Computation Offloading for Mobile Edge Computing: A Deep Learning Approach

Shuai Yu*, Xin Wang*, Rami Langar†

*LIP6 / UPMC University of Paris 6, 4 Place Jussieu, 75005 Paris, France.
†LIGM/UPEM, University Paris Est; Cité Descartes, 77454, Marne-la-Vallée, France.
E-mail: shuai.yu@lip6.fr, rami.langar@u-pem.fr, xin.wang@lip6.fr

***Abstract*—Computation offloading has already shown itself to be successful for enabling resource-intensive applications on mobile devices. Moreover, in view of mobile edge computing (MEC) system, mobile devices can offload compute-intensive tasks to a nearby cloudlet, so as to save the energy and enhance the processing speed. However, due to the varying network conditions and limited computation resources of cloudlets, the offloading actions taken by a mobile user may not achieve the lowest cost. In this paper, we develop a dynamic offloading framework for mobile users, considering the local overhead in the mobile terminal side, as well as the limited communication and computation resources in the network side. We formulate the offloading decision problem as a multi-label classification problem and develop the Deep Supervised Learning (DSL) method to minimize the computation and offloading overhead. Simulation results show that our proposal can reduce system cost up to 49.24%, 23.87%, 15.69%, and 11.18% compared to the "no offloading" scheme, "random offloading" scheme, "total offloading" scheme and "multi-label linear classifier-based offloading" scheme, respectively.**

I. Introduction

Nowadays, computing requirements for emerging applications are increasing tremendously, such as wearable cognitive assistance [1] and augmented reality [2]. Such kind of applications need mobile devices (google glass, smartphones, and so on) not only equip with considerable energy, memory size and computational resources, but also execute the application in a timely manner. A potential solution to address the challenges is through mobile cloud computing (MCC) [3]. Nevertheless, the conventional MCC imposes huge additional load both on radio and backhaul of mobile networks due to the high and variable latency to distant datacenters. Thus, a promising approach to tackle this challenge is to move cloud infrastructure (computation and storage abilities) closer to the end users.

Based on this, the European Telecommunications Standards Institute (ETSI) proposed an emerging concept of mobile edge computing (MEC) [4]–[6]. In this architecture, fixed and powerful servers are located at the network edge in fully distributed manner to reduce communication overhead and execution delay for mobile users. Nevertheless, the edge computing provides only limited radio, storage and computational resources with respect to the conventional MCC. Thus, the offloading actions taken by a mobile user may not always achieve the lowest cost.

978-1-5386-3531-5/17/$31.00

In this paper, we will investigate a dynamic computation offloading framework for single-user single cell MEC network. Our objective is to minimize the offloading cost considering the network resource usage. Our major contributions are summarised as follows:

- We propose a fine-grained computation offloading framework that minimizes the offloading cost for the MEC network. The offloading actions taken by a mobile user consider the local execution overhead as well as varying network conditions (including wireless channel condition, available communication and computation resources). Most of previous machine learning-based offloading works consider either coarse-grained computation offloading [7], [8], or assume an infinite amount of available communication or computation resources in cloudlet.
- We formulate and build a Deep Supervised Learning (DSL) model to obtain an optimal offloading policy for the mobile user. Our method provide a pre-calculated offloading solution which is employed when a certain level of knowledge about the application and network conditions. We formulate the continuous offloading decision problem as a multi-label classification problem. This modelling strategy largely benefits from the emerging deep learning methods in the artificial intelligence field. Our method approaches the optimal solution obtained by the exhaustive strategy performance with a very subtle margin. As the number of fine-grained components n of an application grows, the exhaustive strategy suffers from the exponential time complexity $O(2^n)$. Fortunately, the complexity of our learning system is only $O(mn)^2$, where m is the number of neurons in a hidden layer which indicates the scale of the learning model. In fact, the artificial intelligence problem with both large input/output dimension and large number of examples such as image classification is proved

experimentally feasible using similar technique [9]. To the best of our knowledge, our work is the first of its kind that achieves optimal offloading policy through Deep Learning approach.

The reminder of this paper is organized as follows. Section II introduces an overview of the related works. In Section III, we present the system model. Section IV formulates the optimaztion problems, followed by a description of our proposed algorithm in Section V. Simulation results are presented in Section VI. Finally, conclusions are drawn in Section VII.

II. RELATED WORK

Computation offloading mechanisms have been studied extensively [7], [8], [10], [11] and generally fallen into two categories: coarse-grained offloading [7], [8] and fine-grained (typically at a method-level granularity) offloading [10], [11]. Coarse-grained offloading also refers to the full offloading or total offloading in which full task is migrated to the cloud. This approach does not require estimating the computation overhead prior to the execution. Whereas fine-grained offloading (partial offloading or dynamic offloading) dynamically transmits as little code as possible and offloads only the computation-hungry parts of the application. Despite the introduced burden on the application programmers and additional computational overhead, the fine-grained offloading can reduce unnecessary transmission overhead, achieving a reduced latency and energy consumption.

Previous machine learning based offloading schemes mainly focus on the coarse-grained offloading. For example, authors in [7] proposed machine learning-based runtime scheduler for mobile offloading framework. By running various machine learning algorithms, the results show that instance-based online offloading schedular selects the best scheduling decision. Authors in [8] proposed MALMOS, a machine learning-based mobile offloading scheduler with Online Training. The scheduler adapts offloading decisions based on the observation of previous offloading decisions and their correctness. However, both of the offloading schedulers are based on the local area network (LAN) and wide area network (WAN). These strategies are not able to handle fine-grained offloading problem, because it does not take the dependencies of varying wireless network state and available resources into account during an application execution period.

Recently, deep supervised learning [12] method has achieved its amazing performance. Deep learning has outperformed other machine learning methods in almost all the artificial intelligence fields, including computer vision, speech recognition, natural language processing, etc. With respect to the conventional machine learning-based offloading schemes (e.g. Markov decision process-based offloading scheme in [10]), deep learning scheme carries two advantages: i) noteworthy accuracy capacity of reaching in decision making, and ii) extreme calculation speed for test with a trained model. This motivates us to design a deep learning-based offloading schedular for 5G MEC network.

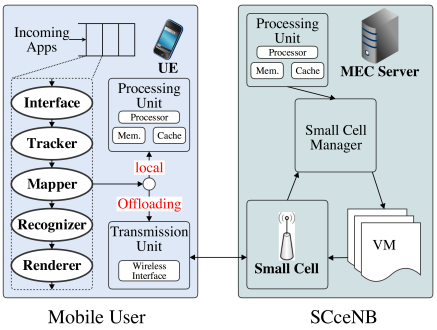

Fig. 1: Mobile offloading in MEC scenario

III. SYSTEM MODEL

In this work, we consider a small cell-based mobile edge computing system, which is also known as small cell cloud. The basic idea is to enhance small cell base stations (e.g., pico, femto) by an additional computation and storage capabilities. The novel base station is called Small Cell cloud-enhanced e-Node B (SCceNB).

In this section, we first present the application model (local execution) and the network model (remote execution) for our dynamic offloading framework, followed by a description of decision making procedure.

A. Application and Local Execution Model

1) Application Model: We assume that a particular mobile user executes a resource-intensive application $\mathcal{G}$. Moreover, the application can be split into multiple components which in the granularity of either method or thread (i.e., a fine-grained partitioning). Each component can be either executed locally or offloaded to the SCceNB. Then, we exploit the concept of call graph [13], which consists in modelling the relationship between components as a weighted directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ denotes the set of components, and $\mathcal{E}$ the data dependencies between components. For example, Fig. 1 illustrates a mobile user offloads an immersive application [14] to a SCceNB. The immersive application is compute-intensive and can be split into different components.

We assume each edge $e_{v-1,v}$ ($e_{v-1,v} \in \mathcal{E}$) represents the data communication (computation result) between two components. We let ϕ_v ($v \in \mathcal{V}$) denotes the weight of component v, which specifies the workload (CPU cycles) for the component v. $e_{v-1,v}$ and $e_{v,v+1}$ represent the input and output data size for component v, respectively. For a given $e_{v-1,v}$, ϕ_v can be derived from [15] as $\phi_v = \omega \cdot e_{v-1,v}$, where the ω in CPU

cycles/byte (cpb) indicates the number of clock cycles a microprocessor will perform per byte of data processed in an algorithm. The parameter depends on the nature of the component, e.g., the complexity of the algorithm. The estimation of this value has been studied in [16] which is thus beyond the scope of our work.

If the current component v is executed on the mobile device, the completion time is $T_l(v) = \frac{\phi_v}{f_u}$, where f_u denotes the CPU rate of mobile devices (in MIPS i.e., Million Instructions Per Second).

B. Network and Remote Execution Model

As illustrated in Fig. 1, a new entity, denoted as Small Cell Manager (SCM) is deployed in the SCceNB. The SCM is in charge of managing the resources and virtualizing the resources by means Virtual Machine (VM). The network deployment is based on the orthogonal frequency division multiple-access (OFDMA). We assume that the available bandwidth B for transmission is divided into N subcarriers. Let $n \in \mathcal{N} = \{1, 2, ..., N\}$ denotes the available subcarriers to be allocated in each component execution period.

Considering the limited computation capacity of SCceNB, we assume that the SCceNB is equipped with M CPU cores, and the CPU rate of each CPU core is given by f_s. Let $m \in \mathcal{M} = \{0, 1, 2, ..., M\}$ denotes the available CPU cores to be allocated in each component execution period, where $m = 0$ denotes the SCceNB is busy, thus the offloading requirement is rejected by the SCceNB.

Let p_u and p_s denote the transmit power for the UE and SCceNB, respectively. The maximum achievable rate (in bps) for uplink over an additive white Gaussian noise (AWGN) channel can be expressed as follows:

$$r_{ul} = n\frac{B}{N}\log_2\left(1 + \frac{p_u|h_{ul}|^2}{\Gamma(g_{ul})d^\beta N_0}\right) \tag{1}$$

where B is the bandwidth, d is the distance between UE and SCceNB, N_0 denotes the noise power. In this paper, we consider the Rayleigh-fading environment, h_{ul} and h_{dl} are the channel fading coefficient for uplink and downlink, respectively, and β is the path loss exponent. Note that $\Gamma(BER) = -\frac{2log(5BER)}{3}$ represents the SNR margin introduced to meet the desired target bit error rate (BER) with a QAM constellation. g_{ul} and g_{dl} are the target BER for uplink and downlink, respectively.

We assume that uplink (UL) and downlink (DL) transmission experience the same noise. The maximum achievable rate (in bps) for downlink is thus given by:

$$r_{dl} = n\frac{B}{N}\log_2\left(1 + \frac{p_s|h_{dl}|^2}{\Gamma(g_{dl})d^\beta N_0}\right) \tag{2}$$

C. Decision Making Procedure

During the application execution, we assume a time interval t which is long enough for the user to finish executing any application component locally or remotely on the SCceNB. Each of such time intervals is defined as a decision period.

At the beginning of each decision period, the user sends a report on its buffer size $e_{v-1,v}$ (i.e., input data size for offloading) and channel quality that it experiences with respect to the SCceNB. In the LTE standard, these information are sent through the BSR (buffer size report) and the CQI (channel quality information) messages [17]. The SCceNB receives the report, allocates communication resource (subcarrier n) and computation (CPU cores m) to the UE, according to its current available resources and the received report. A composite state for the current system states is thus given by $S = (v, e, n, m)$.

Based on the observed composite state S, the user computes the immediate costs of local execution and offloading (as will be explained in Section IV-B), takes an action decision a_v of either executing the component locally on the mobile device (denoted $a_v = 0$) or offloading to the SCceNB (denoted as $a_v = 1$). The offloading decision for the current application can be denoted by $\pi = \{a_v, v = 1, 2, ..., |\mathcal{V}|\}$.

However, an offloading action is not always successful. An offloaded component may fail due to lack of resources (i.e., m or n). As a result, the user has to restore and execute the failed component again (either locally or remotely) in the next decision period.

IV. Problem Formulation

A. State Space and Action Space

The state space of UE in the MEC network is defined as follows:

$$\begin{aligned}\xi =&\{S = (v, e, n, m) \in \mathcal{S} | v \in \mathcal{V}, e \in \mathcal{E}, n \in \mathcal{N},\\ &m \in \mathcal{M}\}\end{aligned} \tag{3}$$

The action space is $\mathcal{A} = \{a_v \in \{0, 1\}, v \in \mathcal{V}\}$, indicating that the UE can execute the current component v locally ($a_v = 0$), or offload the component to SCceNB ($a_v = 1$).

B. Immediate Cost

For the current decision period, an immediate cost $C(S, a_v)$ is expressed as follows:

$$C(S, a_v) = \begin{cases} C_l(S) = T_l(v), & a_v = 0 \\ C_r(S), & a_v = 1 \end{cases} \tag{4}$$

where $C_l(S)$ is the immediate local execution cost (equals to the local execution delay), and $C_r(S)$ is the immediate offloading cost which is defined as follows:

$$\begin{aligned}C_r(S) =&\gamma_1 \cdot [(1 - a_{v-1}) \cdot T_s(e_{v-1,v}) + T_i(v, e_{v,v+1}) +\\ &t_d] + \gamma_2 \cdot C_{pay}^{comp}(m) + \gamma_3 \cdot C_{pay}^{rad}(n),\end{aligned} \tag{5}$$

where γ_1, γ_2 and γ_3 are the weight factors, t_d is the decoding delay for the UE to decode the computation results from SCceNB, a_{v-1} represents the previous offloading decision.

The first term in (5) denotes the delay for the UE to offload the input data $e_{v-1,v}$ and receive the output data $e_{v,v+1}$. $T_s(e_{v-1,v}) = e_{v-1,v}/r_{ul}$ denotes the uplink transmission delay. Note that the SCceNB can temporarily cache the computation results, for the current component v, if its previous component $v-1$ is offloaded to the SCceNB (i.e., $a_{v-1} = 1$), the output data $e_{v-1,v}$ is thus cached in the SCceNB. As a result, the UE will not retransmit the data to SCceNB and the first term equals to 0. $T_i(v, e_{v,v+1})$ denotes the UE's idling delay for the SCceNB executing the component v and transmitting the output data back:

$$T_i(v, e_{v,v+1}) = \frac{\phi_v}{m \cdot f_s} + \frac{e_{v,v+1}}{r_{dl}}. \tag{6}$$

The second term in (5) denotes the computation resource usage, where $C_{pay}^{comp}(m) = m/M$ represents the SCceNB's computation resource payments. Similarly, the last term denotes the radio resource usage, where $C_{pay}^{rad}(n) = n/N$.

C. Optimal Problem Formulation

Our problem can be described as reinforcement learning scenario or Markov Decision Processes (MDPs) [10]. The objective is to find an agent which makes optimal offloading policy for each application. The optimal offloading policy denoted by $\pi^*(\pi^* \in \pi)$ can minimize the system cost given by:

$$\pi^* = \underset{\pi}{\operatorname{argmin}} \sum_{v \in \mathcal{V}} C(S, a_v) \tag{7}$$

Note that in our work, the system cost $\sum_{v \in \mathcal{V}} C(S, a_v)$ is a long-term cost, which is not provided immediately, but until all the components been processed. This cost is formally named as *delayed cost*.

V. Algorithm Design

In this section, we describe our proposed deep supervised learning algorithm for optimal offloading decision making.

We formulate the fine-grained offloading problem as a multi-label classification [18] framework. Specifically, given an application $\mathcal{G}$ which contains $|\mathcal{V}|$ components, the input of our model is the observation of the network states of all components. Our decision is a $|\mathcal{V}|$-dimensional vector. If a component is offloaded, its value is 1, otherwise 0. We evaluate our output with respect to the optimal output by multi-label accuracy, which is defined as the proportion of the predicted correct labels to the total number of labels for that application.

Our algorithm operates in three steps, i.e. initial phase, training phase, and action or testing phase. In the following, we describe these three phases.

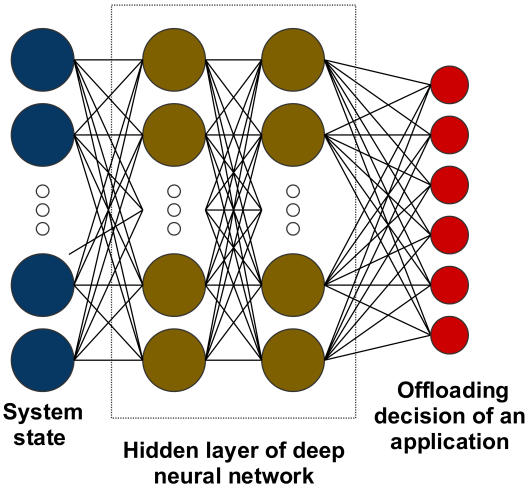

Fig. 2: Deep learning-based offloading framework.

A. Initial Phase

The objective of the initial phase is to obtain the raw data for training our deep supervised learning framework. We run 10,000 times the random and exhaustive-based optimal offloading strategy. Note that we used an exhaustive algorithm to search the optimal offloading policy from all the $2^{|\mathcal{V}|}$ offloading possibilities. In each execution, we vary the network conditions. Specifically, we record network state S and the corresponding optimal offloading strategy. This data is further randomly split into K_{tr} for training phase and K_{te} for testing phase.

B. Training Phase

In this phase, the features of training data are trained using deep neural network. We cross validate our training data to define the number of hidden layers to be 2, and the number of neurons to be both 128, as shown in Fig.2. Conventionally, we use rectified linear unit (ReLU) [19] as the activation function, dropout [20] as the regularization and sigmoid as the output. This network takes the state of S as input, and the offloading decision a_v as output. The objective is to minimize the multi-label accuracy with respect to the optimal decisions. Since this is a multi-label classification problem, we conventionally set the loss function as binary cross entropy [21]. For optimizing the neural network, we use Adam optimizer [22]. The neuron number of the output layer is set to $|\mathcal{V}|$. If one of the output neurons is greater than 0.5, it is decided to offload, otherwise it is not offloaded.

C. Testing Phase

Once the training phase of the deep neural network is finished, we can apply it onto any unseen application. This step is called testing phase. At this time, the deep

TABLE I: Network Parameters

Parameter	Value	Parameter	Value
B	3	β	-2
g_{ul}	10^{-3}	g_{dl}	10^{-3}
N_0	5×10^{-5}	t_d	0.1 s
d	10	γ_1	0.8
γ_2	0.1	γ_3	0.1
p_u	0.01 W	p_s	0.1 W
f_s	10^{10} cyclse/s	f_u	10^{11} cyclse/s
M	16	N	256

TABLE II: Offloading Accuracy

Scheme	Accuracy	Scheme	Accuracy
EOS	100%	DOS	60.1%
MOS	43.5%	TOS	22.4%
ROS	2.3%	NOS	0%

neural network takes the state as the input and outputs the decision for all components in the application. We evaluate the performance of our network based on the outputs of the testing phase.

VI. Performance Evaluation

A. Network Parameters

In our simulations, we consider single-user single cell small cell-based mobile edge cloud network. The parameters used in our simulations are reported in Table I.

For the computing components, we assume that each application consists of 6 random components, both the data dependencies $e_{u,v}(e_{u,v} \in \mathcal{E})$ and required number of CPU cyles per byte (cpb) for the components ω follow the uniform distribution with $e_{u,v} \in [100, 500]$ KB and $\omega \in [4000, 12000]$ cpb as in [23]. All random variables are independent for different components. The number of raw data that used to train our system is 10000, which means that we execute the application 10000 times independently through random offloading and exhaustive-based optimal offloading strategies in the Initial Phase.

We denote our deep learning-based offloading scheme as DOS, and the exhaustive-based offloading scheme as EOS. For comparison, we introduce four benchmark policies from the literature:

- *No Offloading Scheme (NOS):* Local execution, which means that applications are executed on smartphones by letting all the offloading decision variables $a_v = 0, v \in \mathcal{V}$.
- *Random Offloading Scheme (ROS):* The mobile device randomly chooses local execution or offloading action as in [10].
- *Multi-label linear classifier-based offloading scheme (MOS):* Multi-label linear classifier (a "shallow" learning)-based offloading scheme [18] by removing all the hidden layers of our deep model.
- *Total Offloading Scheme (TOS):* Coarse offloading strategies [8], which means that all the components are offloaded to the server side by letting all the offloading decision variables $a_v = 1, v \in \mathcal{V}$.

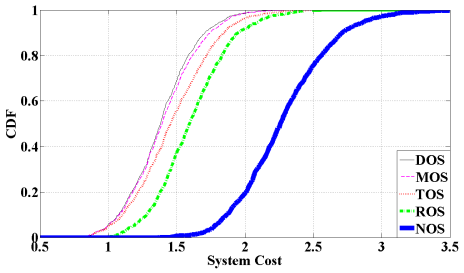

Fig. 3: Comparison of system cost.

B. Simulation Results

In what follows, we present the corresponding simulation results. We define the decision π^* that is made by EOS as an accurate decision. Thus, the offloading accuracy (denoted by η) for an offloading scheme is given by:

$$\eta = \frac{\sum_{\pi \in K_{te}} \pi^*}{K_{te}}, \tag{8}$$

where K_{te} denotes the total number of data that is used to test our system in testing phase.

TABLE. II reports the offloading accuracy for multiple offloading schemes with respect to the optimal offloading scheme (i.e. the EOS). We can observe that compared to the benchmark policies, our DOS scheme gets the highest accuracy.

Fig. 3 shows the cumulative distributed function (CDF) of the system cost shown in (7) for multiple offloading schemes. We can see that, our algorithm performs better compared to the others. Indeed, as shown in Fig. 3, our algorithm can reduce the system cost in average by 49.24%, 23.87%, 15.69%, and 11.18% compared to the NOS, ROS, TOS and MOS schemes respectively.

Fig. 4 shows the impact of the number of data used to train our system (i.e. K_{tr}) on the offloading accuracy metric. We can observe that the accuracy of our DOS scheme increases as K_{tr} grows, whereas K_{tr} has little influence on the other schemes. This means that our deep learning scheme has a more powerful learning capacity than the other schemes.

Fig.5 reports the impact of the distance between UE and SCceNB on the offloading accuracy metric. We can see that DOS scheme always performs better than the coarse-grained offloading schemes (i.e. TOS and NOS). In addition, we observe that the distance d has little

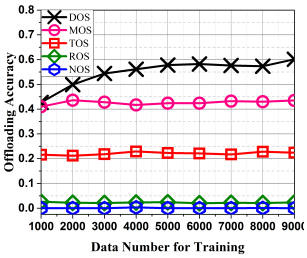

Fig. 4: Offloading accuracy vs. K_{te}.

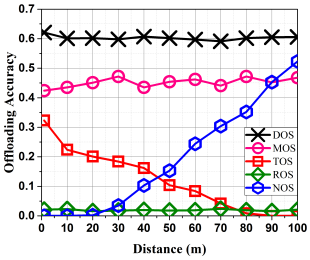

Fig. 5: Offloading accuracy vs. distance d.

influence on our DOS scheme. This means that our DOS scheme supports UE's mobility and remains stable under various network conditions.

VII. Conclusion and Future Work

In this paper, we have studied the offloading problem for a resource intensive applications within MEC networks. We first model the local execution and offloading cost, and proposed our decision making procedure. Then, in order to minimize the overall execution cost, we have formulated the problem as a multi-label classification problem and proposed a deep supervised learning-based algorithm to minimize the system cost. Compared to four alternative solutions, our proposed approach achieves the best performance in offloading accuracy and system cost saving. In particular, we have shown that the overall execution cost can be reduced by at most 49.24%, 23.87%, 15.69%, and 11.18% compared to the NOS, ROS, TOS and MOS schemes, respectively. Future work will address a number of interesting open questions and directions, such as deep learning-based offloading for multi-user multi-cell scenario and mobility management.

Acknowledgment

This work was supported in part by the FUI PODIUM project (Grant no. 15016552). The author shuai Yu would like to thank China Scholarship Council (CSC) for supporting his study in France.

References

[1] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, "Towards wearable cognitive assistance," in *Proc. ACM the 12th annual international conference on Mobile systems, applications, and services, (MobiSys'14)*, Bretton Woods, NH, USA, Jun. 2014, pp. 68–81.

[2] T. Langlotz, D. Wagner, A. Mulloni, and D. Schmalstieg, "Online creation of panoramic augmented reality annotations on mobile phones," *IEEE Pervasive Comput.*, vol. 11, no. 2, pp. 56–63, Feb. 2012.

[3] X. Fu, S. Secci, D. Huang, and R. Jana, "Mobile cloud computing," *IEEE Commun. Mag.*, pp. 61–62, 2015.

[4] ETSI. (2014) Mobile edge computing (mec). [Online]. Available: http://www.etsi.org/technologies-clusters/technologies/mobile-edge-computing

[5] S. Yu, R. Langar, W. Li, and X. Chen, "Coalition-based energy efficient offloading strategy for immersive collaborative applications in femto-cloud," in *Proc. IEEE International Conference on Communications, (ICC'16)*, Kuala Lumpur, Malaysia, May 2016.

[6] S. Yu, R. Langar, and X. Wang, "A d2d-multicast based computation offloading framework for mobile edge computing," in *Proc. IEEE Global Communication Conference, (GLOBECOM'16)*, Washington, DC, USA, Dec. 2016.

[7] H. Eom, P. S. Juste, R. J. O. Figueiredo, O. Tickoo, R. Illikkal, and R. Iyer, "Machine learning-based runtime scheduler for mobile offloading framework," in *IEEE/ACM 6th International Conference on Utility and Cloud Computing, UCC 2013, Dresden, Germany, December 9-12, 2013*, 2013, pp. 17–25.

[8] H. Eom, R. J. O. Figueiredo, H. Cai, Y. Zhang, and G. Huang, "MALMOS: machine learning-based mobile offloading scheduler with online training," in *3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, MobileCloud 2015, San Francisco, CA, USA, March 30 - April 3, 2015*, 2015, pp. 51–60.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems*, ser. NIPS'12. USA: Curran Associates Inc., 2012, pp. 1097–1105. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999134.2999257

[10] Y. Zhang, D. Niyato, and P. Wang, "Offloading in mobile cloudlet systems with intermittent connectivity," *IEEE Trans. Mobile Comput.*, vol. 14, no. 12, pp. 2516–2529, Feb. 2015.

[11] D. Huang, P. Wang, and D. Niyato, "A dynamic offloading algorithm for mobile computing," *IEEE Trans. Wireless Commun.*, vol. 11, no. 6, pp. 1991–1995, Jun. 2012.

[12] L. Yann, B. Yoshua, and G. Hinton, "Deep learning," *Nature*, pp. 436–444, May 2015.

[13] B. G. Ryder, "Constructing the call graph of a program," *IEEE Trans. Softw. Eng.*, vol. SE-5, no. 3, pp. 216–226, May 1979.

[14] T. Verbelen, P. Simoens, F. D. Turck, and B.Dhoedt, "Leveraging cloudlets for immersive collaborative applications," *IEEE Pervasive Comput.*, vol. 12, no. 4, pp. 30–38, Oct. 2013.

[15] A. P. Miettinen and J. K. Nurminen, "Energy efficiency of mobile clients in cloud computing," in *Proc. USENIX the 2nd USENIX conference on Hot topics in cloud computing, (HotCloud'10)*, USENIX Association Berkeley, CA, USA, 2010, pp. 4 – 4.

[16] M. Yang, Y. Wen, J. Cai, and C. H. Foh, "Energy minimization via dynamic voltage scaling for real-time video encoding on mobile devices," in *Proc. IEEE International Conference on Communications, (ICC'12)*, Ottawa, ON, 2012, pp. 2026 – 2031.

[17] L. Lei, Z. Zhong, C. Lin, and X. Shen, "Operator controlled device-to-device communications in lte-advanced networks," *IEEE Wireless Commun.*, vol. 19, no. 3, pp. 96–104, Jun. 2012.

[18] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *Int J Data Warehousing and Mining*, vol. 2007, pp. 1–13, 2007.

[19] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, 2010, pp. 807–814. [Online]. Available: http://www.icml2010.org/papers/432.pdf

[20] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[21] J. Nam, J. Kim, E. Loza Mencía, I. Gurevych, and J. Fürnkranz, *Large-Scale Multi-label Text Classification — Revisiting Neural Networks*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 437–452. [Online]. Available: https://doi.org/10.1007/978-3-662-44851-9_28

[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization." *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1412.html#KingmaB14

[23] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Trans. Wireless Commun.*, vol. 16, no. 3, pp. 1397–1411, Dec. 2016.