Fused-Layer-based DNN Model Parallelism and Partial Computation Offloading

Mingze Li*, Ning Wang†, Huan Zhou*, Yubin Duan§, and Jie Wu§

*College of Computer and Information Technology, China Three Gorges University, Yichang, China
†Department of Computer Science, Rowan University, Glassboro, USA
§Center for Networked Computing, Temple University, Philadelphia, USA

***Abstract*—With the development of Internet of Things (IoT) and the advance of deep learning, there is an urgent need to enable deep learning inference on IoT devices. To address the computation limitation of IoT devices in processing complex Deep Neural Networks (DNNs), partial computation offloading is developed to dynamically adjust computation offloading 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 to increase partial computation offloading flexibility, and thus further create better computation offloading solution. However, there is a trade-off between parallelism computation offloading flexibility and model parallelism overhead. Then, we discuss the optimal DNN model parallelism and the corresponding scheduling and offloading strategies in partial computation offloading. In particular, we present a Minimizing Waiting (MW) method, which explores both the FL strategy, the path scheduling strategy, and the path offloading strategy to reduce time complexity. 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 18.39 times compared with No FL (NFL) algorithm, and is very close to the optimal solution Brute Force (BF) with greatly reduced time complexity.**

***Index Terms*—FL, DNN inference, partial offloading, model parallelism**

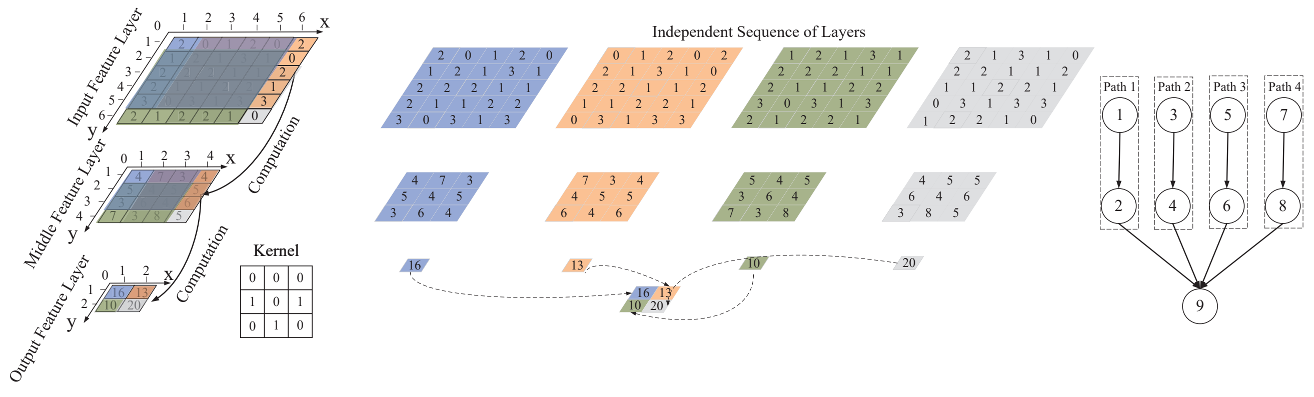
I. Introduction

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 complex tasks, including computer vision, natural language processing, machine translation and many other tasks. One of the obstacles in using deep learning in Internet of Things (IoT) systems is that IoT devices cannot provide real-time and high-precision results at the same time due to their computation resource limitation [3].

To reduce the Deep Neural Network (DNN) inference time, recent studies have explored end device-only computation, full computation offloading, and partial computation offloading. In end device-only computation, existing research accelerates DNN inference by optimizing the DNN structure or using multiple cores [4]. Putic *et al.* [5] proposed DyHard-DNNs to significantly reduce computation time, in which accelerator microarchitectural parameters are dynamically reconfigured during the execution of DNN. Microsoft and Google developed small-scale DNNs for speech recognition on mobile platforms by sacrificing the high prediction [6]. In full computation offloading, raw data is offloaded to the Edge Server (ES) directly in this category [7]–[9]. Fang *et al.* [10] introduced an alternating direction method of multipliers to prune filters in a layerwise manner, and then accelerated the DNNs on the ES. In partial computation offloading, a DNN model is decomposed into layer-level subtasks and the intermediate feature layer is offloaded to the ES by following the corresponding processing dependencies [11]. In general, the intermediate DNN layers have much smaller data sizes and thus lower transmission time [12]–[14]. Duan *et al.* [15] minimized the DNN inference time by jointly optimizing multiple DNNs partitioning and scheduling.

However, in the existing parallel DNN computing studies, the design of fine-grained partitioning of DNN models and parallel computing strategies in edge computing environment still lacks due attention. Therefore, the partial computation offloading's advantage is not maximized. In this paper, we first use 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 result [16]. As a result, we create more flexibility in partial computation offloading by scheduling small layers. However, the FL technique also brings some challenges: (1) Determining the optimal FL strategy is challenging 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 solve the aforementioned two challenges, we propose a Minimizing Waiting (MW) method, which jointly considers the impact of FL strategy, path scheduling strategy and path offloading strategy. Specifically, we first heuristically determine the path scheduling strategy and path offloading strategy. Then, we traverse the possible number of FL paths and FL



(a) Correspondence of values in DNN

(b) Paths divided by the FL technique

Fig. 1. A simple DNN computation with the FL technique

path length to explore the optimal solution. The contributions of this paper are summarized as follows:

- To our best knowledge, we are the first to use DNN model parallelism in partial computation offloading. In particular, we propose to use the FL technique to achieve DNN model parallelism without accuracy loss.
- We propose a heuristic method, MW, to obtain the approximate optimal FL strategy, path scheduling strategy and path offloading strategy with a low time complexity.
- We conduct comprehensive simulations to validate the effectiveness of the proposed method in commonly used DNNs. The results show that the DNN inference time is 18.39 times lower than existing algorithms without model parallelism.

This paper is structured as follows. Section II introduces the system model, including the problem formulation, and Section III presents the proposed method. The results of simulations experiment are described in Section IV. Finally, Section V summarizes this paper.

II. 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 introduce the problem formulation.

A. Fused-Layer Technique

The FL technique was first proposed to enable DNN parallel processing in the multi-core environment [16]. 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. Unlike previous studies on FL technique, we note that FL technique creates opportunities for parallel computing between the end device and ES, and applies FL technique to parallel edge computing environments.

Fig. 1 shows how the DNN inference result is computed by applying the FL technique. In Fig. 1(a), 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**. Fig. 1(b) shows the DNN conversion result by using the FL technique. It generates a sequence of small layers, which can be processed independently. To simplify 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's size. 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's size. The intercepted fused layer's size cannot exceed S_τ^L and S_τ^W.

In Fig. 1(b), after applying the FL technique, the DNN has 4 paths and the FL path length is 2, $S_2 = \{2, 2\}$, and $u_1 = u_2 = u_3 = u_4 = \{1, 1\}$. 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 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 intercepted fused layers' size.

B. 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. In general, we define the computation dependency relationship of layers 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, V is the total number of computation layers. We use c_v and d_v 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 partial computation offloading paradigm, 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.

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:

$$t_v^{end} = \frac{d_v}{f_{end}}, \tag{1}$$

where f_{end} [floating point operations per second (FLOPS)] is the computation resource of the end device (i.e., compute capability of graphics card).

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:

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

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) 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:

$$t_v^{es} = \frac{d_v}{f_{es}}, \tag{3}$$

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 FL path length τ), the path scheduling strategy, and the path offloading strategy. 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.

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:

$$\mathrm{T}_p(v) = \begin{cases} \max \mathrm{T}_p(v') + t_v^{end}, & \{h_{v'}, h_v\} = \{0, 0\}, \\ \max \mathrm{T}_p(v') + t_v^{tr} + t_v^{es}, & \{h_{v'}, h_v\} = \{0, 1\}, \\ \max \mathrm{T}_p(v') + t_v^{es}, & \{h_{v'}, h_v\} = \{1, 1\}, \end{cases} \tag{4}$$

where v' represents the previous computation layer that has the computation dependency on layer v, i.e., $\exists e_{v'v} \in E$. When all FL paths are computed on ES, FL paths will be fused into the fused layer whose values are the same as those normally convoluted into the fused layer. Specifically, $\mathrm{T}(v)$ is the task completion time of layer v after FL paths fused on ES, which can be formulated as:

$$\mathrm{T}(v) = \max\ \mathrm{T}(v') + t_v^{es} \tag{5}$$

To illustrate how to obtain the path offloading strategy $\mathbb{O}$ and task completion time $\mathrm{T}_p(v)$, we use a five-layer DNN as an example. As shown in Fig. 2, the FL path length is 3, and the number of FL paths is 3 (i.e, $\tau = 3$, $P = 3$). The path scheduling order $\mathbb{S} = \{s_1, s_2, s_3\} = \{1, 2, 3\}$. The three FL paths are offloaded to the ES at layer 3, layer 5 and layer 8, respectively. Therefore, the path offloading strategy $\mathbb{O} = \{3, 2, 2\}$. 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 simplicity, we assume that the CPU frequency of the end device and the ES are 1 and 2, the transmission rate between the end device and the ES is 1. Then, the task completion time of layer 2 is $\mathrm{T}_1(2) = \mathrm{T}_1(1) + t_2^{end} = 2 + 2 = 4, \{h_1, h_2\} = \{0, 0\}$. 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 $\mathrm{T}_1(3) = \mathrm{T}_1(2) + t_3^{tr} + t_3^{es} = 4 + 2 + 1 = 7, \{h_2, h_3\} = \{0, 1\}$. 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.

C. 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 (i.e., the task completion time of last layer v), $\mathbb{T}$, with computation dependency can be formulated as follows:

$$\begin{aligned} \min_{\mathbb{F},\mathbb{S},\mathbb{O}} \quad & \mathbb{T} = \max\ \mathrm{T}(v) \\ \textit{s.t.} \quad & C1: \mathrm{T}_p(v') \leq \mathrm{T}_p(v) \quad \forall e_{v'v} \in E \\ & C2: \mathrm{T}(v') \leq \mathrm{T}(v) \quad \forall e_{v'v} \in E \end{aligned} \tag{6}$$

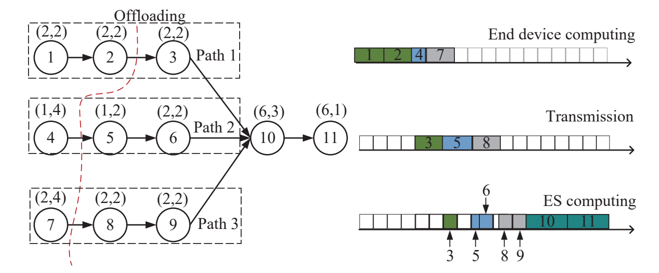


Fig. 2. Illustrating DNN inference with the FL technique in a five-layer DNN

Among them, constraints $C1$, $C2$ denote the computation dependency. The computation layer can only be computed if all of its predecessors have been computed. 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. It is proved that the above optimization problem is NP-Hard [13].

III. Minimizing Waiting Method

This section proposes a heuristic method, Minimizing Waiting (MW) to solve the above optimization problem. The FL strategy can be obtained by enumerating all possible numbers of FL paths and FL path length. Then, the intercepted fused layer's size is obtained as S_τ divided 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 path scheduling strategy and path offloading strategy. Therefore, by updating the FL strategy, we can obtain the minimum DNN inference time.

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 scheduling path and offloaded layer can be determined by using the following criterion. The fewer layers on the path computed on the end device, 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:

$$\min\left(\mathrm{T}_p(v-1)+t_v^{tr}\right),(h_{v-1}=0,h_v=1). \qquad (7)$$

Then, the first scheduling path is recorded as s_1 and the offloaded layer is recorded as o_p. If multiple offloading solutions result in the same transmission completion time, we will choose the offloading strategy which has the minimum local computation time. It 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.

Algorithm 1 Minimizing Waiting

```
Input: Neural network layers $l$ and their parameters.
Output: The minimum completion time $\mathbb{T}_{MW}^{best}$; The best
    solution $\mathbb{U}_{MW}^{best}$, $\tau_{MW}^{best}$, $\mathbb{O}_{MW}^{best}$, $\mathbb{S}_{MW}^{best}$.
1: Initialize $\mathbb{T}_{MW}^{best}$ = NULL
2: for FL path length $\tau = 1 : l$ do
3:   for The number of FL paths $P = 1 : S_\tau^L \times S_\tau^W$ do
4:     The intercepted fused layer's size is obtained as $S_\tau$
       divided into $P$ layers of the same size.
5:     for each FL path do
6:       The first scheduling path and the offloaded layer
         are determined as:
7:           $p, o_p \leftarrow \min\left(\mathrm{T}_p(v-1)+t_v^{tr}\right)$
8:       The scheduling path is recorded as $s_1$ and the
         offloaded layer $o_p$ is recorded in $\mathbb{O}$.
9:     end for
10:    for The $p$-$th$ scheduling path $p = 2 : P$ do
11:      The $p$-$th$ scheduling path and the offloaded layer
         are determined as:
12:          $p, o_p \leftarrow \min\left(\left|\mathrm{T}_{p-1}(v)-t_v^{es}-\mathrm{T}_p(v')\right|\right)$
13:      The scheduling path is recorded as $s_p$ and the
         offloaded layer $o_p$ is recorded in $\mathbb{O}$.
14:    end for
15:    According to the $\mathbb{S}$, $\mathbb{O}$, $\mathbb{U}$, the DNN inference time
       $\mathbb{T}_{MW}$ is obtained.
16:    If $\mathbb{T}_{MW}^{best}$ = NULL
17:        Update $\mathbb{T}_{MW}^{best} \leftarrow \mathbb{T}_{MW}$, $\tau_{MW}^{best} \leftarrow \tau$,
18:               $\mathbb{S}_{MW}^{best} \leftarrow \mathbb{S}$, $\mathbb{U}_{MW}^{best} \leftarrow \mathbb{U}$,
19:    End If
20:    If $\mathbb{T}_{MW} \leq \mathbb{T}_{MW}^{best}$
21:        Update $\mathbb{T}_{MW}^{best} \leftarrow \mathbb{T}_{MW}$, $\tau_{MW}^{best} \leftarrow \tau$,
22:               $\mathbb{S}_{MW}^{best} \leftarrow \mathbb{S}$, $\mathbb{U}_{MW}^{best} \leftarrow \mathbb{U}$, $\mathbb{O}_{MW}^{best} \leftarrow \mathbb{O}$.
23:    End If
24:  end for
25: end for
```

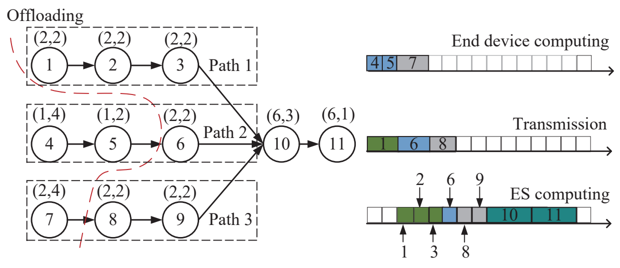
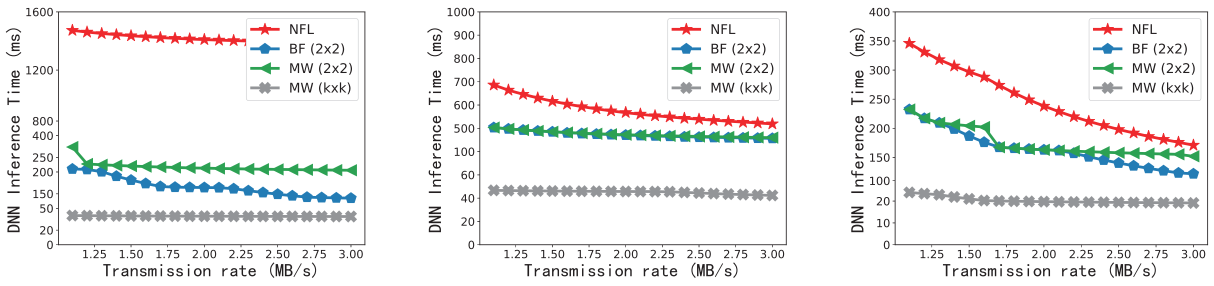


Fig. 3. Illustrating MW in a five-layer DNN

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



(a) AlexNet (b) SqueezeNet (c) YOLOv2

Fig. 4. The DNN inference time with only changing transmission speed. (a) AlexNet. (b) SqueezeNet. (c) YOLOv2.

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 and o_p can be determined by the following formula:

$$\min\left(\left|\mathrm{T}_{p-1}(v) - t_v^{es} - \mathrm{T}_p(v'-1)\right|\right), p \in \{2, 3, ..., P\}, \quad (8)$$

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 layer which has the maximum number of previous layers on the path. Then, the p-th scheduling path can compute more on the end device. Therefore, the path scheduling strategy and the path offloading strategy are obtained.

To illustrate how to obtain the solution, we use a five-layer DNN as an example. As shown in Fig. 3, the FL path length, the number of FL paths, the CPU frequency of the end device and the ES, the transmission rate are the same in Fig. 2. The minimal transmission completion time of the three FL paths are 2, 4, 4, respectively. By using MW, 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. By the same logic, path 3 is the third scheduling path, and layer 8 is the offloaded layer. That is, $o_3 = 2$. When the previous layers of layer 10 have all finished the computing, the output 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 $8 + 3 = 11$.

The pseudocode of MW is shown in Algorithm 1. Line 1 is to initialize $\mathbb{T}_{MW}^{best}$, which is the minimum DNN inference time obtained by MW . Line 4 is to obtain the intercepted fused layer's size $\mathbb{U}$. 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 6. Line 16 to 23 are to update the DNN inference time $\mathbb{T}_{MW}^{best}$, the best solution by using MW.

IV. Performance Evaluation

We conduct extensive simulations to demonstrate the effectiveness of the proposed method in five neural networks which are (1) AlexNet, (2) SqueezeNet, and (5) YOLOv2. The structure of the neural network can be obtained from [17]–[19]. Realistic network parameters are used in our experiments [20], which are shown in Table. I. We compare the performance of our proposed MW with the following benchmark algorithms: (1) $\boldsymbol{No\ FL\ (NFL)}$ **:** Partial computation offloading without the FL technique is used in this algorithm. (2) $\boldsymbol{Brute\ Force\ (BF)}$ **:** The FL technique is used in this algorithm, and the optimal solution is obtained by traversing all feasible solutions.

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) and MW (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 method.

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 [11], etc. Fig. 4 shows the simulation results. Across five different neural networks, when the transmission rate increases from 1.1 MB/s to 3 Mb/s, the DNN inference time of MW is 18.39 times less than that of NFL. However, the improvement depends on the neural network architecture. Fig. 4(a) shows the results in the AlexNet, the DNN inference time of the NFL, BF (2×2), MW (2×2), and MW $(k \times k)$, are reduced from 1470 ms, 260 ms, 335 ms, and 40 ms, to 1384 ms, 160 ms, 255 ms, and 38 ms, respectively, when the transmission rate changes from 1.1 MB/s to 3 MB/s. It can be found that in

TABLE I
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}

YOLOv2, the DNN inference time of NFL, BF (2×2), MW (2×2), and MW ($k \times k$) reduces from 346 ms, 232 ms, 232 ms, and 28 ms to 171 ms, 121 ms, 151ms and 22 ms, respectively. Therefore, the structure of neural network has a significant impact on DNN inference time.

The number of FL paths is very important. From the results in Fig. 4(a), 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 SqueezeNet, the DNN inference time of MW ($k \times k$) is reduced from 46 ms to 42 ms, respectively. Compared with MW (2×2), MW ($k \times k$) further reduces 410 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.

Overall, our approach reduces the DNN inference time well, whether in a lightweight DNN such as the AlexNet and YOLOv2 or a heavyweight DNN such as the SqueezeNet. Lightweight DNNs have low convolution steps with 1 or 2 in most neural layers, using the FL technique can reduce more DNN inference time. In the AlexNet and YOLOv2, the average DNN inference time obtained by MW ($k \times k$) are 39 ms and 40 ms, respectively, which are reduced by 1370 ms and 215 ms times compared with NFL, respectively. Heavyweight DNNs have large convolution steps or neural layers, e.g., 7 convolution steps in the SqueezeNet, 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 MW ($k \times k$) in the SqueezeNet is 44 ms, which is reduced by 523 ms compared with NFL, respectively. Therefore, the effectiveness of MW has been demonstrated.

V. Conclusion

In this paper, we presented a new solution for DNN parallelism and partial computation offloading in MEC. 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 MW method to solve the problem. Specifically, we design the MW algorithm to determine the FL strategy, path scheduling strategy, and path offloading strategy. Finally, we validated the effectiveness and superiority of the method through extensive simulation experiments, and the simulation results showed that our proposed method can reduce the DNN inference time by an average of 18.39 times compared with NFL.

References

[1] 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 of Things Journal*, vol. PP, no. 99, pp. 1–1, 2021.

[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 Communications Magazine*, vol. 56, no. 5, pp. 39–45, 2018.

[3] 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 of Things Journal*, vol. 8, no. 5, pp. 3180–3188, 2021.

[4] C. Guo, Y. Zhou, J. Leng, Y. Zhu, Z. Du, Q. Chen, C. Li, B. Yao, and M. Guo, "Balancing efficiency and flexibility for dnn acceleration via temporal gpu-systolic array integration," in *Proceedings of ACM/IEEE DAC*, 2020, pp. 1–6.

[5] M. Putic, A. Buyuktosunoglu, S. Venkataramani, P. Bose, S. Eldridge, and M. Stan, "Dyhard-dnn: Even more dnn acceleration with dynamic hardware reconfiguration," in *Proceedings of ACM/IEEE DAC*, 2018, pp. 1–6.

[6] P. Aleksic, M. Ghodsi, A. Michaely, C. Allauzen, K. Hall, B. Roark, D. Rybach, and P. Moreno, "Bringing contextual information to google speech recognition," in *Proceedings of ICCT*, 2015, pp. 468–472.

[7] Y. Chen, H. Balakrishnan, L. Ravindranath, and P. Bahl, "Glimpse: Continuous, real-time object recognition on mobile devices," in *Proceedings of ACM CENSS*, 2015, pp. 155–168.

[8] 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 *Proceedings of ACM MobiSys*, 2016, pp. 123–136.

[9] 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 *Proceedings of IEEE INFOCOM*, 2020, pp. 257–266.

[10] 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 of Things Journal*, vol. 8, no. 3, pp. 1259–1271, 2021.

[11] Y. Kang, J. Hauswald, J. Mars, C. Gao, and A. Rovinski, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proceedings of ASPLOS*, 2017, pp. 615–629.

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

[13] N. Wang, Y. Duan, and J. Wu, "Accelerate cooperative deep inference via layer-wise processing schedule optimization," in *Proceedings of IEEE ICCCN*, 2021, pp. 1–9.

[14] Y. Duan and J. Wu, "Computation offloading scheduling for deep neural network inference in mobile computing." in *Proceedings of IEEE/ACM IWQoS*, 2021, pp. 1–10.

[15] Y. Duan and J. Wu, "Joint optimization of dnn partition and scheduling for mobile cloud computing," in *Proceedings of IEEE ICPP*, 2021, pp. 1–10.

[16] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer cnn accelerators," in *Proceedings of IEEE/ACM MICRO*, 2016, pp. 1–12.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of Advances in Neural Information Processing Systems*, 2012, pp. 1–1.

[18] 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. [Online]. Available: https://arxiv.org/abs/1602.07360

[19] 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 *Proceedings of ACM Int. Symp. Field-Program. Gate Arrays*, 2018, pp. 31–40.

[20] M. Mehrabi, S. Shen, V. Latzko, Y. Wang, and F. H. P. Fitzek, "Energy-aware cooperative offloading framework for inter-dependent and delay-sensitive tasks," in *Proceedings of IEEE GLOBECOM*, vol. PP, no. 99, 2020, pp. 1–6.