EURASIP Journal on Wireless Communications and Networking

RESEARCH Open Access

Deep learning-based computation offloading with energy and performance optimization

Yongsheng Gong[1,2,3], Congmin Lv[1,2], Suzhi Cao[1,2*], Lei Yan[1,2] and Houpeng Wang[1,2,3]

Abstract
With the benefit of partially or entirely offloading computations to a nearby server, mobile edge computing gives user equipment (UE) more powerful capability to run computationally intensive applications. However, a critical challenge emerged: how to select the optimal set of components to offload considering the UE performance as well as its battery usage constraints. In this paper, we propose a novel energy and performance efficient deep learning based offloading algorithm. The optimal offloading schemes of components based on remaining energy and its performance can be determined by our proposed algorithm. All of these considerations are modeled as a cost function; then, a deep learning network is trained to compute the solution by which the optimal offloading scheme can be determined. Experimental results show that the proposed method is superior to existing methods in terms of energy and performance constraints.

Keywords: Computation offloading, Deep learning, Mobile edge computing, Energy and performance optimization

1 Introduction

With the development and popularity of smart terminals referred to as user equipment (UE), various network services and applications continue to emerge. Although UEs have experienced a tremendous increase in computational power over the years, it still cannot process intensive computation and huge data in a short time [1–3], for which cloud computing used to be a solution. However, the delay caused by the communication between the UE and the cloud server poses a severe challenge to the feasibility of this typical solution [4]. The European Telecommunications Standards Institute proposed placing small edge servers near end users to reduce network latency, and studied them as mobile edge computing (MEC) [5–7].

MEC refers to deploying computing and storage resources at the edge of mobile networks to provide IT service environments and cloud computing capabilities for mobile networks, thereby providing users with ultra-low latency and high-bandwidth network service solutions. As one of the key technologies, computation offloading [8, 9] refers to the technology in which UEs hand over part or all of their computing tasks to the cloud computing environment to address the shortcomings of mobile devices in terms of resource storage, computation performance, and energy efficiency.

Affected by the way of thinking in cloud computing, the existing offloading solutions generally have the following problems: (1) assuming that the server has unlimited computing power, (2) assuming that users have constant uplink and downlink network conditions, and (3) ignoring different user priorities caused by different energy and network conditions [10–12].

In this paper, we propose a novel energy and performance efficient deep learning based offloading algorithm (EPED), which partially offload computations from UE to MEC under a comprehensive optimization of UE's energy consumption and performance. Based on the concept of

*Correspondence: caosuzhi@csu.ac.cn
[1]Technology and Engineering Center for Space Utilization, Chinese Academy of Sciences, Beijing 100094, China
[2]Key Laboratory of Space Utilization, Chinese Academy of Sciences, Beijing 100094, China
Full list of author information is available at the end of the article

component which refers to each computation step and related data, in this paper, performance is measured by component execution time and energy consumption is also accurately measured by component workload. For each component, there are two choices to deploy it, i.e., either deploy it locally or remotely. We design a cost function for each deployment method, comprehensively considering the performance and energy consumption [13]. Then, the overall cost of computation offloading can be measured by all components, and finally, the best offloading scheme is determined by a deep learning method under the constraint of the smallest overall cost.

The proposed method can adaptively select the optimal combinations of application components to offload, with the smallest cost of execution time and energy consumption. The following summarized our contributions:

1. We split an application into multiple components and designed a mathematical model of cost function for each component, considering energy consumption, execution time, and server-side resource consumption.
2. Based on the cost function of each component, we proposed a mathematical model of the cost function of final offloading scheme, comprehensively evaluating the overall cost of all components. The cost function of final offloading scheme is not designed as the simple linear addition of all components but the interactions and connections between adjacent components, by which, the cost of offloading scheme can be accurately evaluated.
3. Based on the cost function of final offloading scheme, we designed a mathematical model of parameter constraint which can bring the smallest cost.
4. We proposed a supervised deep neural network (DNN) to calculate the parameters of the cost function of final offloading scheme. To train this DNN, the most important problem is how to design an appropriate training dataset, and hence, we also proposed a method of getting the training dataset in our experiments.

The rest of this paper is organized as follows. Section 2 presents our energy and performance efficient deep learning based offloading algorithm. Section 3 describes our experiments, comparison with other methods, and how to prepare the training dataset. Section 4 discusses some related work, and Section 5 concludes the paper.

2 Proposed EPED

The execution process of an application can be divided into several steps. Each of these steps as well as the related data is called a component of the application execution. The component can be either deployed on the local side (UE) or mobile edge server side (MES). An efficient offloading approach should select an optimal part of components to offload to MES but not the whole, aiming to which, EPED is proposed as the following 5 key steps: (1) determines the costs of deploying a component on local side and MES side respectively; (2) designs the cost function formula of offloading scheme, wherein the cost is the dependent variable of offloading decision; (3) searches the best offloading schemes for some specific component states with exhaustive method and (4) the best offloading schemes as well as their component states respectively, and these two parts are then designed as the outputs and inputs of our training dataset; (5) and finally, using a deep neural network, we can get the best offload scheme of any component states from the training dataset.

2.1 Local side execution cost

The local side execution cost consists of energy consumption and execution time. Orsini et al. [14] proposed that the execution time can be evaluated by the input data amount needed for a component. But this method ignored that the input data and the processed data were not equal in amount. If we assume that the output of a component is the input of the next component, then we use $d_{n-1,n}$ to denote the input data amount of component c_n as well as the output data amount of component c_{n-1} . Then, the workload of component c is denoted as :

$$W_c = V \cdot O_n \cdot d_{c-1,c} \tag{1}$$

where W_c is measured in CPU clock cycles and V denotes the number of clock cycles a processor will perform per byte and is measured in cycles per byte. Yang et al. [15] presented the study of this value. O_n is the computational complexity of c_n and represents the data amplification factor of c_n. It is obvious that the input data and the processed data were not equal in amount since the input data may be processed several times by a component; this is why we introduce the denotation of O_n.

Now, if the component c is deployed and run on the local UE side, its execution time is equal to the time to complete the workload W_c, which is given by :

$$T_l(c) = \frac{W_c}{f_l} \tag{2}$$

where f_l is the CPU rate of UE, which is measured in million instructions per second (MIPS).

Let the energy consumption due to this workload be E_c and is given by:

$$E_c = U \cdot W_c \tag{3}$$

where U is the unit power consumption of UE and is measured in MAH per CPU cycle.

If the total energy of the UE is E_t, then the remaining energy for the next component $c+1$ is given by:

$$E_r = E_t - U \cdot W_c \tag{4}$$

After the execution time and energy consumption were determined by formula (2) and (4) respectively, the local side execution cost of component c can be evaluated by:

$$F_l(c) = \gamma_1 T_l(c) + \gamma_2 E_c \tag{5}$$

where γ_1 and γ_2 are weighting coefficients which can balance the contribution of time delay and energy consumption in the local cost function respectively.

2.2 MES side execution cost

Except local side execution, UE can also offload a component to remote side, i.e., MES to execute. Like the local side, the execution cost of the MES side also includes the execution time while this time is much shorter than the local side. We can represent this time similarly as (2) by:

$$T_r(c) = \frac{W_c}{f_r} \tag{6}$$

where f_r is the CPU rate of MES.

The time spent on transfer data from UE to MES should also be considered. This time depends on the mobile internet environment of UE, and this paper only considers the most commonly used 4G environment. 4G communication is implemented by the orthogonal frequency division multiple access (OFDMA) technology. With such technology, the upload and download speed depends on the bandwidth B and the transmission subcarrier number N.

Assuming the same additive white Gaussian noise (AWGN) channel in transmission for uplink and downlink, the maximum achievable uplink and downlink data rate can be easily derived as [16]:

$$r_u = n\frac{B}{N}\log_2(1 + \frac{p_u|h_{ul}|^2}{\Gamma(g_{ul})d^\beta N_o}) \tag{7}$$

$$r_d = n\frac{B}{N}\log_2(1 + \frac{p_s|h_{dl}|^2}{\Gamma(g_{dl})d^\beta N_o}) \tag{8}$$

where B is the bandwidth, β is the path loss exponent, d is the distance between UE and MES, n is the number of subcarriers that will be allocated for transmission from UE to MES, N_o is the noise power, p_u and p_s refer to the transmit power of UE and MES respectively, h_{ul} and h_{dl} are the channel fading coefficient for uplink and downlink respectively, and g_{ul} and g_{dl} are the required bit error rate for uplink and downlink respectively. $\Gamma(g_{ul}) = \frac{-2\log 5 g_{ul}}{3}$ is the SNR margin to satisfy the required bit error rate with quadrature amplitude modulation constellation.

Using (7) and (8), the time spent for UE to send the input data of component c to MES can be derived as:

$$T_s(c) = (1 - p_{c-1})\frac{d_{c-1,c}}{r_u} \tag{9}$$

where p_{c-1} is the offloading decision of the previous component $c-1$. $p_{c-1} = 0$ if component $c-1$ was executed locally and $p_{c-1} = 1$ if component $c-1$ was executed on MES. If the previous component $c-1$ was executed on MES, then its output, i.e., the input of component c need not be transmitted between UE and MES, and hence, the time spent is zero, while the previous component $c-1$ was executed locally, then the data transmission will be actually needed.

Similarly, after the execution of component c, its output should be sent back to UE if the next component $c+1$ will be executed locally, while the transmission is unnecessary if component $c+1$ will be executed on MES; the time spent for UE to accept the output data of component c from MES is derived as:

$$T_a(c) = (1 - p_{c+1})\frac{d_{c,c+1}}{r_u} \tag{10}$$

Finally, the MES side execution cost is derived as:

$$F_r(c) = \gamma_3 T_s(c) + \gamma_4 T_r(c) + \gamma_5 T_a(c) \tag{11}$$

where γ_3, γ_4, and γ_5 are weighting coefficients which can balance the contribution of these three types of time respectively.

2.3 Cost function

We have discussed that a component can be either executed locally or remotely, for which the cost function is shown as (5) and (11) respectively. To derive the cost function of offloading scheme conveniently, we represent the cost of a single component c as:

$$F_c = \begin{cases} F_l(c), \ p_c = 1 \\ F_r(c), \ p_c = 0 \end{cases} \tag{12}$$

The cost function of offloading scheme is the sum evaluation of all components and hence can be represented as:

$$F = \sum F(c) \tag{13}$$

Let the offloading decisions of all components compose the decision space $P = p_1, p_2, \cdots, p_M$, where M is the number of the components. Then, the goal of EPED is to find a special decision space P^* to minimize (13), which can be represented as:

$$P^* = \arg_P \min \sum F(c) \tag{14}$$

2.4 Algorithm implementation

To determine the optimal offloading scheme shown as (14), a DNN structure is employed in this paper. For training this DNN, the most important thing is preparing the training dataset. Our training dataset was got by the following steps:

1. A component has a state (c, v, b, d) representing the mobile environment of this component, wherein c is the component number, v is the input data amount, b is the bandwidth, and d is the distance between UE and MES.

2. The offloading decision of a component depends on its state; therefore, the states of components and the corresponding offloading decisions should be the inputs and outputs of the DNN, respectively.
3. Supposing there are M components in total, and we randomly generate a state for each component, hence, we finally get M different states. Since each component has two offloading choices, there are 2^M different offloading schemes. Using the exhaustive method, we calculated the cost of each scheme and selected the best which can minimize (13).
4. Then, if the DNN is trained well, if we input the above M states, it should output a best offloading scheme.
5. By repeatedly doing step 3 for S times, we can get different states and the corresponding best offloading schemes, which compose S rows training data. The ith row of the training data is denoted as:

$$\text{sample}_i = \{I_i, P_i^*\} \tag{15}$$

where I_i is the state of all M components. Therefore, I_i consists of $4M$ data items since each state has 4 state items. The neuron number of the input layer is also $4M$ to accept I_i accurately. P_i^* is the desired optimal offloading scheme of M components.

$$I_i = \{(c_{i,1}, v_{i,1}, b_{i,1}, d_{i,1}), \cdots, (c_{i,M}, v_{i,M}, b_{i,M}, d_{i,M})\} \tag{16}$$

$$P_i^* = (p_{i,1}^*, p_{i,2}^*, \cdots, p_{i,M}^*) \tag{17}$$

For example, $(c_{i,1}, v_{i,1}, b_{i,1}, d_{i,1})$ is the state of component 1 which is randomly generated in the ith pass in step 5, and $(p_{i,1}, p_{i,2}, \cdots, p_{i,m})$ is the offloading scheme corresponding this pass.

The DNN is designed as Fig. 1, which is a fully connected neural network, but we did not show the real connections between different layers since there are too many neurons. When we input a training record into DNN, a state of a component will be accepted by 4 adjacent neurons. Since there are M components, the number of input nodes is $4M$, and the output layer has M nodes each represents the offloading decision of the corresponding components.

The training dataset is prepared from a limited number of states, but the well-trained DNN can predict the optimal offloading scheme of any combination of states. Although the proposed DNN is designed for fixed number of components, we can introduce a large M to satisfy different scenarios; then, the DNN is competent for any scenarios less than M s. For example, if $M = 10$ and the actual component number is 8, we can just let the input and output of $\text{sample}_i = 0$ if $i > 8$.

The EPED algorithm is summarized as follows:

3 Experiments

3.1 Experiment setup

We implemented the experiments on a workstation with a 32-core CPU and 1 TB RAM. We set M = 100 and randomly generated 10,000 states for each component; then, we get a dataset $\{\text{sample}_i = \{I_i, P_i^*\} | i = 1, 2, \ldots, 10,000\}$. We employed different sparsity of this dataset as our training data, i.e., 10, 20, 50, 100, 200, 500, 1000, and 2000 respectively; samples were selected to train the DNN.

We first verified the accuracy of EPED under different sparsity and then compared the predictive performance of our EPED with other 2 types of representative methods. The 2 compared types of methods are:

1. Total offloading scheme (TOS) [16]: TOS is a coarse grained approach. It makes no decision but selects all the components to offload from UE to MES. No

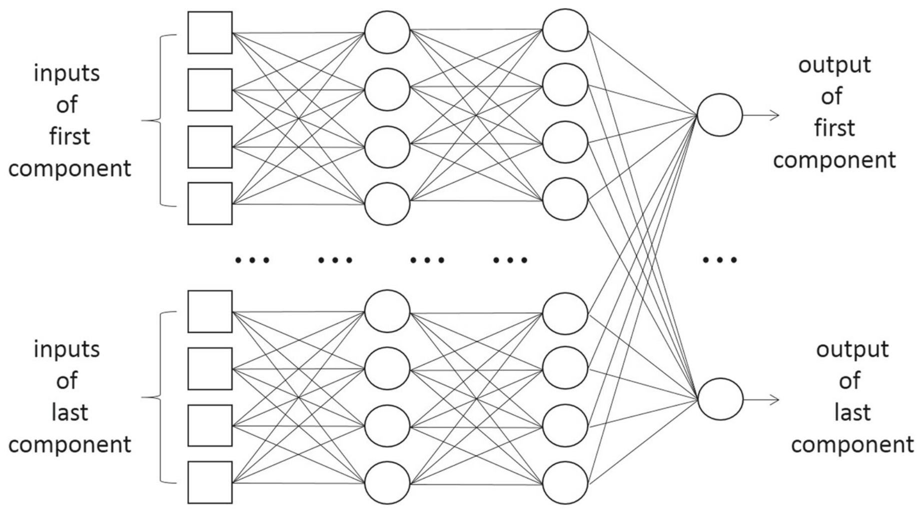

Fig. 1 Proposed deep neural network

Algorithm 1 component matrices computing

```
Input: training dataset$\{sample_i = \{I_i, P_i^*\} | i = 1, 2, \ldots, S\}$
Output: the optimal offloading scheme for any states combination
  Initialize the DNN with random weights
  repeat
    for each $sample_i (1 \leq i \leq S)$ do
      Input $I_i$ and get the corresponding output $P_i$
      Calculate the sum of squared error between $P_i$ and $P_i^*$
      Using back propagation adjust all the weights of the DNN.
    end for
  until convergence
  Input any required states for optimal offloading scheme satisfied:
    $P^* = arg_P min \sum F(c)$
```

components will be executed on UE via this method; therefore, TOS seems to be able to save energy of UEs. However, this method needs a lot of data transmission, which also requires energy consumption. Therefore, we select this method to make a comparison.

2. Random offloading scheme (ROS) [16]: ROS performs offloading by a simple strategy, just randomly select some components to offload.

For the convenience of comparison, we proposed the predictive accuracy as follows:

$$\mathrm{MAE} = \frac{\sum |p_{i,m} - p_{i,m}^*|}{N} \tag{18}$$

$$\mathrm{RMSE} = \sqrt{\frac{\sum (p_{i,m} - p_{i,m}^*)^2}{N}} \tag{19}$$

where $p_{i,m}^*$ is the prediction of offloading scheme of a component, $p_{i,m}$ is its real best offloading scheme, and N is the number of all components. Step 3 of Section 2.4 has shown how to get the real best offloading scheme.

3.2 Experiment implementation

Figure 2 shows the prediction accuracies of EPED under different number is S. Figure 2 indicates that the prediction accuracy of EPDE improves with the increment of the sample number. When the sample number is ≥ 50, the MAE and RMSE are all less than 0.5, which indicates that EPED can make an accurate prediction, since $0 \leq p_{i,m} \leq 1$. However, when the sample number exceeds 1000, the prediction accuracy declines quickly. It means that the performance of EPED is not linear to the sparsity.

We compared EPED between TOS and ROS for accuracy rate and cost consumption. The accuracy rate is defined as:

$$R = \frac{N_p}{M} \tag{20}$$

where N_p is the number of accurately offloaded components.

Figure 3 compared EPED between TOS and ROS for accuracy rate. The accuracy rate is obtained from training dataset of sample number = 100, sample number = 500, and sample number = 1000 respectively. We have verified that the prediction accuracy of EPDE improves with the increment of the sample number. Therefore, the DNN can make more accurate offloading scheme with a large sample number. Figure 3 also indicates that the performance of our proposed EPED is competent for most scenarios, while the other two methods are hard to improve. It is easy to understand, the other two methods select total or random offloading, the more sample number, the more errors they will make.

Figure 4 compared EPED between TOS and ROS for accuracy cost consumption under different sample numbers. If compared to other methods, EPED will run the application with the smallest cost. Another important thing is that EPED has the minimum slope of the curve, which indicates that when the prediction scene becomes complex, EPED can have a relatively small decrease of offloading performance.

4 Related work

To improve the performance of mobile device offloading to utilize the benefits of Clouds, many attempts have been made by researchers. Some studies have focused on methods on how to effectively offload tasks to MES with the minimal energy and time cost. This section introduces several representative works.

To maximize the potential of energy savings, MAUI [17] minimizes the burden on programmers by combining the reflection of programming, portability of code, network costs, serialization, and type safety. In MAUI, applications are prebuilt and executed on HTC smartphones, and the MAUI server uses a dual core desktop running Windows 7 with the v3.5 .NET Framework. Using MAUI, the mobile game components can be offloaded to a remote cloud server and save energy for two types of games. If running computer games, 27% of the energy consumption can be saved, and 45% of the energy consumption can be saved if running chess games. CloneCloud [18] offload the calculation task of the resource-intensive components of a mobile application to a more powerful clone which is created on a cloud. It periodically or on-demand synchronizes all tasks of mobile devices to adjust the current offloading scheme. The advancement of CloneCloud is

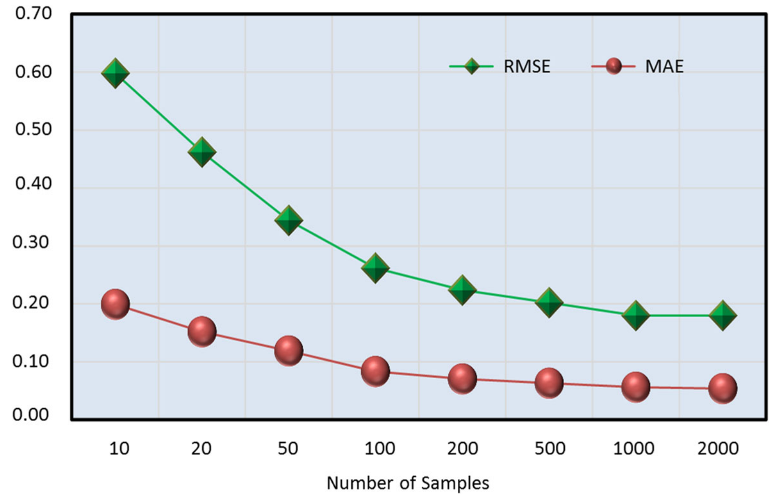

Fig. 2 Different prediction accuracy of EPED under different training dataset sparsity

that an offloaded task can be even partitioned into pieces and select some pieces to run locally while another part of pieces runs on the server side. A related weakness of CloneCloud is that if native resources are not virtualized or are inaccessible for clone, then CloneCloud cannot virtualize such type of resources. It can co-work with thread-granularity migration to improve performance. The difference between MAUI and CloneCloud is that the former only can save energy consumption of mobile applications through automatically offloading, while CloneCloud can minimize either execution time or energy consumption of applications by adaptively determining computation-intensive components. In mobile network, offloading demands usually encountered the inaccessible cloud computing resources, which may bring failure to offloading scheme. COSMOS [19] aims to solve this problem by the risk-based offloading idea. COSMOS can make risk-based offloading strategies to decrease the uncertainties caused by variable network environments. COSMOS offloads task from local to server with the attempt of little energy consumption and the rental fees of cloud resources.

A big trouble of the mobile network environment offloading is the high WAN latency caused by an unstable network environment or a long distance between user equipment and servers. Cloudlets [20] puts forward a clever idea to find the cloud resources close to the users. The distance between user equipment and such cloud resources is only one router hop. The infrastructure of cloudlets is decentralized and widely dispersed. The infrastructure of cloudlets is self-managing and low energy consumption. In a word, the cloudlet is a pre-defined cloud in consisting of several static stations and is generally established in public domains. However, cloudlets cannot guarantee the availability for a nearby

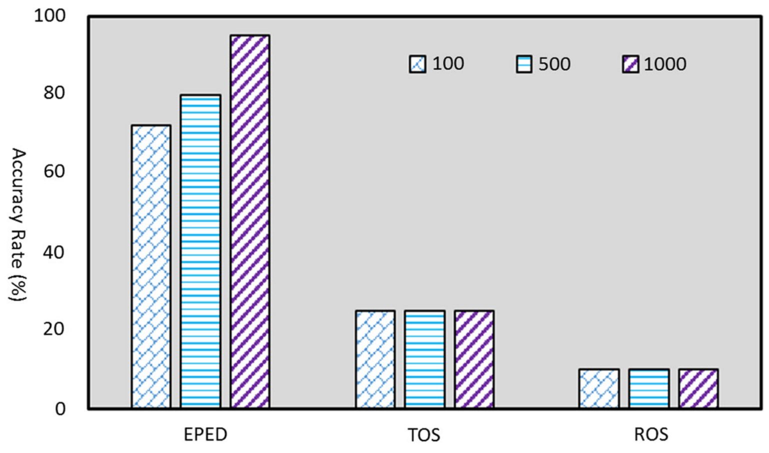

Fig. 3 Comparison of offloading accuracy rates of TOS, ROS, and EPED

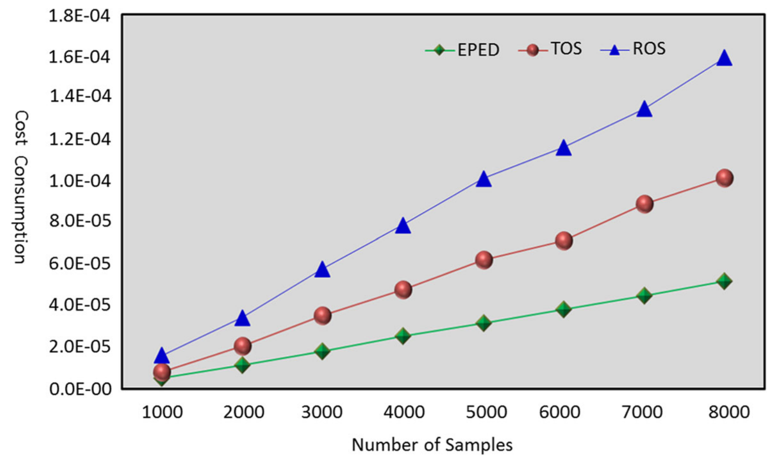

Fig. 4 Comparison of overall cost of TOS, ROS, and EPED

mobile device. Habak et al. [21] considers how to organize a group of colocated devices to provide a cloud service as edge servers. The proposed architecture of femtocloud provides a dynamic, self-configuring mobile cloud which can serve a cluster of mobile devices. The system of femtocloud mainly contains two parts: one is a highly stable and well-configured controller and the other is many mobile and unpredictable devices, i.e., a compute cluster to perform the computation. Spontaneous proximity cloud (SPC) [22] lets a set of neighboring mobile devices work in a collaborative way and decreases the high latency between the user equipment and the cloud server by the data sharing between different users.

5 Conclusion

This paper proposed a new method EPED that can adaptively select the optimal combinations of components to offload, with the smallest cost of execution time and energy consumption. EPED splits an application into multiple components and designs a mathematical model of the cost function for each component, considering energy consumption, execution time, and server-side resource consumption. Based on the cost function of each component, we proposed a mathematical model of cost function of final offloading scheme, comprehensively evaluating the overall cost of all components. The cost function of final offloading scheme is not designed as the simple linear addition of all components but the interactions and connections between adjacent components, by which the cost of offloading scheme can be accurately evaluated. Based on the cost function of final offloading scheme, we designed a mathematical model of parameter constraint which can bring the smallest cost. We proposed a supervised deep neural network (DNN) to calculate the parameters of the cost function of final offloading scheme. We also proposed a method of getting a training dataset in our experiments.

Abbreviations

UE: User equipment; MEC: Mobile edge computing; IT: Information technology; EPED: A novel energy and performance efficient deep learning-based offloading algorithm; DNN: Deep neural network; MES: Mobile edge server; CPU: Central processing unit; MIPS: Million instructions per second; OFDMA: Orthogonal frequency division multiple access; AWGN: Additive white Gaussian noise; SNR: Signal noise ratio; TOS: Total offloading scheme; ROS: Random offloading scheme; RMSE: Root mean squard error; MAE: Mean absolute error; SPC: Spontaneous proximity cloud

Acknowledgements

The authors are grateful to the editor and anonymous referees for their valuable comments and suggestions. Only the authors are responsible for the views expressed and mistakes made.

Authors' contributions

YG and CL developed the idea for the study, performed the experiment, and drafted the manuscript. SC and LY participated in its design and coordination. SC, LY, and HW reviewed and edited the manuscript. The authors read and approved the final manuscript.

Authors' information

Yongsheng Gong is an associate research fellow of the Technology and Engineering Center for Space Utilization, Chinese Academy of Sciences. He received his bachelor's degree from Tsinghua University in 2006 and a master's degree from the University of Chinese Academy of Sciences in 2009. Email: gys@csu.ac.cn
Congmin LV received the Ph.D. degree in space physics from University of Chinese Academy of Sciences, Beijing, China, in 2004. He is a professor in Technology and Engineering Center for Space Utilization, Chinese Academy of Sciences, and acts as the vice chief designer for the space application system of the manned space flight project.
Suzhi Cao received the PH.D. degree in computer application technology from Academy of Opto-Electronics (AOE), Chinese Academy of Science (CAS), Beijing, China.
Dr. Cao is the associate professor of technology and engineering center for space utilization, Chinese Academy of Sciences, Beijing, China.
Lei Yan received the M.S. degree in 2004 form University of Chinese Academy of Science. She is the associate professor of Technology and Engineering Center for Space Utilization.
Houpeng Wang received the bachelor's degree in information security from Beijing University of Posts and Telecommunications in 2019. He is currently studying for the master degree at University of Chinese Academy of Sciences and at Technology and Engineering Center for Space Utilization, Chinese Academy of Sciences.

Funding

This work was supported by the Research Fund of the manned space engineering (no.18022010301) and the Open Fund of IPOC (BUPT).

Availability of data and materials
The data has been gathered from research papers and articles that are mentioned in the references.

Competing interests
The authors declare that they have no competing interests.

Author details
[1]Technology and Engineering Center for Space Utilization, Chinese Academy of Sciences, Beijing 100094, China. [2]Key Laboratory of Space Utilization, Chinese Academy of Sciences, Beijing 100094, China. [3]University of Chinese Academy of Sciences, Beijing 100049, China.

Received: 16 January 2020 Accepted: 6 March 2020
Published online: 30 March 2020

References

1. H. T. Dinh, C. Lee, D. Niyato, P. Wang, A survey of mobile cloud computing: architecture, applications, and approaches. Wirel. Commun. Mob. Comput. **13**(18), 1587–1611 (2013). https://doi.org/10.1002/wcm.1203
2. A. U. R. Khan, M. Othman, S. A. Madani, S. U. Khan, A survey of mobile cloud computing application models. IEEE Commun. Surv. Tutor. **16**(1), 393–413 (2014). https://doi.org/10.1109/surv.2013.062613.00160
3. Y. Wang, I.-R. Chen, D.-C. Wang, A survey of mobile cloud computing applications: perspectives and challenges. Wirel. Pers. Commun. **80**(4), 1607–1623 (2015). https://doi.org/10.1007/s11277-014-2102-7
4. P. Mach, Z. Becvar, Mobile edge computing: a survey on architecture and computation offloading. IEEE Commun. Surv. Tutor. **19**(3), 1628–1656 (2017). https://doi.org/10.1109/comst.2017.2682318
5. J. Xu, S. Wang, B. K. Bhargava, F. Yang, A blockchain-enabled trustless crowd-intelligence ecosystem on mobile edge computing. Ieee Trans. Ind. Inf. **15**(6), 3538–3547 (2019). https://doi.org/10.1109/tii.2019.2896965
6. H. Flores, P. Hui, S. Tarkoma, Y. Li, S. Srirama, R. Buyya, Mobile code offloading: from concept to practice and beyond. IEEE Commun. Mag. **53**(3), 80–88 (2015). https://doi.org/10.1109/mcom.2015.7060486
7. L. Jiao, R. Friedman, X. Fu, S. Secci, Z. Smoreda, H. Tschofenig, in *2013 Future Network & Mobile Summit*. Cloud-based computation offloading for mobile devices: state of the art, challenges and opportunities. (IEEE), pp. 1–11. https://ieeexplore.ieee.org/document/6633526
8. S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, W. Wang, A survey on mobile edge networks: convergence of computing, caching and communications. IEEE Access. **5**, 6757–6779 (2017). https://doi.org/10.1109/access.2017.2685434
9. Y. Mao, C. You, J. Zhang, K. Huang, K. B. Letaief, A survey on mobile edge computing: the communication perspective. IEEE Commun. Surv. Tutor. **19**(4), 2322–2358 (2017). https://doi.org/10.1109/comst.2017.2745201
10. T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, D. Sabella, On multi-access edge computing: a survey of the emerging 5G network edge cloud architecture and orchestration. IEEE Commun. Surv. Tutor. **19**(3), 1657–1681 (2017). https://doi.org/10.1109/comst.2017.2705720
11. Y. Yu, Mobile edge computing towards 5G: vision, recent progress, and open challenges. China Commun. **13**(2), 89–99 (2016)
12. Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, V. Young, Mobile edge computing—a key technology towards 5G. ETSI White Pap. **11**(11), 1–16 (2015)
13. J. Xu, S. Wang, N. Zhang, F. Yang, X. Shen, Reward or penalty: aligning incentives of stakeholders in crowdsourcing. IEEE Trans. Mob. Comput. **18**(4), 974–985 (2019). https://doi.org/10.1109/TMC.2018.2847350
14. G. Orsini, D. Bade, W. Lamersdorf, *CloudAware: A Context-adaptive Middleware for Mobile Edge and Cloud Computing Applications*, (2016), pp. 216–221. https://doi.org/10.1109/fas-w.2016.54
15. M. Yang, Y. Wen, J. Cai, C. H. Foh, *Energy Minimization via Dynamic Voltage Scaling for Real-Time Video Encoding on Mobile Devices*, (2012), pp. 2026–2031. https://doi.org/10.1109/icc.2012.6364132
16. X. Ran, H. Chen, Z. Liu, J. Chen, *Delivering Deep Learning to Mobile Devices Via Offloading*, (2017), pp. 42–47. https://doi.org/10.1145/3097895.3097903
17. E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, P. Bahl, *Maui: making smartphones last longer with code offload*. (ACM), pp. 49–62. https://doi.org/10.1145/1814433.1814441
18. B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, A. Patti, in *Proceedings of the Sixth Conference on Computer Systems*. Clonecloud: elastic execution between mobile device and cloud. (ACM), pp. 301–341. https://doi.org/10.1145/1966445.1966473
19. C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik, E. Zegura, in *Proceedings of the 15th ACM International Symposium on Mobile Ad Hoc Networking and Computing*. Cosmos: computation offloading as a service for mobile devices. (ACM), pp. 287–296. https://doi.org/10.1145/2632951.2632958
20. M. Satyanarayanan, P. Bahl, R. Caceres, N. Davies, The case for vm-based cloudlets in mobile computing. IEEE Pervasive Comput. **8**(4), 14–23 (2009). https://doi.org/10.1109/mprv.2009.82
21. K. Habak, M. Ammar, K. A. Harras, E. Zegura, *FemtoClouds: Leveraging Mobile Devices to Provide Cloud Service at the Edge*, (2015), pp. 9–16. https://doi.org/10.1109/cloud.2015.12
22. R. Golchay, F. Le Mouel, J. Ponge, N. Stouls, in *Lecture Notes of the Institute for Computer Sciences Social Informatics and Telecommunications Engineering, volume 201*. Spontaneous Proximity Clouds: Making Mobile Devices to Collaborate for Resource and Data Sharing, (2017), pp. 480–489. https://doi.org/10.1007/978-3-319-59288-6_45

Publisher's Note