Virtual Network Function Descriptors Mining using Word Embeddings and Deep Neural Networks

Wassim Sellil Atoui[1,2], Imen Grida Ben Yahia[1] and Walid Gaaloul[2]
[1]Orange Labs, Châtillon, France
[2]Telecom SudParis, Samovar, Evry, France
{wassimsellil.atoui, imen.gridabenyahia}@orange.com, walid.gaaloul@mines-telecom.fr

***Abstract*—Agile automation of Virtual Network Functions (VNFs) deployment is a must in the future softwarized networks. The automation is generally enabled using descriptor files associated with the VNFs, called Virtual Network Function Descriptors (VNFDs). These descriptors define the resource requirements, operational behavior, and policies that are required for the deployment. We propose in this experience paper a framework for VNFD mining based on Word Embeddings and Deep Neural Networks. The goal of the framework is to automatically recommend VNF descriptors based on a given description or/and to complete the descriptors with appropriate data. The framework is based on two neural network methods: Long short term memory (LSTM) and Convolutional Neural Networks (CNN), which are both complementary in their abilities and could be combined to enhance the performance of the framework. The experiments show good and promising results.**

I. Introduction

Network function virtualization (NFV) architecture has gained a lot of attention in the telecommunication domain. Driven by virtualization, NFV decouples the control and management functions from the expensive network devices that may deliver only specialized network functions. For that, NFV allows commercial off the shelf hardware devices to be deployed underneath virtual machines that virtualize entire classes of network functions such as firewalls, load balancers, mobile RAN functions, etc. The virtual network functions (VNFs) could be connected or chain together to create new communication services.

To enable and support self-management of network operations, the VNFs need to be automatically deployed and configured. This automation is eased by deployment descriptors that are associated with the VNFs, called VNF descriptors (VNFDs). The VNFDs describe the resource requirements, operational behavior, and policies that are required for the deployment of the VNFs.

We argue that the current approach of using predefined deployment descriptors from service providers is limited and that soon; it will not satisfy the agility and dynamicity of network and service orchestrators. We expect that orchestrators need to be sufficiently intelligent to figure out autonomously and with limited or no external assistance, how to deploy VNFs or network services. This ability could enable the orchestrators to adapt the deployment of the VNFs with the network condition or to anticipate their deployment when the descriptors are damaged or absent.

978-3-903176-15-7

We propose in this work a framework based on deep neural networks that could be used to augment an orchestrator with an ability to select, recommend and complete NFV descriptors from an initial description. The framework is based on two Deep Neural Network (DNN) methods: Long term Short Memory (LSTM) and Convolutional Neural Networks (CNNs). CNN is suited for extracting features from a given task and LSTM for learning sequential data. We apply these methods to learn from given set of deployment descriptors how to recommend and complete descriptors.

The rest of the paper is organized as follow: In section II we present the related work and a brief background information on CNN and LSTM. In section III, we describe our DNN based framework. In section IV, we conduct some experiments to evaluate the performance of the framework and discuss the results. Finally, in section V,we conclude our work and present the future perspectives.

II. Related works and background information

Recent surveys on the current challenges in NFV orchestration have showed that the dynamicity of changing deployment descriptors on the fly to meet the network conditions is a challenging problem that needs to be tackled [1] [2]. In the literature, some contributions have considered the problem of automating the deployment of VNFs. In [3], the authors proposed a tool and a methodology to aid the NFV designer to rapidly design and onboard new services and applications. In [4], a characterization approach based on machine learning is proposed to identify the most appropriate types of resources to be allocated to a VNF at deployment time. Also in [5] , an approach based on machine learning is considered to estimate VNFs needs in term of resource requirements.

Following the ETSI NFV recommendations [6], the VNFD descriptors defines the VNF properties, such as: resources needed (amount and type of Virtual Compute, Storage, Networking), software metadata (External and internal Connection Points, Virtual Links,etc.), lifecycle management behavior (scaling, instantiation,etc.), lifecycle management operations, and their configuration. The absence of a common data model to describe the deployment descriptors have led different template models to emerge. The most used modelling languages for that matter are YANG and TOSCA. In TOSCA, there

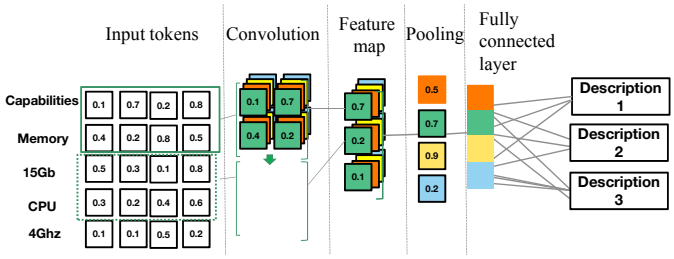

Fig. 1. Convolutional Neural Network Architecture

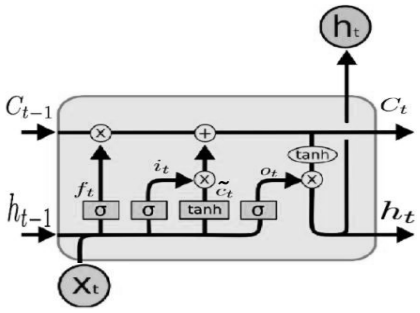

Fig. 2. Long short-term memory cell architecture

are two profiles that are standardized by OASIS: the simple profile in YAML v1.0 [7] for managing the life cycle of cloud applications and services, and the Simple Profile for NFV [8] that is based on ETSI NFV.

To the best of our knowledge, our approach to learn from deployment descriptors using DNN techniques has not been yet investigated in the context of NFV in order to cope with the agility needs.. The asset and power of DNN methods resides on their ability to generalize from examples, handling noises, to achieve different tasks such as classification, regression and clustering. The methods that we use in our framework, i.e. CNN and LSTM, have been already successful in processing natural language [9],[10].

In the following, we give a brief introduction to the used DNN methods.

A. Convolutional Neural Networks (CNN)

CNN are a specialized kind of neural network that were originally used for computer vision in which they made major breakthroughs. The main characteristic of CNN is that it is able to capture and learn relevant features from a given task at multiple levels using convolutions. The feature learning process is achieved through convolution layers. Generally, the layers are composed of three phases as illustrated in Fig. 1. The first phase consists of convolution operations between the input data and learnable filters with customized shapes that act as feature detectors from the original input. The convolution operation is basically a dot product between a region of the input data and the filters, such that the region size has to be equal to one of the filters dimension. The second phase aims to generate a feature map for each filter. The feature map contains the result of the convolution operation between the filter and different regions of the input. Typically, this is done by sliding multiple times the filter over the input with a fixed number of steps. The last phase is called pooling. It is an operation that reduces the dimension of each feature map while taking the most important information. Max-pooling is a simple strategy that is usually employed by considering only the maximal elements of each feature map. The pooling vector can then be fed into a fully-connected layer to perform the classification. CNN can be trained using back propagation of the error from the classification.

B. Long short-term memory (LSTM)

LSTM is a special type of Recurrent Neural Networks (RNN) that is able to recognize and predict sequences of data (time series, text, signals, events, etc.). RNN are neural networks with loops that enable the information flow to persist iteratively. They are utilized for various machine learning tasks such as speech recognition, time series, data prediction, etc. LSTM has the ability to memorize long-term dependencies and overcome the vanishing/exploding gradient problem. It contains an internal state variable that is passed from one cell to the other and modified by operation gates. A simple LSTM cell contain generally three gates, as illustrated in Fig 3:

• Forget gate (f_t): this operation helps to remove irrelevant information from the input and thus keeping only what is important to remember. It takes the output of the previous state, h(t-1) and performs on it a sigmoid function (σ). W and b are respectively the weight matrix and bias term of f_t

$$f_t = \sigma(W_f.[h_{t-1,x_t} + b_f]) \tag{1}$$

• Input gate(i_t): this operation decides which element from the present input to be updated and creates a vector for new candidates to add to the cell state ($\hat{C}_t$).

$$i_t = \sigma(W_i.[h_{t-1,x_t} + b_i]) \tag{2}$$

$$\hat{C}_t = tanh(W_C.[h_{t-1,x_t} + b_C]) \tag{3}$$

• Output gate(o_t): this operation decides what to output from the current state.

$$o_t = \sigma(W_o.[h_{t-1,x_t} + b_o]) \tag{4}$$

The current state of the cell C_t is updated based on the previous gate operations and a filtered version of this state h_t is passed out as output to the next cell:

$$C_t = f_t * C_{t-1} + i_t * \hat{C}_t \tag{5}$$

$$h_t = o_t * tanh(C_t) \tag{6}$$

III. Proposed approach

A. Overview

In this section, we describe our framework that enables network orchestrators to complement a given network deployment descriptor with appropriate metadata and its associated values, select and recommend descriptors to ease and automate the deployment phase of VNFs. The approach is based on word embedding and DNN methods, which both proved to be powerful for identifying patterns in unstructured data (images, sound, video, text). More particularly, we employ two methods of deep learning, CNN and LSTM.

Our framework encompasses three phases: 1) Data preparation phase, 2) Training phase and 3) DNN execution phase. The preparation phase includes two modules: The tokenization module and the indexation module. The first one is an indexation based on token appearance and the second one is an indexation based on word embedding. Essentially, the preparation phase aims to transform the input set (i.e. deployment descriptors) into a data set that is used by the DNN methods in the training phase. During the training phase, the DNN methods learn two models, a CNN model for the recommendation task and a LSTM model for the completion task. During the third phase, the generated LSTM and CNN models are executed.

B. Preparation phase

The objective of this phase is to transform the deployment descriptor files into an understandable and processable format by the DNN methods and that enable them to learn and capture structural patterns from the files. The deployment descriptors could be stored with different format such as YAML, XML, JSON and modeled different template such as TOSCA, YANG. The preparation include two main modules : the tokenization module and the indexation module. The tokenization is the task of breaking down the elements of a text corpus up into pieces, called tokens. We use the tokenization module to decompose the deployment descriptor files into a set of tokens that can capture the characteristics of the description. In the case of YAML files for example, the whitespace indentation is considered as a token, given that it has a logical role in the descriptor file, it denotes the data structure adopted within the descriptor. Hence, the tokenization operation depends on the template format. In Fig.4, as an example, we show a partial VNFD file that is tokenized.

The indexation module has the objective to build a look-up table that contains all the elements that were found by the tokenization modules. It encompasses two different methods, as follows.

1) Indexed Vocabulary Building: This method consists in indexing the elements/tokens based on their appearance in the code corpus. It gives as output a list of indexed vocabulary, where indexes are ordered based on the appearance of tokens in the deployment descriptor files.

2) Word to Vector (word2Vec): This method [11] aims to learn vector representation for each token included in the vocabulary to capture general syntactical and semantic information. The vectors are learned based on the continuous Skip-gram model [11].

C. Training phase

In this phase, we train the DNN methods to learn a recommendation and a completion model for the VNF deployment descriptors.

1) CNN for VNF deployment descriptors recommendation: The objective of the CNN method is to learn a recommendation model from the training data. Given an input partial deployment description, the model can be used to recommend a file of a deployment description from the catalogue. It can be considered as a classification problem, where the descriptor files in the catalogue are classes. The model is trained on the tokenized files that were generated during the preparation phase. Firstly, the tokens are converted to their vector representation based on the look-up table, following one of the two methods: Word2Vec or highest appearance. Then, the vectors are passed to the convolution layer to learn features from the descriptor files using multiple filter sizes. The most obvious features are selected in the pooling layer and concatenated into one vector. Finally, this vector is used in the fully connected layer to perform classification. We can summarize the CNN method into three processes: Token representation, feature extraction and recommendation.

Token Representation: The input of the model is a fixed size vector of tokens $w = \{w_1, w_2, ..., w_D\}$, Where D is the input size. The tokens are initially converted to their numerical representation using the generated look-up table in the network data preparation phase. Thus, the original input is $x = \{x_1, x_2, ..., x_D\}$, where x_i is a vector representation of w_i. The model is iteratively trained with D sequence of tokens from the tokenized files. If a sequence is less than D, then it will be completed with empty characters.

Feature Extraction: This process aims to learn features from the descriptors and merge the best found into a single vector. This vector is passed afterwards to the classification process. Features learning and extraction are accomplished by filters in the convolutional layer.

The input x is convoluted (dot multiplication operation) with multiple filters with different sizes to produce a feature score s_i. A filter is a matrix $W \times R$, such as W is the filter window size and R is the size of the token vector. $R = 1$ in the case of the highest appearance method. Each filter f_i is applied to multiple regions of the input and generates a list of feature scores. Each feature score s_j in the filter list is defined as:

$$s_j = ReLu(x_{j:j+W-1} \cdot f_i + b_s) \tag{7}$$

where $x_{j:j+W-1}$ is the region of the input to which the filter is applied, b_s is a bias, the operator "$\cdot$" denotes the convolution operation and $ReLU$ is a non linear function, $ReLu(x) = max(0, x)$. The training of filters is equivalent to learning a feature for the task of classifying the descriptor files.

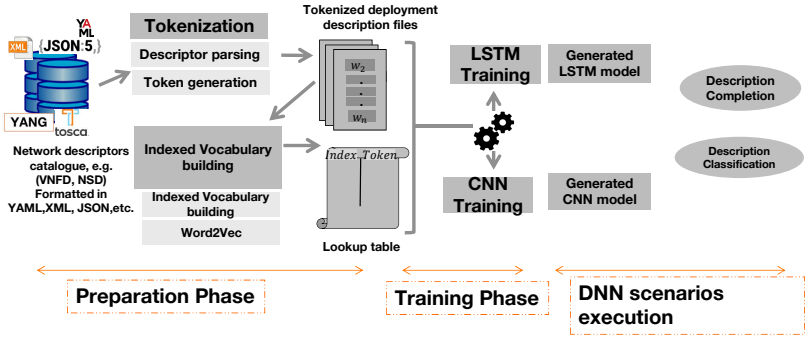

Fig. 3. DNN-based framework for VNF deployment descriptors mining

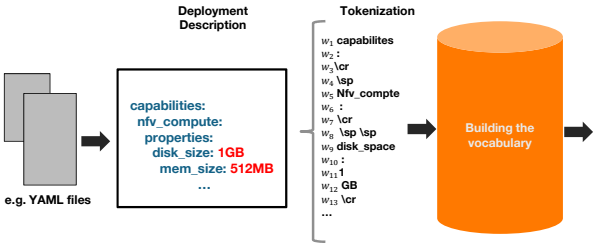

Fig. 4. Example of tokenization using a deployment descriptor file

The features that are extracted with each filter are assembled to form a feature map. After generating the feature maps for all the filters, a max-pooling operation is applied over each feature map to highlight the best resultant feature that is captured by each filter. This operation p_f consists into selecting a feature with a maximum value from the features map.

$$p_f = max(s) = max(s_1, s_2, s_3, ..s_k) \tag{8}$$

The best features of each feature map is merged into a penultimate layer and are passed to a fully connected layer whose output is the probability distribution over all the classes (i.e.deployment descriptor files).

Recommendation: The fully connected layer takes the vector that contains all the max pooled features from all the filters to perform a classification. We use for that a Softmax function to make the classification. The output is $y \in \{D_1, D_2, ..., D_Y\}$, an element from a set of pointers to the deployment descriptor files in the catalogue, Y is the number of files that are in the catalogue.

$$y = softmax(\gamma_i) == \frac{e(\gamma_i)}{\sum_{i'=1}^{Y} e(\gamma_{i'})} \tag{9}$$

where $\gamma_i = p(i|x, \theta)$, i is the number of the class $i \in \{1, 2, 3..Y\}$ and θ is the parameters of the model, including the weights and bias of the fully connected layer. The output is the class (descriptor file) with the highest probability.

The parameters of the model are afterward updated using the random gradient descent method to maximize the optimization objective function $J(\theta)$

$$J(\theta) = \sum_{i'=1}^{Y} log\, p(\gamma^{(i)}|x^{(i)}, \theta) \tag{10}$$

D. LSTM for deployment description completion

LSTM is used to learn a language model from the descriptor files so that it can be able to predict a sequence of tokens given an input description. As described previously, LSTM is basically a recurrent neural network with a special type of memory cell that stores the context of the information. It can be seen as a multiple copies of the same network, where each cell passes the information to its successor enabling the persistence of the information.

The model learns from the tokenized files using multiple sequences of tokens as input. Similarly, to CNN, the first step of the model is to convert the tokens into their numerical representation using the lookup table in the first phase of the framework. LSTM is trained by considering at each iteration a fixed size sequence of tokens. The parameters are tuned during the learning to enable the model to predict with high accuracy the next token given a same sequence size of tokens. The model can be expressed as the following:

$$x(t+1) = F(x(t), x(t-1), ..., x(t-s)) \tag{11}$$

where t is the position of the token, $x(t+1)$ is the predicted token given a s sequence of tokens. The prediction is effectuated using :

$$y = softmax(w_i) = \frac{e(w_i)}{\sum_{i'=1}^{V} e(w_{i'})} \tag{12}$$

where w_i is a token from the vocabulary and V is the vocabulary size The learning is done by minimizing the log-loss function $J(\theta)$ using a simple Stochastic Gradient Descent (SGD), with respect to the parameters of the LSTM model (θ):

$$J(\theta)) = -log\, p(x_1) - \sum_{t=2}^{k} log\, p(x_t|x_{1:t-1}) \tag{13}$$

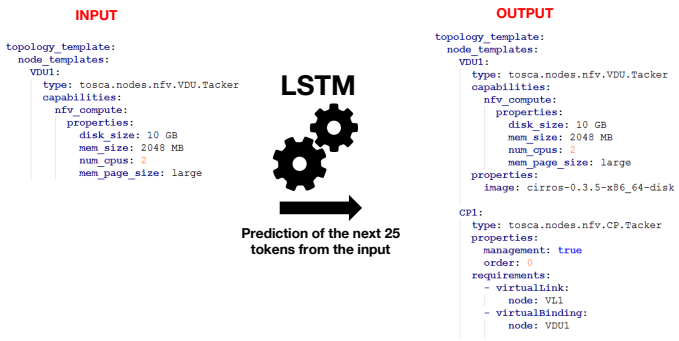

Fig. 5. Using the LSTM method for deployment description completion

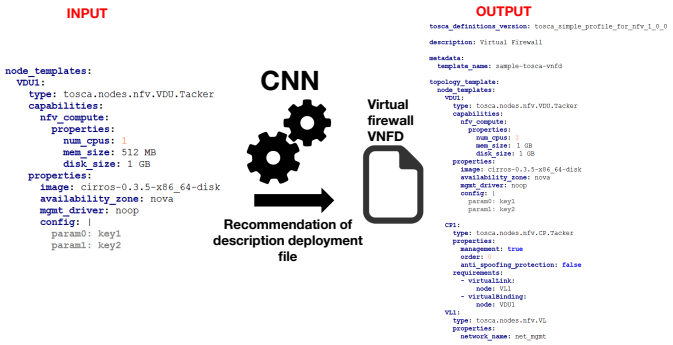

Fig. 6. Using the CNN method for deployment description recommendation

E. DNN execution phase

During this phase, the DNN models are used. To illustrate their usage, we consider example scenarios during design time and run time of an orchestrator. The design time is where the network and service descriptors are to be set and defined towards their deployment by the orchestrator. The Run time is where services and Network functions are already deployed and a change is needed to be done by the orchestrator in response to faulty situations (Noisy VNFs, overloaded VNFs, etc.). We describe here below one scenario for the Design Time and one Scenario for the Run time:

Design Time: VNFD verification and Completion

In this scenario, the resource orchestrator needs to complete a given VNFD with additional information, like policy constraints, forwarding graph, etc. In this case the DNN modules receive the incomplete VNFDs, compute the missing part with LSTM as illustrated in Fig. 5. The resource orchestrator then stores the newly completed VNFD in the catalogue. In this scenario, the LSTM was used to complete the VNFD. Another variation would be to consider the case where there is a need to change completely several parts of the VNFD, here the LSTM is not enough as the sequential aspect is not sufficient to complete the whole VNFD. Hence here, the CNN is invoked to compute the context of the remaining part within the VNFD and propose similar VNFD to complete it as shown in Fig.6.

Run time scenario: VNFD genration

The orchestrator is invoked to change the parameters witin the Descriptos in order to handle faulty situations like resources conflict between virtual machines (e.g. Overloaded VNF). In this case, the orchestrator need to change the VNFD and adapt it to a state of no conflict. To solve the problem, the orchestrator could specify a descriptor for the VNFs with resources and requirements that could potentially end the conflict. The proposed framework could be used to recommend deployment descriptors that satisfy the orchestrator demand.

IV. Experiments

In this section, we evaluate the performance of the proposed approach. More specifically, we focus on the effectiveness of

tab. I. Parameters of the CNN model

Parameters	Value
Train/Test %	80/20 %
Learning rate	0.001
Filter number	150
Batch size (tokens)	500

tab. II. Experiments on the CNN model using different input sizes

Input Size (Tokens)	Embedding Indexation	Appearance based Indexation
20	0.24	0.21
50	0.31	0.29
100	0.44	0.39
200	0.6	0.52
500	0.84	0.69

tab. III. Experiments on the CNN model using different numbers of filters

Number of filters	Embedding Indexation	Appearance based Indexation
50	0.6	0.54
75	0.68	0.59
100	0.71	0.62
125	0.79	0.65
150	0.84	0.69

tab. IV. Parameters of the LSTM model

Parameters	Value
Train/Test %	80/20 %
Learning rate	0.001
sequence length	150
Batch size (tokens)	500

tab. V. Experiments on the LSTM model using different input sequence length

Sequence length	Embedding Indexation	Appearance based Indexation
20	0.4	0.39
35	0.46	0.41
50	0.51	0.49
65	0.71	0.61
80	0.79	0.68

tab. VI. Experiments on the LSTM model using different output length

Output length	Embedding Indexation	Appearance based Indexation
5	0.87	0.79
10	0.82	0.71
30	0.78	0.68
70	0.6	0.42
180	0.43	0.21

tab. VII. Experiments on the CNN model combined with LSTM using different input sizes

input size	Embedding Indexation	Appearance based Indexation
20	0.62	0.56
50	0.67	0.6
100	0.67	0.64
200	0.75	0.64
500	0.84	0.69

the two deep learning models: CNN model for the descriptor recommendation task and LSTM model for the descriptor sequence prediction. We begin by describing the data set and the experimental setup that are used for training the models, and then we continue by discussing the results of the experiments with each model.

A. Data set and experimental Setup

The experiments are performed on a data set constituted of a catalogue of different network function descriptors (VNFDs) containing up to 500 VNFDs, defined in YAML files and following the TOSCA for NFV template [3]. The resulting code corpus includes more than 1,500,000 tokens that generate a vocabulary of 95,000 unique tokens. We use $\%80$ of the data set for training and 20% for testing. The skip gram model is used in the embedding indexation module to learn vector representation for the tokens. The vector dimension is set to 128. The models are implemented in python using Tensorflow, a widely used open source library for numerical computation and large-scale machine learning. The experiments are tested in a machine equipped with an intel i7 CPU and 8 GB of RAM.

B. CNN for descriptor classification

We evaluate the CNN model taking into account the two indexation methods (embedding indexation and appearance based indexation). The parameters of the model are set as shown in tab.1. The accuracy of the classification is measured by varying the input size of the model. The results are shown in tab. 2. We can notice that the input size of the description have an impact on the classification. That makes senses, because the more we add information about the description, the more the CNN can extract features and be able to classify it to the best descriptor file. We can also notice that the embedding base indexation gives more accurate classification than the appearance based classification. This is due to the fact that embeddings enable the CNN to choose the nearest possible descriptor given an input because of the vector representation that encodes the semantic of the tokens.

We also test the performance of CNN in function of the number of filters. Each filter have a window size that is generated randomly following a uniform distribution between $[50, 500]$. The results are presented in tab. 3. We can notice that filters with multiple window sizes can capture the code features and achieve better classification result.

C. LSTM for descriptor sequence prediction

The LSTM model settings are shown in tab. 4. We evaluate here the precision of the model in function of the sequence length. The results in tab. 5 shows that LSTM performs better when it is trained on longer sequence of data. That is actually one of the abilities of LSTM, to handle long sequence of data and capture more information of the context to predict more accurately the next token.

We measure also the performance of LSTM for generating sequence of tokens by variating the length of the output. We fix here the input description to 500 tokens. Tab. 6 shows that the model is able to predict with high accuracy the sequence of tokens in the beginning of the prediction, then the performance decreases with the sequence size.

Combining LSTM with CNN can actually enhance the classification accuracy by completing the input size of CNN using the LSTM model. The results in tab.7 can demonstrate this performance.

V. Conclusion

We proposed in this work a framework based on deep neural networks that could be used to augment an orchestrator with an ability to select, recommend and complete NFV descriptors from an initial description. The promising results from our experiments suggest that the framework is a solution that could be used in practical scenarios to enhance the dynamicity of deploying VNFs. Our next step involves developing a descriptor generation engine that could be used to suggest a new description that fits most the initial requirements.

References

[1] K. Katsalis, N. Nikaein, and A. Edmonds, "Multi-domain orchestration for nfv: Challenges and research directions," in *15th Inter. Conf. on Ubiquitous Comp. and Commun.*, pp. 189–195, Dec 2016.

[2] N. F. S. de Sousa, D. A. L. Perez, R. V. Rosa, M. A. S. Santos, and C. E. Rothenberg, "Network service orchestration: A survey," *arXiv:1803.06596*, 2018.

[3] C. Makaya and D. Freimuth, "Automated virtual network functions onboarding," in *IEEE Conf. on Network Function Virtualization and Software Defined Networks*, pp. 206–211, Nov 2016.

[4] V. Riccobene, M. J. McGrath, M. Kourtis, G. Xilouris, and H. Koumaras, "Automated generation of vnf deployment rules using infrastructure affinity characterization," in *IEEE NetSoft Conference and Workshops*, pp. 226–233, June 2016.

[5] H. Jmila, M. I. Khedher, and M. A. El Yacoubi, "Estimating vnf resource requirements using machine learning techniques," in *Neural Information Processing*, (Cham), pp. 883–892, Springer, 2017.

[6] "Network functions virtualisation (nfv)management and orchestration vnf packaging specification." https://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/011/02.01.01_60/gs_NFV-IFA011v020101p.pdf.

[7] "Tosca simple profile in yaml version 1.0." http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/csprd02/TOSCA-Simple-Profile-YAML-v1.0-csprd02.html.

[8] "Tosca simple profile for network functions virtualization." http://docs.oasis-open.org/tosca/tosca-nfv/v1.0/tosca-nfv-v1.0.html.

[9] Y. Kim, "Convolutional neural networks for sentence classification," *CoRR*, vol. abs/1408.5882, 2014.

[10] S. Ghosh and P. O. Kristensson, "Neural networks for text correction and completion in keyboard decoding," *CoRR*, vol. abs/1709.06429, 2017.

[11] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *NIPS*, pp. 3111–3119, Curran Associates, Inc., 2013.