling overfitting, and performing better than any individual model. Popular ensemble learning methods are random forest and gradient boosting, which both use multiple decision trees.

e) Neural Networks: While there are many different architectures of neural networks, the most common one is the multi-layer perceptron (MLP), which consists of an input and output layer with interconnected hidden layers in between. Input values pass through the neural network layer by layer (no recurrent layers). For each layer, they are multiplied by learned weights and modified by an activation function to produce the final prediction. When designed and configured correctly, MLPs can learn complex, non-linear relationships in data with no feature engineering. However, they usually require lots of data and long training to work well.

Overall, each machine learning approach has advantages and disadvantages. For VNF data with strong linear relationships, linear regression may suffice. For more complex data, SVR, ensemble learning, or MLP are more suitable. We further investigate these different approaches in our evaluation (Sec. V).

4) Hyperparameter Tuning: Each machine learning model has several hyperparameters (e.g., learning rate, batch size, etc.), which can significantly impact the performance of the model. The optimal hyperparameter setting can vary from algorithm to algorithm and from dataset to dataset. Hence, we propose tuning hyperparameters automatically using a combination of grid search and k-fold cross-validation. In particular, this means testing each combination of possible hyperparameter values within a given parameter space by randomly splitting the VNF data into a training dataset and a validation dataset (splits of 70%:30% are typical). The model is then trained with the selected hyperparameter values on the training set and evaluated on the validation set to avoid over-estimation errors. This process is repeated k times for each hyperparameter setting, averaging the resulting validation errors. Finally, the hyperparameter setting with the lowest validation error is chosen and the model can be trained with these hyperparameters on the entire dataset.

Clearly, the size of k affects the total time for tuning the hyperparameters. To reduce tuning time, especially for large datasets, k should be chosen relatively small (e.g., k=5). If the space of hyperparameters is large, it is also useful to randomly sample hyperparameter values rather than performing an exhaustive grid search.

C. Integration with VNF Placement Algorithms

Having trained models for all VNFs of interest, they can be integrated into an existing or new VNF placement algorithm as follows. First of all, the VNF placement algorithm needs load the trained and associated machine learning model of each VNF into memory. To facilitate fast predictions of VNF resource requirements without reloading, all models should be loaded up front during the algorithm's initialization.

Then, the algorithm's routine that decides the amount of resources to allocate to a VNF instance needs to be adjusted. Rather than returning a fixed, pre-defined value, it should query the associated machine learning model of the corresponding VNF. To predict the required resources for each VNF instance, the algorithm needs to pass the traffic rate that each instance has to handle as input to the VNF's trained machine learning model. Typical VNF placement algorithms actively decide what traffic rate to assign to each VNF instance and can directly use this rate as input. Otherwise, they need to obtain the traffic rate from a monitoring component. Loading and swapping different machine learning models is simple and transparent to the rest of the algorithm due to the consistent interface of common machine learning models using the sklearn API [26].

As an example, we adjust the existing B-JointSP [6] algorithm for joint VNF scaling, placement, and resource allocation to use machine learning models for predicting VNF resource requirements. Replacing its existing linear function approximation with machine learning models only required to change less than 100 lines of code. The adjusted algorithm is available online on GitHub [22].

V. Evaluation with Real VNF Data

To illustrate and evaluate our proposed machine learning-based approach for dynamic resource allocation, we perform an evaluation using VNF performance measurements of

two real-world VNFs. In particular, we use performance measurements of the Squid cache[2] and the Nginx proxy[3] from the SNDZoo [5]. Each of these datasets has 4.6 million data points with a subset being performance measurements under varying resource configurations.

In Sec. V-A, we use six different machine learning algorithms to learn the relationship between VNF performance (max. sustainable traffic load) and resource requirements (CPU) from these datasets. We illustrate the importance of hyperparameter tuning and compare their prediction accuracy. In Sec. V-B and V-C, we use the adjusted VNF placement algorithm described in Sec. IV-C with different machine learning models to predict VNF resource requirements and allocate resources accordingly. We evaluate the impact of these different models on the resulting VNF placements and on algorithm runtime, using machines with an Intel Xeon W-2145 CPU and 32 GB RAM.

A. VNF Performance Prediction

To find the most accurate machine learning algorithm for learning from VNF performance measurements, we test and compare six different algorithms from four different families of machine learning approaches: Linear regression, SVR, ensemble learning, and neural networks. Each of the used algorithms is described in more detail in Sec. IV-B2. In addition, we compare these machine learning models with the common fixed model that always assigns a pre-defined, fixed amount of resources to all VNF instances.

Before being able to train machine learning models, the used measurement data has to be preprocessed (see Sec. IV-B2). For faster processing and training, we filter the data to a subset only containing relevant features, i.e., measurements of max. sustainable traffic load for varying CPU configurations, by setting all non-resource related configurations to fixed values. Additionally, we fill any missing values and scale the features, here, the max. sustainable traffic load, to a range of 0 to 1. Feature scaling is especially crucial for neural networks but also for SVR. Without feature scaling, our considered MLP neural network performs terribly – leading to a prediction error that is roughly 150 times worse than with feature scaling. For other machine learning algorithms, e.g., linear regression or ensemble learning, feature scaling only brings negligible benefits.

After preprocessing, we train and evaluate each machine learning model using 5-fold cross validation as described in Sec. IV-B. For each model, we optimize the loss in terms of the root mean squared error (RMSE). The RMSE averages all prediction errors with y_i being the true target value and $\hat{y}_i$ the predicted value (here, required CPU): $\text{RMSE} = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$

1) Squid Cache VNF: Fig. 4 shows the RMSE of the six machine learning models for the Squid cache dataset. In addition to the six machine learning models, the figure also shows the error for the currently prevailing model of assigning fixed resources to each VNF instance (here, configured to 0.8 CPU). Each bar represents the average RMSE of prediction on the validation set in 5-fold cross validation with the standard deviation as error bars.

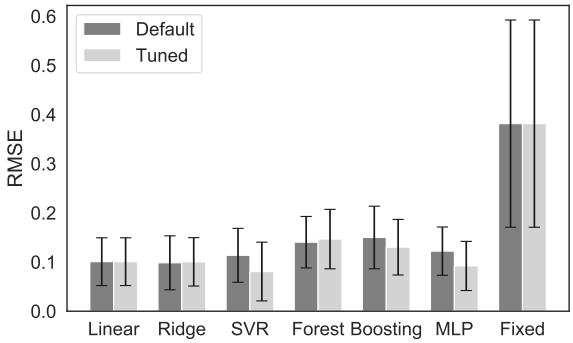

Fig. 4: RMSE of different machine learning models with default and tuned hyperparameters on the Squid VNF dataset.

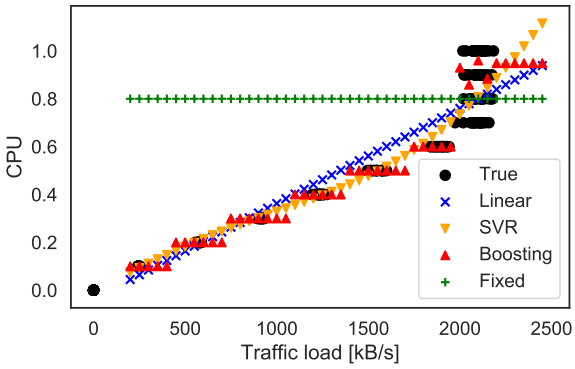

Fig. 5: Squid VNF performance measurements and predictions of different machine learning models.

Not surprisingly, the fixed model does not approximate the true resource requirements of the Squid cache well, which leads to a very high average RMSE of around 0.4. All machine learning models perform significantly better (RMSE between 0.1 and 0.2). Fig. 4 also compares the RMSE of all models with default hyperparameters and after hyperparameter tuning as described in Sec. IV-B4. With default hyperparameters, the linear models (linear and ridge regression) perform best, but after hyperparameter tuning, SVR has a lower average RMSE, closely followed by MLP. SVR and MLP have many hyperparameters such that tuning can improve their performance significantly. In contrast, the simpler, linear models only have very few hyperparameters and do not noticeably benefit from hyperparameter tuning.

As visualized in Fig. 5, the VNF performance measurements of the Squid VNF show a strong linear relationship between max. sustainable traffic load and CPU for most measurements (up to 2000 kB/s), explaining the high accuracy of linear models. Fig. 5 also shows the predictions of some of the tuned and trained machine learning models as well as the fixed model. In contrast to the fixed model, all machine learning models fit the data quite well. While

[2]Squid Cache: http://www.squid-cache.org/ (accessed Nov 25, 2019)
[3]Nginx: https://www.nginx.com/ (accessed Nov 25, 2019)

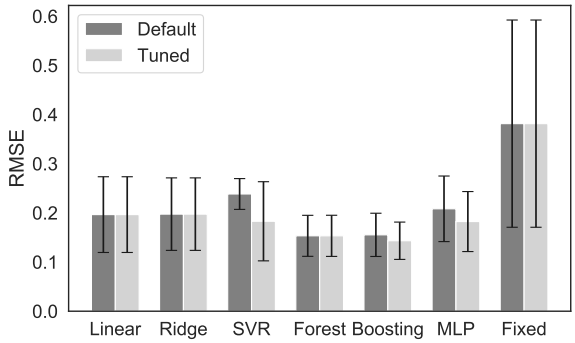

Fig. 6: RMSE of different machine learning models with default and tuned hyperparameters on the Nginx VNF dataset.

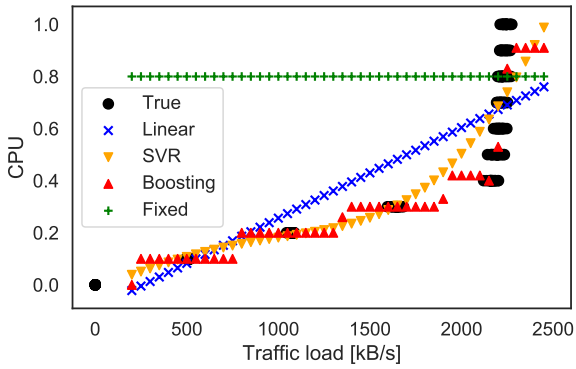

Fig. 7: Nginx VNF performance measurements and predictions of different machine learning models.

the linear model just fits a straight line, SVR uses a non-linear function. Thanks to SVR's regularization, the function is quite smooth. Conversely, the gradient boosting model increases its predictions stepwise, which is due to the decision trees being used within the model, breaking the data into different parts with similar prediction value.

2) Nginx Proxy VNF: Similar to the Squid VNF, we train all machine learning models on the Nginx dataset, with and without hyperparameter tuning, and show their RMSE in Fig. 6. Again, hyperparameter tuning mostly improves the performance of the more complex models, here SVR, MLP, and gradient boosting. For Nginx, the two ensemble methods (gradient boosting and random forest) perform better than all other machine learning models.

As can be seen in Fig. 7, Nignx' true required CPU for increasing max. sustainable traffic load first increases roughly linearly but then rises dramatically for traffic loads above 2000 kB/s. While this is also visible for the Squid cache (Fig. 5), it is much more pronounced in the Nginx dataset. Due to this strong increase, the linear model cannot fit the data as well as in the Squid dataset.

B. Impact on VNF Placement

The previous section revealed significant differences in prediction accuracy between different machine learning models and, in particular, their remarkable advantage over a fixed model. In this section, we evaluate the impact of different models when using them for dynamic VNF resource allocation inside an existing VNF placement algorithm.

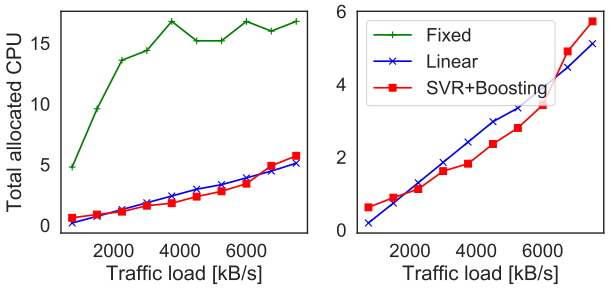

Fig. 8: Total allocated CPU in VNF placements computed with different models. Right: Close-up comparison.

To evaluate the impact on VNF placement, we consider the real-world Abilene network topology from Topology Zoo [27] with unit compute capacity at each node. Further, we consider three traffic sources with an increasing number of flows leaving the sources, leading to growing traffic loads. We assume a web-based network service consisting of the Squid cache and the Nginx proxy chained together.

We use three variants of the adjusted VNF placement algorithm with different models to predict VNF resource requirements and allocate resources for Squid and Nginx, respectively: In the first variant, we select the currently prevailing fixed model with 0.8 CPU per VNF instance. In the second, we choose the trained linear models for both VNFs, which represent the best case that is achievable with recent approaches using linear functions for $r_c(\lambda)$. In the third variant, we use the best models from Sec. V-A, i.e., the trained SVR for predicting Squid's resource requirements $r_{\text{Squid}}(\lambda)$ and gradient boosting for Nginx ($r_{\text{Nginx}}(\lambda)$).

We only know the true resource requirements of Squid and Nginx for measurements in the two datasets but not their true function $r_c(\lambda)$. Compared to other models, SVR and gradient boosting achieved the smallest RMSE in Sec. V-A. Thus, we assume SVR and gradient boosting to be closest to the true resource requirements of the two VNFs, respectively, with negligible over- or under-allocation.

1) Impact on Total Resource Allocation: Fig. 8 (left) shows the total allocated CPU with increasing traffic load for the three different algorithm variants. With more traffic to process, placed VNF instances require more resources in total to properly handle the increasing traffic load. The amount of allocated resources depends on the predicted resource requirements $r_c(\lambda)$ of each model. The figure shows that assigning a pre-defined, fixed amount of CPU resources to each VNF instance leads to a huge amount of total allocated resources that grows rapidly with increasing traffic load, quickly filling up the network's resource capacities. Here, allocating a fixed amount of resources leads to up to 12 times more allocated CPU resources than with SVR and gradient boosting, signifying massive over-allocation.

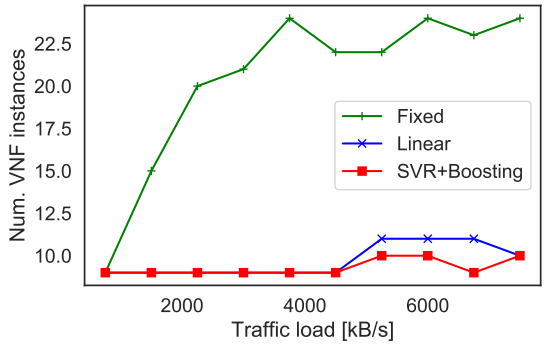

Fig. 9: Model impact on the number of VNF instances.

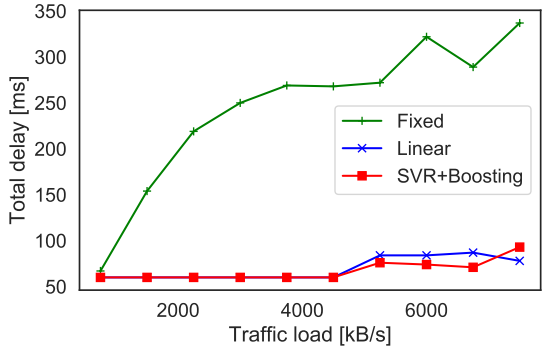

Fig. 10: Model impact on the total delay.

Compared to the fixed model, the differences in total allocated CPU are much smaller between linear approximation and the combined SVR and gradient boosting. Still, Fig. 8 (right) shows that the total allocated CPU resources in VNF placements with the linear models is up to 33% higher or up to 69% lower than in VNF placements computed with SVR and gradient boosting. Since the SVR and gradient boosting models are more accurate than the linear models (Se. V-A), this likely implies over- and under-allocation for VNF placements computed with the linear models.

2) Impact on Number of VNF Instances: With more allocated resources, more VNF instances are required to balance the load across different network nodes. Fig. 9 shows that this is most clearly visible for the case of fixed resource allocation, where 2-3 times more instances are placed than in the SVR and gradient boosting case. Again, the results with the linear models and the combined SVR and gradient boosting are quite similar with respect to the number of placed VNF instances. For higher load, the slight CPU over-allocation of the linear models compared to SVR and gradient boosting adds up and leads to slightly more VNF instances being placed.

3) Impact on Total Delay: Fig. 10 shows the total link delay of the produced VNF placements. As before, VNF placements with the fixed model lead to much worse results and up to 4.5 times higher total delay than with SVR and gradient boosting. Here, the higher total delay is caused by

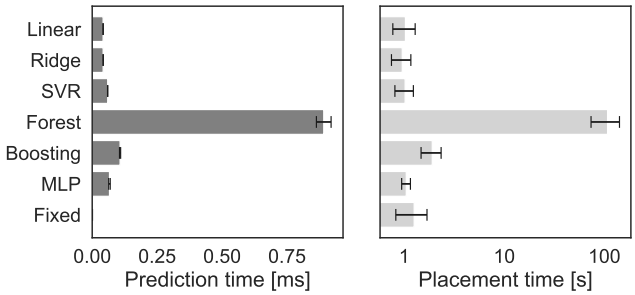

Fig. 11: Prediction and placement times for different models.

the higher number of VNF instances in the corresponding placements. The more VNF instances are placed at different nodes in the network, the more traffic has to be sent via links between them, leading to higher total delay. Again, the linear models and the SVR and gradient boosting models lead to similar total delay for low traffic. For higher traffic, the total delay with the linear models is up to 23% higher than with SVR and gradient boosting due to more instances in the VNF placement with linear approximation.

C. Runtime Analysis

In addition to prediction accuracy and resulting quality of VNF placements, short computational runtime is important to quickly react to changes in demand. While training and hyperparameter tuning can be performed offline up front, predictions with trained models may be frequently required by VNF placement algorithms during online operation.

1) Time per Prediction: Fig. 11 (left) shows the measured time per prediction for each of the trained and tuned models of Sec. V-A. The fixed model returns the same predefined value instantly without any computation. Also all six machine learning models make very fast predictions within 1 ms and, except for random forest, even within 0.1 ms. Differences in prediction time reflect the complexity of the models: The two simple linear models (linear and ridge regression) are fastest and the two ensemble learning models (gradient boosting and random forest) are slowest. Random forest traverses multiple, possibly very deep, decision trees for each prediction, which is relatively timely. While the gradient boosting model uses the same number of decision trees, it limits their depth to speed up predictions significantly. Similarly, the MLP model uses a shallow neural network (1x 128 hidden units), enabling fast predictions.

2) Placement Time: Using these models in our adjusted VNF placement algorithm, Fig. 11 (right) shows the algorithm's runtime on a logarithmic scale. Evidently, the roughly 10+ times higher prediction time of the random forest model compared to other models also leads to much higher algorithm runtimes of over 1 min. All other machine learning models have comparable prediction times, leading to largely similar algorithm runtimes between 1 s and 2 s.

Surprisingly, the algorithm runtimes with the fixed model are higher than with many machine learning models, even though it has almost zero prediction time. This is likely

because the fixed model's high over-allocation leads to many VNF instances, which consequently results in additional placement decisions that increase the algorithm's runtime. Hence, there is a trade-off between prediction time and accuracy depending on a model's complexity. However, many machine learning models are accurate but still fast, which clearly pays off in overall VNF placement quality and even algorithm runtime.

VI. Conclusion

We show how machine learning can be used to automatically learn from real VNF data, deriving models that can accurately predict VNF resource requirements depending on the traffic load. Using these models in VNF placement algorithms can significantly impact the solution quality of resulting VNF placements. Comparing six different machine learning algorithms and a common fixed allocation model, our evaluation using real-world data shows that choosing a suitable model for dynamic resource allocation is crucial. In particular, using fixed resource allocation can lead to massive over-allocation of resources, high numbers of VNF instances, and unnecessary high delay, potentially leading to high costs and bad service quality. While linear approximation works fairly well in our evaluation, it is still outperformed by more powerful approaches like SVR or ensemble learning, which can predict the true VNF resource requirements more accurately. In our evaluation, neural networks achieved good prediction accuracy but not superior to SVR or ensemble learning, while being much more sensitive to correct data preprocessing and hyperparameter tuning.

Overall, machine learning for dynamic resource allocation can help save resources and improve service quality.

Acknowledgments

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. H2020-ICT-2016-2 761493 (5GTANGO), the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing" (SFB 901), and the German Federal Ministry of Education and Research.

References

[1] M. Peuster and H. Karl, "Understand Your Chains: Towards Performance Profile-based Network Service Management," in *European Workshop on Software Defined Networks (EWSDN)*. IEEE, 2016.

[2] R. V. Rosa, C. Bertoldo, and C. E. Rothenberg, "Take Your VNF to the Gym: A Testing Framework for Automated NFV Performance Benchmarking," *IEEE Communications Magazine*, vol. 55, no. 9, pp. 110–117, 2017.

[3] M. G. Khan, S. Bastani, J. Taheri, A. Kassler, and S. Deng, "NFV-inspector: A systematic approach to profile and analyze virtual network functions," in *IEEE International Conference on Cloud Networking (CloudNet)*. IEEE, 2018, pp. 1–7.

[4] S. Dräxler and H. Karl, "SPRING: Scaling, placement, and routing of heterogeneous services with flexible structures," in *IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2019, pp. 115–123.

[5] M. Peuster, S. Schneider, and H. Karl, "The softwarised network data zoo," in *IEEE/IFIP International Conference on Network and Service Management (CNSM)*. IEEE/IFIP, 2019.

[6] S. Dräxler, S. Schneider, and H. Karl, "Scaling and placing bidirectional services with stateful virtual and physical network functions," in *IEEE Conference on Network Softwarization (NetSoft)*, 2018.

[7] J. G. Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532, 2016.

[8] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2015, pp. 98–106.

[9] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *IEEE Conference on Computer Communications (INFOCOM)*, 2015, pp. 1346–1354.

[10] R.-H. Gau, "Optimal traffic engineering and placement of virtual machines in SDNs with service chaining," in *IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2017, pp. 1–9.

[11] M. Dieye, S. Ahvar, J. Sahoo, E. Ahvar, R. Glitho, H. Elbiaze, and N. Crespi, "CPVNF: Cost-efficient proactive VNF placement and chaining for value-added services in content delivery networks," *IEEE Transactions on Network and Service Management*, vol. 15, no. 2, pp. 774–786, 2018.

[12] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, and R. Boutaba, "Elastic virtual network function placement," in *IEEE International Conference on Cloud Networking (CloudNet)*, 2015.

[13] S. Dräxler, H. Karl, and Z. A. Mann, "Joint optimization of scaling and placement of virtual network services," in *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2017.

[14] V. Eramo, E. Miucci, M. Ammar, and F. G. Lavacca, "An approach for service function chain routing and virtual function network instance migration in network function virtualization architectures," *IEEE/ACM Transactions on Networking*, vol. 25, no. 4, pp. 2008–2025, 2017.

[15] S. Ayoubi, N. Limam, M. A. Salahuddin, N. Shahriar, R. Boutaba, F. Estrada-Solano, and O. M. Caicedo, "Machine learning for cognitive network management," *IEEE Communications Magazine*, vol. 56, no. 1, pp. 158–165, 2018.

[16] T. Lei, Y. Hsu, I. Wang, and C. Wen, "Deploying QoS-assured service function chains with stochastic prediction models on VNF latency," in *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2017.

[17] X. Zhang, C. Wu, Z. Li, and F. C. Lau, "Proactive VNF provisioning with multi-timescale cloud resources: Fusing online learning and online optimization," in *IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2017, pp. 1–9.

[18] X. Fei, F. Liu, H. Xu, and H. Jin, "Adaptive VNF scaling and flow routing with proactive demand prediction," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2018.

[19] K. Sembiring and A. Beyer, "Dynamic resource allocation for cloud-based media processing," in *ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2013, pp. 49–54.

[20] R. Mijumbi, S. Hasija, S. Davy, A. Davy, B. Jennings, and R. Boutaba, "Topology-aware prediction of virtual network function resource requirements," *IEEE Transactions on Network and Service Management*, vol. 14, no. 1, pp. 106–120, 2017.

[21] S. Van Rossem, W. Tavernier, D. Colle, M. Pickavet, and P. Demeester, "Profile-based resource allocation for virtualized network functions," *IEEE Transactions on Network and Service Management*, 2019.

[22] S. Schneider, "Open-source repository with all code and results," https://github.com/CN-UPB/ml-for-resource-allocation, 2019.

[23] M. Peuster and H. Karl, "Understand your chains and keep your deadlines: Introducing time-constrained profiling for NFV," in *IEEE/IFIP International Conference on Network and Service Management (CNSM)*. IEEE/IFIP, 2018, pp. 240–246.

[24] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning (ICML)*, 2015, pp. 448–456.

[25] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *International Conference on Computational Statistics (COMPSTAT)*. Springer, 2010, pp. 177–186.

[26] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.

[27] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.