DNN Acceleration In Vehicle Edge Computing With Mobility-Awareness: A Synergistic Vehicle-Edge And Edge-Edge Framework

Yuxin Zheng[a], Lin Cui[a,b,*], Fung Po Tso[c], Zhetao Li[a,b], Weijia Jia[d]

[a] *Department of Computer Science,Jinan University, Guangzhou, China*
[b] *Guangdong Key Laboratory of Data Security and Privacy Preserving*
[c] *Department of Computer Science, Loughborough University, UK*
[d] *BNU-UIC Institute of Artificial Intelligence and Future Networks, Beijing Normal University (BNU Zhuhai) and BNU-HKBU United International College, Zhuhai, China*

Abstract

In recent years, vehicular networks have seen a proliferation of applications and services such as image tagging, lane detection, and speech recognition. Many of these applications rely on Deep Neural Networks (DNNs) and demand low-latency computation. To meet these requirements, Vehicular Edge Computing (VEC) has been introduced to augment the abundant computation capacity of vehicular networks to complement the limited computation resources on vehicles. Nevertheless, offloading DNN tasks to MEC (Multi-access Edge Computing) servers effectively and efficiently remains a challenging topic due to the dynamic nature of vehicular mobility and varying loads on the servers. In this paper, we propose a novel and efficient distributed DNN Partitioning And Offloading (DPAO), leveraging the mobility of vehicles and the synergy between vehicle-edge and edge-edge computing. We exploit the variations in both computation time and output data size across different layers of DNN to make optimized decisions for accelerating DNN computations while reducing the transmission time of intermediate data. Meanwhile, we dynamically partition and offload tasks between MEC servers based on their load differences. We have conducted extensive simulations and testbed experiments to demonstrate the effectiveness of DPAO. The evaluation results show that, compared to performing computation solely on the vehicle, DPAO reduces the latency of DNN tasks by 1.7x. DPAO with queue reservation can further reduce the task average completion time by 10%.

Keywords: Vehicular edge computing, deep neural networks, task partitioning.

1. Introduction

With vehicles becoming more and more intelligent, recent years have seen a large number of emerging vehicular applications, ranging from information services to driving safety, such as speech recognition [1], image tagging [2], lane detection [3], and traffic monitoring [4]. For example, self-driving vehicles are equipped with cameras to monitor the surrounding environment, which sends video analytics to steer wheels and feedback signals to pedals [5]. Most of these applications are based on Deep Neural Networks (DNNs), characterized by high accuracy and resource consumption [6]. However, due to limited computation resources on vehicles, such DNN tasks struggle to be completed within an acceptable time threshold [7]. One approach to tackle this is to offload tasks to remote servers in cloud data centers, but it will incur a longer latency for data communication over wide area networks [8].

A more promising approach is to offload DNN tasks to MEC (Multi-access Edge Computing) servers in the proximity of mobile vehicles through the radio access network [9]. This approach is also known as Vehicular Edge Computing (VEC), as it integrates MEC into vehicular networks [10]. Vehicles usually have limited computation resources, while Road-Side Units (RSUs) equipped with MEC servers can provide sufficient resources for vehicles with low latency. Fig. 1 shows an example of VEC in which a well-trained DNN model has been pre-deployed on vehicles and MEC servers. Vehicles can offload their computation-intensive and latency-sensitive DNN tasks to RSUs to reduce task completion time (see Section 3.2 for details).

*Corresponding author
[1]E-mail addresses: zhengyuxin@stu2019.jnu.edu.cn (Y. X. Zheng), tcuilin@jnu.edu.cn (L. Cui), p.tso@lboro.ac.uk (F. P. Tso), liztchina@hotmail.com (Z. T. Li), jiawj@uic.edu.cn (W. J. Jia)

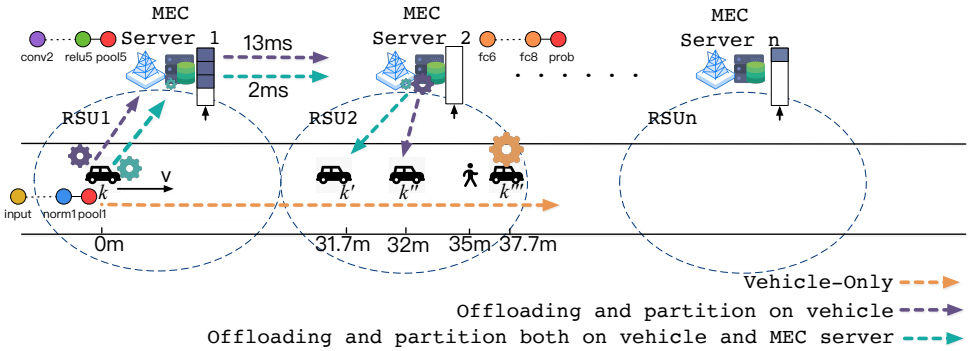

Figure 1: The DNN tasks offloading in vehicular edge network.

However, offloading the whole DNN task to the MEC server still causes a significant fraction of latency due to a large amount of data to be transmitted. A DNN model consists of a number of layers. The input data are passed through DNN layer by layer. Previous studies [11, 12, 13] have shown that the output data size of some intermediate layers of DNN is much smaller than the original input data (see Fig. 2). So some works optimize DNN tasks through partitioning DNN layers [14, 15, 16, 17, 18]. Thus, by offloading a portion of DNN layers to RSUs, such schemes can accelerate DNN computation (see Fig. 3). However, two significant challenges need to be considered when applying such schemes to VEC for DNN optimization.

First, due to the high mobility of vehicles, they may soon leave the communication range of the MEC server after offloading DNN tasks. This means that results need to be relayed by other MEC servers in the direction of vehicle movement. Thus, both the *moving direction* and *speed of vehicles* need to be considered when determining appropriate MEC servers for DNN tasks partition and offloading strategy. Second, uneven vehicle distribution caused by dynamic road traffic conditions can easily cause an *unbalanced load* on MEC servers. This means that task queuing time for servers in hotspot areas may be longer than others. Therefore, how to coordinate the load differences between servers and reduce the average completion time of tasks is another major challenge.

To overcome the above challenges, in this paper, we propose a synergistic vehicle-edge and edge-edge framework for DNN partitioning and offloading, utilizing the mobility of vehicles. First, considering the enhanced computational capabilities of the MEC server and the varying sizes of output data of the intermediate layers, each DNN task is partitioned between the vehicle and the MEC server to accelerate DNN computations and reduce the transmission time of intermediate data. Second, we offload tasks to MEC servers in the direction of the vehicle movement and take into account the impact of the speed of the vehicle on end-to-end latency. Third, based on the load of MEC servers and predicted locations of vehicles, DNN tasks on each MEC server are further dynamically partitioned and offloaded among MEC servers to accelerate the computation of each DNN task. In addition, we explore layer-by-layer partition and bidirectional offloading mechanisms to improve the performance of DPAO. We have used AlexNet [19] and CaffeNet [20] to demonstrate the effectiveness of the proposed partition and offloading scheme in both simulation and testbed experiments, respectively.

To sum up, the contributions of this paper are as follows:

1. Considering vehicle mobility and load imbalance among MEC servers, we exploit vehicle-edge and edge-edge collaboration to partition and offload tasks. Based on the load difference, a task is allowed to be further partitioned after it is offloaded to the MEC server. The optimization goal is to minimize the average completion time of all tasks.
2. We propose several DNN partitioning and offloading (DPAO) algorithms for VEC. Each partition decision aims to reduce the average completion time of all tasks in the current vehicle or MEC server. To mitigate the impact of fluctuating queueing times for the MEC server on the partitioning results, a reservation strategy is introduced to improve the performance further.
3. Extensive evaluations with the AlexNet model are conducted. We also implemented a testbed based on Raspberry Pi to demonstrate the effectiveness of DPAO. The results show that compared to computing on the vehicle, DPAO can reduce task average

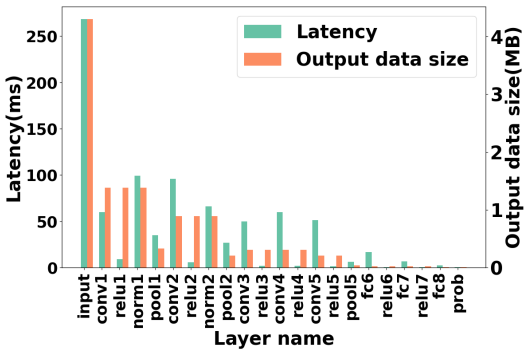

Figure 2: Latency and output data size of each layer of AlexNet[1].

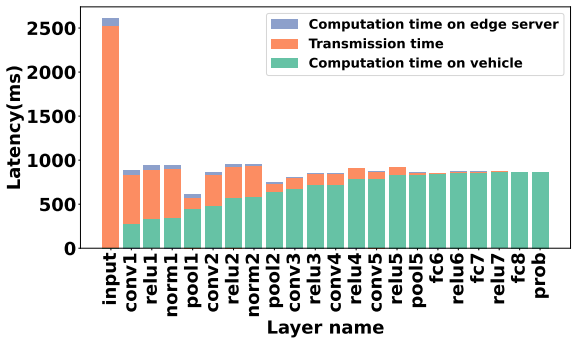

Figure 3: Latency breakdown when partitioning at different layers of AlexNet[2].

completion time by 1.7x.

The remainder of this paper is organized as follows. Section 2 introduces related works for both DNN partition and offloading. Background and a motivating example are explained in Section 3, followed by the problem model in Section 4. Section 5 and Section 6 introduce the proposed DPAO scheme and a queue reservation optimization mechanism. Evaluations of DPAO in both simulation and testbed experiments are provided in Section 7. Finally, the paper is concluded in Section 8.

2. Related Works

Model Partition. To alleviate the pressure of resource-constrained end devices, partitioning a complete DNN model and offloading the computation-intensive part to powerful MEC servers or nearby mobile devices will reduce inference time [25]. Neurosurgeon [11] represents a landmark effort. First, it establishes a regression prediction model for the execution time and energy consumption of each layer of DNN. Then, Neurosurgeon evaluates the inference latency of partitioning the DNN between the end device and the MEC server in each candidate point. Finally, it selects an optimal partition layer to meet the latency requirement or energy requirement. Fan et al. [26] optimized DNN partition, communication, and resource allocation to minimize the processing delay of all the deep learning tasks in a multi-base station (BS) and multiservice edge–cloud-assisted IoT environment. Hu et al. [27] described the deep learning model with a directed acyclic graph and then used the min-cut method to minimize inference delay under the lightly loaded condition and maximize throughput under the heavily loaded condition. Su et al. [28] investigated how to jointly optimize DNN partition and computing resource allocation in energy-constrained hierarchical edge-cloud systems to minimize the long-term average end-to-end delay of multiple types of deep learning tasks. They proposed a DNN partition and resource allocation algorithm based on deep deterministic policy gradient (DDPG) to dynamically decide the DNN partition by observing the environment such as task arrival rate and energy budgets. To reduce the reliance on remote clouds, Wang et al. [24] proposed a dynamic resource allocation scheme to select the best partition point of DNN inference tasks by vehicle-edge collaborative computing to minimize the average response time of all DNN tasks. Kim et al. [29] proposed a DNN partition placement and resource allocation strategy that considers different processing powers, memory and battery levels for heterogeneous IoT devices. The goal is to optimize the utility of running multiple DNN-based applications in IoT environments while addressing the challenges associated with limited resources and heterogeneity. Liao et al. [30] proposed a partitioning-and-offloading scheme for the heterogeneous tasks-server system to reduce the overall system latency and energy consumption on DNN inference.

Existing works mainly focus on fixed partitions (e.g., partitioning the DNN into two or three parts and executing them in the order of local, edge, and clouds). Considering the uneven load of edge servers caused by road traffic conditions, our work not only enables the partitioning of DNN between the vehicle and the edge server but also allows for dynamic division of the DNN among multiple edge servers based on server loads.

Task offloading in vehicular edge computing. Task offloading methods in vehicular edge computing can be divided into two categories: vehicle-based offloading

Table 1: Comparison of related works on DNN partition in VEC.

Works	Scenarios	DNN partition	Vehicle-edge synergy	Edge-edge synergy	Vehicle mobility
Zhang et al. (2017) [21]	Single task	✗	✓	✓	✓
MDPO (2021) [22]	Single tasks	✓	✓	✓	✓
Dai et al. (2019) [23]	Multiple tasks	✗	✓	✗	✓
Wang et al. (2021) [24]	Multiple tasks	✓	✓	✗	✗
DPAO (this paper)	Multiple tasks	✓	✓	✓	✓

[31] and server-based offloading [21, 23, 32, 33, 22, 34]. Server-based offloading is to offload tasks to RSUs on both sides of the road, which can provide rich computation and storage resources. Liu et al. [32] formulated the task offloading problem as a multi-user computation offloading game and proved the existence of Nash equilibrium (NE) of the game.

Considering the mobility of the vehicle, it can be challenging to send computation results back to the vehicle from the original server. This is because the vehicle may have moved out of range of the original server by the time the computation is complete. To address this challenge, Zhang et al. [21] proposed an efficient predictive combination-mode relegation scheme, where the tasks are adaptively offloaded to the MEC servers through direct uploading or predictive relay transmissions. In direct uploading mode, the computation result needs to be transmitted from the RSU that executed the task to the RSU where the vehicle arrives, which goes through the wireless backhauls. This work only transmitted computation results between MEC servers without performing task partitioning and offloading to utilize the computing resources of MEC servers more efficiently. Tian et al. [22] developed a mobility-included DNN partition offloading algorithm (MDPO) to adapt to the user's mobility. This method constructed a directed acyclic graph (DAG) to represent the collaborative execution paths by the mobile device and the MEC server at the layer level and used the shortest path algorithm to find the optimal partition offloading decision. Zeng et al. [35] proposed an efficient edge-cloud collaborative caching strategy in the field of vehicular edge computing to minimize the acquisition delay of supporting data.

In the scenario of multi-task offloading, if all vehicles choose the same server for offloading, that server may become overloaded, causing some tasks to exceed the permissible latency threshold. To avoid this problem, Dai et al. [23] proposed integrating load balancing with offloading, which allows each vehicle to select one VEC server and offload parts of its computation task for edge execution. If a vehicle tries to offload a task to an RSU that is located ahead of the vehicle's current position, the task cannot be offloaded until the vehicle reaches the RSU's communication range. This delay may result in additional travel time for task completion.

Table 1 shows the comparison between this paper and the state-of-the-art in terms of task scenarios, model partition, vehicle-edge synergy, edge-edge synergy and vehicle mobility. Most of the work focused on partitioning and offloading either a single task or multiple tasks between vehicles and the MEC server. However, in multi-task vehicle edge computing, how to better utilize MEC server resources and leverage their imbalanced load to accelerate DNN tasks still needs to be investigated. In addition to partitioning the DNN tasks of multiple vehicles between vehicles and MEC servers, we also further enable DNN tasks to be partitioned and offloaded between MEC servers based on vehicle mobility and server load. Therefore, in this paper, we jointly study DNN model partitioning and offloading in vehicular edge computing scenarios by vehicle-edge synergy and edge-edge synergy to reduce average task completion time.

3. Backgrounds and Motivating Examples

3.1. Backgrounds

DNNs consist of multiple layers of interconnected nodes that process input data to produce output predictions. Each layer performs a specific computation on the input data and passes the result to the next layer until the final output is produced. The computational requirements of each layer can vary depending on its position in the network and the size of its input and output data.

A vehicle equipped with ten high-resolution cameras can generate 2 *Gpixels/s* of data, requiring approximately 250 trillion operations per second (TOPS) for processing [36]. Meanwhile, L4 and L5 autonomous driving require approximately 320 TOPS and 4000+

TOPS, respectively [37]. However, NVIDIA's edge-accelerator kit AGX Orin only delivers a theoretical 275 TOPS of performance [38].

To optimize DNN task execution in vehicular networks, it is necessary to partition DNN layers between vehicles and MEC servers based on their computational requirements and data size. This partitioning can help reduce task execution time and minimize the transmission time of intermediate data between vehicles and MEC servers.

3.2. Motivating Examples

To show the motivation and the main idea of this paper, a simple example is provided in Fig. 1. The vehicle is driven at a speed of 120 kilometers per hour. There is a pedestrian 35 meters away from the vehicle. The vehicle collects video information through the camera and initiates a DNN task, which should respond in time. In this example, we use the Alexnet model, which is already pre-deployed on vehicles and MEC servers.

There are three possible different strategies. One intuitive way is to compute the entire DNN task on the vehicle. This method does not require any data transmission but is limited by the computation capability of the vehicle, leading to a long computation time of about 1133ms (i.e., 300ms+ 833ms)[3]. When the computation result is obtained, the vehicle has already driven 37.7 meters and collided with the pedestrian, as shown for vehicle k''' in Fig. 1.

DNN tasks can be partitioned into two parts. The former part of the DNN task is computed on the vehicle, and the latter part is offloaded to MEC servers because MEC servers have more computation power than vehicles. Suppose the computation time on the vehicle is 433ms (e.g., from the *input* layer to the *pool1* layer). The time for data transmission among vehicles and MEC servers needs to be considered. The transmission time is determined by the amount of data. In this case, the transmission time of the output data of the *pool1* layer is 133ms (i.e., 170KB over 10$Mbps$). In this example, MEC server 1 is assumed to be busy with an average queuing delay of 50ms, while MEC server 2 is idle. When the task is offloaded to the MEC server 1, there are two approaches for offloading: partitioning DNN again between two MEC servers or offloading completely to the MEC server 2. For ease of description, we assume that MEC servers can only process one task at a time based on the first-in-first-out (FIFO) principle.

On the MEC server 1, in case of complete offloading, the data of the DNN task received from the vehicle is forwarded completely to the MEC server 2 (i.e., dashed purple line in Fig. 1). After the DNN task is computed by the MEC server 2. It returns the result to the vehicle. The whole processing time from initiating the DNN request to the time receiving results is about 963ms (i.e., 300ms+433ms+133ms+50ms+13ms+33ms+1ms)[4]. Thus, the DNN result can be obtained after the vehicle has driven 32m before reaching the pedestrian's position, as shown for vehicle k'' in Fig. 1. This shows that the completion time of DNN can be reduced with the help of MEC servers.

Alternatively, the DNN task can be partitioned into two parts again on MEC server 1 (i.e., the dashed green line in Fig. 1). Suppose the computation time of the two parts on the MEC server is 23ms (e.g., from the *conv2* layer to the *pool5* layer) and 10ms (e.g., from the *fc6* layer to the *output* layer) respectively, and the transmission time of the output data of the fc6 layer between adjacent MEC servers is 2ms (e.g., 26KB over 100Mps), which is less than the time of transmitting data received from vehicle. So, when the task enters the queue of the MEC server 1, layers of the first part are computed on MEC server 1, and then the intermediate result is transmitted to MEC server 2 for subsequent computations. In this way, the whole process time of the DNN task is about 952ms (i.e., 300ms+433ms+133ms+50ms+20ms+2ms+13ms+1ms)[5]. The vehicle has driven 31.7m when obtaining the results, leaving more response time and distance for the driver, as shown for vehicle k' in Fig. 1.

[1]Results are measured on a docker container with an Intel Xeon E5-2698 2.20GHz processor.

[2]In this experiment, we used a docker container with an Intel Xeon E5-2698 2.20GHz processor as the vehicle, and another docker container, which is equipped with an Intel Xeon E5-2698 2.20GHz processor and a Tesla V100-DGXS GPU, as the MEC server.

[3]300ms is waiting time on vehicle. 833ms is computation time on vehicle (e.g., measured on Xeon E5-2698 2.20GHz processor).

[4]300ms is the waiting time on vehicle. 433ms is the computation time on vehicle. 133ms is the transmission time from vehicle to MEC server 1. 50ms is the waiting time on MEC server 1. 13ms is the transmission time from MEC server 1 to MEC server 2 (e.g., 170KB transmitted over 100$Mbps$). 33ms is the computation time on MEC server 2 (e.g., measured on a docker container equipped with an Intel Xeon E5-2698 2.20GHz processor and a Tesla V100-DGXS GPU). Finally, 1ms is the transmission time from MEC server 2 to vehicle (e.g., the result of 10KB transmitted over 10$Mbps$).

[5]Waiting time on vehicle is 300ms. 433ms is the computation time on vehicle. 133ms is the transmission time from vehicle to MEC server 1. 50ms is the waiting time on MEC server 1. 20ms is the computation time on MEC server 1. 2ms is the transmission time from MEC server 1 to MEC server 2. 13ms is computation time on MEC server 2. Finally, 1ms is transmission time from MEC server 2 to vehicle.

Through this motivating example, we can derive two observations: (1) Partitioning and offloading tasks between vehicles and MEC servers can take advantage of different computational capabilities of the MEC servers to reduce task completion time. (2) Selecting appropriate partition points and offloading tasks between edge servers can reduce task transmission time compared to direct offloading.

4. Problem Model

This section introduces the problem model for DNN partition and offloading with respect to both vehicle-edge and edge-edge collaboration in VEC. The notations used in the paper are listed in Table 2.

4.1. VEC Network

The VEC scenario is shown in Fig. 1. There are several Road-Side Units (RSUs)[6] deployed along the road. The distance between two adjacent RSUs is M. We assume that each RSU has the same radio power. The radio coverage areas of adjacent RSUs may partially overlap with others [39]. Each vehicle communicates with one RSU at a time based on transmission power. Each RSU is equipped with an MEC server with limited computation capacity. RSUs can communicate with each other through dedicated communication channels such as Wi-Fi or cellular networks [40].

Vehicles on the road follow the Switched Batch Bernoulli Process (SBBP) [41]. Without loss of generality, all vehicles are assumed to drive at a speed of V. We assume that the vehicle is at the leftmost point of the communication range of RSU. The vehicle communicates with the RSU for M/V time before moving to the next RSU. The initial position of the vehicle on the road is random, hence the duration of time it takes to pass through an RSU can vary between 0 and M/V. Each vehicle generates a homogeneous DNN task every specific time interval, e.g., the front camera takes a photo every $100ms$ [42]. Each DNN task contains L layers. Denote d_l to be the predicted size of the output data by the l-th layer and d_0 is the size of the original input data for the task, i.e., the data required by the input layer of the DNN. Let c_l be the amount of computation resources required by the l-th layer (in terms of floating-point operations per second (FLOPS)). Multiple tasks are queued before being processed by a vehicle. A vehicle only processes one task at a time. Denote N^v to be the number of tasks in the queue on a vehicle and the i-th task is defined as D_i:

$$D_i = \{(d_l, c_l) | l = 1, 2, \ldots, L\}, \tag{1}$$

where $i \in \{1, 2, .., N^v\}$.

The DNN model has been pre-deployed on all vehicles and MEC servers. Thus, the computation of each DNN task can be completed locally or offloaded to MEC servers. The queue of DNN tasks can build up at both vehicles and MEC servers. Varied queue schedule schemes can be applied. For simplicity, we assume all vehicles and MEC servers use the FIFO rule on queue.

4.2. Vehicle-Edge Collaboration

DNN tasks are both computationally intensive and delay sensitive. Vehicles normally have limited computation resources. Thus, each vehicle can reduce the average task completion time in the task queue of the current vehicle by offloading DNN tasks to resource-rich MEC servers. The task completion time refers to the time from when the DNN task is generated on a vehicle to when the vehicle receives the computation result.

Let u_v and u_e be computation resources provided by a vehicle and a MEC server per unit time, in terms of FLOPS, respectively. Specially, $u_v < u_e$ since the MEC server has more computation resources than the vehicle. On a vehicle, each DNN task can be partitioned into two parts. One is computed locally by the vehicle, and the other one is offloaded to the linked MEC server. Let p_i^v be the partition layer of task D_i on the vehicle. So the 1-st layer to the p_i^v-th layer are executed locally on the vehicle, and the $(p_i^v + 1)$-th layer to the L-th layer are offloaded to the linked MEC server[6].

The total task completion time mainly consists of the following parts: First, the execution time t_i^v from the 1-st layer to the p_i^v-th layer on vehicle, including the computation time for these layers and the waiting time in the queue of the vehicle. Waiting time is the sum of the computation time required for tasks before task D_i in the queue of the current vehicle.

Let a_i^v denote the time when the i-th task arrives at the vehicle. The departure time of the i-th task can be formulated as:

$$d_i^v = \begin{cases} \max\{d_{i-1}^v, a_i^v\} + \sum_{l=1}^{l=p_i^v} \frac{c_l}{u_v} & i>1 \\ a_1^v + \sum_{l=1}^{l=p_1^v} \frac{c_l}{u_v} & i = 1 \end{cases}. \tag{2}$$

[6]The terms RSU and MEC server are used interchangeably in this paper.

[6]The second part of the task may be further partitioned and offloaded by the linked MEC server (see Section 4.3 for details).

Table 2: Notations and parameters.

Notations	Descriptions
M	Distance between MEC servers
V	Speed of vehicle
N^v	Number of task in vehicle
N_x^e	Number of task in MEC server x
D_i	The i-th task of the vehicle or MEC server
L	Total number of layers of DNN
d_l	Output data size of the l-th layer
c_l	Computation resources required by the l-th layer
a_i^v	The time when the i-th task arrives at the vehicle
a_i^e	The time when the i-th task arrives at the MEC server
d_i^v	The time when the i-th task leaves the vehicle
d_i^e	The time when the i-th task leaves the MEC server
u_v, u_e	Computation resources provided byvehicle and MEC server
r_v	Bandwidth between vehicle and MEC server
r_e	Bandwidth between between MEC servers
p_i	Partition layer of the i-th task of vehicle
p_i^v, p_i^e	Partition layer of the i-th task in vehicle and MEC server
q_i^v, q_i^e	Queuing time of i-th task in vehicle and MEC server
$q_{i,x}^e$	Queuing time of i-th task in MEC server x
t_i^v, t_i^e	Execution time of task i in vehicle and MEC server
t_i^u	Data upload time from vehicle to MEC server
t_i^d	Data download time from MEC server to vehicle
t_i^c	The time to obtain final result in MEC server
t_i^b	The time to transmit final result to vehicle
T_{V_l}, T_{E_l}	Computation time of l-th layer in vehicle and MEC server

We can deduce that the waiting time of the i-th task:

$$q_i^v = \begin{cases} \max\{d_{i-1}^v - a_i^v, 0\} & i>1 \\ 0 & i=1 \end{cases}. \tag{3}$$

The processing time of the i-th task in the vehicle is:

$$t_i^v = \begin{cases} \sum_{l=1}^{l=p_i^v} \frac{c_l}{u_v} + q_i^v & 1 \le p_i^v \le L \\ q_i^v & p_i^v = 0 \end{cases}, \tag{4}$$

where q_i^v is the waiting time of task D_i in the queue of vehicle. Specifically, $t_i^v = 0$ if $p_i^v = 0$ means the entire DNN task is offloaded to the MEC server without partition.

Second, the upload time for intermediate data output by the p_i^v-th layer, which is denoted by t_i^u.

$$t_i^u = \begin{cases} d_{p_i^v}/r_v & 0 \le p_i^v < L \\ 0 & p_i^v = L \end{cases}, \tag{5}$$

where $d_{p_i^v}$ is output data size of p_i^v-th layer. r_v is the data transfer rate between the vehicle and the MEC server.

Third, the execution time t_i^e from the (p_i^v+1)-th layer to the L-th layer on the MEC server, including the computation time of these layers and the waiting time in the queue of MEC server. The vehicle can obtain the waiting time of the MEC server through periodic communication.

$$t_i^e = \begin{cases} \sum_{l=p_i^v+1}^{l=L} \frac{c_l}{u_e} + q_i^e & 0 \le p_i^v < L \\ 0 & p_i^v = L \end{cases}, \tag{6}$$

where q_i^e is the waiting time of task D_i in queue of the linked MEC server. Similarly, $t_i^e = 0$ if $p_i^v = L$, indicating that there is no offloading.

A vehicle moving quickly might have traveled through several MEC servers at the moment when the DNN task is completed. In this case, the final result needs to be forwarded by multiple MEC servers before being sent back to the vehicle. Since many network devices, such as routers and switches, can provide line rate switching, forwarding time can be ignored and only transmission time is considered here.

According to Equation (4)~(6), the time to calculate

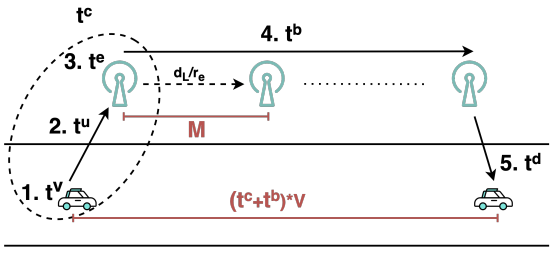

Figure 4: The problem model.

the final result is $t_i^c = t_i^v + t_i^u + t_i^e$. During this period, the distance traveled by the vehicle is $t_i^c * V$.

Denote the size of the final output of DNN task D_i is d_L. Therefore, the transmission time of the final result between two adjacent MEC servers is d_L/r_e, where r_e is the data transfer rate between MEC servers.

Suppose that it takes t_i^b time to forward the final result of task D_i to catch up with the vehicle, i.e., both the final result and vehicle are located in the same MEC. Considering that the final result should arrive at the MEC server no later than the vehicle, the following equation needs to be satisfied:

$$\lceil \frac{(t_i^c + t_i^b) * V}{M} \rceil = \lfloor \frac{t_i^b}{d_L/r_e} \rfloor. \tag{7}$$

As shown in Fig. 4, the most number of MEC servers that the vehicle passes through during DNN task computation and transmission $(t_i^c + t_i^b)$ is shown on the left side of Equation (7). The least number of MEC servers that the computation result should be forwarded to catch up with the vehicle is shown on the right side of Equation (7). Therefore, in the worst case, the following equation holds:

$$\frac{(t_i^c + t_i^b) * V}{M} = \frac{t_i^b}{d_L/r_e} - 1. \tag{8}$$

Then the total transmission time of the final result can be derived:

$$t_i^b = \frac{d_L(Vt_i^c + M)}{r_e M - V d_L}. \tag{9}$$

This ensures that the computation result can catch up with the vehicle within t_i^b. At this time, if the vehicle is within the RSU communication range where the computation result is located, the computation result can be transmitted to the vehicle; otherwise, the computation result is ahead of the vehicle and will be transmitted when the vehicle arrives.

Similarly to data upload, the time to download the final result from the MEC server to the vehicle is t_i^d:

$$t_i^d = \begin{cases} d_L/r_v & 0 \le p_i^v < L \\ 0 & p_i^v = L \end{cases}. \tag{10}$$

So the completion time of the i-task on the vehicle is:

$$T_i^v = t_i^v + t_i^u + t_i^e + t_i^b + t_i^d. \tag{11}$$

The objective of a vehicle is to minimize the average completion time of all tasks on the queue of the vehicle.

$$\min \frac{1}{N^v} \sum_{i=1}^{i=N^v} T_i^v \tag{12}$$

$$s.t. \quad 0 \le p_i^v \le L \tag{13}$$

4.3. Edge-Edge Collaboration

Since loads of MEC servers are different and dynamically changing, when an offloaded task arrives at one MEC server, it may need to be partitioned and offloaded again to the next MEC server with lower loads to reduce task completion time. Considering two adjacent MEC servers S_x and S_{x+1} which have N_x^e and N_{x+1}^e tasks in the queue, respectively. These tasks are processed in the FIFO manner. When task D_i ($i \in \{1, 2, .., N_x^e\}$) on MEC server S_x is offloaded to MEC server S_{x+1}, the waiting time of task D_i on S_x is $q_{i,x}^e$. Waiting time is the sum of the time required for other tasks in the queue of the MEC server to compute.

Suppose task D_i in the queue of S_x is partitioned into two parts in the p_i^e-th layer (assume the task has been divided once in the p_i^v-th layer in thevehicle). Thus, the (p_i^v+1)-th layer to p_i^e-th layer are executed in S_x and the $(p_i^e + 1)$-th layer to the L-th layer are offloaded to next MEC server S_{x+1}.

Hence, the total processing time on MEC servers includes the execution time from the $(p_i^v + 1)$-th layer to the p_i^e-th layer on S_x, the transmission time for output data of the p_i^e-th layer and the execution from $(p_i^e + 1)$-th layer to the L-th layer on S_{x+1}. For simplicity, all MEC servers are assumed to be homogeneous, i.e., they have the same computation capacity. Then, the total time for the task offloading from S_x to S_{x+1} is given as:

$$T_i^e = \begin{cases} \sum_{l=p_i^v+1}^{l=L} \frac{c_l}{u_e} + q_{i,x}^e + \frac{d_{p_i^e}}{r_e} + q_{i,x+1}^e & p_i^v \le p_i^e < L \\ \sum_{l=p_i^v+1}^{l=L} \frac{c_l}{u_e} + q_{i,x}^e & p_i^e = L \end{cases}. \tag{14}$$

$q_{i,x}^e$ is waiting time of task D_i in current MEC server, which can be expressed as:

$$d_i^e = \begin{cases} \max\{d_{i-1}^e, a_i^e\} + \sum_{l=p_i^v+1}^{l=L} \frac{c_l}{u_e} & i > 1 \\ a_1^e + \sum_{l=p_1^v+1}^{l=L} \frac{c_l}{u_e} & i = 1 \end{cases}, \tag{15}$$

$$q_{i,x}^{e} = \begin{cases} \max\{d_{i-1}^{e} - a_{i}^{e}, 0\} & i>1 \\ 0 & i=1 \end{cases}. \tag{16}$$

$q_{i,x+1}^{e}$ is the waiting time of task D_i in the next MEC server, which can be obtained through periodic communication exchanges among MEC servers. $d_{p_i^e}$ represents output data size of the p_i^e-th layer. Specially, $T_i^e = \sum_{l=p_i^v+1}^{l=L} \frac{c_l}{u_e} + q_{i,x}^e$, if $p_i^e = L$. This means that task D_i is only calculated on the MEC server S_x. In addition, layers of task D_i that are offloaded to S_{x+1} may be further partitioned and offloaded to the following MEC servers to reduce the average task completion time.

The objective for an MEC server S_x is to minimize the average total processing time of all tasks through offloading:

$$\min \frac{1}{N_x^e} \sum_{i=1}^{N_x^e} T_i^e \tag{17}$$

$$s.t. \quad p_i^v \le p_i^e \le L \tag{18}$$

It can be seen that the partition points are integers, and the queuing times are nonlinear. Therefore, both Problem (12) and Problem (17) are complex pure integer nonlinear programming (PINLP) problems, which are NP-hard in general [43]. In this paper, we propose a DPAO Scheme to solve these problems.

5. DPAO Scheme

This section proposes a low complexity DNN Partitioning And Offloading (DPAO) scheme to optimize task completion time through both vehicle-edge and edge-edge collaboration.

5.1. DPAO Overview

Fig. 5 shows the overall architecture of DPAO, which depicts both the vehicle-edge collaboration and the edge-edge collaboration.

DPAO is decentralized, allowing each vehicle and MEC server to make partitioning and offloading decisions locally. The *DNN analyzer* runs a DNN task to measure the execution time and the output data size of each layer of the DNN task (see Section 7.1 for details), and delivers the measurement results to the offloading controller. Note that tasks with different original data sizes have the same computing resource requirements and output data size for each layer in the same DNN processing [44]. The *state synchronizer* is used to synchronize the information between the vehicle and the MEC server or between the MEC servers, such as the queuing time and computation time of each layer of the DNN model, etc. The *offloading controller* decides at which layer to partition the DNN task based on the information from the DNN analyzer and the state synchronizer. The MEC server will periodically send the status information required for partition decisions to adjacent MEC servers or vehicles within the communication range. To reduce task completion time, vehicles can partition the original DNN task into a maximum of two parts and offload the second part to the linked MEC server.

The unbalanced distribution of vehicles may cause the load of the current MEC server to be greater than that of the next one. To reduce the completion time of all tasks, such as the waiting time of other tasks on the current MEC server, the linked MEC server can further partition the offloaded task into a maximum of two pieces and offload the second portion to the subsequent MEC server. This process will continue until the whole DNN task is finished.

During the above process, choosing a suitable partition layer is crucial for reducing task completion time. Since partition decisions are made by each vehicle and MEC server, DPAO designs both vehicle-edge partition algorithm and edge-edge partition algorithm. The goal of both algorithms is to reduce the average task completion time for all tasks in the local queue (vehicle or MEC server) through partition and offloading of DNN tasks.

Denote T_{V_l} and T_{E_l} ($l = 1, 2 \ldots L$) to be the computation time of l-th layer of DNN on vehicle and MEC server, respectively. The output data size of l-th layer is d_l. For a DNN task, the reward function is defined to be the total reduction in the task completion time for each possible partition decision.

$$Reward_l = ReducedTime_l - AddedTime_l, l = 1, 2, \ldots, L. \tag{19}$$

AddedTime refers to additional network transmission and queuing time caused by the partition decision. *ReducedTime* represents the decrease in task completion time for all tasks, e.g., the decrease in waiting time for tasks queued after the offloaded task. Consequently, $Reward_l$ indicates the variation in the average completion time of the task partitioned in the l-th layer. The goal of each partition decision is to maximize the reward function for local queue tasks.

5.2. Vehicle-Edge Partition

On a vehicle, DPAO will process the task from the head of its queue, say D_i, to make partition and offloading decisions for the task. The main operations of the vehicle-edge partition are described in Algorithm 1, which is executed on each vehicle. It will first analyze

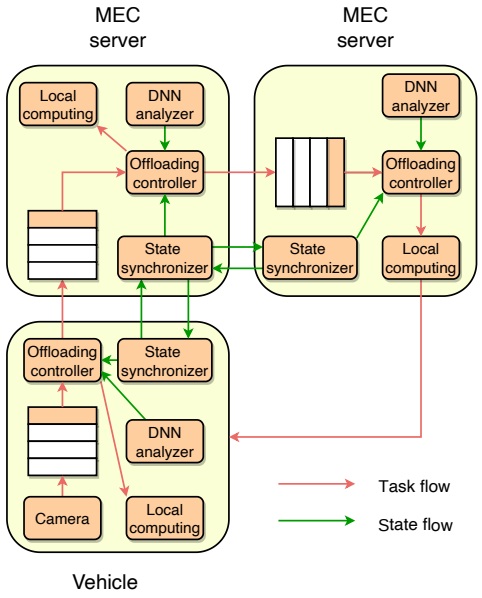

Figure 5: The architecture of DPAO. The task flow represents the computation process of the task, including task uploading, task offloading, and task downloading. The state flow represents the process of synchronizing the state of adjacent devices.

Algorithm 1 DPAO for vehicle-edge DNN partition

Input: D_i, L, T_V, T_E, d_l, r_v, r_e, V, M, N^v, q_i^e
Output: p_i^v: the selected layer of partition decision

1: **for** l in $1...L$ **do**
2: **if** $d_l/r_v + \sum_{j=1}^{l} T_{V_j} < q_i^e$ **then**
3: $AddedTime[l] \leftarrow q_i^e - \sum_{j=1}^{l} T_{V_j} - d_l/r_v$
4: **else**
5: $AddedTime[l] \leftarrow d_l/r_v$
6: **end if**
7: $t_i^b \leftarrow$ Calculate the time to forward the computation result back to the vehicle according to Equation (9)
8: $AddedTime[l] += t_i^b$
9: $ReducedTime[l] \leftarrow \sum_{j=l+1}^{L}(T_{V_j} - T_{E_j}) + (N^v - 1) * \sum_{j=l+1}^{L} T_{V_j}$
10: **end for**
11: $p_i^v \leftarrow \arg\min_{l=1,...L}(ReducedTime[l] - AddedTime[l])$

the task completion time when it is partitioned in each possible layer (line 1∼10). If the task is partitioned in the l-th layer, the vehicle will process the 1st layer to the l-th layer. The output data of the l-th layer is then transmitted to the linked MEC server for the processing of subsequent layers. This task will be first pushed to the end of the MEC server's task queue.

To obtain the suitable layer for partition and offloading, DPAO calculates rewards for each potential partition layer. The $AddedTime$ and $ReducedTime$ are obtained as follows:

The added time. Compared to local processing, offloading a task to the MEC server will introduce an additional transmission time from the vehicle to the MEC server and waiting time on the MEC server. The queuing time q_i^e on the MEC server can be obtained through periodical communication between vehicle and MEC server. Since the first portion of the task needs to be first processed locally in the vehicle before being offloaded, when the partitioned task arrives at the MEC server, the remaining waiting time of the queue becomes: $max(q_i^e - \sum_{j=1}^{l} T_{V_j} - d_l/b_e, 0)$.

If the condition in line 2 holds, it indicates that there are still tasks in the queue when the partitioned task arrives at the MEC server. The added time is the remaining waiting time in the queue (line 3). Otherwise, the MEC server should be idle, and the added time is the data transmission time (line 5).

Another additional latency introduced is the time to send the computation result back from the MEC server to the vehicle. First, we calculate t_i^c which is the time to obtain the computation result on the MEC server. According to Equation (9), the time to forward the computation result back to the vehicle, i.e., t_i^b, can be obtained (line 8).

The reduced time. Since the computation power of MEC server is greater than vehicle, the computation time of the offloaded task will be reduced by $\sum_{j=l+1}^{L}(T_{V_j} - T_{E_j})$ as shown in line 9. For the remaining tasks in the vehicle queue, since the head task in the queue is offloaded, their waiting time will be reduced by $\sum_{j=l+1}^{L} T_{V_j}$. Suppose that there are N^v tasks in the vehicle queue. The sum of reduced time is $(N^v - 1) * \sum_{j=l+1}^{L} T_{V_j}$.

Finally, DPAO will choose the layer with the maximum value of reward for partition (line 11).

In Algorithm 1, the partition result can be obtained after comparing the execution time of each possible partition layer. Therefore, the time complexity of Algorithm 1 is $O(L)$.

5.3. Edge-Edge Partition

An MEC server may partition and offload tasks in the head of the queue to the next MEC server. The edge-edge partition is shown in Algorithm 2. Similar to vehicle-edge partition, Algorithm 2 will check each possible partition layer of the task, calculating both added

Algorithm 2 DPAO for edge-edge partition

Input: D_i, L, T_E, d_l, d_e, N_x^e, $q_{i,x+1}^e$, p_i^v
N_x^e: the number of task of current MEC server
$q_{i,x+1}^e$: waiting time of the next MEC server
Output: p_i^e: the selected layer of partition decision

1: **for** l in $p^v ... L$ **do**
2: **if** $\sum_{j=p_i^v}^{l} T_{E_j} + d_l/r_e < q_{i,x+1}^e$ **then**
3: $AddedTime[l] \leftarrow q_{i,x+1}^e - \sum_{j=p_i^v}^{l} T_{E_j} - d_l/r_e$
4: **else**
5: $AddedTime[l] \leftarrow d_l/r_e$
6: **end if**
7: $ReducedTime[l] \leftarrow (N_x^e - 1) * \sum_{j=l+1}^{L} T_{E_j}$
8: **end for**
9: $p_i^e \leftarrow \arg\min_{l=p_i^v,...L}(ReducedTime[l] - AddedTime[l])$

time and reduced time to determine offloading decision with maximum reward.

The added time mainly consists of data transmission time between two MEC servers and waiting time in the queue of the next MEC server. If the condition in line 2 holds, it indicates that there are still tasks in the queue when the partitioned task arrives at the next MEC server. The added time is the remaining waiting time in the queue (line 3). Otherwise, the next MEC server should be idle, and the added time is the data transmission time (line 5).

The reduced time. If a portion of the head task is offloaded to the next MEC server, the waiting time of the remaining task in the queue of the current MEC server will be reduced. Given that there are N_x^e tasks in the current MEC server queue, the total reduced time is $(N_x^e - 1) * \sum_{j=l+1}^{L} T_{E_j}$.

After both added time and reduced time are calculated, DPAO will choose the layer with the maximum value of reward for partitioning and offloading, i.e., line 9 in Algorithm 2.

Since the partition result can be obtained after comparing the execution time of all possible layers, the time complexity of Algorithm 2 is $O(L)$.

5.4. Analysis of layer-by-layer partition

The above DPAO method partitions tasks at the beginning and then computes and offloads them based on the partition points. However, the environment, including the number of tasks on the current device and the queueing time on the next device, may be changed and different from the moment when tasks are partitioned. As a result, the partitioning results may not be optimal. Therefore, a layer-by-layer estimation method can be used for DPAO, which determines whether to offload tasks at the end of each layer. This tragedy can help DPAO to obtain more accurate partition results. However, this requires that each task perform the partition decision L times, which incurs overhead. Especially when the environment is relatively stable, the benefits of real-time partitioning are limited while overhead remains constant. We tested the performance of the layer-by-layer partitioning mechanism in experiments.

6. DPAO with Queue Reservation Scheme

The DPAO operations described in the previous section do not consider the instability of the MEC server queue and the queuing time of tasks cannot be accurately estimated. For example, a new task may arrive at the next MEC server before the offloaded task, leading to an incorrect estimate of the waiting time in the queue. Thus, we further propose DPAO with a *queue reservation* scheme to overcome these issues.

6.1. Virtual Task for Queue Reservation

When the partition decision is made, a *virtual task* will be generated and pushed to the queue of the next MEC server. The *virtual task* only contains the partition result and the expected arrival time for the real offloaded task. Since the amount of data carried by the *virtual task* is very small, it can be transmitted to the next MEC server soon after the partition decision is made and reserve the position in the queue of the next MEC server, ensuring that the waiting time of the offloaded task will not be affected by other tasks. The expected arrival time of a real offloaded task is estimated based on the processing time of the previous layers of the DNN and the transmission time. Once the real offloaded task arrives, which carries the output data of previous layers of the DNN, it will replace the *virtual task* in the queue. DPAO will iterate through the tasks in the queue one by one from the beginning, skipping all *virtual tasks* and processing the first real task encountered. All skipped virtual tasks will continue to remain in the queue. Virtual tasks will be dropped if they are not replaced by real tasks within a certain time period (see Section 6.2).

6.2. Task Scheduling

In the queue reservation scheme, DPAO introduces a *partition queue* and a *compute queue*. Each arriving task will be first processed by the partition queue, where the partition decision is made. Depending on the partition decision, tasks in the compute queue will compute either some layers or all layers locally.

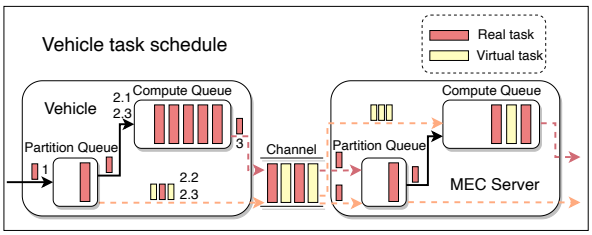

Figure 6: Task schedule on vehicle.

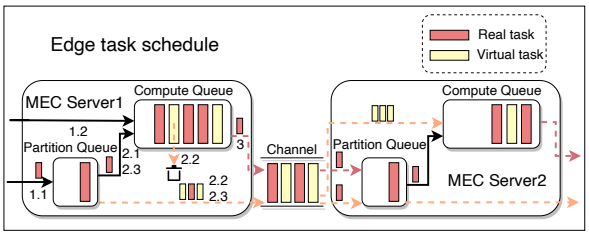

Figure 7: Task schedule on MEC servers.

Vehicle Task Scheduling. Fig. 6 shows the task scheduling strategy on a vehicle that handles DNN task partition and offloading between the vehicle and the linked MEC server.

Step 1: On a vehicle, new arriving tasks are first processed by the partition queue.

Step 2: When the partition decision is made for the head task in the partition queue: (2.1) If the whole DNN task is to be computed on the vehicle, it will be pushed into the compute queue; (2.2) If the vehicle decides to offload the whole DNN task, the DNN task, together with its virtual task, will be sent to the linked MEC server (the waiting time of the task on vehicle q_i^v is 0 in Equation (4)); (2.3) Otherwise, the task is partitioned and the first part of the DNN layers will be pushed into the compute queue and a virtual task will be sent to the linked MEC server. Virtual tasks will be pushed into the compute queue bypassing the partition queue on the MEC server.

Step 3: After the task is calculated to the partition point in the computation queue, it will be sent to the partition queue of the linked MEC server, carrying the output data of the partitioned layer.

Edge Task Scheduling. Fig. 7 shows the task scheduling strategy on an MEC server that performs DNN task partition and offloading with the next MEC server.

Step 1: When a task arrives (either from vehicles or the previous MEC server): (1.1) If it is a virtual task, it will be pushed into the compute queue; (1.2) Otherwise, it will be first processed by the partition queue.

Step 2: When a partition decision is made for the head task in the partition queue: (2.1) If the whole DNN task is to be computed on the current MEC server, it will be pushed into the compute queue or replace its virtual task in the compute queue if the virtual task exists; (2.2) If the vehicle decides to offload the whole DNN task, the DNN task, together with its virtual task, will be sent to the next MEC server. In the meantime, its virtual task will be removed from the compute queue if exists (the waiting time of the task on MEC server $q_{i,x}^e$ is 0 in Equation (14)); (2.3) Otherwise, the DNN task is partitioned and the first part of the DNN layers will be pushed into the compute queue or replace its virtual task in the compute queue if exists. Meanwhile, a virtual task will be sent to the next MEC server.

Step 3: After the task is calculated to the partition point in the computation queue, it will be sent to the partition queue of the next MEC server, carrying the output data of the partitioned layer.

6.3. Estimation of Waiting Time

Tasks in the partition queue can be processed quickly, e.g., 91us in our experiment. Thus, the majority of the waiting time for a task is the queuing time in the compute queue. Accurate estimation of waiting time in both the local compute queue and the compute queue of the linked (or next) MEC server is important for DNN task partition decision.

Let Q_x be the compute queue of MEC server S_x. For each task $D_i \in Q_x$, it can be represented by (c_i, a_i), where c_i denotes task computation time on S_x and a_i is the expected arrival time for D_i. Specifically, $a_i = 0$ means that this is a real offloaded task. Suppose thata task D_j is to be offloaded to S_x after time τ_j from a vehicle or MEC server S_{x-1}. τ_j can be obtained by summation of waiting time in the current node (e.g., a vehicle or S_{x-1}) and transmission time to S_x. When D_j is offloaded to S_x, it needs to wait for the computation of all real tasks in Q_x that arrives before D_j. Thus, for task D_j, its estimated task waiting time W_j on S_x, in the worst case, can be defined as:

$$W_j = \sum_{D_i \in \{D_i \in Q_x | a_i \leq \tau_j\}} c_i - \tau_j. \tag{20}$$

Fig. 8 illustrates the calculation of waiting time using a vehicle, its linked MEC server and a previous MEC server as an example. There are three tasks in the compute queue of MEC server: Task 1, Task 2, and Task 3. Consider that the computation time on the linked MEC server for each task is 20ms. Task 2 and Task 3 are virtual tasks, and their real tasks are transmitted to the

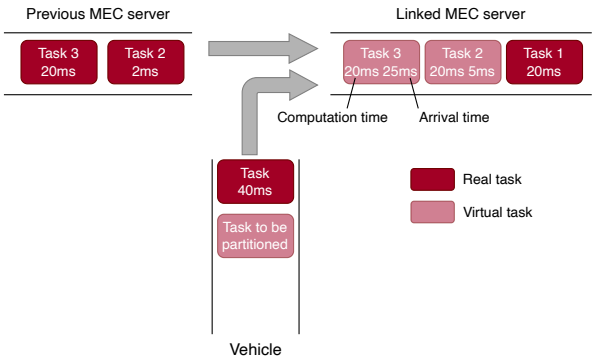

Figure 8: Estimation of the waiting time. Computation time is the duration required for a task to be computed locally. Arrival time refers to the expected time for the corresponding real task to arrive, which includes the computation and transmission time.

linked MEC server after being computed on the previous one. Suppose that the computation time on the previous MEC server and transmission time of Task 2 are 2*ms* and 3*ms*, respectively, then its arrival time is 5ms (i.e., 2*ms*+3*ms*). Suppose that the computation time on the previous MEC server and transmission time of Task 3 are 20*ms* and 3*ms*, respectively. Task 3 needs to wait for Task 2 to complete before starting its computation, and its arrival time is 25*ms* (i.e., 2*ms*+20*ms*+3*ms*). There are two tasks in the compute queue of the vehicle: one task has a computation time of 40*ms*. A DNN task arrives at the vehicle and a partition decision needs to be made.

For complete offloading, the task is completely offloaded to the MEC server instead of being processed by the vehicle. We assume that the transmission time of the original input data for the task between the vehicle and the MEC server is 8*ms*. When the offloaded task arrives at the MEC server, the corresponding real task of Task 2 has arrived, but Task 3 has not. Consequently, the offloaded task can only be executed once both Task 1 and Task 2 have been completed. The waiting time is 32*ms* (i.e., 20*ms*+20*ms*-8*ms*).

For partial offloading, we assume that the offloaded task is partitioned into two parts, the first of which is executed in the vehicle with a computation time of 10*ms* and the second of which is executed in the MEC server with a transmission time of 5*ms*. The offloaded task arrives at the MEC server after 55*ms* (i.e., 40*ms*+10*ms*+5*ms*). At this moment, the corresponding real tasks for the three tasks have all arrived. Therefore, the waiting time is 5*ms* (i.e., 20*ms*+20*ms*+20*ms*-55*ms*).

6.4. Analysis of bidirectional offloading

We assume that tasks are offloaded in the direction of movement of vehicles and not in the opposite direction. This facilitates reducing the time required to transmit computation results back to vehicles. We refer to the MEC servers aligned with the direction of vehicle movement as the next MEC servers and the MEC servers in the opposite direction as the previous MEC servers. However, it is possible that when the load on the previous MEC server is lower than that on the next MEC server, offloading tasks to the previous MEC server can result in shorter completion times, even if it increases the transmission time for returning computation results (vehicles could be too far away once the computation is over). Therefore, based on DPAO with queue reservation scheme, we also enable tasks to select the appropriate offloading direction by considering the load of neighboring MEC servers in both the preceding and the subsequent directions. We tested the performance of the bidirectional offloading mechanism in experiments.

7. Performance evaluation

We have implemented DPAO, using AlexNet in simulation conducted in Python and CaffeNet in a testbed based on the Raspberry Pi to demonstrate its effectiveness. CaffeNet is a variant of AlexNet that was implemented using the Caffe deep learning framework [20]. The primary parameters are summarized in Table 3.

7.1. Experiment Setup

DPAO is implemented using modified instances of Caffe, which is a widely used deep learning framework that provides a set of tools and libraries for training and deploying deep neural networks [20]. The DNN structure in Caffe is recorded in a "prototxt" file, which is used to process the layers. We inserted a "stop layer" after each partitioned layer in the "prototxt" file. This allows the instance of Caffe to stop processing at the desired layers. We have trained AlexNet, a chain-topology DNN with 22 layers, for image recognition over the ILSVRC 2012 dataset [45]. The execution time and output data size of each layer of AlexNet are first collected, which are then used in the simulation in Section 7.2.

Execution time of each layer. The execution time of each layer of DNN is collected for both the vehicle and the MEC server. We utilized a docker container as the vehicle, equipped with an Intel Xeon E5-2698 2.20GHz processor, running on the Ubuntu 18.04 operating system. The Caffe function *set_mode_cpu*()[7] is

Table 3: Simulation paremeters.

Parameter	Value
Number of roadside infrastructures	10
Distance between ajacent MEC servers	$100m$
Number of vehicles	22
Speed of vehicles	$120km/h$
Computation capacity of vehicles	$8 \times 10^8 FLOPS$
Computation capacity of MEC server	$2 \times 10^{10} FLOPS$
BandWidth between vehicle and MEC server	$10Mbps$
BandWidth between MEC servers	$100Mbps$
Arrive interval time between adjacent task	$[40ms, 120ms]$

used to execute DNN only by CPU. In contrast, the docker container of the MEC server was also running on the Ubuntu 18.04 operating system, but it was equipped with a Tesla V100-DGXS GPU in addition to the Intel Xeon E5-2698 2.20GHz processor. The Caffe function $set\_mode\_gpu()$[7] is executed to accelerate DNN processing using GPU. The value of FLOPS for vehicle and MEC servers are about 8×10^8 and 2×10^{10}, respectively. To execute a specific layer of DNN, the function $net.forward()$[8] in Caffe is used to specify the start layer and end layer. The measured execution time of each layer for both the vehicle and the MEC server is shown in Fig. 3.

Output data size for each layer. In Caffe, the intermediate layer output data is presented in blob format. They are first converted to byte streams so that they can be transmitted on the network. Then, the function $getsizeof()$ is used to obtain their size. The output data size of each layer is shown in Fig. 2.

Vehicular Network. We consider a one-way, straight road and deploy RSUs along its sides. Each RSU is equipped with a MEC server and is separated by $100m$ from its neighbors. Each RSU can only communicate with neighboring RSUs and vehicles within its range of communication. The bandwidth between the vehicle and the MEC servers is set to be $10Mbps$, and the bandwidth between adjacent MEC servers is $100Mbps$.

Vehicle generation and movement. We define the process representing the number of vehicles that arrive at the first MEC server as a Switched Batch Bernoulli Process (SBBP) [41, 46]. The number of arriving vehicles follows a Poisson distribution with mean ω_k for each individual arrival phase k ($k = 1, 2$). The transition probability of arrival phases is denoted as:

$$Pr = \begin{bmatrix} 1/2 & 1/2 \\ 1/3 & 2/3 \end{bmatrix}. \tag{21}$$

Vehicles on the road are driving at a speed of 120 kilometers per hour.

DNN task generation. We consider the occurrence of a sequence of discrete tasks to be modeled as a Poisson process. The arrival of two such tasks is treated as independent random variables, which are drawn from an exponentially distributed population with the density function $f(x) = \lambda e^{-\lambda x}$ for some fixed constant λ, which is also called as task arrival rate. The interval time between two adjacent tasks is $1/\lambda$, which is ranged from $60ms$ to $120ms$. 5000 tasks are generated continuously per experiment.

Evaluation Metrics. We evaluate average task completion time on vehicles. The task completion time is the period of time between task initiation and receipt of task results by a vehicle.

We compare the following methods:

(i) Vehicle-Only: The DNN task is executed entirely on vehicle;

(ii) Edge-Only: The DNN task is executed entirely on the linked MEC server;

(iii) DPAO;

(iv) DPAO (layer-by-layer partition): partition decisions are made at the end of the computation of each layer, as opposed to only at the beginning.

(v) DPAO with queue reservation (forward offloading): tasks are be offloaded in the same direction of movement of vehicles;

(vi) DPAO with queue reservation (bidirectional offloading): tasks are offloaded to the one with the smallest load among adjacent edge servers;

(vii) MDPO [22][10]: MDPO regards splitting DNN tasks as a minimum cutting problem and uses the maximum flow algorithm and topological sorting algorithm

[7] https://github.com/BVLC/caffe/blob/master/python/caffe/_caffe.cpp
[8] https://github.com/BVLC/caffe/blob/master/python/caffe/pycaffe.py

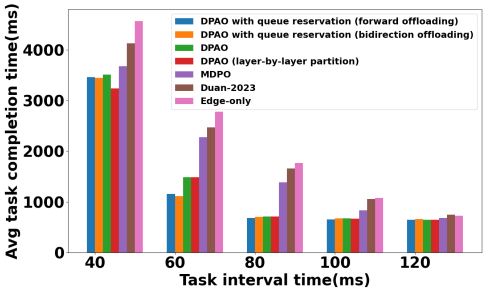

Figure 9: Average task completion time under different task interval time

to execute them together on both local and edge servers. It cuts the DNN model once on the mobile device and does not consider the dynamic change of MEC server load.

7.2. Evaluation Results and Analysis

The following results are the average of 100 simulation runs for each scenario. Each run consists of 5,000 tasks.

The variation of average task completion time with respect to task interval time is shown in Fig. 9. DPAO and DPAO with queue reservation consider both transmission time and the difference in workload between MEC servers. Thus, results show that DPAO can reduce average task completion time by 1.7 and 1.5 times compared with Edge-Only and MDPO, respectively. DPAO with queue reservation reserves position on MEC servers and estimates the queuing time accurately, so the average completion time is reduced by 10% compared with DPAO. When task interval time is small, layer-by-layer partition can significantly reduce the average completion time compared to DPAO, but it may have side effects when task interval time is large. This is because the queuing time at the edge server fluctuates significantly when the task interval time is short. Layer-by-layer partition can sensitively capture the changes of queuing time, thereby making better partitioning decisions. In addition, bidirectional offloading DPAO with queue reservation can further reduce the average task completion time compared to forward offloading. Specifically, with a task interval time of 40ms, bidirectional offloading decreases the average task completion time by 5% relative to forward offloading. Furthermore, if all tasks are processed in the vehicle, the average completion time is 145s when the task interval time is 100ms.

The reduction of waiting time contributes a significant part to the optimization of task completion time. The average waiting time for tasks (including in the vehicle and MEC server) is shown in Fig. 10. The results show that DPAO can reduce waiting time on both vehicle and MEC server by 2.4 times and 3.3 times compared with MDPO and Edge-Only. Compared with DPAO, DPAO with queue reservation (forward offloading) can reduce waiting time on both vehicle and MEC server by 79%. To further illustrate the waiting times for different methods, Fig. 11 and Fig. 12 shows the average waiting time on vehicle and MEC server under different task interval time, respectively. Layer-by-layer partition can accurately estimate the queuing time on the MEC server before offloading tasks. Especially when the task time is 40ms and 60ms, layer-by-layer partition can further reduce the average waiting time by 48% and 18% based on DPAO. As the task interval time increases, the queueing time of the MEC server changes slowly. The benefits of layer-by-layer partition diminish while the overhead of making multiple division decisions remains constant. It ultimately leads to a higher average task waiting time for layer-by-layer partition than DPAO. It can be seen from Fig. 12 that the average waiting time on the MEC server of our proposed four methods is smaller than MDPO and Edge-Only. This is because our proposed methods consider the load differences between MEC servers and perform dynamic partitioning and offloading.

Since DPAO allows a DNN task to be partitioned and offloaded among multiple MEC servers, Fig. 13 shows the distribution of the number of MEC servers participating in task computation under different task interval times for DPAO with queue reservation. The results indicate that the greater the task interval, the fewer MEC servers participate in the computation. This is because when the task interval time is increased, the load of MEC servers is reduced. It is sufficient to complete the computation on a few servers to avoid more data transmission.

The distribution of partitioned layers on both the vehicle and MEC server under different task interval times for DPAO with queue reservation are shown in Fig. 14 and Fig. 15, respectively. For vehicles, most partitioned layers are the *input* layer and the *conv1* layer. When the task interval time increases, more and more tasks are partitioned in the *conv1* layer. When the task interval time is 120ms, 48% of tasks are partitioned at the *conv1* layer. This is because when the task interval time is

[10] Although MDPO [22] optimizes the completion time of a single task, it can be extended to multitask scenarios. Other works in Table 1 either do not involve DNN partition [23] or do not consider the mobility of vehicles [24], so they are not suitable for comparison in the evaluations.

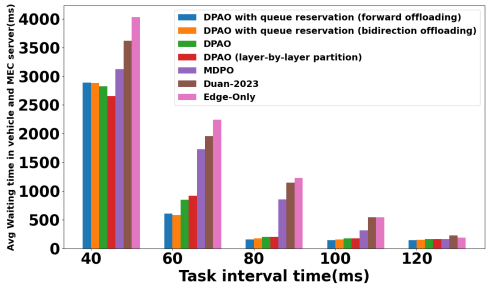

Figure 10: Waiting time in both vehicle and MEC server under different task interval time

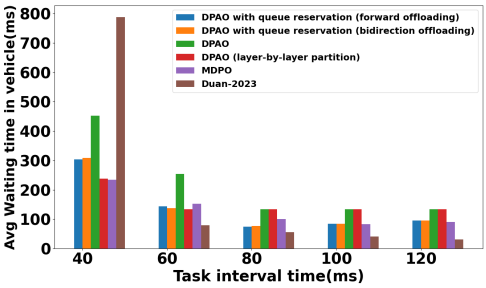

Figure 11: Waiting time in vehicle under different task interval time

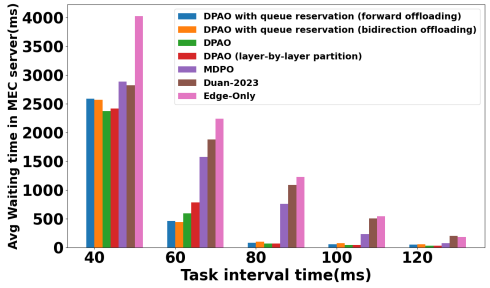

Figure 12: Waiting time in vehicle under different task interval time

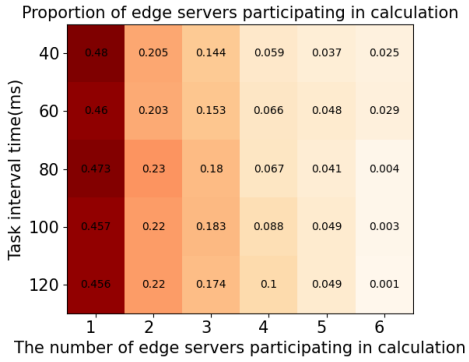

Figure 13: Num. of edge servers involved in computation for DPAO with queue reservation

large, the computational burden of the vehicle is small, and more layers can be processed locally to avoid data transmission time. For MEC servers, about half of the tasks are always partitioned at the *poo1* layer for any task interval time. This is because the output data of the *poo1* layer is very small, even though it introduces computing overhead. When task interval time is small, say 40*ms*, 35% of tasks are partitioned at the *input* layer (i.e., offloaded completely), while there are only 3% at the *conv1* layer. However, when the task interval time is increased, the load on current devices (vehicles or MEC servers) is reduced, so more and more tasks are partitioned at the *conv1* layer. For example, 26% of tasks are partitioned at the *conv1* layer when the task interval time is 120*ms*.

In addition to waiting time, transmission time and execution time on both vehicle and MEC server are also investigated and results are shown in Fig. 16. The result shows that when task interval time is increased, task execution time on the vehicle increases slowly, but transmission time decreases slowly. However, execution time on MEC servers drops rapidly. This is because the queuing time of the vehicle and MEC servers is balanced when tasks are partitioned on the vehicle. When the queuing time of the MEC server is short, it tends to offload tasks; otherwise, it is computed locally. This is because the larger the task interval time, the later of the partition layer in DNN layers, which is observed in Fig. 14. Therefore, more layers are computed on vehicles, decreasing computation time on MEC servers.

Fig. 17 compares the overhead and performance benefit brought by the queue reservation scheme. The overhead is mainly caused by the transmission of virtual tasks, and thus is measured by the average time for transmitting virtual tasks during the computation process for each task. The performance benefit is measured by the average task completion time saved by the DPAO with the queue reservation scheme compared to the DPAO. The result shows that the performance benefit brought by the queue reservation scheme is much greater than the overhead.

Fig. 18 shows the distribution of the number of MEC

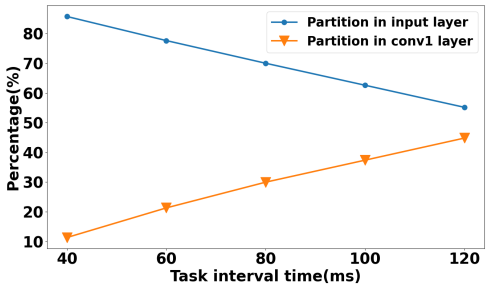

Figure 14: Proportion of partition layers on vehicle for DPAO with queue reservation

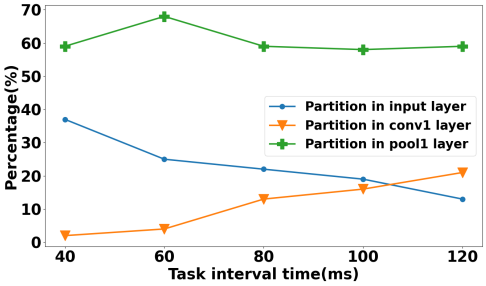

Figure 15: Proportion of partition layers on MEC server for DPAO with queue reservation

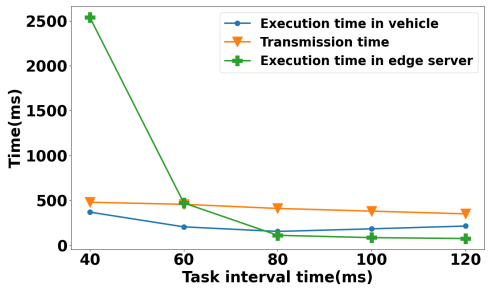

Figure 16: Execution and transmission time for DPAO with queue reservation (forward offloading)

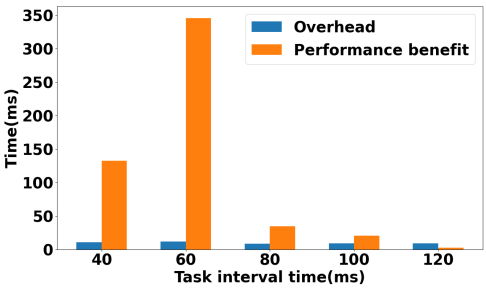

Figure 17: Overhead and performance of queue reservation scheme

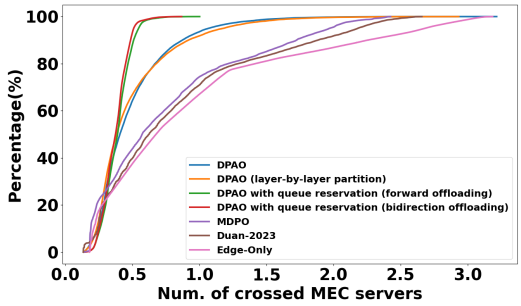

Figure 18: The number of crossed MEC servers during task completion time (task interval time = 40*ms*)

servers that vehicles pass by during the task completion time when the task interval time is 40*ms*. For DPAO with queue reservation, approximately 97% of tasks have passed through fewer than two edge servers within the completion time, whereas for DPAO, 86% of tasks have passed through fewer than 3 edge servers within the completion time. Additionally, a task can pass through up to 6 edge servers within the completion time for MDPO and Edge-Only. It means that our proposed method, especially DPAO with queue reservation, can enable most tasks to obtain computational results within a shorter driving distance.

7.3. Evaluation with Raspberry Pi

We have also implemented a simple prototype to verify the feasibility of DAPO. As shown in Fig. 19, we have used a Raspberry Pi 3 model B Plus, which has a quad-core ARM processor at 1.4 GHz with 1 GB of RAM and a camera to represent a vehicle. This camera can take photos at the resolution of 2592×1944 and record 1080p video. Two desktop computers are used to emulate MEC servers, each of which has a quad-core Intel 1.6 GHz processor with 3.8 GB of RAM and runs the Ubuntu operating system. The vehicle and MEC servers are connected through WiFi. The DNN model used in the testbed is CaffeNet.

The workflow of the vehicle is as follows: 1) Collect video information and extract frames. In the Raspberry Pi, the command "*raspistill -t 1000*" is used to take a picture every 1*s*; 2) make DNN partition decision; 3) compute layers of DNN assigned to vehicle; and 4) transmit the partition result and the output data of the intermediate layer to the MEC server.

Figure 19: Raspberry Pi as a vehicle with a camera

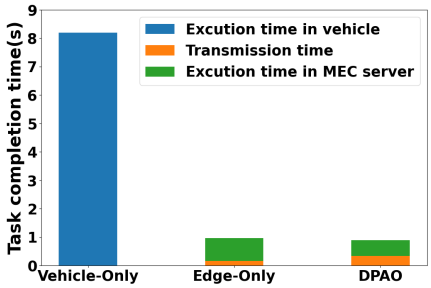

Figure 20: Average task completion time

Accordingly, the workflow of the MEC server is as follows: 1) receive the partition result and the output data of the intermediate layer from vehicle; 3) make partition decision and compute assigned layers of DNN; 4) transmit the partition result and the output data of the intermediate layer to the next MEC server and 5) the last MEC server will transmit inference result back to the vehicle.

Fig. 20 shows the average task completion time of the three tasks under DPAO, Vehicle-Only and Edge-Only methods. DPAO can effectively obtain the results within 890*ms*, reducing the task completion time by 8.3x compared to the Vehicle-Only method (8.2*s*) and 8% compared to Edge-Only method (965*ms*). One observation is that DPAO partitions DNN tasks in the *input* layer on Raspberry Pi, i.e., offloading all layers to the MEC server. This is mainly because that the computation power of the MEC server is much higher than that of vehicle (i.e., Raspberry Pi). As a result, DPAO chooses such a partition strategy to reduce execution time on the vehicle.

8. Conclusions

In this paper, we study the collaborative offloading of multiple DNN tasks in the vehicle edge computing scenario. We present a comprehensive study of partition and offloading problem under VEC while considering the mobility of vehicles. We also propose DPAO (DNN Partition and Offloading) to find the partition layer by both vehicle-edge collaboration and edge-edge collaboration. We then conduct a numerical simulation experiment and testbed experiment with Raspberry Pi. Results show that DPAO can reduce the latency by 1.9x compared to computing on the vehicle. Currently, this paper only considers the DNN of the chain topology. In the future, we plan to partition and offload the DNN with directed acyclic graph (DAG) topology between multiple MEC servers. In addition, the unpredictable driving path of the vehicle and the speed change are also worth exploring.

References

[1] Y. Yin, M. Wu, X. Wang, X. Huang, Speech recognition for power customer service based on DNN and CNN models, in: International Forum on Digital TV and Wireless Multimedia Communications, Springer, 2022, pp. 453–468.

[2] L. Chen, W. Zhan, W. Tian, Y. He, Q. Zou, Deep integration: A multi-label architecture for road scene recognition, IEEE Transactions on Image Processing 28 (10) (2019) 4883–4898. doi:10.1109/TIP.2019.2913079.

[3] G. Singal, H. Singhal, R. Kushwaha, V. Veeramsetty, T. Badal, S. Lamba, RoadWay: lane detection for autonomous driving vehicles via deep learning, Multimedia Tools and Applications 82 (4) (2023) 4965–4978. doi:10.1007/s11042-022-12171-0.

[4] Y. Hu, W. Pang, X. Liu, R. Ghosh, B. Ko, W. H. Lee, R. Govindan, Rim: Offloading Inference to the Edge, IoTDI 2021 - Proceedings of the 2021 International Conference on Internet-of-Things Design and Implementation (2021) 80–92doi:10.1145/3450268.3453521.

[5] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, W. Shi, Edge computing for autonomous driving: opportunities and challenges, Proceedings of the IEEE 107 (8) (2019) 1697–1716. doi:10.1109/JPROC.2019.2915983.

[6] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, J. Zhang, Edge intelligence: Paving the last mile of artificial intelligence with edge computing, Proceedings of the IEEE 107 (8) (2019) 1738–1762.

[7] S. Yi, Z. Hao, Z. Qin, Q. Li, Fog computing: Platform and applications, in: IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb), 2015. doi:10.1109/HotWeb.2015.22.

[8] B. Zhang, J. Xin, S. Ratnasamy, J. Wawrzynek, E. A. Lee, Awstream: adaptive wide-area streaming analytics, in: the Conference of the ACM Special Interest Group on Data Communication, 2018.

[9] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, V. Young, Mobile edge computing—a key technology towards 5g, ETSI white paper 11 (11) (2015) 1–16.

[10] K. Zhang, Y. Mao, S. Leng, S. Maharjan, Y. Zhang, Optimal delay constrained offloading for vehicular edge computing networks, in: IEEE International Conference on Communications (ICC), 2017. doi:10.1109/ICC.2017.7997360.

[11] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, L. Tang, Neurosurgeon: Collaborative intelligence between the cloud and mobile edge, ACM SIGOPS Operating Systems Review (2017).

[12] W. He, S. Guo, S. Guo, X. Qiu, F. Qi, Joint DNN partition deployment and resource allocation for delay-sensitive deep learning inference in IoT, IEEE Internet of Things Journal 7 (10) (2020) 9241–9254. doi:10.1109/JIOT.2020.2981338.

[13] B. Zhang, T. Xiang, H. Zhang, T. Li, S. Zhu, J. Gu, Dynamic DNN decomposition for lossless synergistic inference, in: 2021 IEEE 41st International Conference on Distributed Computing Systems Workshops (ICDCSW), IEEE, 2021, pp. 13–20.

[14] E. Li, Z. Zhou, X. Chen, Edge intelligence: On-demand deep learning model co-inference with device-edge synergy, in: Proceedings of the 2018 Workshop on Mobile Edge Communications, 2018, pp. 31–36.

[15] L. Zeng, E. Li, Z. Zhou, X. Chen, Boomerang: On-demand cooperative deep neural network inference for edge intelligence on the industrial internet of things, IEEE Network 33 (5) (2019) 96–103. doi:10.1109/MNET.001.1800506.

[16] J. Mao, Z. Yang, W. Wen, C. Wu, L. Song, K. W. Nixon, Y. Chen, MeDNN: A distributed mobile system with enhanced partition and deployment for large-scale DNNs, in: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2017. doi:10.1109/ICCAD.2017.8203852.

[17] J. Mao, X. Chen, K. W. Nixon, C. Krieger, Y. Chen, MoDNN: Local distributed mobile computing system for deep neural network, in: Design, Automation Test in Europe Conference Exhibition (DATE), 2017. doi:10.23919/DATE.2017.7927211.

[18] F. Dong, H. Wang, D. Shen, Z. Huang, Q. He, J. Zhang, L. Wen, T. Zhang, Multi-exit DNN inference acceleration based on multi-dimensional optimization for edge intelligence, IEEE Transactions on Mobile Computing (2022) 1–1doi:10.1109/TMC.2022.3172402.

[19] T. F. Gonzalez, Handbook of approximation algorithms and metaheuristics, Handbook of Approximation Algorithms and Metaheuristics (2007) 1–1432doi:10.1201/9781420010749.

[20] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, Caffe: Convolutional architecture for fast feature embedding, ACM (2014).

[21] K. Zhang, Y. Mao, S. Leng, Y. He, Y. ZHANG, Mobile-edge computing for vehicular networks: A promising network paradigm with predictive offloading, IEEE Vehicular Technology Magazine 12 (2) (2017) 36–44. doi:10.1109/MVT.2017.2668838.

[22] X. Tian, J. Zhu, T. Xu, Y. Li, Mobility-included DNN partition offloading from mobile devices to edge clouds, Sensors 21 (1) (2021) 229.

[23] Y. Dai, D. Xu, S. Maharjan, Y. Zhang, Joint load balancing and offloading in vehicular edge computing and networks, IEEE Internet of Things Journal 6 (3) (2019) 4377–4387. doi:10.1109/JIOT.2018.2876298.

[24] Q. Wang, Z. Li, K. Nai, Y. Chen, M. Wen, Dynamic resource allocation for jointing vehicle-edge deep neural network inference, Journal of Systems Architecture 117 (2021) 102133.

[25] C. Feng, P. Han, X. Zhang, B. Yang, Y. Liu, L. Guo, Computation offloading in mobile edge computing networks: A survey, Journal of Network and Computer Applications 202 (April) (2022) 103366. doi:10.1016/j.jnca.2022.103366.
URL https://doi.org/10.1016/j.jnca.2022.103366

[26] W. Fan, L. Gao, Y. Su, F. Wu, Y. Liu, Joint DNN Partition and Resource Allocation for Task Offloading in Edge-Cloud-Assisted IoT Environments, IEEE Internet of Things Journal 10 (12) (2023) 10146–10159. doi:10.1109/JIOT.2023.3237361.

[27] C. Hu, W. Bao, D. Wang, F. Liu, Dynamic adaptive DNN surgery for inference acceleration on the edge, Proceedings - IEEE INFOCOM April (2019). doi:10.1109/INFOCOM.2019.8737614.

[28] Y. Su, W. Fan, L. Gao, L. Qiao, Y. Liu, F. Wu, Joint DNN partition and resource allocation optimization for energy-constrained hierarchical edge-cloud systems, IEEE Transactions on Vehicular Technology 72 (3) (2022) 3930–3944.

[29] T. Kim, H. Park, Y. Jin, S.-s. Lee, S. Lee, Partition Placement and Resource Allocation for Multiple DNN-Based Applications in Heterogeneous IoT Environments, IEEE Internet of Things Journal 10 (11) (2023) 1–1. doi:10.1109/jiot.2023.3235993.

[30] Z. Liao, W. Hu, J. Huang, J. Wang, Joint multi-user DNN partitioning and task offloading in mobile edge computing, Ad Hoc Networks 144 (March) (2023) 103156. doi:10.1016/j.adhoc.2023.103156.
URL https://doi.org/10.1016/j.adhoc.2023.103156

[31] G. Qiao, S. Leng, K. Zhang, Y. He, Collaborative task offloading in vehicular edge multi-access networks, IEEE Communications Magazine 56 (8) (2018) 48–54. doi:10.1109/MCOM.2018.1701130.

[32] Y. Liu, S. Wang, J. Huang, F. Yang, A computation offloading algorithm based on game theory for vehicular edge networks, in: IEEE International Conference on Communications (ICC), 2018. doi:10.1109/ICC.2018.8422240.

[33] W. Ju, D. Yuan, W. Bao, L. Ge, B. B. Zhou, DeepSave: Saving DNN inference during handovers on the edge, Proceedings of the 4th ACM/IEEE Symposium on Edge Computing, SEC 2019 (2019) 166–178doi:10.1145/3318216.3363301.

[34] J. Zhao, Q. Li, Y. Gong, K. Zhang, Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks, IEEE Transactions on Vehicular Technology 68 (8) (2019) 7944–7956.

[35] F. Zeng, K. Zhang, L. Wu, J. Wu, Efficient caching in vehicular edge computing based on edge-cloud collaboration, IEEE Transactions on Vehicular Technology 72 (2) (2023) 2468–2481. doi:10.1109/TVT.2022.3213130.

[36] J. Zhang, K. B. Letaief, Mobile edge intelligence and computing for the internet of vehicles, Proceedings of the IEEE 108 (2) (2019) 246–261.

[37] J. Jhung, H. Suk, H. Park, S. Kim, Hardware accelerators for autonomous vehicles, in: Artificial Intelligence and Hardware Accelerators, Springer, 2023, pp. 269–317.

[38] S. K. Prashanthi, S. A. Kesanapalli, Y. Simmhan, Characterizing the Performance of Accelerated Jetson Edge Devices for Training Deep Learning Models, Proceedings of the ACM on Measurement and Analysis of Computing Systems 6 (3) (2022). doi:10.1145/3570604.

[39] M. Su, C. Cao, M. Dai, J. Li, Y. Li, Towards fast and energy-efficient offloading for vehicular edge computing, in: 2022 IEEE 28th International Conference on Parallel and Distributed Systems (ICPADS), 2023, pp. 649–656. doi:10.1109/ICPADS56603.2022.00090.

[40] S. Munawar, Z. Ali, M. Waqas, S. Tu, S. A. Hassan, G. Abbas, Cooperative Computational Offloading in Mobile Edge Computing for Vehicles: A Model-Based DNN Approach, IEEE Transactions on Vehicular Technology 72 (3) (2023) 3376–3391. doi:10.1109/TVT.2022.3217323.

[41] O. Hashida, Y. Takahashi, S. Shimogawa, Switched bath

bernoulli process (sbbp) and the discrete-time sbbp/g/1 queue with application to statistical multiplexer performance, IEEE Journal on Selected Areas in Communications 9 (3) (1991) 394–401. doi:10.1109/49.76638.

[42] C. B. Kuhn, M. Hofbauer, G. Petrovic, E. Steinbach, Introspective failure prediction for autonomous driving using late fusion of state and camera information, IEEE Transactions on Intelligent Transportation Systems (2020) 1–15doi:10.1109/TITS.2020.3044813.

[43] H. Li, C. Sun, X. Li, Q. Xiong, J. Wen, X. Wang, V. C. M. Leung, Mobility-aware content caching and user association for ultra-dense mobile edge computing networks, in: GLOBECOM 2020 - 2020 IEEE Global Communications Conference, 2020, pp. 1–6. doi:10.1109/GLOBECOM42002.2020.9348257.

[44] W. Fan, Z. Chen, Z. Hao, Y. Su, F. Wu, B. Tang, Y. Liu, DNN deployment, task offloading, and resource allocation for joint task inference in iiot, IEEE Transactions on Industrial Informatics 19 (2) (2023) 1634–1646. doi:10.1109/TII.2022.3192882.

[45] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, F. F. Li, Imagenet: A large-scale hierarchical image database, in: IEEE Conference on Computer Vision and Pattern Recognition, 2009. doi:10.1109/CVPR.2009.5206848.

[46] F. Busacca, G. Faraci, C. Grasso, S. Palazzo, G. Schembra, Designing a multi-layer edge-computing platform for energy-efficient and delay-aware offloading in vehicular networks, Computer Networks 198 (2021) 108330. doi:https://doi.org/10.1016/j.comnet.2021.108330.
URL https://www.sciencedirect.com/science/article/pii/S1389128621003315