DeepNFV: A Lightweight Framework for Intelligent Edge Network Functions Virtualization

DeepNFV: A Light-weight Framework for Intelligent Edge Network Functions Virtualization

Liangzhi Li, *Student Member, IEEE,* Kaoru Ota, *Member, IEEE,* Mianxiong Dong, *Member, IEEE*

***Abstract*—Traditional Network functions virtualization (NFV) implementations are somehow too heavy and do not have enough functionality to conduct complex tasks. In this work, we propose a light-weight NFV framework named DeepNFV, which is based on the Docker container running on the network edge, and integrate the state-of-the-art deep learning models with the NFV containers to address some complicated problems, such as traffic classification, link analysis, etc. We compare the DeepNFV framework with several existing works, and detail its structures and functions in the paper. The most significant advantage of DeepNFV is its light-weight design, resulted from the virtualization and low-cost nature of the container technology. Also, we design this framework to be compatible with edge devices, in order to decrease the computational overhead of the central servers. Another merit is its strong analysis ability brought by the deep learning models, which make it suitable for much more scenarios than the traditional NFV approaches. In addition, we also describe some typical application scenarios, regarding how the NFV container works and how to utilize its learning ability. Simulations demonstrate its high efficiency, as well as the outstanding recognition performance in a typical use case.**

***Index Terms*—Network Functions Virtualization (NFV), edge computing, deep learning, packet classification.**

I. Introduction

With the rapidly developing network needs, the telecommunications service providers (TSPs) pick up their pace to introduce new network features and functions. Innovations in theories and implementations help a lot to improve the quality and functionality of the network services, however, at the same time, they also bring a huge cost to the TSPs when upgrading their hardware and infrastructure for the deployment of these new network functions. Several TSPs express some early interests to build the network functions with software implementations, which can significantly decrease the capital expenditures (capex) and operating expenditures (opex) when introducing new network functions [1], and collaborate on a white paper in 2012 [2], where the concept of network functions virtualization (NFV) is presented, as well as its benefits and enablers. NFV decouples the implementation of network functions from the underlying hardware, and for proposing or modifying new functions, the TSPs can simply modify the software installed in network devices, rather than totally replace the deployed hardware.

Liangzhi Li, Kaoru Ota and Mianxiong Dong are with the Department of Information and Electronic Engineering, Muroran Institute of Technology, Japan.
E-mail:{16096502, ota, mxdong}@mmm.muroran-it.ac.jp
Manuscript received xx xx, 20xx; revised xx xx, 20xx.

Combining NFV with the state-of-the-art deep learning technologies can empower the network devices with more intelligence to deal with complicated network traffic. Due to their superior abilities of automatic feature extraction, deep learning models can find the hidden patterns in the raw data, and based on that, infer the possible relationships between the input and the pre-defined labels [3], i.e., the classification. In this paper, we employ the deep models for the NFV scenarios, and successfully implement a series of NFV instances with outstanding analysis ability to conduct various tasks, such as traffic classification, link analysis, quality of service (QoS) control, firewall, routing table, etc.

However, the traditional NFV approaches are somehow too heavy, because they usually adopt some general-purposed full-featured virtual machines (VMs), resulting in non-negligible resource costs to the underlying hardware. This might not be a serious problem when running in central servers, because the traditional NFV functions are usually not very powerful and cost few resources, and the central servers are powerful hardware with strong computing abilities, as well as huge storage space. But when the deep learning is adopted, which is extremely compute-intensive, the central servers will face serious performance problems due to the super large computation burden. One obvious solution is to offload the computation to the network edge, and use the edge computing approaches for the NFV functions with deep learning abilities. It is exactly the edge environment where the traditional VM-based NFV frameworks struggle, because the VM instances account for too much resource, which is limited in edge devices. Therefore, a light-weight NFV framework is essential to implement NFV in the edge devices. Our solution is the docker container, which is very efficient and cost-saving due to its virtualization and low-cost nature.

As mentioned above, we adopt NFV with edge-computing-based deep learning models, in order to implement a powerful yet light-weight framework, which can be suitable for general edge devices, such as routers, gateways, etc. As shown in Fig. 1, the NFV containers, each of which represents a virtual network function (vNF), are located among the central servers and the edge devices. When one user calls a specific network function, the related NFV containers will be organized as a chain. The containers on the chain can be all deployed in the edge devices, or part of them can be put in the central servers. Each container has the ability of learning and interfering with some pre-trained deep models, and its output result can be used by other containers in the chain for further process. For example, the results of protocol analysis are very useful for the following QoS and firewall functions, so the relevant NFV

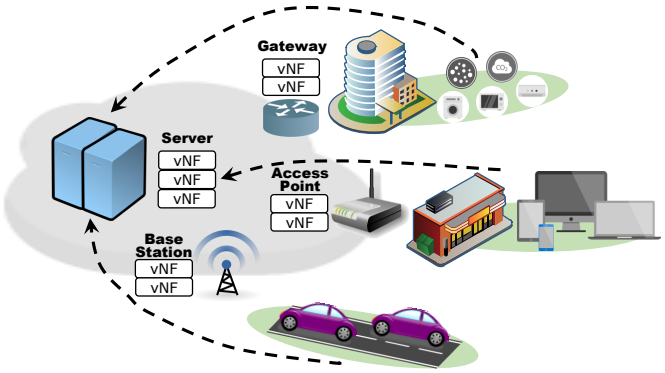


Fig. 1. The application scenario of the proposed edge NFV framework.

containers will share their results.

The main contributions of our work include:

- We present a deep-learning-enabled NFV framework. This framework is light-weight and resource-efficient, while keeping the strong analysis abilities brought by the deployed deep models. We adopt this framework for the edge-centric scenarios, and the results show its high efficiency and relatively low costs.
- We introduce a typical use case to demonstrate the feasibility of the proposed framework, and to show how the NFV containers work and how to use them. The simulations prove the deep model works well in the proposed framework and shows a good recognition performance.

II. Recent Advances in NFV and Efficient Deep Learning Systems

As the paper mainly consists of two subtopics, i.e. NFV and edge-computing-based deep learning, some related works regarding these areas will be reviewed in this chapter.

A. NFV Methods and Platforms

Currently, there are mainly three kinds of NFV frameworks, which are differentiated in the underlying platform. The first kind is based on the full-featured VMs, which can be deployed on general hardware; the second kind is for the specialized VMs, which require specific operating systems (OSs); the last one is used with the container, which can also be installed on general hardware while consuming much less resource. In the following paragraphs, we will respectively introduce their basic ideas and some existing implementations.

General-purposed VM The general-purposed-VM based NFV frameworks, such as the Cloud4NFV platform [4], normally follow the NFV standard guidelines, which can run on general-purposed hardware. Generally, NFVs are supposed to offer control functions and user/data module functions. Some NFV approaches, called full virtualization, move all control and user/data module functions to the virtual resource, while other NFV approaches, called partial virtualization, only move control functions or user/data module functions to the virtual resource. According to the reference architecture guideline of an NFV framework, three functionalities are necessary for realizing virtualization. The first one is NFV Infrastructure (NFVI), consisting of the hardware and software resource that are needed to build the environment for deployment and management. vNF is the second one, which obtains the resources from NFVI. And the VNF instances are managed by a VNF Manager. The last one is Operation Support System / Business Support System (OSS/BSS), as well as the description of Service, VNF and Infrastructure. In addition, a service layer is provided as an additional block to indicate the location of services offered to the users. The disadvantages of this kind of VNF frameworks include high resource consuming, slow deployment and running speed, etc., compared to the light-weight solutions, which is demonstrated by the results in the performance evaluation section.

Specialized VM As we know, middleboxes based on the custom hardware suffer from various issues with high opex and low scalability. That is the reason why specialized VMs, such as the ClickOS [5], are desirable. ClickOS supports a variety of middlebox functions based on the Click software, and is essentially a VM platform combining MiniOS, which is a Xen-based OS with Click modular router. MiniOS provides all the functions that are required for running Click, removing any other functions in a traditional Linux kernel. Therefore, both the CPU cycles and memory resources can be significantly saved. Many instances of ClickOS can be operated on the same physical machine, and ClickOS VMs can be created and destroyed by command-line interface (CLI) command or control threads of MiniOS according to the requirement of users. Thus, ClickeOS can achieve high scalability with multiple VMs. However, this kind of VMs, which are also called unikernels, requires some specific software environments. For example, ClickOS needs the Click software, which is a serious restriction for wide deployment.

Container Container is a software technology provided by the company Docker, which puts applications into a virtual environment and achieves the operating-system-level virtualization. Specifically, the virtual container can be transplanted into any devices running Linux or Windows system, which guarantees the flexibility of Docker container. Therefore, as a newly-emerged solution [6], network functions can be packaged with light-weight Docker containers, in order to obtain the high performance, such as high throughput and platform independence. Actions are performed to the Docker base image, and each file-system layer has the complete information about recreating the actions. Hence, compared with other full-featured VMs, it only needs to deliver the updates of each layer, which results in light-weight images for Docker container. Compared with other NFV frameworks, the container-based framework has the following advantages: first, it can decrease performance overhead to a large extent; second, it can achieve the fast deployment; third, the resource, such as the CPU cycles and memory, can be used with high utilization.

The work in [6] gives a good approach regarding how to deploy NFV with containers, and serves as a helpful example for us to design our deep learning enabled NFV containers. We build our framework on the basis of their edge structure, and successfully combine it with the specially-designed deep learning models.

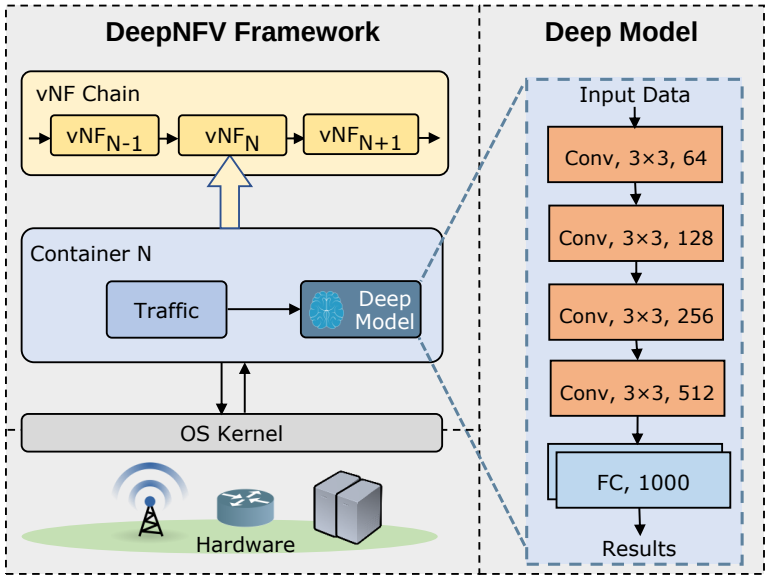


Fig. 2. The design and structure of the proposed DeepNFV framework.

B. Edge Computing and Efficient Deep Learning

In the recent years, human world is increasingly exposed to a boom of various mobile applications, e.g., social network, taxi booking, health care, crowdsensing [7], etc. As a result, the network traffic and computing load are rapidly increasing, and are predicted to double every year. To resolve this problem, edge computing is proposed to move the computing ability from the centralized servers to the devices near the user-end [8]. Edge computing brings two major improvements to the existing cloud network. The first one is that the edge nodes and devices obtain the ability to pre-process the large data before sending it to the central servers in the cloud. The other one is that the cloud resources are optimized by enabling the edge devices with computing ability [9]. Due to the potentiality brought by the edge computing, the aforementioned problems of the cloud infrastructure can be well addressed. Several instructive examples have been presented. The authors in [10] present an edge computing based scheme to support crowdsensing applications. They formulate the model into a mixed-integer nonlinear program and focus on cost-efficient resource provisioning. As a result, they work out a method to minimize the overall cost while satisfying the QoS requirement. Simulations prove the edge computing based approach can outperform traditional cloud computing methods. There are also several works focused on the network delay for the edge computing environment [11]. The authors attempt to decrease the service delay from two aspects, including the computational delay and the transmission delay. In their work, the former one is addressed with the VM migration and the second one is improved using the transmission power control.

Because the calculation of the deep models is computation-intensive, it is necessary to decrease the computing cost by removing weights, decreasing model complexity or removing redundant layers, in order to adapt the deep model into the edge computing environment. Recently, Sze et al. [12] design an energy-aware pruning method to decrease the energy cost of convolutinal neural network (CNN) models. They model the energy estimation methodology using the actual data extrapolate from the hardware measurements. This method successfully reduces the calculation cost by around 2 times.

Notably, the deep learning methods have been broadly used in the network related areas. For example, the authors adopt CNN model in [13], and design an intelligent traffic control system.

Based on these results, we successfully design an cost-efficient deep models for NFV containers, which are mainly deployed in the edge environments, to conduct various network tasks.

III. Deep-Learning-Enabled NFV Framework: Design and Applications

The design of the proposed DeepNFV framework is shown in Fig. 2. This framework mainly consists of two parts, i.e., the DeepNFV framework, and the adopted deep models.

The infrastructure layer in the framework deals with the underlying network devices and network connections. It is about the actual network hardware, including the base station, router, gateway, central servers, etc. These devices may be in the cloud, or in the network edge, in accordance with their distances to the end users. They are responsible for conducting the actual computation tasks and making the control commands.

The vNF is built on the docker container, which can package all the dependencies for one or some applications and run independently on the Linux and Windows operating systems. As shown in Fig. 2, the docker container can adopt several interfaces to access the underlying system kernel, in order to implement the virtualization functions and platforms for the higher-level applications. With these features, we built a deep learning subsystem in each vNF container. The deployed deep models will first analyze the input network traffic, then analyze the hidden patterns, and ultimately, output the recognition

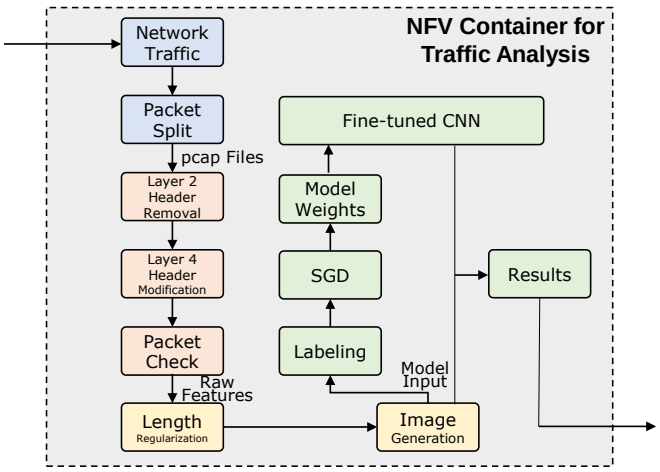
or prediction results for various pre-designed network functions. The deep models are pre-trained using manually-labeled dataset, and can be fine-tuned using newly captured packets in the vNF container. The adopted model is not limited. We give an example in the right part of Fig. 2, which is a simple CNN model with six layers.

We build our framework on the basis of the Glasgow network funtions (GNF). GNF is proposed in [6] for container-based vNF implementation, which can take the control of the vNF containers and publishes the vNFs to the users.

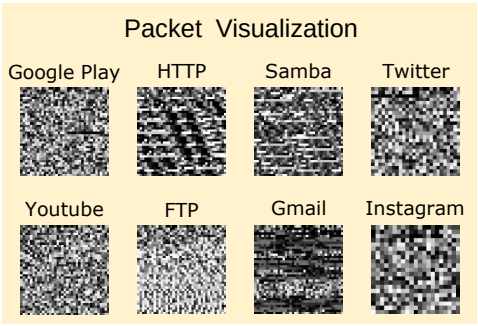
In this section, we will introduce some use cases for the proposed DeepNFV framework. There are a lot of scenarios that deep learning methods can be used to implement some useful network functions, such as intrusion detection, critical link analysis, protocol classification, network firewall, routing table, etc. Although the existing approaches work well in traditional network environments, they have struggled in the current network communications, where a lot of applications exist and generate huge network traffic which need to be classified for various objectives. In this section, we will look at one typical use case, i.e., the traffic analysis, to illustrate the application of the proposed NFV framework.

Traffic analysis is an important task in the field of network security. How to accurately recognize the specific protocols in a captured data packet has attracted lots of researchers. We built our solution on top of several well-developed theories in relevant areas [14], [15], and design an efficient model for the traffic recognition. This CNN model consists of six layers, including four convolutional layers and two full-connected layers.

The illustration of the traffic analysis container is shown in Fig.3a, and some typical images generated from the raw network traffic are shown in Fig.3b. We can see that there are some obvious patterns in this images which can be analyzed by the deep learning models for detection or classification by this container. This container belongs to an NFV chain that the system selects for one specific user, therefore, its input may be the user device or the output which is generated from some other NFV containers. The first task to process the raw network traffic is to split them into some discrete units, i.e., network packets. These units are recorded as the packet capture (pcap) file, which is the universal file format in traffic analysis and can be used with the libpcap library in Unix-like systems. Second, the packet headers should be modified in order to remove unnecessary and interfering information. Data link layer is the second layer of the open systems interconnection model (OSI model), and there are some Ethernet related data in its header, for example, the Media Access Control (MAC) address, which is no use for the protocol judgments and, therefore, should be removed. In addition, the transport layer, which is the fourth layer of the OSI model, provides headers with different length for Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) packets, and should also be modified to the same length or simply removed for simplification. The last step to clean the packets is to delete the duplicated or empty pcap files, then the resulted files will be regularized to the same size s. The files larger than s are trimmed to size s, and the files smaller than s are enlarged by adding duplicate



(a)



(b)

Fig. 3. The NFV container for traffic analysis. (a) The preprocess and recognition procedure. (b) Some image examples generated from network packets of various protocols.

bits. After these preprocess, the pcap files can be transformed into two-dimension data, 30×30 for example, which can be regarded as gray-scale images if mapped to $[0, 1]$.

In the training phase of the proposed deep model, the generated images will first be marked with different labels, according to the specific protocols of their raw packets. Then the generated images, as well as the corresponding labels, will be imported into the CNN model for fine-tuning. The Stochastic Gradient Descent (SGD) method is adopted in the training process, and can minimize the network loss by optimizing the model weights. After several iterations of the training process, we can get a fine-tuned CNN model for the final classification. In the interfering phase, the image generation module will directly transfer the generated images to the well-trained CNN model, which can give deep insight into the pattern details and output accurate recognition results.

Ultimately, this container transfers the results to other NFV containers in the chain, and these containers can make corresponding decisions to the raw network packets, according to the recognition results generated in the traffic analysis container. For example, there may be an NFV container for quality of service (QoS) optimization in the following chain. It can carry out different strategies for the network packets, e.g., putting Voice over IP (VoIP) applications at a higher tier of priority for network traffic and moving down the tier

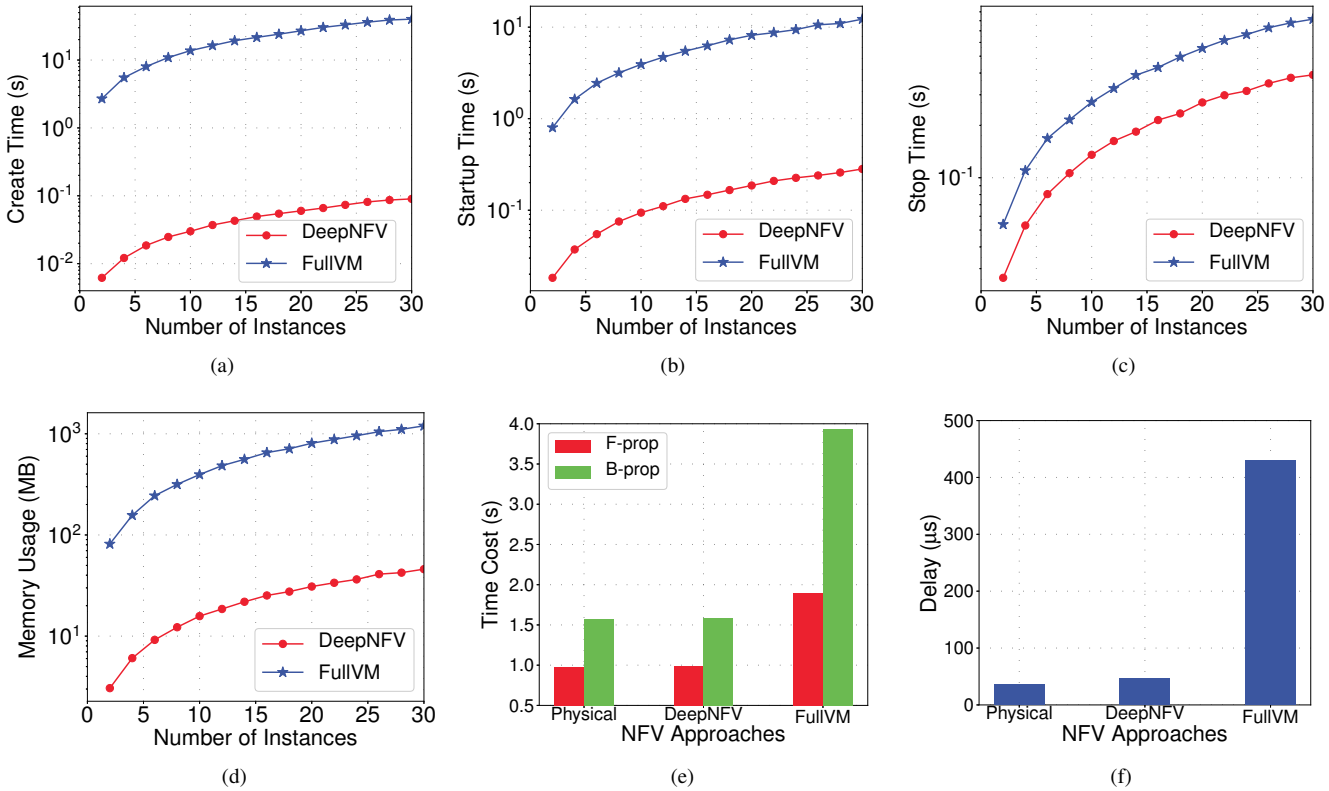


Fig. 4. The results of performance evaluation. (a∼d) The deployment performance and the scalability. (e) The computational ability. (f) The network performance.

for BitTorrent downloading traffic in case that they take up too much bandwidth. Another common scenario is that the network manager desires to completely get rid of some specific type of protocol, such as gaming or proxy connections, they can set up the firewall NFV containers to ban these traffic, also according to the recognition results.

IV. Performance Evaluation

In this section, we conduct a series of experiments to demonstrate the proposed DeepNFV framework, mainly including two aspects, i.e., the deployment performance and scalability of the proposed framework, and the recognition performance of the deep model installed in the DeepNFV container.

A. Container Performances

First, we conduct a series of experiments to test the performance of the proposed DeepNFV framework, as shown in Fig. 4. In (a∼d), we present the results regarding the deployment performance and the scalability. The x-axis is the total number of instances, which could be our DeepNFV framework based on container, or KVM-based VM. KVM is a famous VM platform and has been widely used in commodity servers. The y-axis is the measurement index, including the time cost and the memory usage.

We create 30 instances in each test, and record their initialization time, startup time, stop time, and the memory usages. As we can see from Fig. 4a, our container-based DeepNFV framework takes the shortest time for initialization. Thanks to the light-weight nature of the docker container, DeepNFV only needs one percent of the time of the traditional full VMs, which make most infrastructure and functions visualized, resulting in a big VM image and slow creation time. Also, in both (b) and (c), our DeepNFV show a significant advantage in startup and stopping time, compared with the FullVM approach. In (d), we present the memory usage for DeepNFV and FullVM. It can be seen that DeepNFV shows an obvious advantage against the full VMs, due to their light-weight design.

As out main objective is to adopt the deep learning models for the NFV containers, we conduct another test to evaluate the computing performance when running deep learning calculations. The computational cost of the deep models mainly consist of two parts, i.e., the forward propagation and the back propagation. The former one can represent the model running speed in the interfering phase, and the latter one is mainly used in the training phase. We can see that the time costs in DeepNFV are smaller than the ones of FullVM, and are very close to the performances in the physical machines, which demonstrate its efficiency in performing deep learning related tasks.

The last test is about the network performance. We use the round-trip time (RTT) to show the network latency. Once again, we see the delay value of DeepNFV is smaller than the FullVM. Similarly, there is few difference between the physical machine and the proposed container-based approach.

TABLE I
THE ACCURACY OF TRAFFIC ANALYSIS.

Protocol	DeepNFV		CART	
	Precision	Recall	Precision	Recall
Samba	1.00	1.00	1.00	1.00
Twitter	0.92	0.96	1.00	0.80
Google Play	0.84	0.99	0.49	1.00
Youtube	1.00	0.91	0.94	0.87
Gmail	0.97	0.85	0.61	0.51
FTP	0.94	1.00	1.00	0.73
HTTP	0.99	1.00	1.00	1.00
Instagram	0.96	0.88	0.91	0.62
Average	0.95	0.95	0.87	0.82

B. Classification Performance

In this paper, we introduce a DeepNFV application, which can recognize the protocols of network traffic, as an user case. In order to demonstrate its feasibility, we conduct a traffic recognition experiment. We record a large number of network packets, which mainly includes the following protocols or applications, i.e., Samba, Twitter, Google Play, Youtube, Gmail, FTP, HTTP, Instagram, etc.

We compare our DeepNFV framework, which can use the state-of-the-art deep learning methods for automatic feature extraction and classification, with the traditional classification and regression trees (CART) algorithm, in order to show the advantage to adopt the deep models for vNF implementations. CART is a statistical classifier, and can generate the decision tree for classification. In Table. I, we present the recognition results for these two methods. The classification performance is usually measured in two aspects, i.e., precision and recall. Precision represents the accuracy of the classifier's judgments, and recall represents its ability to find out as many as possible packets in one specific category. As we can see, the deep learning methods outperform the traditional CART algorithm in most protocols and measurements, which proves the necessity to enable the vNF with deep learning support. More precisely, the deep model based method improves the precision and recall by 8% and 13% respectively, compared to the CART approach. Also, our method performs robustly in all these scenarios, and as a comparison, the CART approach has a poor performance in some tests, such as the Gmail test.

V. CONCLUSION

In the paper, we propose a deep-learning-enabled NFV framework named DeepNFV, which can be deployed in the network edge. In order to work out a light-weight implementation, we select a container-based design, i.e., build the vNF in the containers rather than the full-featured VMs. Docker container can be installed on general hardware while consuming little resource, therefore, is a good solution for the edge-centric scenarios. Based on that, we combine the edge-computing-based NFV with the deep learning models, empowering the DeepNFV framework with the abilities to analyze complicated network traffic and conduct various tasks. Simulations demonstrate its feasibility, efficiency, as well as the high recognition precision in a typical use case.

Future work includes an optimization method to decrease the computation cost of the adopted deep models on the edge servers, in order to make DeepNFV framework more energy-efficient and sustainable.

ACKNOWLEDGMENT

This work is partially supported by JSPS KAKENHI Grant Number JP16K00117, JP15K15976, and KDDI Foundation.

REFERENCES

[1] J. d. J. Gil Herrera and J. F. B. Vega, "Network functions virtualization: A survey," *IEEE Latin America Transactions*, vol. 14, no. 2, pp. 983–997, Feb 2016.

[2] M. Chiosi, D. Clarke, P. Willis, A. Reid, J. Feger, M. Bugenhagen, W. Khan, M. Fargano, C. Cui, H. Deng *et al.*, "Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action," in *SDN and OpenFlow World Congress*, 2012, pp. 22–24.

[3] Z. M. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2432–2455, Fourthquarter 2017.

[4] J. Soares, M. Dias, J. Carapinha, B. Parreira, and S. Sargento, "Cloud4nfv: A platform for virtual network functions," in *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*, Oct 2014, pp. 288–293.

[5] J. Martins, M. Ahmed, C. Raiciu, and F. Huici, "Enabling fast, dynamic network processing with clickos," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13. New York, NY, USA: ACM, 2013, pp. 67–72. [Online]. Available: http://doi.acm.org/10.1145/2491185.2491195

[6] R. Cziva and D. P. Pezaros, "Container network functions: Bringing nfv to the network edge," *IEEE Communications Magazine*, vol. 55, no. 6, pp. 24–31, 2017.

[7] H. Li, K. Ota, M. Dong, and M. Guo, "Mobile crowdsensing in software defined opportunistic networks," *IEEE Communications Magazine*, vol. 55, no. 6, pp. 140–145, 2017.

[8] X. Tao, K. Ota, M. Dong, H. Qi, and K. Li, "Performance guaranteed computation offloading for mobile-edge cloud computing," *IEEE Wireless Communications Letters*, vol. PP, no. 99, pp. 1–1, 2017.

[9] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili, "Collaborative mobile edge computing in 5g networks: New paradigms, scenarios, and challenges," *IEEE Communications Magazine*, vol. 55, no. 4, pp. 54–61, 2017.

[10] H. R. Arkian, A. Diyanat, and A. Pourkhalili, "Mist: Fog-based data analytics scheme with cost-efficient resource provisioning for iot crowdsensing applications," *Journal of Network and Computer Applications*, vol. 82, pp. 152–165, 2017.

[11] T. G. Rodrigues, K. Suto, H. Nishiyama, and N. Kato, "Hybrid method for minimizing service delay in edge cloud computing through vm migration and transmission power control," *IEEE Transactions on Computers*, vol. 66, no. 5, pp. 810–819, May 2017.

[12] V. Sze, T.-J. Yang, and Y.-H. Chen, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[13] F. Tang, B. Mao, Z. M. Fadlullah, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "On removing routing protocol from future wireless networks: A real-time deep learning approach for intelligent traffic control," *IEEE Wireless Communications*, vol. PP, no. 99, pp. 1–7, 2017.

[14] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "End-to-end encrypted traffic classification with one-dimensional convolution neural networks," in *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*, July 2017, pp. 43–48.

[15] M. Lotfollahi, R. S. H. Zade, M. J. Siavoshani, and M. Saberian, "Deep packet: A novel approach for encrypted traffic classification using deep learning," *CoRR*, vol. abs/1709.02656, 2017. [Online]. Available: http://arxiv.org/abs/1709.02656

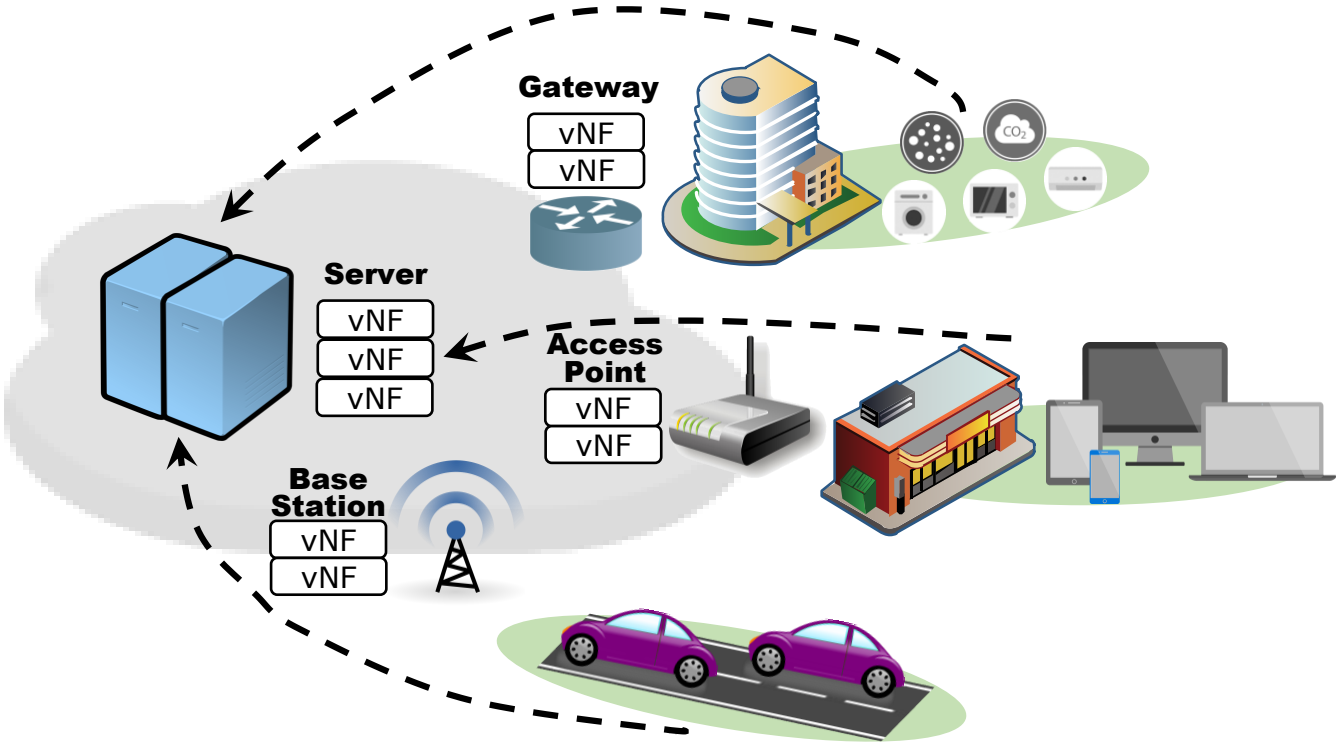


Fig. 1. The application scenario of the proposed edge NFV framework.

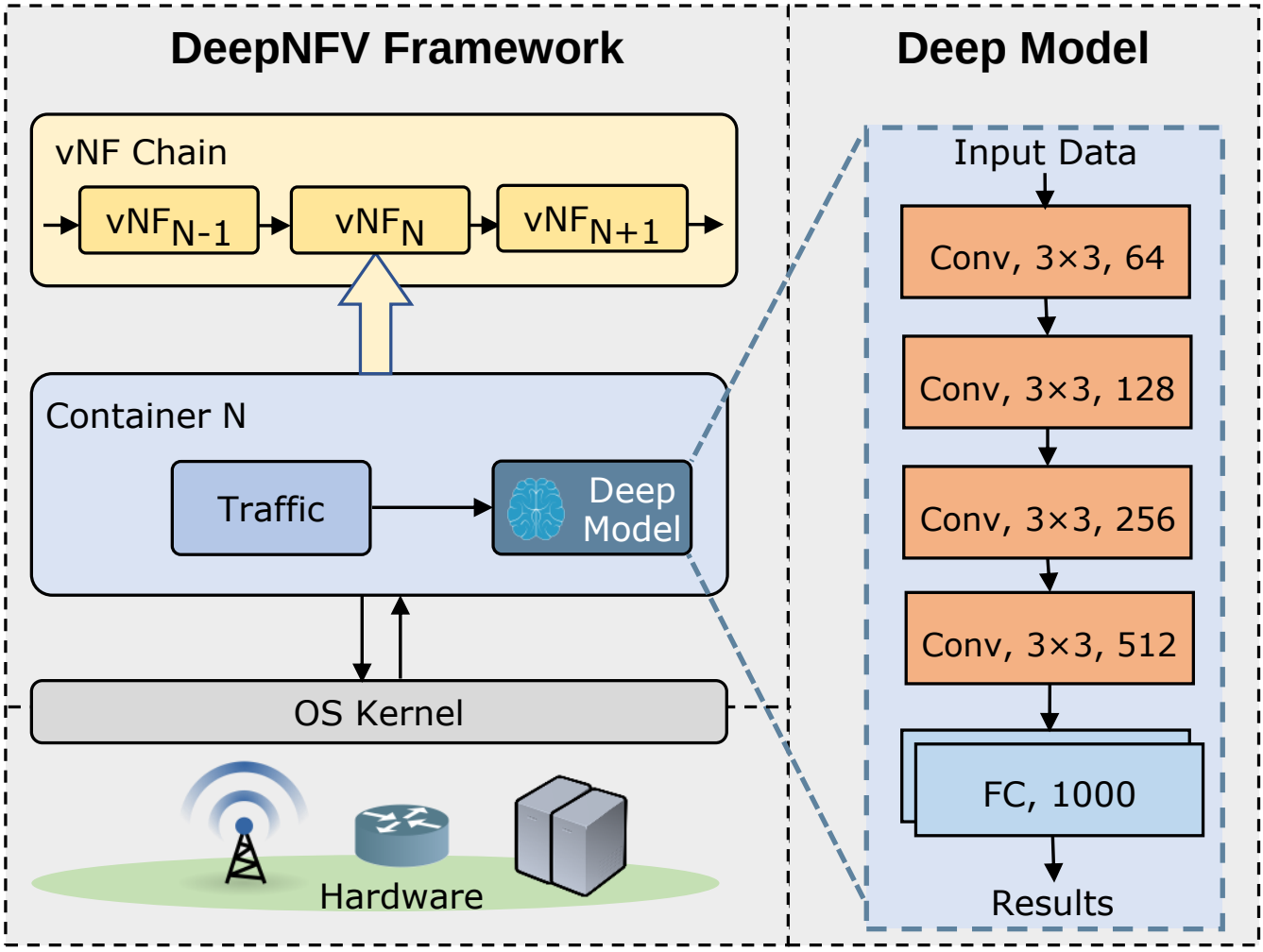
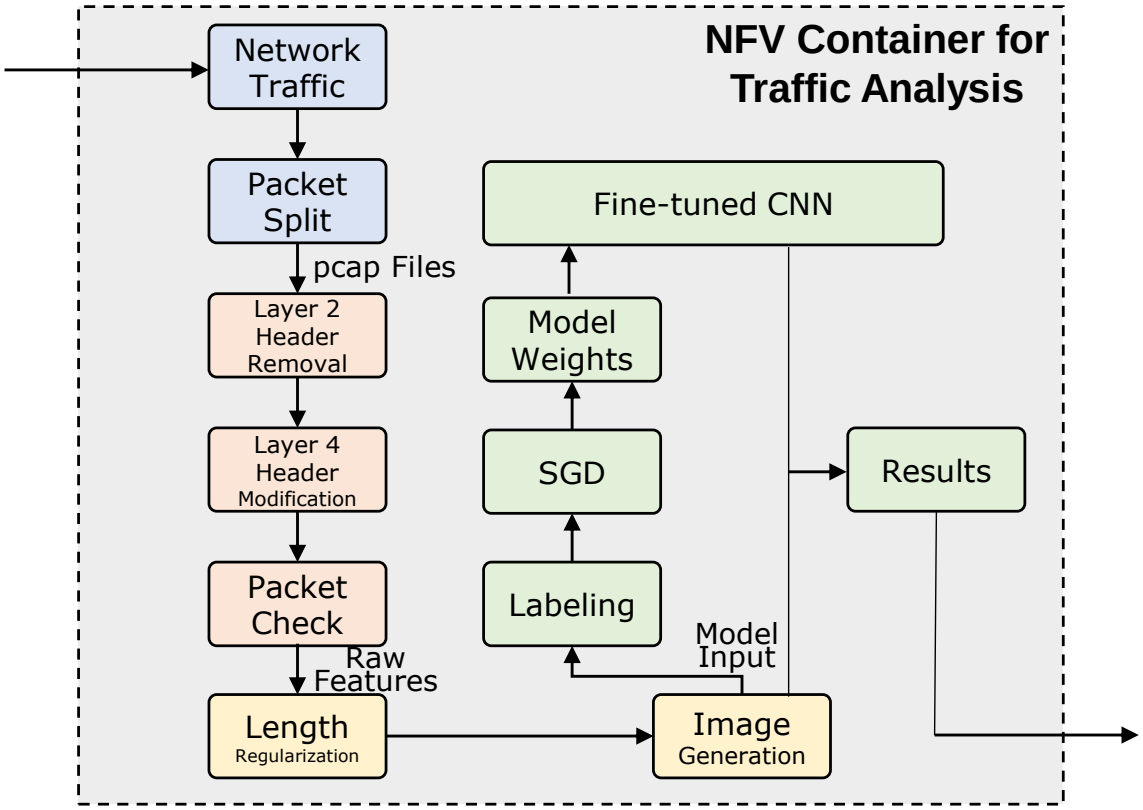
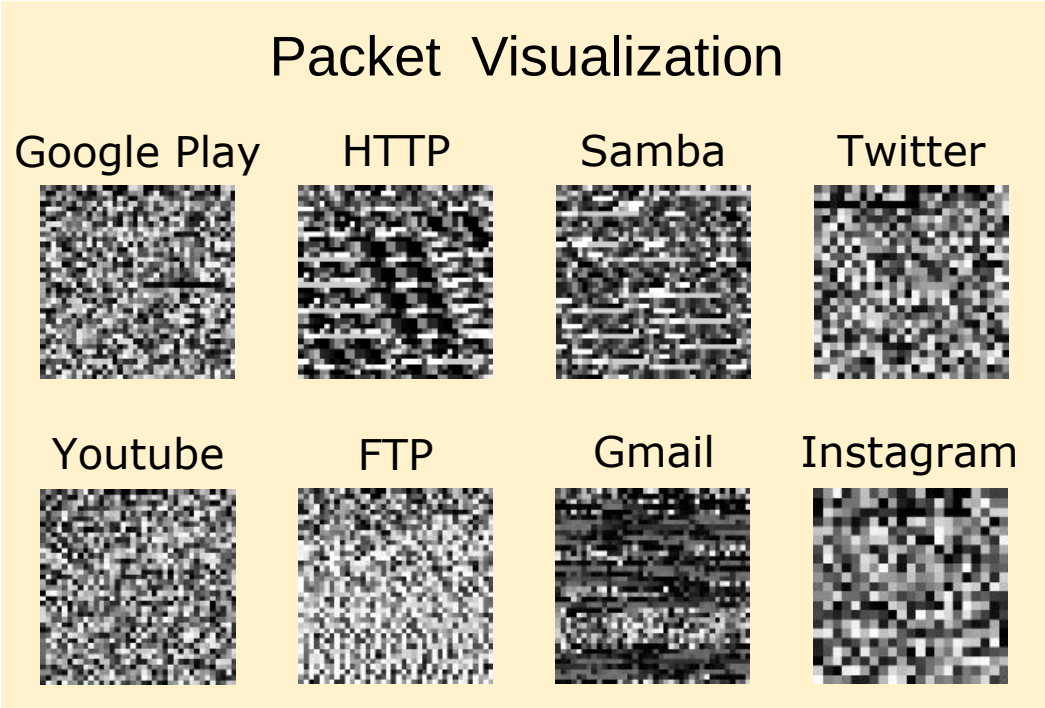


Fig. 2. The design and structure of the proposed DeepNFV framework.



(a)



(b)

Fig. 3. The NFV container for traffic analysis. (a) The preprocess and recognition procedure. (b) Some image examples generated from network packets of various protocols.

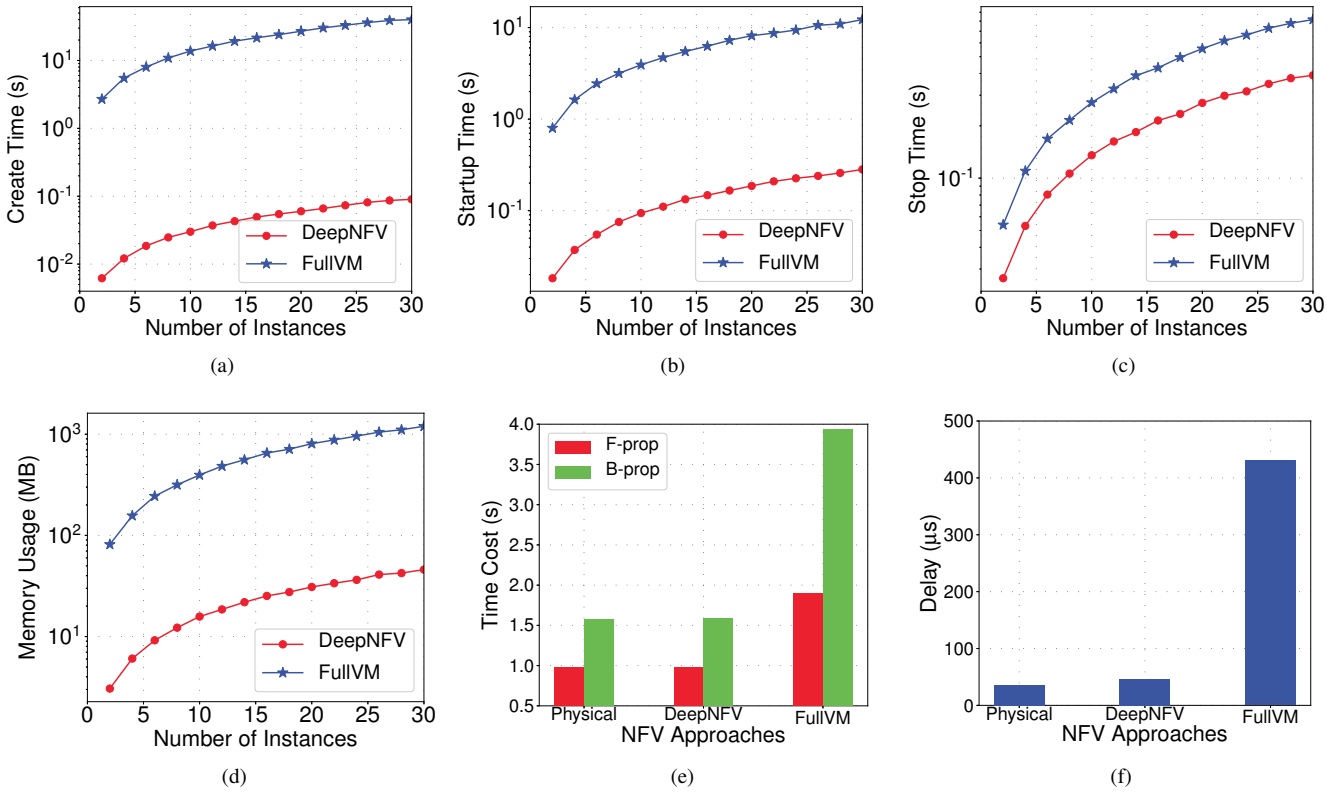


Fig. 4. The results of performance evaluation. (a∼d) The deployment performance and the scalability. (e) The computational ability. (f) The network performance.

TABLE I
THE ACCURACY OF TRAFFIC ANALYSIS.

Protocol	DeepNFV		CART	
	Precision	Recall	Precision	Recall
Samba	1.00	1.00	1.00	1.00
Twitter	0.92	0.96	1.00	0.80
Google Play	0.84	0.99	0.49	1.00
Youtube	1.00	0.91	0.94	0.87
Gmail	0.97	0.85	0.61	0.51
FTP	0.94	1.00	1.00	0.73
HTTP	0.99	1.00	1.00	1.00
Instagram	0.96	0.88	0.91	0.62
Average	0.95	0.95	0.87	0.82