Accelerating Deep Learning Inference via Model Parallelism and Partial Computation Offloading

Huan Zhou, *Member, IEEE*, Mingze Li, Ning Wang, *Member, IEEE*, Geyong Min, *Member, IEEE*, and Jie Wu, *Fellow, IEEE*

Abstract—With the rapid development of Internet-of-Things (IoT) and the explosive advance of deep learning, there is an urgent need to enable deep learning inference on IoT devices in Mobile Edge Computing (MEC). To address the computation limitation of IoT devices in processing complex Deep Neural Networks (DNNs), computation offloading is proposed as a promising approach. Recently, partial computation offloading is developed to dynamically adjust task assignment strategy in different channel conditions for better performance. In this paper, we take advantage of intrinsic DNN computation characteristics and propose a novel Fused-Layer-based (FL-based) DNN model parallelism method to accelerate inference. The key idea is that a DNN layer can be converted to several smaller layers in order to increase partial computation offloading flexibility, and thus further create the better computation offloading solution. However, there is a trade-off between computation offloading flexibility as well as model parallelism overhead. Then, we investigate the optimal DNN model parallelism and the corresponding scheduling and offloading strategies in partial computation offloading. In particular, we propose a Particle Swarm Optimization with Minimizing Waiting (PSOMW) method, which explores and updates the FL strategy, path scheduling strategy, and path offloading strategy to reduce time complexity and avoid invalid solutions. Finally, we validate the effectiveness of the proposed method in commonly used DNNs. The results show that the proposed method can reduce the DNN inference time by an average of 12.75 times compared to the legacy No FL (NFL) algorithm, and is very close to the optimal solution achieved by the Brute Force (BF) algorithm with the difference of less than 0.04%.

Index Terms—Mobile edge computing, fused-layer, DNN inference, partial offloading, model parallelism

1 Introduction

With the popularity of mobile devices and the advance of wireless access technique, the booming mobile applications have led to the explosive growth of data traffic [1]. According to the International Data Corporation's report [2], the global data center traffic will reach 163 zettabytes by 2025, and more than 75% of the data will be processed at the edge of the network. Deep learning has shown success in addressing complex tasks [3], [4], including computer vision [5], natural language processing [6], [7], machine translation [8] and many other tasks. One of the obstacles in using deep learning in the Internet-of-Things (IoT) systems is that IoT devices cannot provide real-time and high-precision results simultaneously due to their computation resource limitation [9], [10]. However, many IoT systems, such as traffic monitoring, need not only high processing speed, but also high precision.

To address the aforementioned challenges, Mobile Edge Computing (MEC) has been proposed recently[11], [12], [13]. MEC pushes computation [14], [15], caching [16], [17], [18], [19], and networking functions towards the network edges to perform task processing and provide services, avoiding unnecessary transmission delay [20]. However, MEC introduces additional transmission overhead and delay caused by the communication between the Edge Server (ES) and end devices, which may not be ignored due to large amount of data (e.g., video) and slow transmission speed [21]. For example, when the WiFi transmission rate is 10 MB/s, it takes about 0.6 seconds to transmit a 6 MB picture.

To reduce the inference time of Deep Neural Networks (DNNs) , recent studies have explored partial computation offloading, in which part of the task is computed on IoT end device, and the rest is offloaded to the ES [22]. In partial computation offloading, a DNN model is decomposed into layer-level subtasks and is simultaneously processed in the IoT end device and the ES based on the corresponding processing dependencies. There are three stages in the DNN partial computation offloading. In the first stage, the end device partially processes a DNN model into an intermediate feature layer. In the second stage, the intermediate result obtained from the computation of the end device is transferred to the ES. In the third stage, the ES continues to process the DNN model and obtains the final inference results.

- *Huan Zhou and Mingze Li are with the Hubei Key Laboratory of Intelligent Vision Based Monitoring for Hydroelectric Engineering, the College of Computer and Information Technology, China Three Gorges University, Yichang, Hubei 443002, China. E-mail: zhouhuan117@gmail.com, limingze927@163.com.*
- *Ning Wang is with the Department of Computer Science, Rowan University, Glassboro, NJ 08028 USA. E-mail: wangn@rowan.edu.*
- *Geyong Min is with the Department of Computer Science, University of Exeter, EX4 4PY Exeter, U.K. E-mail: G.Min@exeter.ac.uk.*
- *Jie Wu is with the Center for Networked Computing, Temple University, Philadelphia, PA 19122 USA. E-mail: jiewu@temple.edu.*

Manuscript received 3 January 2022; revised 20 September 2022; accepted 30 October 2022. Date of publication 16 November 2022; date of current version 19 December 2022.
This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grants 62172255 and 61872221, and in part by EU Horizon 2020 Research and Innovation Programme under the Marie Sklodowska-Curie under Grant 101008297.
(Corresponding authors: Huan Zhou and Ning Wang.)
Recommended for acceptance by S. Pallickara.
Digital Object Identifier no. 10.1109/TPDS.2022.3222509

1045-9219 © 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Temple University. Downloaded on February 27,2023 at 19:08:26 UTC from IEEE Xplore. Restrictions apply.

The rationale of offloading the intermediate feature layer's data instead of the raw data is that the intermediate DNN layers have much smaller data sizes and thus lower transmission time [23], [24]. Furthermore, it also enables DNN parallel processing [22], [25]. Therefore, partial computation offloading can significantly reduce the DNN inference time.

Some studies on partial computation offloading have been proposed to offload the entire intermediate DNN layer to ES [23], [24]. However, current DNNs are not optimized for parallel processing and do not support dividing the intermediate DNN layer into multiple layers in the MEC environment. Therefore, the advantage of partial computation offloading is not maximized. In this paper, we first leverage the Fused-Layer (FL) technique to convert a single sequence of DNN layers into multiple sequences, called paths, where each FL path consists of a sequence of small layers without modifying the inference results [26]. As a consequence, we create more flexibility in partial computation offloading by scheduling small layers. However, the FL technique also brings grand challenges including: (1) Determining the optimal FL strategy is difficult due to the trade-off between parallelism computation offloading flexibility and model parallelism overhead; (2) The FL technique leads to a more complicated DNN architecture, which is abstracted as a Directed Acyclic Graph (DAG), and thus it is non-trivial to determine the optimal path offloading strategy and path scheduling strategy.

To tackle the aforementioned challenges, we propose an innovative Particle Swarm Optimization with Minimizing Waiting (PSOMW) method, which jointly considers the impact of FL strategy, path scheduling strategy and path offloading strategy. Specifically, a new Minimizing Waiting (MW) algorithm is developed to heuristically determine the path scheduling order and the number of FL paths. Then, PSOMW is designed by combining the Particle Swarm Optimization (PSO) algorithm with MW algorithm to dynamically update the FL path length, intercepted fused layer's size and path offloading strategy in order to explore the optimal solution and avoid converging at a local minimum.

The major contributions of this paper are summarized as follows:

- To the best of our knowledge, this work is the first of its kind to accelerate partial DNN inference via model parallelism. Parallel computation of DNN by exploiting the FL technique can reduce DNN inference time without accuracy loss.
- We use the MW algorithm to obtain the path scheduling strategy and the number of FL paths. This heuristic algorithm can obtain solutions with low time complexity.
- We propose a novel heuristic method, namely PSOMW, to obtain the approximate optimal FL path length, intercepted fused layer's size and path offloading strategy. PSOMW prevents solutions from falling into local optimum.
- We conduct comprehensive simulation experiments to validate the effectiveness of the proposed method in commonly used DNNs. The results show that the DNN inference time achieved by the proposed method is 12.75 times lower than the existing algorithms without model parallelism.

The rest of this paper is structured as follows. Section 2 describes the work related to computation offloading in MEC. Section 3 introduces the system model, including the problem formulation, and Section 4 presents the proposed method. The simulation results are analysed in Section 5. Finally, Section 6 summarizes this paper.

2 Related Works

In this section, we briefly summarize three DNN computation offloading strategies in IoTs, including (1) end device only computation, (2) full computation offloading, and (3) partial computation offloading.

End Device Only Computation: Kang et al. [27] proposed an adaptive spatiotemporal workload reuse method to maintain the high offloading rate of multiple DNNs in a single adversarial network model. Putic et al. [28] proposed DyHard-DNNs to significantly reduce energy consumption and computation time, in which the accelerator microarchitectural parameters are dynamically reconfigured during the execution of DNN. Guo et al. [29] proposed a simultaneous multimodal architecture (SMA), which provides general-purpose programmability on DNN accelerator to accelerate end-to-end applications. Microsoft and Google developed small-scale DNNs for speech recognition on mobile platforms by sacrificing the high prediction [30]. However, with a reduced number of parameters, the inference accuracy decreases as well.

Full Computation Offloading: Raw data is offloaded to the ES directly in this category. Han et al. [31] proposed alternative DNN models to balance the accuracy and performance/energy. Filter pruning has been recognized as a useful technique to compress and accelerate the DNNs. Fang et al. [32] introduced an alternating direction method of multipliers to prune filters in a layerwise manner, and then accelerate the DNNs on the ES. Ren et al. [33] proposed a federated learning-based method, which offloads training parameters of end devices to the ES. In [34] and [35], the authors proposed adaptive methods which can adjust the accuracy requirement based on the wireless link condition. However, they focused on minimizing the traffic rather than the delay, and their methods are very sensitive to the network environment [36].

Partial Computation Offloading: Kang et al. [37] first considered the large transmission delay during the offloading and thus proposed to use end devices to conduct partial computing. They designed the optimal line-structure DNN partition strategy with the goal of minimizing the transmission delay. Li et al. [38] reduced computation delay via early exiting inference at an appropriate intermediate DNN layer on the basis of adaptive partitioning of DNN neural networks. Hu et al. [39] abstracted a DNN as a DAG, and transformed the minimum delay problem into a min-cut optimization problem. They then proposed a max-flow approach. Although the DNN is divided and the middle feature layer is selected to offloading to the ES, the transmission time of the middle feature layer is still large. To solve the problem of feature compression transmission, various BottleNeck structures have been introduced into DNN models. BottleNet [40] uses the Encoder-Decoder structure for feature coding and compression. It can solve the problem of low compression rate of DNN feature transmission

Authorized licensed use limited to: Temple University. Downloaded on February 27,2023 at 19:08:26 UTC from IEEE Xplore. Restrictions apply.

TABLE 1
Symbol Table

Name	Definition
r	The kernel size.
g	The step size.
$\mathbb{P}$	The set of FL paths in the DAG.
P	The number of FL paths.
V	The total number of computation layers.
c_v	The output data size of layer v.
d_v	The amount of computation for layer v.
$e_{v'v}$	The computation dependency from layer v' to v.
h_v	The layer v is assigned to the end device or ES.
t_v^{end}	The computation time of layer v on the end device.
t_v^{tr}	The transmission time of layer v from the end device to ES.
t_v^{es}	The computation time of layer v on the ES.
$\mathbb{S}$	The path scheduling strategy.
$\mathbb{O}$	The path offloading strategy.
τ	The FL path length.
$\mathbb{U}$	The intercepted fused layer's.
$T_p(v)$	The task completion time of layer v.
$\mathbb{Q}$	The path offloading strategy variation.
π	The FL path length variation.
$\mathbb{W}$	The intercepted fused layer's size vector variation.
$\mathbb{O}_k^{best}$	The best path offloading strategy of solution k.
τ_k^{best}	The best FL path length of solution k.
$\mathbb{U}_k^{best}$	The best intercepted fused layer's size vector of solution k.
$\mathbb{T}_k^{best}$	The minimum DNN inference time of solution k.
$\mathbb{O}^{best}$	The best path offloading strategy.
τ^{best}	The best FL path length.
$\mathbb{U}^{best}$	The best intercepted fused layer's size vector.
$\mathbb{T}^{best}$	The minimum DNN inference time.
$\mathbb{T}_{MW}^{best}$	The minimum DNN inference time by MW algorithm.
l	The number of neural layers without the FL technique.
E	The set of dependencies.
K	The number of solutions.
I	The iterations of PSOMW.
S_τ	The size of fused layer with FL path length τ.

without bottleneck structure to a certain extent, but it will make the offloading point relatively fixed. As a result, the offloading strategy has weak adaptability to different deployment platforms and dynamic network environment. Duan et al. [25], [41] considered the impact of scheduling order on DNN inference time after DNN partitioning. Fu et al. [42] jointly considered network conditions and computation capabilities of the end device and the ES to reduce the DNN inference time by partitioning the DNN and computing it on the end device in poor network conditions. Zeng et al. [43] proposed CoEdge, which divides each neural network layer and assigns it to multiple ES for computation with intermediate results sharing. Wang et al. [22] proposed a DNN layer-wise processing schedule model to solve pipeline-based DAG schedule problem, which offloads the entire middle feature layer to the ES.

Different from the existing work, we propose a partial computation offloading model for accelerating DNN inference without accuracy loss, and we further consider model parallelism with more fine-grained. Specifically, FL technique is used to implement multi-core GPU parallel computing locally, and we use FL technique to achieve models parallelism in MEC. In our model, DNN is divided into independent sequence of layers for offloading, rather than directly offloading an intermediate layer. Each independent sequence of layers can be offloaded to the ES after partial computing on the end device. The end device and ES can make full use of their computing resources and reduce DNN inference time by model parallelism, which is very important for applications with low latency requirements.

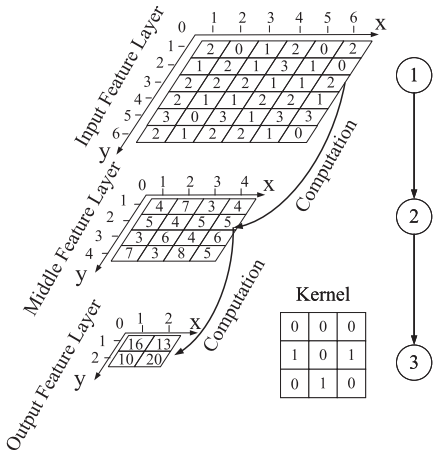


Fig. 1. A simple DNN computation without the FL technique.

3 System Model

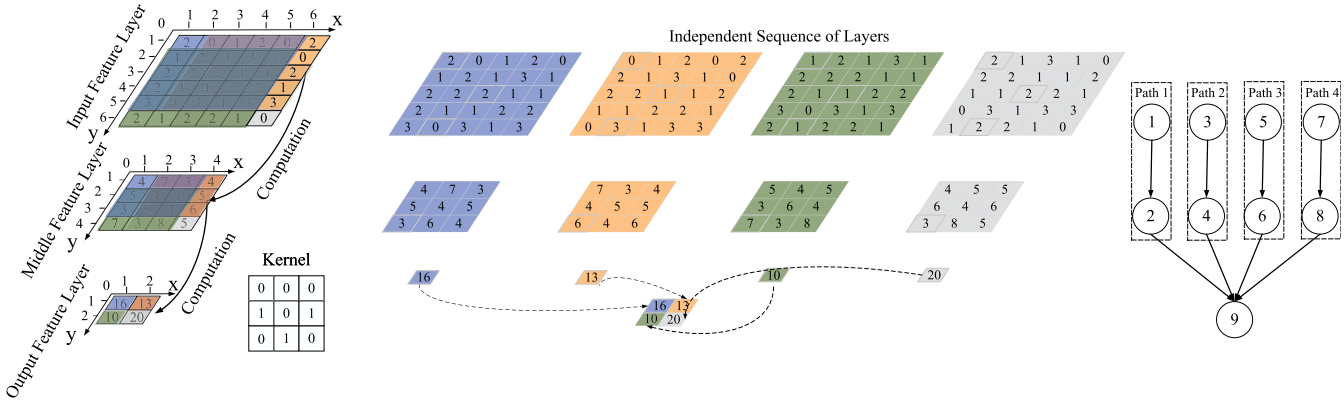
In this section, we first introduce the FL technique for DNN processing, which can enable DNN parallel processing. Then, we further discuss the partial computation offloading model used in this paper, and present the problem formulation. The symbols used in this paper are shown in Table 1.

3.1 Fused-Layer Technique

The FL technique has been originally proposed to accelerate DNN inference locally without accuracy loss. Specifically, FL technique divides DNN into separate sequence of layers, and accelerates DNN inference locally by parallel computing with multi-core GPUs [26]. Its key idea is to take advantage of processing locality for DNN operations such as convolution and pooling. For these operations, each output value only depends on the value in the corresponding area of the previous layer. With this observation, the FL technique computes the output feature layer by splitting the input feature layers into independent small layers and further fuses their corresponding results back to get the original output. Different from [24], this paper applies the FL technique to the MEC environment, in which the end device with limited computation resources offloads its computation layers to ES. It is worth noting that this paper mainly focuses on convolutional neural networks in which we can apply the FL technique.

We first show how a simple DNN is computed traditionally in Fig. 1. Without loss of generality, we use a three-layer DNN and the convolution kernel is shown in the lower-right corner of the figure. We establish a 2D cartesian coordinate system with the upper-left corner as the origin point. The sizes of input feature layer, middle feature layer, and output feature layer are 6×6, 4×4 and 2×2, respectively, by applying a 3×3 convolution kernel with a step size of 1.

Authorized licensed use limited to: Temple University. Downloaded on February 27,2023 at 19:08:26 UTC from IEEE Xplore. Restrictions apply.



(a) Correspondence of values in DNN (b) Paths divided by the FL technique

Fig. 2. A simple DNN computation with the FL technique and homogeneously intercepted fused layer's size.

DNN is computed by multiplying and adding the characteristic layer and the convolution kernel. For example, as shown in Fig. 1, the value of the middle feature layer at coordinate (1,1) is obtained by calculating the convolution result of the rectangular area surrounded by vertex $\{(1,1),(1,3),(3,1),(3,3)\}$ in the input feature layer and convolution kernel (i.e., $4 = 2\times0+0\times0+1\times0+1\times1+2\times0+1\times1+2\times0+2\times1+1\times0$).

Fig. 2 shows how the same DNN inference result is computed by applying the FL technique. In Fig. 2a, we show the processing dependency of four rectangular areas denoted by four colors in three layers, respectively. These four areas can be processed independently in the input and middle feature layers, and then fused on the *output feature layer*, which is also defined as the *fused layer*. In the input feature layer, the blue rectangular area whose four vertices are the coordinates of $\{(1,1),(1,5),(5,1),(5,5)\}$ is computed to get the corresponding blue rectangular area in the middle feature layer with four vertices having the coordinates of $\{(1,1),(1,3),(3,1),(3,3)\}$ by using the convolution kernel. Similarly, the blue rectangular area in the middle feature layer is computed by using the same kernel to get the blue rectangular area in the fused layer with one vertex having coordinates of (1,1).

In general, for a value at the coordinate of (x,y) in any layer, the four vertex coordinates of the corresponding rectangular area in the previous layer are:

$$\begin{cases} ((x-1)\times g+1,(y-1)\times g+1),\\ (x-1)\times g+1,(y-1)\times g+r),\\ ((x-1)\times g+r,(y-1)\times g+1),\\ (x-1)\times g+r,(y-1)\times g+r), \end{cases} \tag{1}$$

where r and g denote the kernel size and step size, respectively.

Fig. 2b shows the DNN conversion result by using the FL technique. It generates a sequence of small layers, which can be processed independently. To clarify the description in this paper, we further define the FL path as follows.

Definition 1. *The independent sequence of layers divided by the FL technique are abstracted as the FL paths $\mathbb{P}=\{1,2,...,p,...,P\}$ in a DAG, where P represents the number of paths.*

We define the FL path length as τ, and the size of the fused layer with the FL path length τ as $S_\tau=\{S_\tau^L,S_\tau^W\}$, in which S_τ^L is the length and S_τ^W is the width of the fused layer. The total layers without applying the FL technique is l, so the upper bound of the FL path length τ cannot exceed l, i.e.,

$$0\le\tau\le l. \tag{2}$$

The set of the intercepted fused layer's size vector as $\mathbb{U}=\{u_1,u_2,...,u_p,...,u_P\}$, where $u_p=\{u_p^L,u_p^W\}$ is the intercepted fused layer's size vector of FL path p, in which u_p^L is the length and u_p^W is the width of intercepted fused layer. The intercepted fused layer's size cannot exceed S_τ^L and S_τ^W, i.e.,

$$\sum_{p=1}^{P}u_p^L=S_\tau^L,\sum_{p=1}^{P}u_p^W=S_\tau^W. \tag{3}$$

In Fig. 2b, after applying the FL technique, the DNN has 4 paths and the FL path length is 2, $S_2=\{2,2\}$, and the homogeneously intercepted fused layer's size vector $u_1=u_2=u_3=u_4=\{1,1\}$. It is worth noticing that DNN can also be divided into paths with heterogeneously intercepted fused layer's size. For example, if DNN is divided into 3 FL paths with heterogeneously intercepted fused layer's size, the intercepted fused layer's size vector can be $u_1=\{1,2\},\ u_2=u_3=\{1,1\}$. It is worth noting that the FL technique will lead to additional computation redundancy. For example, in the input feature layer, the rectangular area with vertices at the coordinates of $\{(2,1),(2,5),(5,2),(5,5)\}$ is the area where the blue rectangular area and the green rectangular area are computed repeatedly. Therefore, it is non-trivial to find the optimal FL strategy, i.e., the FL path length, the number of FL paths, and the size of intercepted fused layers.

3.2 DNN Partial Computation Offloading Model

In this paper, we propose to apply partial computation offloading paradigm to accelerate DNN inference. An end device and an ES will work collaboratively to finish DNN inference task. The FL technique mentioned in Section 3.1 is applied to create partial computation opportunities between the end device and the ES, and converts a line-structure DNN to a more complicated DAG DNN. In general, we define the computation dependency relationship of layers

Authorized licensed use limited to: Temple University. Downloaded on February 27,2023 at 19:08:26 UTC from IEEE Xplore. Restrictions apply.

in a DNN as a DAG, and use $\mathbb{V} = \{1, 2, ..., v, ..., V\}$ to denote the set of layers, where v represents a certain computation layer, and V is the total number of computation layers. We use c_v and d_v [floating point operations (FLOPs)] to denote the transmission data size of layer v, and the amount of computation of layer v, respectively. $e_{v'v} = (v', v) \in E$ represents the computation dependency relationship from v' to v, which means that layer v can only be computed after layer v' is computed completely, where E is the set of dependencies.

In the partial computation offloading paradigm, the end device can offload computation layers to the ES. The ES can process the current offloaded layer as long as its previous layer has been processed. Computation offloading strategy can be defined as $\mathbb{H} = \{h_1, h_2, ..., h_V\}$, where $h_v = 0$ if layer v is computed locally on the end device, and $h_v = 1$ if layer v is offloaded to the ES.

In order to obtain the DNN inference time, we first need to get the computation and transmission time of each computation layer.

3.2.1 Computation Time of the End Device

We assume that only one computation layer can be computed at the same time for the end device. If layer v is computed locally on the end device, then the computation time t_v^{end} of layer v on the end device is calculated as:

$$t_v^{end} = \frac{d_v}{f_{end}}, \tag{4}$$

where f_{end} [floating point operations per second (FLOPS)] is the computation resource of the end device.

3.2.2 Transmission Time Between the End Device and the ES

The computation layers are transmitted based on the first-come-first-process order. If layer v is offloaded to the ES, then the transmission time t_v^{tr} of layer v from the end device to the ES is calculated as:

$$t_v^{tr} = \frac{c_v}{R}, \quad \text{where} R = B\log_2\left(1 + \frac{QG}{\varepsilon^2}\right), \tag{5}$$

R is the transmission rate between the end device and the ES, which can be calculated by using the Shannon's theorem. B represents the bandwidth of the channel between the end device and the ES, Q is the transmission power of the end device, G is the channel gain between the end device and the ES, and ε^2 represents the standard deviation of Gaussian channel noise.

3.2.3 Computation Time of the ES

Similarly, if layer v is offloaded to the ES, then the computation time t_v^{es} of layer v on the ES is calculated as:

$$t_v^{es} = \frac{d_v}{f_{es}}, \tag{6}$$

where f_{es} [FLOPS] is the computation resource of the ES.

Then, we need to obtain the FL strategy $\mathbb{F}$ (e.g., the number of FL paths P, the intercepted fused layer's size, the FL path length τ), the path scheduling strategy, and the path offloading strategy. Due to the observation that the computation order of the FL paths on the end device is the same as the transmission order and the computation order on the ES. Hence, the path scheduling strategy $\mathbb{S} = \{s_1, s_2, ..., s_p, ..., s_P\}$ is defined as the computation order of the FL paths on the end device, where s_p is the p-th scheduling path. Moreover, we define the path offloading strategy as $\mathbb{O} = \{o_1, o_2, ..., o_p, ..., o_P\}$, where o_p is the number of layers between the first computation layer and the offloaded computation layer on path p, which can be calculated based on the computation layers' offloading strategy as:

$$o_p = \sum_{v=1}^{V}(1 - h_v) + 1, v \in p, p \in \mathbb{P}. \tag{7}$$

The upper bound of o_p cannot exceed τ, i.e.,

$$0 \le o_p \le \tau. \tag{8}$$

Specifically, $\mathrm{T}_p(v)$ is the task completion time of layer v on path p, which can be computed recursively and formally as follows:

$$\mathrm{T}_p(v) = \begin{cases} \max\mathrm{T}_p(v') + t_v^{end}, & \{h_{v'}, h_v\} = \{0, 0\}, \\ \max\mathrm{T}_p(v') + t_v^{tr} + t_v^{es}, & \{h_{v'}, h_v\} = \{0, 1\}, \\ \max\mathrm{T}_p(v') + t_v^{es}, & \{h_{v'}, h_v\} = \{1, 1\}, \end{cases} \tag{9}$$

where v' represents the previous computation layer that has the computation dependency on layer v, i.e., $\exists e_{v'v} \in E$.

After all FL paths are computed on ES, the output of FL paths will be fused as a fused layer with the same values as the normal convolution to the fused layer. If the number of FL paths and FL path length are obtained, the value of fused layer can be calculated as $\tau P + 1$.

Let us denote the task completion time of fused layer and the layer after fused layer as $\mathrm{T}(v)$. Then, the task completion time of fused layer can be calculated as:

$$\mathrm{T}(v) = \max\mathrm{T}_p(v') + t_v^{es}, v = \tau P + 1, \tag{10}$$

The task completion time of the layer after fused layer can be calculated as:

$$\mathrm{T}(v) = \mathrm{T}(v') + t_v^{es}, v > \tau P + 1. \tag{11}$$

To illustrate how to obtain the path offloading strategy $\mathbb{O}$ and task completion time, we use a five-layer DNN as an example. As shown in Fig. 3, the FL path length is 3, and the number of FL paths is 3 (i.e., $\tau = 3$, $P = 3$). The path scheduling order is path 1, path 2, path 3. That is, $\mathbb{S} = \{s_1, s_2, s_3\} = \{1, 2, 3\}$. The three FL paths are offloaded to the ES at layer 3, layer 5 and layer 8, respectively. Therefore, the computation layers' offloading strategy $\mathbb{H} = \{0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1\}$ and the path offloading strategy $\mathbb{O} = \{3, 2, 2\}$ are shown by using a red dotted line. The couple $(2, 2)$ near layer 1 means that the amount of computation of layer 1 is 2 and the transmission data size is 2. For explanation simplicity, we assume that the CPU frequency of the end device, the transmission rate

Authorized licensed use limited to: Temple University. Downloaded on February 27,2023 at 19:08:26 UTC from IEEE Xplore. Restrictions apply.

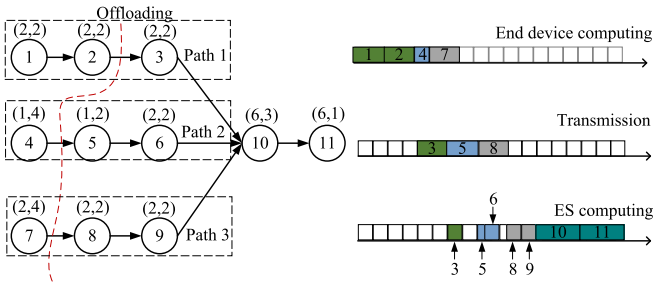


Fig. 3. Illustrating DNN inference with the FL technique in a five-layer DNN.

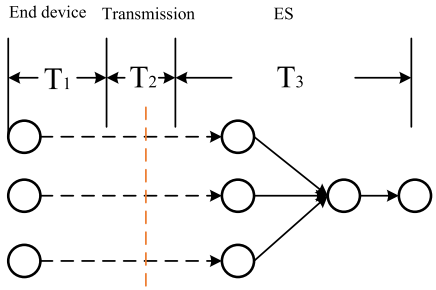


Fig. 4. An example of NP-hard reduction.

between the end device and the ES are 1, and the CPU frequency of the ES is 2. Then, the task completion time of layer 2 is $\mathrm{T}_1(2) = \mathrm{T}_1(1) + t_2^{end} = 2 + 2 = 4$. Layer 3 is offloaded to the ES, so the task completion time of layer 3 includes the transmission time and the ES computation time, which can be calculated as $\mathrm{T}_1(3) = \mathrm{T}_1(2) + t_3^{tr} + t_3^{es} = 4 + 2 + 1 = 7$. Layer 10 ($3 \times 3 + 1$) is the fused layer, so the task completion time of layer 10 is $\mathrm{T}(10) = \max\{\mathrm{T}_1(3), \mathrm{T}_2(6), \mathrm{T}_3(9)\} + t_{10}^{es} = 12 + 3 = 15$. Layer 11 is the layer after fused layer, so the task completion time of layer 11 is $\mathrm{T}(11) = \mathrm{T}(10) + t_{11}^{es} = 15 + 3 = 18$. The right side shows the computation layer's end computation time, transmission time between the end device and the ES, and the ES computation time on the time axis. Each cell represents a unit time and it can be seen that the DNN inference time is 18.

3.3 Problem Formulation

The objective of this paper is to minimize the DNN inference time in partial computation offloading while considering DNN model parallelism optimization. The DNN inference time can be interpreted as the task completion time of the last computation layer. Then, we use $\mathbb{T}$ to indicate the DNN inference time (i.e., the task completion time of the last layer) with computation dependency, which can be formulated as follows:

$$\begin{aligned} &\min_{\mathrm{F,S,O}} \quad \mathbb{T} = \max \mathrm{T}(v) \\ &s.t. \quad C1: \mathit{Eq.}\ (9) - \mathit{Eq.}\ (11), \\ &\quad C2: \mathrm{T}_p(v') \le \mathrm{T}_p(v), \quad \forall e_{v'v} \in E, \\ &\quad C3: \mathrm{T}_p(v') \le \mathrm{T}(v), \quad \forall e_{v'v} \in E, \\ &\quad C4: \mathrm{T}(v') \le \mathrm{T}(v), \quad \forall e_{v'v} \in E. \end{aligned} \tag{12}$$

Among them, constraint $C1$ formulates the task completion time of each layer, constraints $C2$, $C3$, and $C4$ are the computation dependency. The computation layer can only be computed if all of its predecessors have been computed. To sum up, the DNN inference time can be obtained in relation to the FL strategy, the path scheduling strategy and the path offloading strategy.

3.4 Problem Hardness

Theorem 1. *The proposed minimizing DNN inference time problem is NP-hard.*

Proof: Since the 3-machine flow-shop problem is a well-known NP hard problem [22], this paper transforms the optimization problem into the 3-machine flow-shop problem, thus proving that the complexity of this problem is NP-Hard. For any instance of the 3-machine flow-shop problem with p jobs, we can reduce it to a special case of this problem by building a special DAG which has p separate paths as shown in Fig. 4. Any path will have three stages in such a special DNN. It is worth noting that we consider the network partition in multi-path, and use T_1, T_2, and T_3 to denote the corresponding end device computing time, the transmission time between end device and ES, and the ES computing time for that path in three stages, respectively. Particularly, (1) $T_1 = \sum_{v=b}^{c-1} t_v^{end}$, (2) $T_2 = t_c^{tr}$, and (3) $T_3 = \sum_{v=c}^{e} t_v^{es}$, where b, c, and e are the beginning layer, the offloading layer, and the last layer of this path, respectively. We can build layers so that T_1, T_2, and T_3 are equal to the processing times of job i in three machines in polynomial time.

3-machine flow-shop $\Rightarrow$ *our problem*: Suppose that there is an optimal dispatch in a 3-machine flow-shop. Then, the optimal solution of the 3-machine flow-shop is applied to this problem to obtain the minimum computation time for the above special cases.

Our problem $\Rightarrow$ *3-machine flow-shop*: Similarly, suppose that we find the optimal solution for this particular case of the problem. Then, based on the above reduction, we can derive the optimal solution of the corresponding 3-machine flow-shop problem.

Therefore, we prove that our problem is as hard as 3-machine flow-shop.

4 Particle Swarm Optimization With Minimizing Waiting

In this section, we will propose a heuristic method, Particle Swarm Optimization with Minimizing Waiting (PSOMW) to solve the above optimization problem, and obtain the near-optimal solution including: (1) the path scheduling strategy; (2) the FL strategy; (3) the path offloading strategy. The motivation is that the above optimization problem can be proved to be NP-Hard. When a DNN has few layers, the optimal solution can be obtained by using the Brute Force (BF) algorithm. However, the complexity of finding the optimal solution increases exponentially with the increase of the total number of layers and the FL paths. Therefore, an efficient heuristic method is necessary to satisfy the real-time requirement of practical application.

4.1 Minimizing Waiting Algorithm

This part proposes the Minimizing Waiting (MW) algorithm to determine the path scheduling strategy and the number of FL paths. In MW algorithm, the FL strategy can be

Authorized licensed use limited to: Temple University. Downloaded on February 27,2023 at 19:08:26 UTC from IEEE Xplore. Restrictions apply.

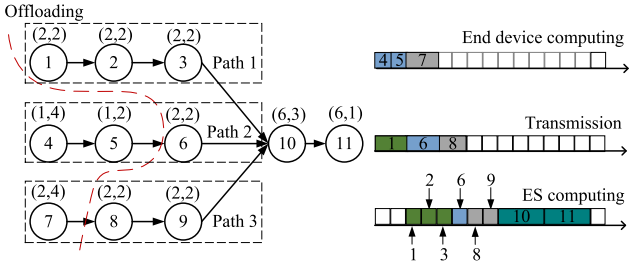


Fig. 5. Illustrating MW in a five-layer DNN.

obtained by enumerating all possible numbers of FL paths and FL path length. Then, the intercepted fused layer's size is obtained by dividing S_τ into P layers with the same size, so the intercepted fused layer's size vector $\mathbb{U}$ is obtained. Once the FL strategy is obtained, DNN inference time can be obtained by determining the path scheduling strategy and path offloading strategy. Therefore, by updating the FL strategy, we can obtain the minimum DNN inference time.

Algorithm 1. Minimizing Waiting Algorithm

Input: Neural network layers l and their parameters.
Output: The number of FL paths P and the path scheduling strategy $\mathbb{S}_{MW}^{best}$.

Initialize $\mathbb{T}_{MW}^{best} = \text{NULL}$
for FL path length $\tau = 1 : l$ **do**
  for The number of FL paths $P = 1 : S_\tau^L \times S_\tau^W$ **do**
    The intercepted fused layer's size is obtained as S_τ divided into P layers of the same size.
    for each FL path **do**
      The first scheduling path and the offloaded layer are determined as:
        $p, o_p \leftarrow \min(\mathrm{T}_p(v-1) + t_v^{tr})$
      The scheduling path is recorded as s_1 and the offloaded layer o_p is recorded in $\mathbb{O}$.
    end for
    for The p-th scheduling path $p = 2 : P$ **do**
      The p-th scheduling path and the offloaded layer are determined as:
        $p, o_p \leftarrow \min(|\mathrm{T}_{p-1}(v) - t_v^{es} - \mathrm{T}_p(v')|)$
      The scheduling path is recorded as s_p and the offloaded layer o_p is recorded in $\mathbb{O}$.
    end for
    According to the $\mathbb{S}$, $\mathbb{O}$, $\mathbb{U}$, the DNN inference time $\mathbb{T}_{MW}$ is obtained.
    If $\mathbb{T}_{MW}^{best} = \text{NULL}$
      Update $\mathbb{T}_{MW}^{best} \leftarrow \mathbb{T}_{MW}$, $\tau_{MW}^{best} \leftarrow \tau$,
        $\mathbb{S}_{MW}^{best} \leftarrow \mathbb{S}$, $\mathbb{U}_{MW}^{best} \leftarrow \mathbb{U}$.
    End If
    If $\mathbb{T}_{MW} \leq \mathbb{T}_{MW}^{best}$
      Update $\mathbb{T}_{MW}^{best} \leftarrow \mathbb{T}_{MW}$, $\tau_{MW}^{best} \leftarrow \tau$,
        $\mathbb{S}_{MW}^{best} \leftarrow \mathbb{S}$, $\mathbb{U}_{MW}^{best} \leftarrow \mathbb{U}$.
    End If
  end for
end for

For the path scheduling strategy and path offloading strategy, the main idea is that once the current path has completed its transmission, the next path should finish its computation and start to transmit without waiting. The first path can be determined by using the following criterion. The fewer layers on the path computed on the end device, the faster the transmission to the ES, and the shorter the time for parallel computing of the next path on the end device. However, too few layers computed on the end device will lead to too much transmission data, thus increasing the transmission time between the end device and the ES. Therefore, we need to find an appropriate number of offloaded layers of each path. The offloaded layer v on path p should satisfy the minimal transmission completion time, which can be formulated as:

$$\min\left(\mathrm{T}_p(v-1) + t_v^{tr}\right), (h_{v-1} = 0, h_v = 1). \tag{13}$$

Then, the first scheduling path is recorded as s_1. If multiple offloading solutions result in the same transmission completion time, we will choose the offloading strategy which has the minimum local computation time. This is because in this case, the ES can start processing as soon as possible and at the same time, the next path can compute on the end device as soon as possible.

The offloading strategy of the p-$th, (p \in \{2, 3, ..., P\})$ scheduling path is determined by the transmission completion time of the $(p-1)$-th scheduling path so that the waiting time between two paths is minimized. In particular, the task completion time on the end device of the p-th scheduling path should be as close as possible to the transmission completion time of the $(p-1)$-th scheduling path. Then, the p-th scheduling path can start transmission as soon as possible after the $(p-1)$-th scheduling path is completed. The offloaded layer v of the p-th scheduling path denoted as s_p can be determined by the following formula:

$$\min\left(\left|\mathrm{T}_{p-1}(v) - t_v^{es} - \mathrm{T}_p(v'-1)\right|\right), p \in \{2, 3, ..., P\}, \tag{14}$$

where v is the offloaded layer of the $(p-1)$-th scheduling path and v' is the offloaded layer of the p-th scheduling path. If multiple layers have the same task completion time on the end device, we will choose the path which has the maximum number of previous layers on the path. Then, the p-th scheduling path can compute more on the end device and reduce the computing time of ES.

To illustrate the proposed MW algorithm, we use a five-layer DNN as an example. As shown in Fig. 5, the FL path length, the number of FL paths, the CPU frequency of the end device, the transmission rate and the the CPU frequency of the ES are the same in Fig. 3. The minimal transmission completion time of the three FL paths are 2, 4, and 4, respectively. By using the MW algorithm, path 1 is the first scheduling path. If we choose to offload layer 1, the transmission completion time of path 1 is $0 + 2 = 2$ since the task completion time of layer 1 on the end device is 0, and the transmission time of layer 1 is 2. By following the same logic, if we choose to offload layer 2 or layer 3, the transmission completion time is $2 + 2 = 4$ or $2 + 2 + 2 = 6$, respectively.

Therefore, $o_1 = 1$. For the second scheduling path, the task completion time of layers on the end device should be as close to 2 as possible. The task completion time of layer 5 and layer 7 on the end device are $1 + 1 = 2$ and 2. Therefore, we determine path 2 as the second scheduling path, and layer 6 is offloaded to the ES. That is, $o_2 = 3$ and the transmission completion time of layer 6 is $2 + 2 = 4$. Path 3 is the

Authorized licensed use limited to: Temple University. Downloaded on February 27,2023 at 19:08:26 UTC from IEEE Xplore. Restrictions apply.

third scheduling path, and the task completion time of layer 7, layer 8 and layer 9 on the end device are $2+2=4$, $2+2+2=6$ and $2+2+2+2=8$, respectively. The task completion time of layer 8 on the end device is closest to 4, so the offloaded layer on path 3 is layer 8. That is, $o_3=2$. When the previous layers of layer 10 are done, the outputs of layer 3, layer 6 and layer 9 are fused as layer 10, then the fused layer 10 can start the computing, and the task completion time of layer 10 is $8+3=11$.

Algorithm 2. Particle Swarm Optimization With Minimizing Waiting

Input: Neural network layers l and their parameters; Iterations I; Number of solutions K; Number of FL paths P
Output: The minimum completion time $\mathbb{T}^{best}$; The best solution $\mathbb{U}^{best}$, τ^{best}, $\mathbb{O}^{best}$, $\mathbb{S}^{best}$

1: **for** solution $k=1:1:K$ **do**
2: $\mathbb{O}_k$, $\mathbb{U}_k$, τ_k is randomly generated
3: **end for**
4: **for** solution $k=1:1:K$ **do**
5: The path scheduling strategy $\mathbb{S}_k$ is obtained by using the MW algorithm.
6: According to $\mathbb{O}_k, \mathbb{U}_k, \tau_k, \mathbb{S}_k$, the DNN inference time $\mathbb{T}_k$ is obtained.
7: Update $\mathbb{O}_k^{best} \leftarrow \mathbb{O}_k, \mathbb{U}_k^{best} \leftarrow \mathbb{U}_k$,
8: $\tau_k^{best} \leftarrow \tau_k, \mathbb{S}_k^{best} \leftarrow \mathbb{S}_k$ and $\mathbb{T}_k^{best} \leftarrow \mathbb{T}_k$.
9: **If** $k=1$
10: Update $\mathbb{O}^{best} \leftarrow \mathbb{O}_k, \mathbb{U}^{best} \leftarrow \mathbb{U}_k$,
11: $\tau^{best} \leftarrow \tau_k, \mathbb{S}^{best} \leftarrow \mathbb{S}_k$ and $\mathbb{T}^{best} \leftarrow \mathbb{T}_k$.
12: **ElseIf** $\mathbb{T}_k \leq \mathbb{T}^{best}$
13: Update $\mathbb{O}^{best} \leftarrow \mathbb{O}_k, \mathbb{U}^{best} \leftarrow \mathbb{U}_k$,
14: $\tau^{best} \leftarrow \tau_k, \mathbb{S}^{best} \leftarrow \mathbb{S}_k$ and $\mathbb{T}^{best} \leftarrow \mathbb{T}_k$.
15: **End If**
16: **end for**
17: **for** Iteration i from 2, 3, to I **do**
18: **for** solution $k=1:1:K$ **do**
19: The path scheduling strategy $\mathbb{S}_k$ is determined by MW algorithm.
20: According to $\mathbb{O}_k, \mathbb{U}_k, \tau_k, \mathbb{S}_k$, the DNN inference time $\mathbb{T}_k$ is obtained.
21: **If** $\mathbb{T}_k \leq \mathbb{T}_k^{best}$
22: Update $\mathbb{O}_k^{best} \leftarrow \mathbb{O}_k, \mathbb{U}_k^{best} \leftarrow \mathbb{U}_k$,
23: $\tau_k^{best} \leftarrow \tau_k, \mathbb{S}_k^{best} \leftarrow \mathbb{S}_k$ and $\mathbb{T}_k^{best} \leftarrow \mathbb{T}_k$.
24: **End If**
25: **If** $\mathbb{T}_k \leq \mathbb{T}^{best}$
26: Update $\mathbb{O}^{best} \leftarrow \mathbb{O}_k, \mathbb{U}^{best} \leftarrow \mathbb{U}_k$,
27: $\tau^{best} \leftarrow \tau_k, \mathbb{S}^{best} \leftarrow \mathbb{S}_k$ and $\mathbb{T}^{best} \leftarrow \mathbb{T}_k$.
28: **End If**
29: **end for**
30: **Each** solution **does** Update $\mathbb{O}$, τ and $\mathbb{U}$.
31: $\mathbb{O}_k^{*\prime} \leftarrow \gamma_1 \mathbb{O}_k^* + \gamma_2(\mathbb{O}_k^{best} - \mathbb{O}_k) + \gamma_3(\mathbb{O}^{best} - \mathbb{O}_k)$
32: $\tau_k^{*\prime} \leftarrow \gamma_1 \tau_k^* + \gamma_2(\tau_k^{best} - \tau_k) + \gamma_3(\tau^{best} - \tau_k)$
33: $\mathbb{U}_k^{*\prime} \leftarrow \gamma_1 \mathbb{U}_k^* + \gamma_2(\mathbb{U}^{best} - \mathbb{U}_k) + \gamma_3(\mathbb{U}^{best} - \mathbb{U}_k)$
34: $\mathbb{O}_k' \leftarrow \lfloor \mathbb{O}_k + \mathbb{O}_k^{*\prime} \rfloor$
35: $\tau_k' \leftarrow \lfloor \tau_k + \tau_k^{*\prime} \rfloor$
36: $\mathbb{U}_k' \leftarrow \lfloor \mathbb{U}_k + \mathbb{U}_k^{*\prime} \rfloor$
37: **end for**

The pseudocode of the MW algorithm is shown in Algorithm 1. Line 1 is to initialize $\mathbb{T}_{MW}^{best}$, where $\mathbb{T}_{MW}^{best}$ is the minimum DNN inference time obtained by the MW algorithm. Line 4 is to obtain the FL path length $\mathbb{U}_{MW}$. Lines 5 to 9 are to determine the first scheduling path, the offloaded layer and record them in $\mathbb{S}$, $\mathbb{O}$, respectively. Lines 10 to 14 are to determine the path scheduling order from the second path to the last path and record the corresponding values in $\mathbb{S}$, $\mathbb{O}$, respectively. Line 15 is to obtain the DNN inference time by Eq. (12). Lines 16 to 23 are to update the DNN inference time $\mathbb{T}_{MW}^{best}$, the best solution by using the MW algorithm.

It is worth noting that although the MW algorithm can get the path scheduling strategy and the path offloading strategy, the path offloading strategy obtained by MW is not always the optimal solution, so the MW algorithm is only used to determine the path scheduling strategy $\mathbb{S}$ and the number of FL paths P.

4.2 Particle Swarm Optimization With Minimizing Waiting

The FL path length, the intercepted fused layer's size and the path offloading strategy are further optimized by the PSOMW algorithm. The basic idea of the Particle Swarm Optimization (PSO) algorithm is to simulate the predatory behavior of birds. Birds adjust their search path through their own experience and communication among populations, so as to find the place with the most food. The PSO algorithm is a probability-based global optimization algorithm. It has a strong global search ability for nonlinear and multimodal problems, and has a high probability of obtaining the global optimal solution [44].

In this part, we combine the PSO algorithm with MW algorithm, and propose the PSOMW algorithm. In PSOMW, we first initialize the solution space by using the MW algorithm. The number of FL paths is obtained by using the MW algorithm with the minimum DNN inference time, which traverses all FL paths and the FL path length. Then, the initial solution randomly generates the path offloading strategy $\mathbb{O}$ for each solution, and the path scheduling strategy is determined by using the MW algorithm. Moreover, the DNN inference time can be obtained, which is the evaluation index in PSOMW. Solution k will record its minimum DNN inference time in history $\mathbb{T}_k^{best}$ and the solution includes $\mathbb{U}_k^{best}, \tau_k^{best}, \mathbb{S}_k^{best}, \mathbb{O}_k^{best}$. In addition, the minimum DNN inference time for all solutions' history $\mathbb{T}^{best}$ and the corresponding solution $\mathbb{U}^{best}, \tau^{best}, \mathbb{S}^{best}, \mathbb{O}^{best}$ will also be recorded.

Next, for each solution, we will update the FL path length, the intercepted fused layers' size and the path offloading strategy to find the optimal solution. The following variables are defined to show how solutions are updated. The path offloading strategy variation is defined as $\mathbb{O}^* = \{o_1^*, o_2^*, \ldots, o_p^*, \ldots, o_P^*\}$, o_p^* represents the variation of o_p. The intercepted fused layers' size variation is defined as $\mathbb{U}^* = \{u_1^*, u_2^*, \ldots, u_p^*, \ldots, u_P^*\}$, $u_p^* = \{u_p^{L*}, u_p^{W*}\}$, where u_p^{L*} is the length vector variation and u_p^{W*} is the width vector variation of u_p. The FL path length variation τ^* is defined as the variation of the FL path length τ. The FL path length, the intercepted fused layer's and the path offloading strategy variations are determined by the inertia of themselves, the best in their solution history $\mathbb{O}_k^{best}, \mathbb{U}_k^{best}, \tau_k^{best}$, and the best in all solution history $\mathbb{O}^{best}, \mathbb{U}^{best}, \tau^{best}$. The FL path length, the intercepted fused layers' size and the path offloading strategy are updated as follows:

Authorized licensed use limited to: Temple University. Downloaded on February 27,2023 at 19:08:26 UTC from IEEE Xplore. Restrictions apply.

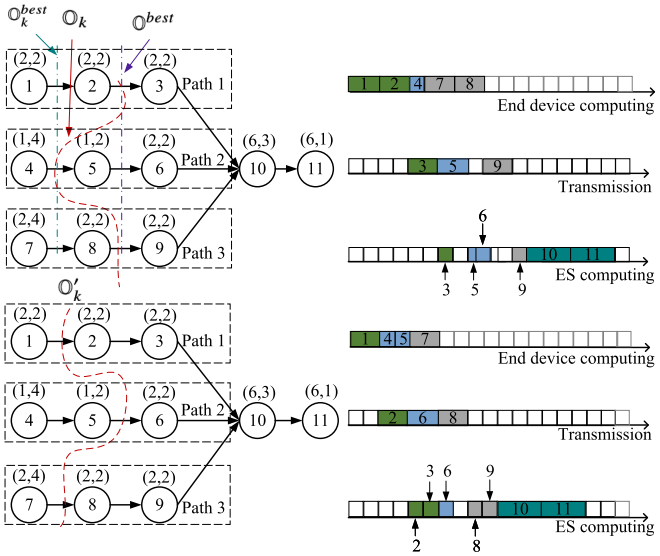


Fig. 6. Illustrating path offloading strategy updating in a five-layer DNN.

$$\mathbb{O}_k^{*\prime} \leftarrow \gamma_1 \mathbb{O}_k^{*} + \gamma_2\left(\mathbb{O}_k^{best} - \mathbb{O}_k\right) + \gamma_3\left(\mathbb{O}^{best} - \mathbb{O}_k\right), \quad (15)$$

$$\tau_k^{*\prime} \leftarrow \gamma_1 \tau_k^{*} + \gamma_2\left(\tau_k^{best} - \tau_k\right) + \gamma_3\left(\tau^{best} - \tau_k\right), \quad (16)$$

$$\mathbb{U}_k^{*\prime} \leftarrow \gamma_1 \mathbb{U}_k^{*} + \gamma_2\left(\mathbb{U}_k^{best} - \mathbb{U}_k\right) + \gamma_3\left(\mathbb{U}^{best} - \mathbb{U}_k\right), \quad (17)$$

$$\mathbb{O}'_k \leftarrow \left\lfloor \mathbb{O}_k + \mathbb{O}_k^{*\prime} \right\rfloor, \quad (18)$$

$$\tau'_k \leftarrow \left\lfloor \tau_k + \tau_k^{*\prime} \right\rfloor, \quad (19)$$

$$\mathbb{U}'_k \leftarrow \left\lfloor \mathbb{U}_k + \mathbb{U}_k^{*\prime} \right\rfloor, \quad (20)$$

where γ_1 is the inertia factor of solution, γ_2 is the influence factor of the best solution in its history, and γ_3 is the influence factor of the best solution in all solution's history.

It is worth noting that the inference solution boundary of τ, o_p and u_p prevents the generation of invalid solutions. Our algorithm uses the aforementioned constraints to ensure that the generated result is valid. For example, it is meaningless to have the negative FL path length or to exceed the neural layers of the DNN. If o_p exceeds the FL path length, all FL paths are offloaded to the ES after the task is completed on the end device without parallelization. The largest path size is reduced until the boundary constraint is satisfied.

Fig. 6 illustrates an example about the path offloading strategy $\mathbb{O}$. The CPU frequency of the end device, the transmission rate, and the CPU frequency of the ES in the example are the same as in Fig. 5. Before updating, the current path offloading strategy $\mathbb{O}_k$, the best strategy in its history $\mathbb{O}_k^{best}$ and the best solution in all solution's history $\mathbb{O}^{best}$ are shown in Fig. 6. For convenience, γ_1, γ_2, γ_3 and τ^* is set to be 0, 1, 1, 0, and each FL path has the same intercepted fused layer's size, respectively. Then, $\mathbb{O}^* = \{-1, 1, -1\}$ (i.e., $o_1^* = 1 \times (1-2) + 1 \times (2-2) = -1$). Therefore, the path offloading strategy is changed from $\mathbb{O} = \{3, 2, 3\}$ to $\mathbb{O}' = \{2, 3, 2\}$. The DNN inference time before updating is 18, and the DNN inference time after updating is 16.

The pseudocode of the proposed method is shown in Algorithm 2. Lines 1 to 3 are to randomly generate $\mathbb{O}_k$, $\mathbb{U}_k$, τ_k. Lines 4 to 16 are the first iteration, and record $\mathbb{T}_k^{best}$, $\mathbb{T}^{best}$, $\mathbb{U}_k^{best}$, $\mathbb{U}^{best}$, τ_k^{best}, τ^{best}, $\mathbb{O}_k^{best}$, $\mathbb{O}^{best}$, $\mathbb{S}_k^{best}$, $\mathbb{S}^{best}$. Line 17 starts the iteration. Lines 18 to 29 are to compute the DNN inference time of each solution and update the corresponding results. Lines 30 to 36 are to update the FL path length, the intercepted fused layer's and the path offloading strategy of each solution.

Although PSO is used in many existing studies, there are significant differences between PSOMW and traditional PSO. In traditional PSO, the initial space of the solution is randomly generated, whereas in PSOMW, the scheduling strategy $\mathbb{S}$ and the number of FL paths P are determined by the MW algorithm. In the iteration, the traditional PSO has no limit on the change of solution. In PSOMW, the sum of u_p^L and u_p^W of all paths cannot exceed S_τ^L and S_τ^W. As a result, PSOMW is more suitable for our model.

TABLE 2
Simulation Parameters

Parameter	Definition	Value
f_{end}	computation resource of the end device	2.23×10^8
f_{ES}	computation resource of the ES	4.32×10^9
B	The bandwidth of wireless links	5MHz
Q	The transmission power of the end device	0.1W
ε^2	The power of background noise	10^{-9}
G	The channel gain between end device and ES	10^{-6}
γ_1	The inertia of solution	0.5
γ_2	Influencing factors of itself	0.5
γ_3	Influencing factors of all solutions	0.5

5 Performance Evaluation

In this section, we conduct extensive simulations to demonstrate the effectiveness of the proposed method in five neural networks which are (1) AlexNet, (2) MobileNet, (3) SqueezeNet, (4) VGG16, and (5) YOLOv2. The structure of the neural network can be obtained from [45], [46], [47], [48], [49] (i.e., the number of convolution layers, convolution kernel size, convolution step, etc.). Following the existing relevant studies [37] and [50], simulation parameters used in our experiment are shown in Table 2.

5.1 Performance Comparison

We compare the performance of our proposed PSOMW with the following benchmark algorithms:

1) *No FL (NFL)*: Partial computation offloading without the FL technique is used in this algorithm. After the end device has computed to an intermediate layers, the entire intermediate layer is offloaded to the ES for the remaining computations. The minimum DNN inference time is obtained by traversing all feasible solutions.
2) *Brute Force (BF)*: Partial computation offloading with the FL technique is used in this algorithm, DNN is divided into multiple paths, and the optimal FL strategy, path scheduling strategy and offloading strategy are obtained by traversing all feasible solutions.
3) *Minimizing Waiting (MW)*: Partial computation offloading with the FL technique is used in this algorithm. The MW algorithm is used to determine the path offloading strategy and the path scheduling

Authorized licensed use limited to: Temple University. Downloaded on February 27,2023 at 19:08:26 UTC from IEEE Xplore. Restrictions apply.

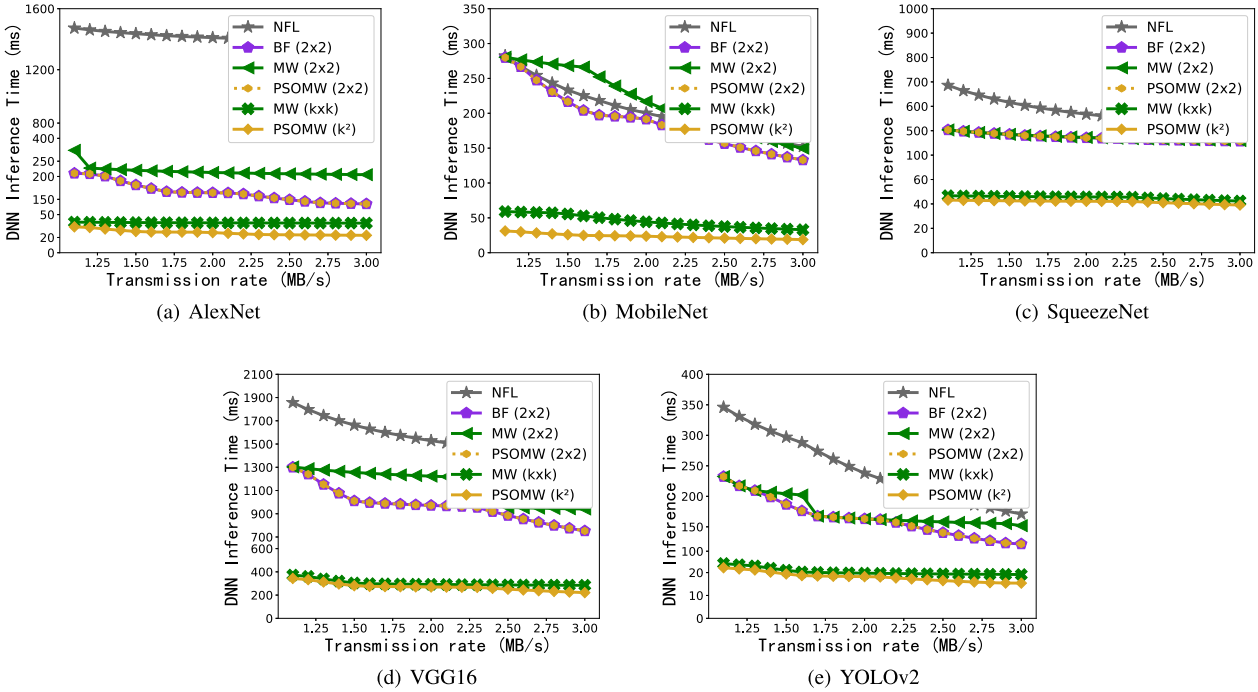


Fig. 7. The DNN inference time with only changing transmission rate.

strategy, and the FL strategy is determined by traversing all DNN layers.

With the increase of the total number of layers and the FL paths, the complexity of finding the optimal solution increases exponentially. Therefore, we choose the case of four FL paths with homogeneously intercepted fused layer's size to compare the performance of BF (2×2), MW (2×2) and PSOMW (2×2). Furthermore, we use MW ($k \times k$) to represent MW with homogeneously intercepted fused layer's size, where the number of FL paths $k \times k$ is determined by MW algorithm. PSOMW (k^2) represents PSOMW with heterogeneously intercepted fused layer's size, and the number of FL paths k^2 is determined by MW. Since the time complexity of BF is too large, the result of BF with heterogeneously intercepted fused layer's size cannot be obtained, the superiority of PSOMW is verified by comparing the results of BF (2×2) and PSOMW (2×2).

5.2 Simulation Results

5.2.1 *Varying the Transmission Rate*

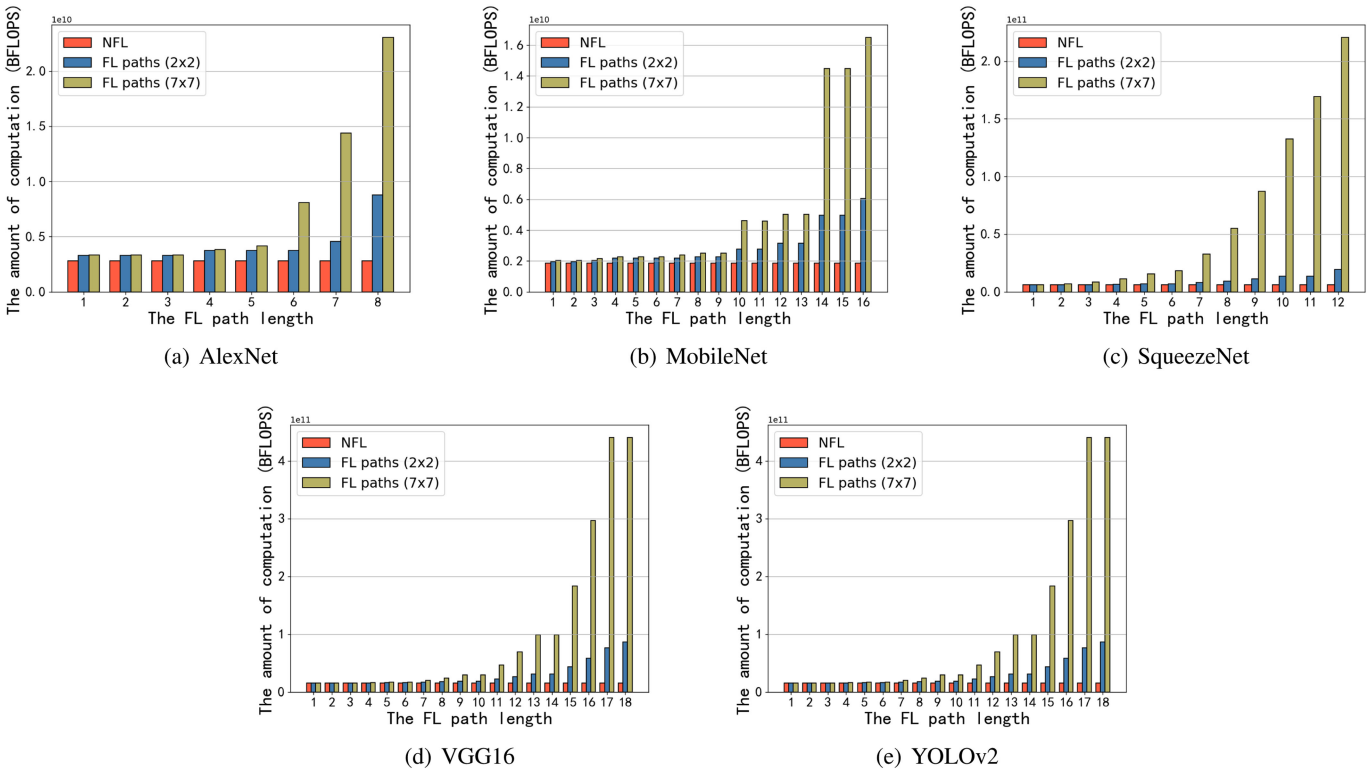
The transmission rate is varied from 1.1 MB/s to 3 MB/s to simulate a variety of scenarios, which covers common network environments, such as 4G 1.3 MB/s and WiFi 1.8 MB/s [37], etc. Fig. 7 shows the simulation results. Across five different neural networks, when the transmission rate increases from 1.1 MB/s to 3 Mb/s, PSOMW (k^2) can reduce the DNN inference time by an average of 12.75 times compared with NFL, especially 55 times in AlexNet. However, the improvement depends on the neural network architecture. Fig. 7a shows the results in the AlexNet, the DNN inference time of the NFL, BF (2×2), MW (2×2), MW ($k \times k$), and PSOMW (k^2) are reduced from 1470 ms, 260 ms, 335 ms, 40 ms, and 32 ms to 1384 ms, 160 ms, 255 ms, 38 ms, and 21 ms, respectively, when the transmission rate changes from 1.1 MB/s to 3 MB/s. The VGG16 has 18 neural layers, which is relatively larger than that of AlexNet, and the increased number of neural layers will cause more computational workloads. It can be found that, the DNN inference time of NFL, BF (2×2), MW (2×2), and PSOMW (2×2) reduces from 1856 ms, 1300 ms, 1302 ms, and 1300 ms to 1207 ms, 752 ms, 940 ms and 752 ms in VGG16, respectively. Therefore, the DNN inference time in AlexNet is shorter than that in VGG16.

The number of FL paths is very important. From the results shown in Fig. 7a, the following observation can be obtained. When the number of FL paths is 4, the DNN inference time of BF (2×2) is 5 times less than that of NFL. When the transmission rate changes from 1.1 MB/s to 3 MB/s at the MobileNet, the DNN inference time of MW ($k \times k$) and PSOMW (k^2) is reduced from 59 ms and 30 ms to 33 ms and 17 ms, respectively. Compared with MW (2×2) and PSOMW (2×2), MW ($k \times k$) and PSOMW (k^2) further reduces average 171 ms and 165 ms DNN inference time. The reason is that a lager number of FL paths leads to more flexibility in the path scheduling, and thus causes better results.

Furthermore, the simulation results shown in Fig. 7 illustrate that the DNN inference time of BF (2×2) and PSOMW (2×2) are basically the same, and the average difference of the DNN inference time are 0, 0.02%, 0, 0, 0.04% in the AlexNet, MobileNet, SqueezeNet, VGG16, YOLOv2, respectively. Therefore, the superiority of the path scheduling strategy and path offloading strategy that we obtained has been demonstrated. The results of MW and PSOMW are very different, and the difference between the results of MW ($k \times k$) and PSOMW (k^2) is 51.3%, 90.1%, 4.7%, 12.2% and 13.1% in the AlexNet, MobileNet, SqueezeNet, VGG16, YOLOv2, respectively. Therefore, the path offloading strategy obtained by MW is not always the optimal solution.

Overall, our approach reduces the DNN inference time well, whether in a lightweight DNN such as the AlexNet,

Authorized licensed use limited to: Temple University. Downloaded on February 27,2023 at 19:08:26 UTC from IEEE Xplore. Restrictions apply.



(a) AlexNet (b) MobileNet (c) SqueezeNet

(d) VGG16 (e) YOLOv2

Fig. 8. The amount of computation with different FL path length and number of FL paths.

MobileNet and YOLOv2 or a heavyweight DNN such as the VGG16 and SqueezeNet. Lightweight DNNs have a small number of convolution steps with 1 or 2 in most neural layers, using the FL technique can reduce more DNN inference time. In the AlexNet, MobileNet and YOLOv2, the average DNN inference time obtained by PSOMW (k^2) are 26 ms, 24 ms and 21 ms, respectively, which are reduced by 1388 ms, 180 ms and 222 ms compared with NFL, respectively. Heavyweight DNNs have a large number of convolution steps or neural layers, e.g., 18 neural layers in the VGG16, but the DNN inference time can still be reduced greatly by parallel computing on the end device and the ES. For example, the average DNN inference time of PSOMW (k^2) in the SqueezeNet and VGG16 are 42 ms and 270 ms, which are reduced by 534 ms and 1251 ms compared with NFL, respectively. Therefore, the effectiveness of PSOMW has been demonstrated.

5.2.2 Impact of Redundancy

Fig. 8 reveals the redundant computation overhead introduced by different FL strategies. As shown in the figure, the network structure of DNNs, including the size of the neural layer, size of the convolution kernel r and step size of the convolution g has a significant impact on the amount of redundant computation. The larger FL path length, kernel size and step size of convolution, the greater the amount of redundant computation.

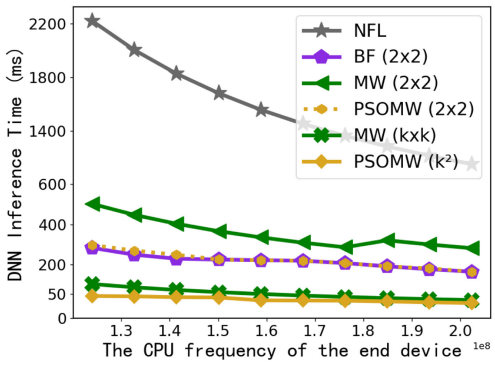
In addition, the FL path length and the number of FL paths also affect the amount of redundant computation. For the FL path, a large number of FL paths (i.e., FL paths (7×7)) leads to the increase of repeated area in the input layer. However, a large number of FL paths can reduce the DNN inference time through potential better path scheduling and offloading. The FL path length also plays a vital part in redundant computation. As shown in Fig. 8b, when the FL path length is less than 9, the overhead introduced by the FL technique is less than 35%, and the DNN inference time is 215 ms less than NFL when the transmission rate is 1.1 MB/s. In summary, an increase in the number and length of FL paths will result in an increase in the computation overhead, which increases both computation and transmission costs. However, a small number of FL paths will lead to less flexible model parallelism strategy, while a shorter FL path will lead to a large amount of transmission data. Therefore, it is necessary to balance the increased DNN inference time of redundant computation and the reduced DNN inference time of parallelization.

For the kernel size and step size of convolution, Eq. (1) shows that a larger convolution kernel size and convolution step size will lead to a larger corresponding area of the current feature layer in the previous feature layer, which will result in a larger amount of redundant computation. Therefore, the structure of the neural network itself can affect the amount of redundant computation caused by FL technique. Considering that choosing to fuse several intermediate feature layers in a small convolution step instead of starting from the input feature layer may reduce the impact, but it can lead to more complex problems, which would be an interesting direction for future work.

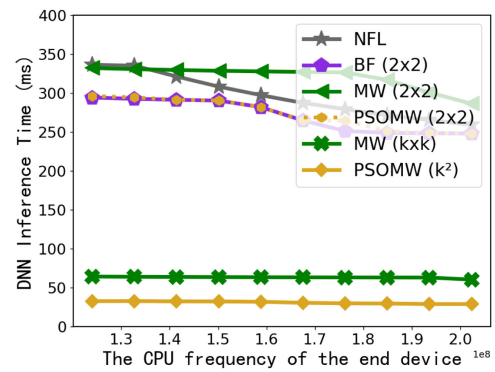
5.2.3 Varying the CPU Frequency of the End Device

To simulate different computing environments of end devices, the CPU frequency of end devices is set at 1.24×10^8 to 2.11×10^8. The DNN inference time is shown in Fig. 9 by

Authorized licensed use limited to: Temple University. Downloaded on February 27,2023 at 19:08:26 UTC from IEEE Xplore. Restrictions apply.



(a) AlexNet



(b) MobileNet

Fig. 9. The DNN inference time with only varying the CPU frequency of the end device.

using AlexNet and MobileNet. The DNN inference time of NFL, BF (2×2), MW (2×2), and PSOMW (2×2) reduces from 2220 ms, 349 ms, 568 ms, and 363 ms to 1150 ms, 230 ms, 348 ms, and 230ms in AlexNet, respectively. For the MobileNet, the DNN inference time of NFL, BF (2×2), MW (2×2), and PSOMW (2×2) reduces from 336 ms, 294 ms, 332 ms, and 296 ms to 259 ms, 248 ms, 286 ms, and 248 ms, respectively. The results show that the solution obtained by PSOMW (2×2) remains stable and is almost the same as the optimal solution obtained by BF (2×2). For MW ($k \times k$) and PSOMW (k^2), the DNN inference time reduces from 84 ms, 54 ms to 44 ms, 37ms in AlexNet and 64 ms, 32 ms to 60 ms, 28 ms in MobileNet. Therefore, PSOMW (k^2) performs best, and MW ($k \times k$) is close to PSOMW (k^2).

5.2.4 Heterogeneous Case

To compare the impact of homogeneously and heterogeneously intercepted fused layer's size on DNN inference time, the transmission rate is varied from 1.1 MB/s to 3 MB/s in VGG16. PSOMW (2^2) and PSOMW (2×2) represent PSOMW with four heterogeneously and homogeneously intercepted fused layer's size. In Fig. 10, the DNN inference time of PSOMW (2×2) and PSOMW (2^2) reduces from 1300 ms and 1203 ms to 752 ms and 703 ms. The results shows that PSOMW (2^2) can further reduce about 9.2% inference time, compared with PSOMW (2×2). To further explain why the DNN inference time of PSOMW (2^2) is less than PSOMW (2×2), we use Fig. 11 to show the computation and offloading process of the optimal solution of homogeneously and heterogeneously intercepted fused layer's

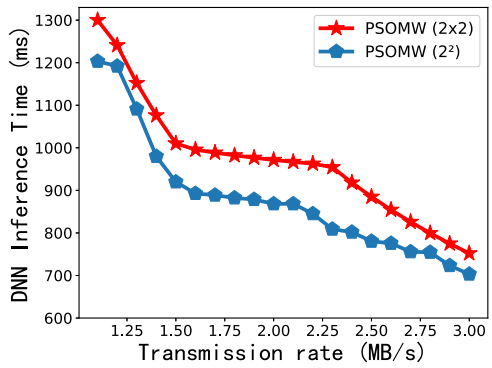


Fig. 10. The DNN inference time of homogeneously and heterogeneously intercepted fused layer's size.

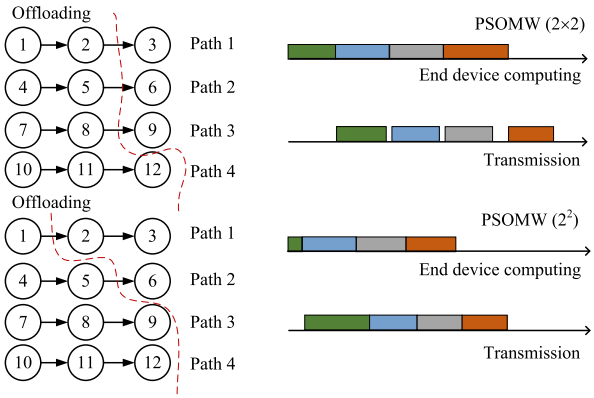


Fig. 11. Comparison of homogeneously and heterogeneously intercepted fused layer's size.

size when the transmission rate is 1.3 MB/s. In PSOMW (2×2), the transmission completion time of Path 1 is 444 ms, and the end device task completion time of Path 2 is 524 ms. This means that when the end device completes the computation of Path 2 and prepares for transmission, Path 1 has completed transmission and waited for Path 2 for 80 milliseconds However, in PSOMW (2^2), Path 2 can offload to the ES immediately when Path 1 completes the transmission. Therefore, the transmission channel has no idle waiting time. It can be seen that the transmission resources and the computing resources of the ES are not fully utilized with homogeneously intercepted fused layer's size, and the transmission channel needs to wait for the current FL path to complete its computing. As a result, PSOMW with heterogeneously intercepted fused layer's size can reduce more DNN inference time than PSOMW with homogeneously intercepted fused layer's size.

6 Conclusion

In this paper, we presented a new solution for DNN parallelism and partial computation offloading in MEC. To this end, we proposed a DNN neural network partitioning model based on the FL technique and the corresponding computation model when the DNN neural network is transformed into a DAG. Subsequently, we proposed the PSOMW method to solve the problem. Specifically, we designed the MW algorithm to determine the path

Authorized licensed use limited to: Temple University. Downloaded on February 27,2023 at 19:08:26 UTC from IEEE Xplore. Restrictions apply.

scheduling strategy and the number of FL paths, and then we combined PSO with MW to determine the FL path length, the intercepted fused layer's size and the path offloading strategy. Finally, we validated the effectiveness and superiority of the method through extensive simulation experiments. The performance results demonstrated that our proposed method can reduce the DNN inference time by an average of 12.75 times compared with NFL.

ACKNOWLEDGMENTS

This article reflects only the authors' view. The European Union Commission is not responsible for any use that may be made of the information it contains.

REFERENCES

[1] Y. Zhang, X. Lan, J. Ren, and L. Cai, "Efficient computing resource sharing for mobile edge-cloud computing networks," *IEEE/ACM Trans. Netw.*, vol. 28, no. 3, pp. 1227–1240, Jun. 2020.

[2] K. Zhang, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Mobile edge computing and networking for green and low-latency Internet of Things," *IEEE Commun. Mag.*, vol. 56, no. 5, pp. 39–45, May 2018.

[3] H. Zhou, K. Jiang, X. Liu, X. Li, and V. C. M. Leung, "Deep reinforcement learning for energy-efficient computation offloading in mobile edge computing," *IEEE Internet Things J.*, vol. 9, no. 2, pp. 1517–1530, Jan. 2022.

[4] L. Gu, D. Zeng, W. Li, S. Guo, A. Y. Zomaya, and H. Jin, "Intelligent vnf orchestration and flow scheduling via model-assisted deep reinforcement learning," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 2, pp. 279–291, Feb. 2020.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Int. Conf. Neural Inf. Process.*, 2012, pp. 1097–1105.

[6] C. Ronan and W. Jason, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proc. Int. Conf. Mach. Learn.*, 2008, pp. 160–167.

[7] A. Graves, A. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2013, pp. 6645–6649.

[8] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2014, *arXiv:1409.0473*.

[9] H. Qiu, Q. Zheng, T. Zhang, M. Qiu, G. Memmi, and J. Lu, "Toward secure and efficient deep learning inference in dependable IoT systems," *IEEE Internet Things J.*, vol. 8, no. 5, pp. 3180–3188, Mar. 2021.

[10] J. Yao and N. Ansari, "Caching in dynamic IoT networks by deep reinforcement learning," *IEEE Internet Things J.*, vol. 8, no. 5, pp. 3268–3275, Mar. 2021.

[11] H. Zhou, Z. Wang, N. Cheng, D. Zeng, and P. Fan, "Stackelberg game-based computation offloading method in cloud-edge computing networks," *IEEE Internet Things J.*, vol. 9, no. 17, pp. 16 510–16 520, Sep. 2022.

[12] H. Zhou, Z. Zhang, D. Li, and Z. Su, "Joint optimization of computing offloading and service caching in edge computing-based smart grid," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2022.3163750.

[13] H. Zhou, T. Wu, X. Chen, S. He, D. Guo, and J. Wu, "Reverse auction-based computation offloading and resource allocation in mobile cloud-edge computing," *IEEE Trans. Mobile Comput.*, to be published, doi: 10.1109/TMC.2022.3189050.

[14] K. Jiang, C. Sun, H. Zhou, X. Li, M. Dong, and V. C. M. Leung, "Intelligence-empowered mobile edge computing: Framework, issues, implementation, and outlook," *IEEE Netw.*, vol. 35, no. 5, pp. 74–82, Sep./Oct. 2021.

[15] A. Naouri, H. Wu, N. A. Nouri, S. Dhelim, and H. Ning, "A novel framework for mobile-edge computing by optimizing task offloading," *IEEE Internet Things J.*, vol. 8, no. 16, pp. 13 065–13 076, Aug. 2021.

[16] Y. M. Saputra, D. T. Hoang, D. N. Nguyen, and E. Dutkiewicz, "A novel mobile edge network architecture with joint caching-delivering and horizontal cooperation," *IEEE Trans. Mobile Comput.*, vol. 20, no. 1, pp. 19–31, Jan. 2021.

[17] G. Qiao, S. Leng, S. Maharjan, Y. Zhang, and N. Ansari, "Deep reinforcement learning for cooperative content caching in vehicular edge computing and networks," *IEEE Internet Things J.*, vol. 7, no. 1, pp. 247–257, Jan. 2020.

[18] H. Zhou, T. Wu, H. Zhang, and J. Wu, "Incentive-driven deep reinforcement learning for content caching and D2D offloading," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 8, pp. 2445–2460, Aug. 2021.

[19] H. Zhou, Z. Wang, H. Zheng, S. He, and M. Dong, "Cost minimization-oriented computation offloading and service caching in mobile cloud-edge computing: An A3C-based approach," *IEEE Trans. Netw. Sci. Eng.*, 2022.

[20] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 450–465, Feb. 2018.

[21] F. Sun et al., "Cooperative task scheduling for computation offloading in vehicular cloud," *IEEE Trans. Veh. Technol*, vol. 67, no. 11, pp. 11 049–11 061, Nov. 2018.

[22] N. Wang, Y. Duan, and J. Wu, "Accelerate cooperative deep inference via layer-wise processing schedule optimization," in *Proc. IEEE Int. Conf. Comput. Commun. Netw.*, 2021, pp. 1–9.

[23] L. Yang, J. Cao, H. Cheng, and Y. Ji, "Multi-user computation partitioning for latency sensitive mobile cloud applications," *IEEE Trans. Comput.*, vol. 64, no. 8, pp. 2253–2266, Aug. 2015.

[24] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *Proc. IEEE Int. Symp. Inf. Theory*, 2016, pp. 1451–1455.

[25] Y. Duan and J. Wu, "Joint optimization of DNN partition and scheduling for mobile cloud computing," in *Proc. Int. Conf. Parallel Process.*, 2021, pp. 1–10.

[26] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer CNN accelerators," in *Proc. IEEE/ACM Int. Symp. Microarchit.*, 2016, pp. 1–12.

[27] S. Kang et al., "GANPU: An energy-efficient multi-dnn training processor for GANs with speculative dual-sparsity exploitation," *IEEE J. Solid-State Circuits*, vol. 56, no. 9, pp. 2845–2857, Sep. 2021.

[28] M. Putic, A. Buyuktosunoglu, S. Venkataramani, P. Bose, S. Eldridge, and M. Stan, "Dyhard-DNN: Even more DNN acceleration with dynamic hardware reconfiguration," in *Proc. IEEE/ACM Annu. Des. Automat. Conf.*, 2018, pp. 1–6.

[29] C. Guo et al., "Balancing efficiency and flexibility for DNN acceleration via temporal GPU-systolic array integration," in *Proc. IEEE/ACM Annu. Des. Automat. Conf.*, 2020, pp. 1–6.

[30] P. Aleksic et al., "Bringing contextual information to google speech recognition," in *Proc. Int. Conf. Commun. Technol.*, 2015, pp. 468–472.

[31] S. Han, H. Shen, M. Philipose, S. Agarwal, and A. Krishnamurthy, "MCDNN: An approximation-based execution framework for deep stream processing under resource constraints," in *Proc. ACM Annu. Int. Conf. Mobile Syst. Appl. Service*, 2016, pp. 123–136.

[32] F. Yu, L. Cui, P. Wang, C. Han, R. Huang, and X. Huang, "EasiEdge: A novel global deep neural networks pruning method for efficient edge computing," *IEEE Internet Things J.*, vol. 8, no. 3, pp. 1259–1271, Feb. 2021.

[33] J. Ren, G. Yu, and G. Ding, "Accelerating DNN training in wireless federated edge learning systems," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 1, pp. 219–232, Jan. 2021.

[34] Y. Chen, H. Balakrishnan, L. Ravindranath, and P. Bahl, "Glimpse: Continuous, real-time object recognition on mobile devices," in *Proc. ACM Conf. Embedded Netw. Sensor Syst.*, 2015, pp. 155–168.

[35] C. Wang, S. Zhang, Y. Chen, Z. Qian, J. Wu, and M. Xiao, "Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 257–266.

[36] K. Imagane, K. Kanai, J. Katto, and T. Tsuda, "Evaluation and analysis of system latency of edge computing for multimedia data processing," in *Proc. IEEE Glob. Conf. Consum. Electron.*, 2016, pp. 1–2.

[37] Y. Kang, J. Hauswald, J. Mars, C. Gao, and A. Rovinski, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proc. Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2017, pp. 615–629.

[38] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge AI: On-demand accelerating deep neural network inference via edge computing," *IEEE Trans. Wireless Commun.*, vol. 19, no. 1, pp. 447–457, Jan. 2020.

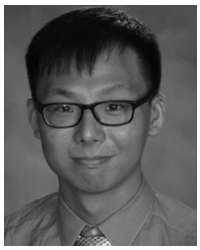
[39] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive DNN surgery for inference acceleration on the edge," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 1423–1431.

Authorized licensed use limited to: Temple University. Downloaded on February 27,2023 at 19:08:26 UTC from IEEE Xplore. Restrictions apply.

[40] A. E. Eshratifar, A. Esmaili, and M. Pedram, "BottleNet: A deep learning architecture for intelligent mobile cloud computing services," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Des.*, 2019, pp. 1–6.
[41] Y. Duan and J. Wu, "Computation offloading scheduling for deep neural network inference in mobile computing," in *Proc. IEEE/ACM Int. Symp. Qual. Service*, 2021, pp. 1–10.
[42] Z. Fu, Y. Zhou, C. Wu, and Y. Zhang, "Joint optimization of data transfer and co-execution for DNN in edge computing," in *Proc. IEEE Int. Conf. Commun.*, 2021, pp. 1–6.
[43] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, "CoEdge: Cooperative DNN inference with adaptive workload partitioning over heterogeneous edge devices," *IEEE/ACM Trans. Netw.*, vol. 29, no. 2, pp. 595–608, Apr. 2021.
[44] J. Liang, A. Qin, P. Suganthan, and S. Baskar, "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions," *IEEE Trans. Evol. Comput.*, vol. 10, no. 3, pp. 281–295, Jun. 2006.
[45] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[46] A. G. Howard, Z. Menglong, and C. Bo, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.
[47] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5mb model size," 2016, *arXiv:1602.07360*.
[48] H. Qassim, A. Verma, and D. Feinzimer, "Compressed residual-VGG16 CNN model for big data places image recognition," in *Proc. Annu. Comput. Commun. Workshop Conf.*, 2018, pp. 169–175.
[49] H. Nakahara, H. Yonekawa, T. Fujii, and S. Sato, "A lightweight YOLOv2: A binarized CNN with a parallel support vector regression for an FPGA," in *Proc. ACM Int. Symp. Field-Prog. Gate Arrays*, 2018, pp. 31–40.
[50] M. Mehrabi, S. Shen, V. Latzko, Y. Wang, and F. H. P. Fitzek, "Energy-aware cooperative offloading framework for interdependent and delay-sensitive tasks," in *Proc. IEEE Glob. Commun. Conf.*, 2020, pp. 1–6.

Huan Zhou (Member, IEEE) received the PhD degree from the Department of Control Science and Engineering, Zhejiang University. He was a visiting scholar with the Temple University from 2012 to 2013, and a CSC supported postdoc fellow with the University of British Columbia from 2016 to 2017. Currently, he is a full professor with the College of Computer and Information Technology, China Three Gorges University. He was a lead guest editor of Pervasive and Mobile Computing, TPC chair of EAI GameNets 2022, EAI BDTA 2020, local arrangement chair of I-SPAN 2018, and TPC member of IEEE Globecom, ICC, ICCCN, etc. He has published more than 70 research papers in some international journals and conferences, including *IEEE Journal on Selected Areas in Communications*, *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Mobile Computing*, *IEEE Transactions on Wireless Communications* and so on. His research interests include mobile edge computing, opportunistic mobile networks, mobile data offloading and VANETs. He receives the Best Paper Award of I-SPAN 2014 and I-SPAN 2018, and is currently serving as associate editor for *IEEE Access* and *EURASIP Journal on Wireless Communications and Networking*.

Mingze Li received the BS Degree from Northeast Electric Power University. Currently, he is working toward the graduation degree with the College of Computer and Information Technology, China Three Gorges University. His main research interests are edge computing, computation offloading and parallel computing.



Ning Wang (Member, IEEE) received the BE degree from the University of Electronic Science and Technology of China, Chengdu, China, in 2013 and the PhD degree from Temple University, Philadelphia, PA, USA, in 2018. He is currently an assistant professor with the Department of Computer Science, Rowan University, Glassboro, NJ, USA. He has authored or coauthored about 30 papers in high-impact conferences and journals, such as, IEEE ICDCS, INFOCOM, IWQoS, *IEEE Transactions on Big Data*, and *Journal of Parallel and Distributed Computing*. His current research interests include optimization problems in Internet-of-Things systems and smart cities applications. He was a program committee member for top international conferences, such as, IEEE ICDCS, and WCNC and a reviewers of premier journals, such as, *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Wireless Communications*, *IEEE Transactions on Mobile Computing*, *IEEE Transactions on Intelligent Transportation Systems*, *ACM Transactions on Internet Technology*, and *IEEE Transactions on Services Computing*.

Geyong Min (Member, IEEE) received the BSc degree in computer science from the Huazhong University of Science and Technology, China, in 1995 and the PhD degree in computing science from the University of Glasgow, U.K., in 2003. He is currently a professor of high-performance computing and networking with the Department of Computer Science, University of Exeter, U.K. His research interests include computer networks, wireless communications, parallel and distributed computing, ubiquitous computing, multimedia systems, modeling and performance engineering.

Jie Wu (Fellow, IEEE) is currently the director of Center for Networked Computing and a Laura H. Carnell professor with the Temple University. He also serves as the director of International Affairs with the College of Science and Technology. He served as the chair for the Department of Computer and Information Sciences from the summer of 2009 to the summer of 2016 and the associate vice provost for International Affairs from the fall of 2015 to the summer of 2017. Prior to joining the Temple University, he was the program director of the National Science Foundation. He was a distinguished professor with Florida Atlantic University. He regularly publishes in scholarly journals, conference proceedings, and books. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. He serves on several editorial boards including *IEEE Transactions on Service Computing* and *Journal of Parallel and Distributed Computing*. He was an IEEE Computer Society distinguished visitor, ACM distinguished speaker, and the chair of the IEEE Technical Committee on Distributed Processing (TCDP). He is a CCF distinguished speaker. He was the general co-chair of IEEE MASS 2006, IEEE IPDPS 2008, IEEE ICDCS 2013, ACM MobiHoc 2014, ICPP 2016, and IEEE CNS 2016, and the program co-chair of the IEEE INFOCOM 2011 and CCF CNCC 2013. He was a recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/csdl.

Authorized licensed use limited to: Temple University. Downloaded on February 27,2023 at 19:08:26 UTC from IEEE Xplore. Restrictions apply.