

# Optimization of Resource Allocation for Edge Computing in Efficient Deep Learning Service

Qinglin Yang

A DISSERTATION  
SUBMITTED IN FULFILLMENT OF THE REQUIREMENTS  
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY  
IN COMPUTER SCIENCE AND ENGINEERING

Graduate Department of Computer and Information Systems

The University of Aizu

2021



© Copyright by Qinglin Yang, March 2021

All Rights Reserved.

The thesis titled

*Optimization of Resource Allocation for Edge Computing in  
Efficient Deep Learning Service*

by

Qinglin Yang

is reviewed and approved by:

Main referee

Professor

Pengli

Professor

Hayakawa

Professor

Anh T. Pham

Professor

Lei Jing

THE UNIVERSITY OF AIZU

March 2021





# Contents

<b>List of Abbreviations</b>	<b>ix</b>
<b>Chapter 1 Introduction</b>	<b>1</b>
1.1 Introduction . . . . .	1
1.2 Organization . . . . .	8
1.3 Publications . . . . .	10
<b>Chapter 2 Background and Related Work</b>	<b>13</b>
2.1 Background . . . . .	13
2.1.1 Efficient Deep Learning . . . . .	13
2.2 Related Works . . . . .	16
2.2.1 Efficient Deep Learning Applications . . . . .	16
2.2.2 Edge Intelligence and Energy Efficiency of Edge . . . . .	20
<b>Chapter 3 Computation Offloading for Fast CNN Inference in Edge Computing</b>	<b>26</b>
3.1 Introduction . . . . .	26
3.2 Background . . . . .	30
3.3 System Model . . . . .	32
3.4 Near-Optimal algorithm design . . . . .	34
3.5 Online algorithm design . . . . .	37
3.6 Performance Evaluation . . . . .	39
3.6.1 Experimental Setup . . . . .	39
3.6.2 Experiments . . . . .	40
3.7 Summary . . . . .	44
<b>Chapter 4 Cooperation of Mobile Devices for Fast Inference of Deep Learning Application</b>	<b>45</b>
4.1 Introduction . . . . .	45
4.2 Background . . . . .	48
4.3 System Model . . . . .	50
4.4 Centralized Algorithm Design . . . . .	52
4.5 Distributed Algorithm Design . . . . .	54
4.6 Performance Evaluation . . . . .	56
4.6.1 Simulation settings . . . . .	56
4.6.2 Simulation results . . . . .	56
4.7 Summary . . . . .	59
<b>Chapter 5 Deep Reinforcement Learning based Task Scheduling for Energy Efficiency Edge Computing</b>	<b>61</b>

5.1	Introduction . . . . .	61
5.2	Related Work . . . . .	64
5.3	System Model and Formulation . . . . .	66
5.3.1	Network Model . . . . .	66
5.3.2	Power Model . . . . .	66
5.4	Reinforcement Learning-based Methods . . . . .	69
5.4.1	Markov Decision Process Description . . . . .	69
5.4.2	Deep Reinforcement Learning Design . . . . .	72
5.5	Evaluation . . . . .	74
5.5.1	Settings . . . . .	74
5.5.2	Analysis and Discussion . . . . .	75
5.6	Summary . . . . .	76
<b>Chapter 6 Conclusions</b>		<b>77</b>

# List of Figures

Figure 1.1 General process for mobile deep learning applications . . . .	2
Figure 1.2 Capabilities comparison of cloud-based approach and edge-based approach . . . . .	9
Figure 1.3 Organization of thesis structure . . . . .	11
Figure 2.1 Schematic diagram of DNNs . . . . .	14
Figure 2.2 Schematic diagram of Computing . . . . .	16
Figure 3.1 The intermediate output of CNN (Lenet,Alexnet and GoogleNet) layers by caffe2 framework . . . . .	28
Figure 3.2 Inference time <i>vs.</i> Batch size. . . . .	29
Figure 3.3 System overview of cooperation of mobile devices for fast CNN inference . . . . .	30
Figure 3.4 Approximation of nonlinear constraints. . . . .	35
Figure 3.5 The expected task processing time under different $t_{ik}$ . . . . .	38
Figure 3.6 The average task completion time under different number of mobile users. . . . .	40
Figure 3.7 The percentage of mobile users choosing splitting under different network bandwidth. . . . .	41
Figure 3.8 The average tasks completion time under different mobile computing capability. . . . .	42
Figure 3.9 The average tasks completion time under different network bandwidth. . . . .	43
Figure 3.10 The average tasks completion time under different network bandwidth based for online scenario. . . . .	43
Figure 3.11 The average task completion time of online settings under different number of users. . . . .	44
Figure 4.1 Inference time <i>vs.</i> Batch size. . . . .	46
Figure 4.2 System overview of D2D-based computation offloading for CNN inference . . . . .	49
Figure 4.3 a simple illustration for line 7 of Algorithm 2 . . . . .	53
Figure 4.4 Performance analysis comparison of the PSO algorithm and Random Condition. . . . .	57
Figure 4.5 With the growth of the number of users $N$ , which ranging from 2 to 30, the minimum time of PSO resides between Random and Optimal. . . . .	57
Figure 4.6 Number of aggregation nodes selected over Optimal, PSO and Random. . . . .	58

Figure 4.7 With the growth of the number of users $N$ , which ranging from 2 to 30, the minimum time of Distributed Algorithm resides between NonCollaborative and Optimal. . . . .	58
Figure 4.8 Number of aggregation nodes selected over Distributed Algorithm and Optimal. . . . .	59
Figure 5.1 Model Overview . . . . .	64
Figure 5.2 Schematic of DQN . . . . .	71
Figure 5.3 The loss during training process . . . . .	75
Figure 5.4 The percentage of energy saved compared with Non-SO2 at different time slot (hour) . . . . .	76



# List of Tables

Table 3.1	The percentage of mobile users choosing different splitting layers. . . . .	41
Table 4.1	Variables and symbols . . . . .	52
Table 5.1	Notations . . . . .	66

# List of Abbreviations

AI	Applications Intelligence
CNNs	Convolutional neural networks
DL	Deep learning
DRL	Deep Reinforcement Learning
DNNs	Deep neural networks
EC	Edge Computing
FPGA	Field-Programmable Gate Array
GPU	Graphic Process Unit
MCC	Mobile Cloud Computing
ML	Machine learning
NLP	Natural Language Processing
PSO	Partial Swarm Optimization
RAN	Radio Access Network
RNNs	Recurrent neural networks
VR	Virtual Reality

# Abstract

Algorithmic breakthroughs of deep learning make people enjoy a more convenient as well as smarter mobile life. Convolutional Neural Networks (CNNs) is an important computation model for many popular mobile artificial intelligence applications. As the accuracy is ensured by the CNN model, the performance of deep learning service mainly depends on the response time for handling user demands, which includes the network transmission time, task scheduling time, inference time (the execution time of the deep neural network (DNN) inference), and so forth. In response time, inference time usually occupies the dominant portion, especially for a complicated DNN model. In other words, CNN inference, i.e., processing input data based on well-trained CNN models, is computation-intensive and incurs a heavy overhead for mobile devices with limited hardware resources (e.g., storage, battery). Compared a natural way to tackle this challenge is to employ cloud computing by offloading the computation tasks to remote servers which has two major concerns like high bandwidth requirements and transmission latency.

We find the discrepancy memory size of each layer for different CNN models which means mobile devices can conduct inference applications over a few layers so that the intermediate data smaller to reduce the transmission time. Therefore, in order to maximize the utilization of edge and mobile resources (computation resource, communication, and storage) in the efficient deep learning service, we

first propose to offload a portion of CNN inference computation of mobile devices to the edge computing site due to the findings that batching tasks on GPUs can significantly reduce average inference time. We design an algorithm that jointly considers the tasks on all mobile devices and the corresponding batching benefit on the edge site, different from existing work on the collaborative inference that lets each mobile device independently make offloading decisions. Different from simply offloading the whole task to the edge, we focus on partial offloading, i.e., the mobile devices conduct inference computation over a few layers of CNN models and then send the intermediate results to the edge computing site, which completes the computation of the rest layers. Such a kind of collaborative inference approach has been shown very effective in further reducing inference time and improving the energy efficiency of mobile devices. Finally, extensive simulations are conducted to evaluate the performance of our proposed algorithms and the results show they outperform existing work under different settings.

Subsequently, mobile devices become much powerful than ever with greater computing capacity and longer battery life to install deep learning-based applications. As a supplement of MEC, when mobile devices are relatively close to each other it could be sensible to have a direct communication link instead of delivering data via a base station in order to achieve low latency and save transmission power of both devices and base stations, radio access as well as core network resources. Many mobile devices now are able to collaborate on computing tasks by sharing their resources (e.g., CPU, GPU), which motivate us to aggregate workloads of mobile devices to efficiently accelerate the deep learning inference process, considering the findings that batched workload with GPUs can reduce the overhead of

GPU memory access.

To meet this demand, we propose to employ partial swarm optimization (PSO) which is a versatile population-based stochastic optimization technique, to help design our collaborative inference scheme. Moreover, extensive simulations are implemented to evaluate the performance of the designed algorithm. By performance evaluation, we find that the collaborative inference scheme can reduce global dealing time in the given field compared with handling the data which is affected by the high transmission latency between mobile devices and the cloud.

Since the edge servers are not always keeping a high-load operation status which still consumes energy, in order to study the energy efficiency problem of edge nodes with switching ON/OFF (SO2) strategy by migrating the tasks, we propose to apply the Deep Reinforcement Learning-based method to tackle the challenge that we have no knowledge of incoming tasks that can be formulated into a sequential decision-making problem since the edge nodes continue to receive tasks from consumers that cause the workloads to vary in space and time. Furthermore, simulation experiments are conducted to evaluate the proposed scheme, and the results show that the switching ON/OFF strategy can save energy compared with the general situations.

# 概要

ディープラーニングのアルゴリズムの飛躍的進歩により、人々はより便利でスマートなモバイルライフを楽しむことができます。畳み込みニューラルネットワーク（CNN）は、多くの一般的なモバイル人工知能アプリケーションにとって重要な計算モデルです。CNN モデルによって精度が保証されるため、深層学習サービスのパフォーマンスは、主に、ネットワーク送信時間、タスクスケジューリング時間、推論時間（深層ニューラルネットワークの実行時間）など、ユーザーの要求を処理するための応答時間に依存します。DNN）推論）など。応答時間では、特に複雑な DNN モデルの場合、通常、推論時間が支配的な部分を占めます。言い換えると、CNN 推論、つまり、十分にトレーニングされた CNN モデルに基づいて入力データを処理することは、計算量が多く、ハードウェアリソース（ストレージ、バッテリーなど）が限られているモバイルデバイスでは大きなオーバーヘッドが発生します。この課題に取り組む自然な方法と比較すると、計算タスクをリモートサーバーにオフロードしてクラウドコンピューティングを採用することです。これには、高帯域幅要件と伝送遅延などの 2 つの大きな懸念があります。

異なる CNN モデルの各層のメモリサイズの不一致が見つかりました。これは、モバイルデバイスが数層にわたって推論アプリケーションを実行できるため、中間データが小さくなり、送信時間が短縮されることを意味します。したがって、効率的な深層学習サービスでエッジおよびモバイルリソース（計

算リソース、通信、およびストレージ)を最大限に活用するために、まず、モバイルデバイスの CNN 推論計算の一部をエッジコンピューティングサイトにオフロードすることを提案します。GPU でタスクをバッチ処理すると、平均推論時間を大幅に短縮できるという調査結果。すべてのモバイルデバイスでのタスクと、エッジサイトでの対応するバッチ処理のメリットを共同で検討するアルゴリズムを設計します。これは、各モバイルデバイスが独立してオフロードの決定を行うことができる共同推論に関する既存の作業とは異なります。タスク全体を単にエッジにオフロードするのとは異なり、部分的なオフロードに焦点を当てます。つまり、モバイルデバイスは CNN モデルのいくつかのレイヤーに対して推論計算を実行し、中間結果をエッジコンピューティングサイトに送信して、計算を完了します。残りのレイヤー。このような種類の協調的推論アプローチは、推論時間をさらに短縮し、モバイルデバイスのエネルギー効率を改善するのに非常に効果的であることが示されています。最後に、提案されたアルゴリズムのパフォーマンスを評価するために広範なシミュレーションが実行され、その結果は、さまざまな設定で既存の作業よりも優れていることを示しています。

その後、モバイルデバイスはこれまでになく強力になり、ディープラーニングベースのアプリケーションをインストールするためのコンピューティング容量とバッテリー寿命が向上します。MEC の補足として、モバイルデバイスが互いに比較的近い場合、低遅延を実現し、デバイスと基地局の両方の送信電力を節約するために、基地局を介してデータを配信する代わりに、直接通信リンクを使用することが賢明です。無線アクセスとコアネットワークリソース。多くのモバイルデバイスは、リソース (CPU、GPU など) を共有することでコンピューティングタスクで共同作業できるようになりました。こ



れにより、モバイルデバイスのワークロードを集約して、ディープラーニング推論プロセスを効率的に加速することができます。GPU メモリアクセスのオーバーヘッドを削減します。

この需要を満たすために、多目的な母集団ベースの確率的最適化手法である部分群最適化 (PSO) を採用して、協調的推論スキームの設計を支援することを提案します。さらに、設計されたアルゴリズムのパフォーマンスを評価するために、広範なシミュレーションが実装されています。パフォーマンス評価により、協調的推論スキームは、モバイルデバイスとクラウド間の高い伝送遅延の影響を受けるデータを処理する場合と比較して、特定のフィールドでのグローバルな取引時間を短縮できることがわかりました。

エッジサーバーは常にエネルギーを消費する高負荷動作状態を維持しているわけではないため、タスクを移行してオン/オフ (SO2) 戦略を切り替えるエッジノードのエネルギー効率の問題を調査するために、ディープを適用することを提案します。エッジノードは、ワークロードの空間と時間の変化を引き起こすタスクを消費者から受け取り続けるため、順次意思決定の問題に定式化できる着信タスクの知識がないという課題に取り組むための強化学習ベースの方法。さらに、提案方式を評価するためにシミュレーション実験を実施し、その結果、ON / OFF の切り替え戦略が一般的な状況と比較してエネルギーを節約できることを示しています。

# Chapter 1

## Introduction

### 1.1 Introduction

The deep learning techniques have drawn ever-increasing research interests thanks to their inherent capability of overcoming the drawback of conventional machine learning algorithms dependent on hand-designed features. Deep learning belongs to a subset of AI and machine learning that applies multi-layers neural networks to deliver state-of-the-art accuracy in tasks, such as objective recognition, language translation, voice recognition, and others.

In most current breakthrough, application intelligence depends on deep neural networks (DNNs) as a critical part of the solution. The major architectures of networks that deep learning offers are convolutional neural networks (CNNs). Simple applications of CNNs that impact our everyday life are obvious choices such as facial recognition software, image classification, speech recognition programs. A typical working process of CNN-based mobile applications usually contains two stages, as depicted in Figure 1.1. The first stage is training, i.e., software developers train the CNN models using a large amount of training data and computing power. The well-trained models are then released to users who can download from the cloud platform and install these models on their mobile devices. In the second stage, users capture data, e.g., an image or a voice, using their mobile devices.

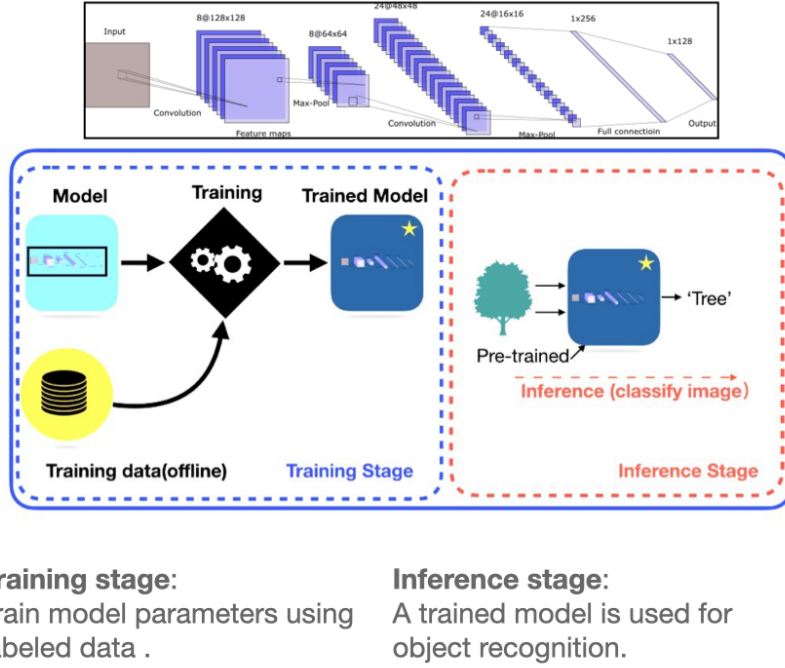


Figure 1.1: General process for mobile deep learning applications

These data are processed by CPU or Graphic Process Unit (GPU) according to the CNN models for object recognition or content understanding. This stage is also named inference, which deep learning service for users. In [1], the authors have been shown that CNN inference contains intensive computation, which incurs a heavy computational burden for mobile devices with limited power and hardware resources. Hence, in order to make the DNN inference efficient to service users with high quality, many existing works have been focused on the following areas:

- **Device-based Approach:**

Due to smart devices, particular smartphones with novel deep learning-based applications (e.g., Siri and Google Assistant) deployed, become our everyday companions and the ubiquitous mobile Internet and computing applications pervade people's daily life. It will reduce the response time (inference time, transmission time) when the trained models are directly deployed on mobile devices since raw data can be processed locally once generated without transmitted to the remote servers. Although today's mobile devices become

more powerful than ever with greater computing capacity and longer battery life, the energy constraints still remain unprecedented challenges to implement the novel mobile applications in an energy-efficient manner, because it has been shown that CNN inference contains intensive computation, which incurs a heavy computational burden for mobile devices with limited power and hardware resources [1].

- **Cloud-based Approach:**

By migrating computational tasks from mobile devices to the infrastructure-based cloud servers, the performance of mobile applications can be improved with mobile cloud computing (MCC) and the energy consumption of mobile devices will be reduced [2]. MCC as a computational paradigm and an alternative to conventional computing that enables ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and GPUs), has the potential to address the aforementioned challenges [3]. However, this cloud-only approach requires a massive amount of raw data (aka, images, videos, and context) to be transmitted to the cloud servers via the wireless network, bring significant data transmission overhead to the network and computation burden on the data-center. Meanwhile, the infrastructure-based cloud servers are always located centrally in the core network and far away from mobile devices, which causes high transmission latency and detains the latency-sensitive applications (aka, augmented reality (AR)), significantly affecting the performance of learning-based applications. The average inference latency has been found in the order of 1.5 seconds, in which the high percentile latency exceeds the 2 seconds front [4].

- **Edge-based Approach:**

To address the drawback that is hailed from MCC systems, mobile edge computing (MEC) as a promising approach to improve the offloading efficiency

is proposed with cloud resources, e.g., storage and processing capacity to the edge within the radio access network (RAN), in which computing and storage nodes are placed at the Internet's edge in close proximity to mobile devices or sensors. With MEC, mobile devices can execute their respective tasks locally by themselves or offload part or all of them by the access point, rather than utilizing the servers in the core network. In the MEC scenario, there exist many significant benefits (aka, ultra-low latency, network congestion reduction), which are necessary for emerging applications such as IoT, online gaming, AR and connected cars [5]. In [6], Rimal *et al.* investigates performance gains of centralized cloud and MEC enabled integrated fiber-wireless (FiWi) access networks. They propose a novel unified resource management scheme that incorporates both centralized cloud and MEC computation offloading tasks into the underlying FiWi dynamic bandwidth allocation process. In addition, Figure 1.2 depicts the capacities comparison of cloud-based and edge-based approaches. While the empirical study shows that the performance of edge-based DNN inference is still highly sensitive to the available bandwidth between edge servers and mobile devices with considering the benefits of edge intelligence. With the bandwidth dropping from 1 Mbps to 50 Kbps, the latency of edge-based DNN inference grows from 0.123s to 2.317s and becomes on par with the latency of local processing on the device [7].

- **Edge-based CNN-splitting:**

As the accuracy of deep learning-based applications is ensured by the CNN model, the performance of deep learning service mainly depends on the response time for handling user requests, which includes the network transmission time, task scheduling time, inference time (the execution time of the DNNs inference), and so forth. In response time, inference time usually occupies the dominant portion, especially for a complicated DNN model. The discrepancy memory size of each layer for different CNN model which

means mobile devices can conduct inference applications over a few layers so that the intermediate data smaller to reduce the transmission time, in Figure 3.1. Considering the computation graph of CNN provides insights to identify a better computation splitting between mobile devices and the cloud at the layer level. Such a kind of collaborative inference approach has been shown very effective in further reducing inference time and improving the energy efficiency of mobile devices. The findings show that data size is generally decreasing at the front-end, while the mobile latency of per-layer is generally higher at the back-end, which means there is a unique opportunity for computation splitting in the middle of the CNN between mobile devices and the cloud [8].

In addition, the energy consumption of servers is becoming the major concern due to the increased requirements for modern computing-intensive applications [9]. In [10], Ehsan Ahvar *et al.* analyze four different Cloud-related architectures (i.e., fully centralized (FC), partly distributed (PD), Fully Distributed with Centralized Controller (FDCC) and fully distributed (FD)) by using the energy model. The results showed that an FD architecture consumes between 14% and 25% less energy than FC and partly distributed (PD) architectures respectively. With the result of this analysis, future work on greening emerging cloud architectures should focus on the improvement of the usage of telecommunication networks, the efficiency of computing infrastructures, and the exploitation of heterogeneous infrastructures to better meet the users' demands. The energy efficiency and energy proportionality (EP) of edge server vary in different time and space, thanks to the configuration and workload of the edge server are different [11]. For example, the small-size and medium-sized servers are not always keeping a high-load operating state which still consumes much energy when they are idle. For small- and medium-sized edge servers, are not always in a high-load operating state and thus the servers still consume a lot of energy when they are idle. The server utilization of small scale is much lower than the large scale's as well as their power use effectiveness [12].

We find that the existing works on the collaborative inference just let each mobile device independently make offloading decisions. Different from existing works [8] that only focus on the partition scheme between a single mobile device and a cloud server, we consider multiple devices and edge servers where workloads are performed by batching. Mobile devices become much powerful than ever with greater computing capacity and longer battery life to install deep learning-based applications. Different from traditional device-to-device cooperation [13,14], many mobile devices now are able to collaborate on computing tasks by sharing their resources (e.g., CPU, GPU), which motivates us to aggregate workloads of mobile devices to efficiently accelerate the deep learning inference process, considering the findings that batched workload with GPUs can reduce the overhead of GPU memory access. Compared with the edge site, local mobile devices are closer in crowded scenarios (e.g., shopping malls, stadiums) which leads to low latency. The incoming requirements might be sporadic and the workloads of the server fluctuate on an hourly, daily, or weekly basis. In [15], Luiz *et al.* finds that real systems attain peak efficiency at peak utilization but quickly lose efficiency as utilization drops because they are unable to reduce power consumption proportionately. They believe that an ideal energy-proportional (EP) system should always use energy in proportion to the work done by maintaining the peak efficiency, even at a reduced load. Although there are some existing works on increasing the network traffic, optimizing the data transmission, or model optimization, there still is a lack of studies towards carrying out computing tasks offloading and aggregating task schemes, especially utilizing the GPU to accelerate the deep learning inference. Hence, we mainly study three works to implement fast DNN inference and energy efficiency of edge nodes with the following challenges.

To fully exploit the benefit of workload batching at GPU and accelerate the deep learning inference process, we propose to jointly consider the CNN offloading and workload batching, with the objective of minimizing the average inference time of a given number of tasks. Thanks to the diverse user demands in time



and space, it remains challenging to find a suitable batching scheme for the edge sites to achieve efficient DNN inference with offloading a portion of CNN inference computation of mobile devices to the edge computing site.

In order to exploit the local offloading that enables collaborative CNN inference among local mobile devices, for mobile devices, it's difficult to select a suitable batch size for aggregation nodes (like a server) who is responsible for collecting tasks from their neighbors, and it is a time-consuming process to collect the global offload information of devices. Different from the existing works that let each mobile device independently makes offloading decisions according to the network connections, ignoring the acceleration chances by aggregating, our work also has the challenge to address the nonlinear relationship among the offloaded tasks by non-aggregation nodes.

The goal of the third work is to investigate the energy efficiency problem with the consideration of task migration in the MEC environment, according to the status of edge servers (idle or low-load). Due to the edge nodes continue to receive tasks from consumers that causes the workloads to vary in space and time, the main challenge is that we have no knowledge of the incoming tasks that can be formulated into a sequential decision-making problem. Meanwhile, the conventional tasks scheduling algorithm can't solve the dynamic tasks migration problem efficiently.

To tackle the first challenge, we design an algorithm that jointly considers the tasks on all mobile devices and the corresponding batching benefit on the edge site. Instead of simply offloading the whole task to the edge, we focus on partial offloading, i.e., the mobile devices conduct inference computation over a few layers of CNN models and then send the intermediate results to the edge computing site, which completes the computation of the rest layers. Next, an online algorithm has been proposed to handle the scenario that users submit their requests at a different time; Finally, extensive trace-driven simulations have been conducted and the results show that our proposal significantly outperforms existing approaches.

In order to address the second challenge, we investigate employing local offloading to enable collaborative inference among local mobile devices due to the local offloading like fog computing and mobile edge computing is developed. It can achieve energy efficiency indirectly by reducing the execution of workloads on mobile devices, thanks to the average power consumption of the CPU circle is stable. We design a heuristics algorithm based on PSO to efficiently minimize the total time costs for collaborative inference, with a dynamic procedure of selecting the optimal computing nodes from local mobile devices. In addition, we propose a distributed algorithm in which each mobile device uses its one-hop information to make offloading decisions. We finally conduct extensive simulations to evaluate the performances of our proposed algorithm and demonstrate its comprehensive advantages to existing solutions.

In the final work, different from the existing works, we not only exploit the switching ON/OFF strategies, but the migration portion of tasks of the inactive edge nodes. The goal is to fully minimize the energy cost by dynamically switching ON/OFF the edge nodes by considering the status of workload. Since deep reinforcement learning (DRL) is being able to address a wide range of complex decision-making tasks by integrating deep learning with reinforcement learning [16], we exploit to address this problem by using a learning-based method. To solve the reinforcement learning problem, we formulate it as an optimization problem that can be learned by conventional learning methods.

## 1.2 Organization

In this dissertation, we mainly focus on the collaborative inference for deep learning applications between edge servers and mobile devices and split the CNNs model so as to improve the efficiency of deep learning service based on the finding that batching tasks on GPU can significantly reduce average inference time on GPUs. As illustrated in Figure 1.3, the rest of the thesis is organized in the

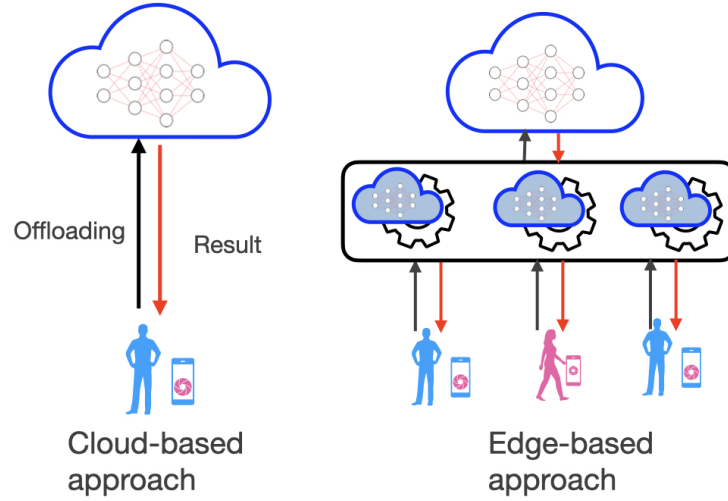


Figure 1.2: Capabilities comparison of cloud-based approach and edge-based approach

following pattern.

In **Chapter 2**, the research background is given, which mainly includes the requirements of efficient Applications Intelligence (AI) within Edge Computing (EC) environment.

Next, in **Chapter 3**, we propose to offload a portion of CNN inference computation of mobile devices to the edge computing site. We find that batching tasks on GPU can significantly reduce average inference time on GPUs. Based on this important observation, we design an algorithm that jointly considers the tasks on all mobile devices and the corresponding batching benefit on the edge site, different from existing work on the collaborative inference that lets each mobile device independently make offloading decisions. Furthermore, an online algorithm is proposed to handle the scenario that CNN inference tasks arrive at different times. It can significantly reduce average inference time without the knowledge of future task arrivals.

Then, in **Chapter 4**, there is the supplement case happens under the edge computing network in which mobile device can share their computation resource to support learning-based applications. Since the typical smartphones, today have 6 to 8 CPU cores or massively parallel GPUs to accelerate learning-based appli-

cations, which has made processing data locally at the end or at the edge of the network more efficient [17]. When communicating devices are relatively close to each other it could be sensible to have a direct communication link instead of conveying data via a base station in order to save transmission power of both user devices and base stations, radio access as well as core network resources [18]. Hence, we mainly propose to employ local offloading to enable collaborative inference among local mobile devices due to the local offloading like fog computing and mobile edge computing is developed. To meet this demand, we investigate employing PSO that is a versatile population-based stochastic optimization technique, to help design our collaborative inference scheme.

In **Chapter 5**, we study the practical scenario that the edge nodes continue to receive tasks from consumers, due to the workloads vary in space and time, to minimize the energy cost of edge nodes by considering the status of workloads by switching ON/OFF strategies.

In this dynamic context, the main challenge is that we have no knowledge of incoming tasks that can be formulated into a sequential decision-making problem, and conventional task scheduling algorithms can't solve the dynamic tasks migration problem efficiently. Since deep reinforcement learning (DRL) is being able to address a wide range of complex decision-making tasks by integrating deep learning with reinforcement learning [16], we exploit to address this problem by using a learning-based method. To solve the reinforcement learning problem, we formulate it as an optimization problem that can be learned by conventional learning methods.

Finally, **Chapter 6** concludes this dissertation.

## 1.3 Publications

The following papers have been published or accepted by journal or conference.

1. Major journal paper

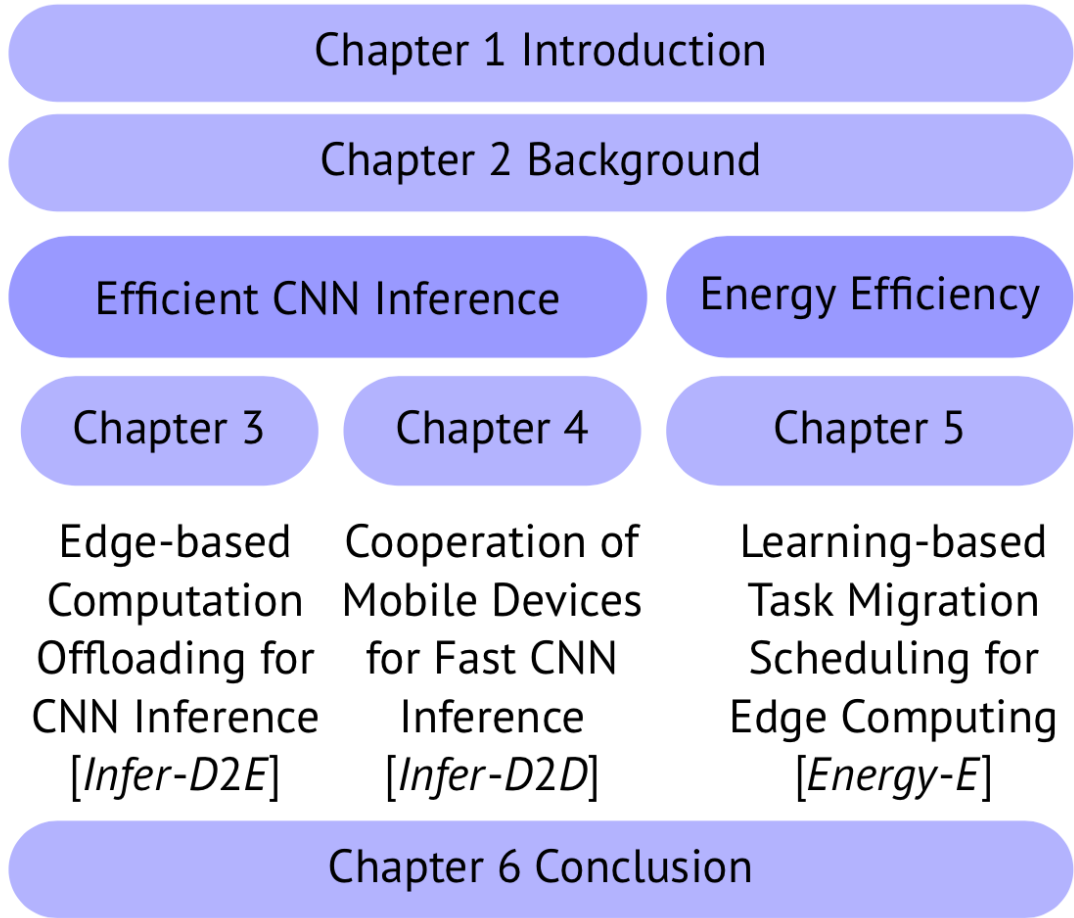


Figure 1.3: Organization of thesis structure

- **Qinglin Yang**, Xiaofei Luo, Peng Li, *et al.* Cooperation of Mobile Devices for Fast Inference of Deep Learning Applications [J]. Mobile Networks and Applications (published)

## 2. Major conference paper

- **Qinglin Yang**, Xiaofei Luo, Peng Li, *et al.* Collaborative Inference for Mobile Deep Learning Applications (Conference) 2nd International Conference on 5G for Ubiquitous Connectivity.
- **Qinglin Yang**, Xiaofei Luo, Peng Li, *et al.* Computation offloading for fast CNN inference in edge computing (ACM Conference) International Conference on Research in Adaptive and Convergent Systems, Chongqing China, September 2019.
- **Qinglin Yang**, Peng Li. Deep Reinforcement Learning based Energy

Scheduling for Edge Computing (*IEEE* Conference) SmartCloud 2020

- Xiaofei Luo, **Qinglin Yang**, Peng Li, et al. Grouping Strategy for RFID-Based Activity Recognition in Smart Home (*IEEE* Conference). 2019 International Conference on Internet of Things (iThings)
- Hongzhi Xiao, Chen Qiu, **Qinglin Yang**, et al. Deep Reinforcement Learning for Optimal Resource Allocation in Blockchain-based IoV Secure Systems (*IEEE* Conference). The 16th International Conference on Mobility, Sensing and Networking (MSN 2020)

# Chapter 2

## Background and Related Work

### 2.1 Background

#### 2.1.1 Efficient Deep Learning

Intelligence applications have an aspect much of people's life due to the advantage of learning-based (Machine learning ) algorithm. Machine learning as introduced by Arthur Samuel [19], is a "Field of study that gives computers the ability to learn without being explicitly programmed", which is widely applied in the real-world to produce predictive models for applications intelligence such as classification and natural language processing (NLP). With Machine Learning technology, human-level artificial intelligence has been improved as well as [20].

Deep learning algorithms are used to extract high-level features from these data without human manipulation in order to obtain hierarchical representations, which is a subset of Machine learning. As shown in Figure 2.1, the schematic diagram of general deep neural networks full connected neural networks (FCNNs), recurrent neural networks (RNNs), and convolutional neural networks (CNNs) are given. FCNNs is a multi-layer network with many hidden layers, whose weights are fully connected. In the last few years, thanks to the increase in computational power, the amount of available training data, and the tricks for training deep nets, CNNs have managed to achieve superhuman performance on some complex visual



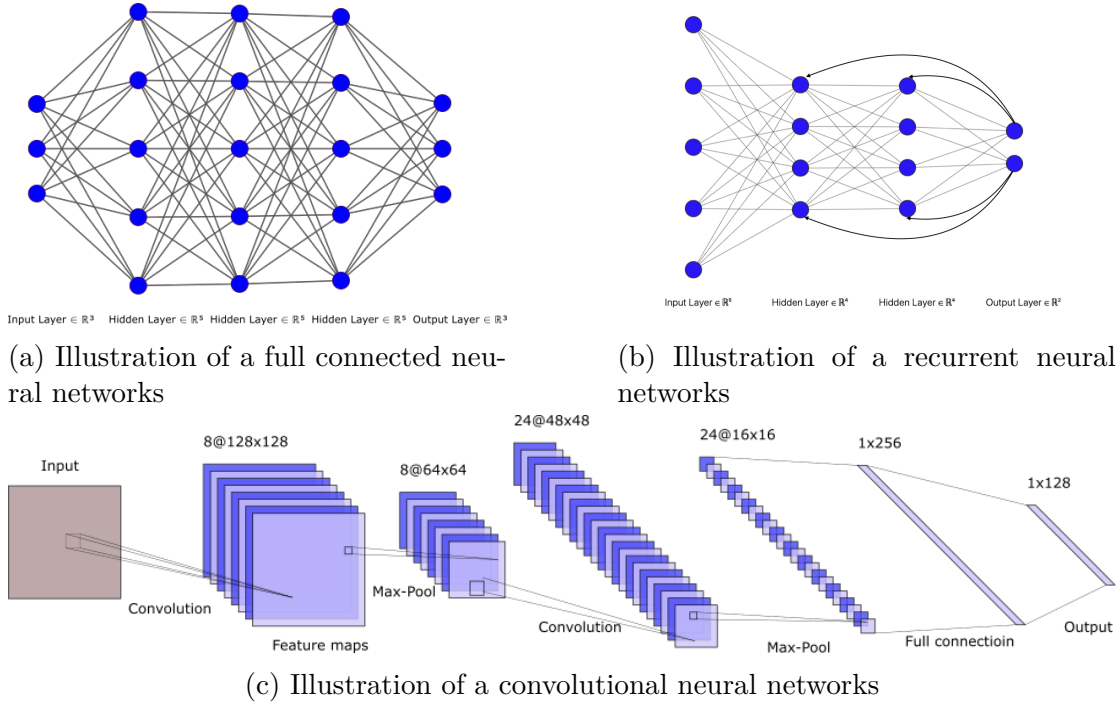


Figure 2.1: Schematic diagram of DNNs

tasks. Generally, CNNs includes two operations named *convolution* and *pooling* to reduce an image into its essential features which are used to understand and classify the image. The *convolution* operation is a dot product of the pre-layer values with a filter that filled with weights. The results are summed up into one number that includes all the information the filter observed. *pooling* is the process of further down-sampling and reducing the size of the feature matrix. There might be multiple activations and pooling layers, according to the CNN architecture.

CNNs have shown predominant performance in processing images and videos, thanks to their local connectivity and weight sharing. The CNN topology reduces the number of parameters in the neural network by leveraging spatial relationships so as to improve the performance and deployed on mobile devices easily. While it still remains drawbacks to deploy a CNN-based (trained) model on a mobile device since the requirement of high computational resources and memory compared to conventional methods. An inference process with deep learning involves tens of billions of mathematical operations and tens of millions of parameter reads. Hewitt *et al.* [21] propose three types of established CNN architectures and

comparatively evaluate them on a large, in-the-wild benchmark dataset of facial images with the difference that models deployed to mobile devices must minimize storage requirements while retaining high performance. There also are some cloud-based approaches to deploy the trained CNN-based model in the cloud to mitigate the burden of the mobile device. However, the cloud-based approaches incur drawbacks (high latency, congestion) due to the remote location of servers which requires a significant amount of raw data to be transmitted via the wireless network and the centralized architecture brings a significant computational burden on the data-center.

Nowadays, the Internet of Things (IoT) blends in with our daily lives, providing important measurement and collection tools to inform our every decision. Millions of sensors and devices are continuously producing data and exchanging important messages via complex networks supporting machine-to-machine (M2M) communications and monitoring and controlling critical smart-world infrastructures [22].

The notion of cloud computing as depicted in Figure 2.2a has not only reshaped the field of distributed systems but also fundamentally changed how businesses utilize computing and communications today. For example, in order to mitigate the burden of mobile devices, the computation tasks are often offloaded to the cloud site. However, the infrastructure-based cloud servers are always located centrally in the core network and far away from mobile devices. The long transmission from the mobile devices to the cloud servers may cause delay fluctuation and invoke extra transmission energy cost [23].

Moreover, due to the surge of mobile devices, conventional centralized cloud computing is struggling to meet the QoS for many applications. Thus, computation offloading efficiency can severely degrade. Compared with the well-known cloud computing, edge computing (EC) will bring data computation or storage to the network 'edge' close to the end-users. The three-layer architecture of EC is depicted in Figure 2.2b. As shown, different types of mobile devices and sensors, e.g., Internet of Vehicles (IoVs), big data, and social platforms, are connected

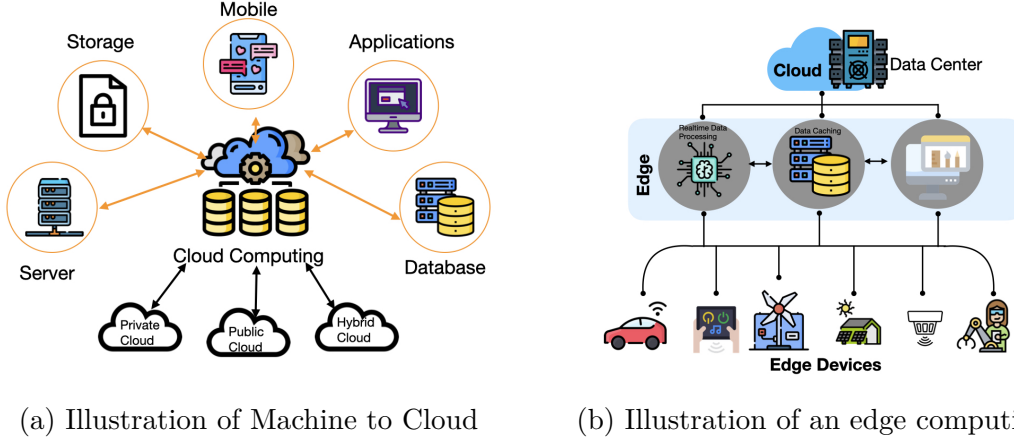


Figure 2.2: Schematic diagram of Computing

to the core network (i.e., mobile Internet) through the edge network, i.e., the RAN and MEC, and the core network are connected to the private cloud network. Moreover, each edge platform represents an edge cloud with specific applications and services to the target mobile environment. This novel feature enables computation-intensive and latency-critical DNN-based applications to be executed in a real-time responsive manner (i.e., edge intelligence) [24] in comparison with traditional cloud services. By leveraging edge computing, we believe that it is significant to achieve efficient deep learning service for supporting real-time edge AI applications.

## 2.2 Related Works

In this section, we review the related work on efficient DNN application and the energy efficiency of edge nodes.

### 2.2.1 Efficient Deep Learning Applications

Different from traditional machine learning and artificial intelligence approaches, deep learning technologies have recently been progressing massively with successful applications to speech recognition, NLP, information retrieval, compute vision, and image analysis [25, 26]. In [7, 27], Deep Learning technologies are introduced

under the edge computing environment, to achieve edge intelligence. It's successful to apply deep learning techniques to phone recognition and large vocabulary speech recognition tasks by leveraging the superiority of DNNs with the sequential modeling ability of hidden Markov models [28]. With the collaboration of researchers between academic and industrial, deep learning and DNNs affect speech recognition gradually from 2010. It leads to better phone recognition performance by designing the pooling scheme in the deep CNNs to balance the trade-off between invariance and discrimination among speech sounds [28]. Andrew *et al.* propose an approach to speech recognition that uses only a neural network to map acoustic input to characters, a character-level language model, and a beam search decoding procedure by eliminating much of the complex infrastructure of modern speech recognition systems, making it possible to directly train a speech recognizer using errors generated by spoken language understanding tasks [29]. Dario Amodei *et al.* show that it is possible to recognize either English or Mandarin Chinese speech—two vastly different languages with an end-to-end deep learning approach. Then, they apply HPC techniques to enable experiments that previously took weeks to now run in days [30].

Hu *et al.* implement an extensive evaluation of CNN-based face recognition systems (CNN-FRS) on a general ground to make their work easily reproducible and public database LFW (Labeled Faces in the Wild) are applied to train CNNs instead of private databases in most existing work [31]. Liu *et al.* propose a two-stage approach to extracts low dimensional but very discriminative features for face verification and recognition by combines a multi-patch deep CNN and deep metric learning [32].

Fu *et al.* propose a novel convolutional neural network with recurrent attention, which learns discriminative region attention and region-based feature representation at multiple scales in a mutually reinforced manner. The learning is composed of a classification sub-network and an attention proposal sub-network (APN) which starts from full images and iteratively generates region attention

from coarse to fine by taking the previous prediction as a reference, while the finer scale network takes as input an amplified attended region from the previous scale in a recurrent way at each scale [33].

Tian *et al.* propose DeepParts, which can be trained on weekly labeled data, i.e. only pedestrian bounding boxes without part annotations are provided and be able to handle low IoU positive proposals that shift away from the ground truth. Furthermore, only observing a part of a proposal can satisfy the requirement of pedestrian detection [34].

Chu *et al.* propose a CNN-based framework for online MOT. This framework utilizes the merits of single object trackers in adapting appearance models and searching for a target in the next frame. Simply applying a single object tracker for MOT will encounter the problem in computational efficiency and drifted results caused by occlusion. Their framework achieves computational efficiency by sharing features and using ROI-Pooling to obtain individual features for each target. Some online learned target-specific CNN layers are used for adapting the appearance model for each target. In the framework, they introduce a spatial-temporal attention mechanism (STAM) to handle the drift caused by occlusion and interaction among targets. The visibility map of the target is learned and used for inferring the spatial attention map. The spatial attention map is then applied to weigh the features. Besides, the occlusion status can be estimated from the visibility map, which controls the online updating process via weighted loss on training samples with different occlusion statuses in different frames. It can be considered as a temporal attention mechanism [35].

Mehdi *et al.* mainly evaluate the performance of Google speech recognition and Apple Siri the two of the most popular cloud-based speech recognition systems under different network conditions considering command recognition accuracy and round trip delay [36]. More details of evolution, models applications and future trends for deep learning technologies can be found in [37–42].

In [43], Adarsh Kumar *et al.* find that caching intermediate layer outputs can

help us avoid running all the layers of a DNN for a sizeable fraction of inference requests and this can potentially reduce the number of effective layers by half for 91.58% of CIFAR-10 requests run on ResNet-18. Then, they present a system named Freeze Inference that studies approximate caching at each intermediate layer and discuss techniques to reduce the cache size and improve the cache hit rate. In [44], Angad *et al.* first analyze the applicability of AMS approaches to larger networks by proposing a generic AMS error model, implementing it in an existing training framework, and investigating its effect on ImageNet classification with ResNet-50. Furthermore, they show that significant accuracy recovery by exposing the network to AMS error during retraining and demonstrate that batch normalization layers are responsible for this accuracy recovery. Dalton *et al.* propose a novel remote sensing data flow (RESFlow) by partitioning data into homogeneous distributions for fitting simple models for advancing machine learning to compute with massive amounts of remotely sensed imagery [45].

Thomas *et al.* investigate to employ deep learning technologies to leverage observations and model data by predicting unresolved turbulent processes and subsurface flow fields. They find that it is successful to replicate the spatiotemporal variability of the subgrid eddy momentum forcing are capable to generalize a range of dynamical behaviors by CNNs [46]. In [47], the authors propose a new fine-grained dynamic pruning technique for CNN inference, named channel gating (CG), and present an accelerator architecture that can effectively exploit the dynamic sparsity. In order to maximize computation savings while minimizing accuracy loss, CG learns the gating thresholds together with weights automatically by training. Sayeh Sharify *et al.* proposes Laconic which is a hardware accelerator that implements to boost energy efficiency for transparently identifying ineffectual computations during the inference process with deep learning models [48]. In [49], Sumanth Gudaparthi *et al.* propose a novel wire-aware CNN accelerator by employing a deep and distributed memory hierarchy, WAX, which enables g data movement over short wires in the general context.

In [50], Hyoukjun Kwon *et al.* introduce a set of data-centric directives to concisely specify the DNN data-flow space in a compiler-friendly form. They show how these directives can be analyzed to infer various forms of reuse and to exploit them using hardware capabilities. Then, the authors structure this analysis into an analytical cost model, MAESTRO (Modeling Accelerator Efficiency via Spatial-Temporal Reuse and Occupancy), that estimates various cost-benefit trade-offs of a data-flow including execution time and energy efficiency for a DNN model and hardware configuration. Li *et al.* find that soft errors can lead to catastrophic failures in DNN systems, which incurred by high-energy particles have been increasing in hardware systems. They give two efficient protection techniques for DNN systems, according to the observation that the error resilience of a DNN system depends on the data types, values, data reuses, and types of layers in the design [51]. Lillian Pentecost *et al.* design t MaxNVM, a principled co-design of sparse encoding, protective logic, and fault-prone MLC eNVM technologies (i.e., RRAM and CTT) to enable highly-efficient DNN inference. They improve the storage density (i.e., bits-per-cell) with minimal overhead by using protective logic to avoid the limitation caused by bit reduction techniques (e.g., clustering and sparse compression) [52].

### 2.2.2 Edge Intelligence and Energy Efficiency of Edge

In [53, 54], many existing works on resources allocation and scheduling problems in cloud computing are investigated. Anton *et al.* first define an architectural framework and principles for energy-efficient CC. The main study resources provisioning and allocation algorithms for energy-efficient management of Cloud computing environments [55, 56]. In [57], Xu *et al.* first propose an algorithm of job scheduling based on the Berger model, considering the commercialization and the virtualization characteristics of CC. The characteristics of the Fog computing are described: 1) low latency; 2) Wide-spread geographical distribution; 3) devices mobility; 4) massive amount of nodes; 5) heterogeneity; 6) wireless access [58].



MEC has received an increase in attention in decades. To jointly tackle computation offloading and content caching strategies under wireless cellular networks with mobile edge computing, Wang *et al.* formulate the computation offloading decision, resources allocation, and content caching strategy as an optimization problem, the total revenue of the network is considered [59]. Yu *et al.* implement a dynamic offloading framework for mobile users with the local overhead in the mobile terminal side considered, as well as the limited communication and computation resources on the network side. The offloading problem is formulated as a multi-label classification problem and develop the Deep Supervised Learning (DSL) method by minimizing the computation and offloading overhead [60]. In [61], Zhang *et al.* apply a deep Q-learning approach to design optimal offloading schemes, with the joint selection of the target server and determination of data transmission mode considered.

In order to provide satisfactory computation performance as well as achieving green computing, Mao *et al.* propose a green MEC system with EH devices and develop an effective computation offloading strategy, which is of significant importance to seek renewable energy sources to power mobile devices via energy harvesting (EH) technologies [62]. Hao *et al.* propose a new concept of task caching which refers to the caching of completed task application and their related data in the edge cloud. Then, they study the problem as a joint optimization of task caching and offloading on edge cloud with the computing and storage resources constraints [63]. Zhang *et al.* investigate energy-efficient computation offloading (EECO) schemes for MEC in 5G heterogeneous networks. In order to minimize the energy consumption of the offloading system, where the energy cost of both task computing and file transmission is taken into consideration, an optimization problem is proposed [3]. Zhang *et al.* show an energy-aware offloading scheme, which jointly optimizes communication and computation resources allocation under the limited energy and sensitive latency to study the trade-off between energy consumption and latency [64]. Zhao *et al.* investigate a multi-mobile-users MEC

system, in which multiple smart mobile devices (SMDs) ask for computation offloading from a MEC server. They jointly optimize the offloading selection, radio resources allocation, and computational resources allocation coordinately to minimize the energy consumption on SMDs. The energy consumption minimization problem is formulated as a mixed-integer nonlinear programming (MINLP) problem, which is subject to specific application latency constraints [65].

Chen *et al.* propose a novel D2D Crowd framework for 5G mobile edge computing to achieve energy-efficient collaborative task executions at the network edge for mobile users, where a massive crowd of devices at the network edge leverage network-assisted D2D collaboration for computation and communication resources sharing [66]. Anselme *et al.* formulate the problem of joint Computing, Caching, Communication, and Control (4C) in big data MEC as an optimization problem to jointly optimize a linear combination of the bandwidth consumption and network latency [67]. Wang *et al.* propose a unified MEC-wireless power transfer designed by considering a wireless powered multiuser MEC system, where a multi-antenna access point (AP) (integrated with a MEC server) broadcasts wireless power to charge multiple users and each user node relies on the harvested energy to execute computation tasks [68].

In [69], Tuyen X. Tran *et al.* propose a MEC enabled multi-cell wireless network, in which each base station (BS) is equipped with a MEC server that assists mobile users in executing computation-intensive tasks via task offloading. To maximize the users' task offloading gains, which is measured by a weighted sum of reductions in task completion time and energy consumption, jointly optimizing task offloading and resource allocation. In [70], Chen *et al.* investigate the task offloading problem aiming to minimize the delay while saving the battery life of the user's equipment by leveraging the idea of the software-defined network under an ultra-dense network. Specifically, they formulate the task offloading problem as a mixed-integer non-linear program which is NP-hard. In [71], time-division multiple access (TDMA) and orthogonal frequency-division multiple access (OFDMA)

based resources allocation is investigated for a multi-user Mobile-edge computation offloading (MECO) system. First, in order to minimize the weighted sum mobile energy consumption under the constraint on computation latency for the TDMA MECO system which has infinite or finite cloud computation capacity, you *et al.* formulate the optimal resources allocation as a convex optimization problem. While, for the OFDMA MECO system, the optimal resource allocation is formulated as a mixed-integer problem.

In [72], Qiao *et al.* introduce a framework of device-to-device edge computing and networks (D2D-ECN), a new paradigm for computation offloading and data processing with a group of resources-rich devices towards collaborative optimization between communication and computation. They design a reinforcement learning framework in a point-to-point offloading system to overcome challenges of the dynamic nature and uncertainty of renewable energy, channel state, and task generation rates. Thanh Quang Dinh *et al.* propose an optimization framework of offloading from a single mobile device (MD) to multiple edge devices. In order to minimize both total tasks' execution latency and the MD's energy consumption by jointly optimizing the task allocation decision and the MD's central process unit (CPU) frequency, two cases for the MD, i.e., fixed CPU frequency and elastic CPU frequency are considered [73].

Li *et al.* propose a unified energy management framework that supports cooperation between the energy supply system and the edge computing system so that renewable energy can be fully utilized while offering improved quality of service for time-constrained IoT applications, in order to enable a sustainable edge computing paradigm with distributed renewable energy resources [74]. Chen *et al.* propose a novel D2D Crowd framework for 5G mobile edge computing, in which a massive crowd of devices at the network edge leverage network-assisted D2D collaboration for computation and communication resources sharing. A goal of this framework is to achieve energy-efficient collaborative task executions at the network edge for mobile devices [66].

In order to host AI on mobile devices, mobile DNN computation deploys DNN models close to users so that more flexible execution, as well as more secure interaction, can be achieved [24]. However, it remains challenging to execute the computation-intensive DNNs on mobile devices directly since the limited computation resources. There are mainly three aspects in the literature runtime management, model architecture optimization, and hardware acceleration. Runtime management means offloading computation tasks from mobile devices to the cloud or edge server which obtains better performance by using external computation resources. Model architecture optimization exploits to develop novel DNN structure in order to achieve desired accuracy with moderate computation [75]. In [76], the authors propose a novel mobile deep inference platform that achieves good inference performance, named MODI. By MODI, it can orchestrate a centralized model repository and periodically updates models at edge locations, ensuring up-to-date models for mobile applications without incurring high network latency. Meanwhile, it extends the number of models each mobile application can use by caching high-quality models at the edge servers. Li *et al.* proposes a framework, Edgent, which leverages edge computing for DNN collaborative inference with the synergy of device-edge. Based on the two design knobs: (1) DNN partitioning that adaptively splits computation between device and edge for the purpose of coordinating the powerful cloud resource and the proximal edge resource for real-time DNN inference; (2) DNN right-sizing that further reduces computing latency via early exiting inference at an appropriate intermediate DNN layer [77], Edgent can reach effectiveness in enabling on-demand low-latency edge intelligence.

As one of the run-time optimizations approaches, DNN splitting technology segment-specific DNN model into some successive parts and deploys each part on multiple participated devices. In spite of how many devices are involved, DNN splitting dedicates to maximize the utilization of external computation resources with the aiming of accelerating mobile computation. It becomes more and more practicable with the increasing importance of EC through deep learning model

partitioning and offloading to cloud or edge network servers.

Compared with the existing works, we exploit the deep learning model splitting and offload them to edge servers with the batching scheme to achieve efficient DNN inference in the edge environment. Meanwhile, we apply this proposal within the D2D environment since the devices are closer than the edge in local areas. In addition, we investigate the energy efficiency (ON/OFF) problem with the tasks migration in the MEC environment, according to the status of edge servers (idle or low-load). We exploit the trade-off among the switching ON/OFF the idle or low-load servers to save energy with their tasks migrated to the high-load server by considering their status of workload.

# Chapter 3

## Computation Offloading for Fast CNN Inference in Edge Computing

### 3.1 Introduction

Mobile artificial intelligence applications (e.g., Apple Siri) recently become prevalent thanks to a fundamental technology called Convolutional Neural Networks (CNNs) [78], which enables high accuracy in recognizing objects in images or understanding contents from voice records with local connectivity and weights sharing [79]. A typical working process of CNN-based mobile applications usually contains two stages. The first stage is training, i.e., software developers train the CNN models using a large amount of training data. The well-trained models are then released to users who can download and install these models on their mobile devices. In the second stage, users capture data, e.g., an image or a voice, using their mobile devices. These data are processed by CPU or GPU according to the CNN models for object recognition or content understanding. This stage is also called inference. It has been shown that CNN inference contains intensive computation, which incurs a heavy computational burden for mobile devices with

limited power and hardware resources [1].

To improve the efficiency of CNN inference, a widely adopted approach is to offload its computation to the cloud [80–83]. However, the cloud usually locates far from mobile users and the data transmission may incur a high latency. Edge computing has been proposed to offer low-latency computation services for mobile users by deploying a number of modest-size computing sites at the network edge [84]. Compared to the cloud, edge computing sites are closer to mobile users, leading to low network latency, but they have limited computational resources, which should be efficiently scheduled.

In this chapter, we study to offload CNN inference tasks generated by mobile devices to the edge computing site. Instead of simply offloading the whole task to the edge, we focus on partial offloading, i.e., the mobile devices conduct inference computation over a few layers of CNN models and then send the intermediate results to the edge computing site, which completes the computation of the rest layers. By considering the characteristics of each layer in the CNN model, it provides insights to identify a better computation splitting between the mobile device and the cloud at the layer level. Such a kind of collaborative inference approach has been shown very effective in further reducing inference time and improving the energy efficiency of mobile devices. Their findings show that data size is generally decreasing at the front-end, while the mobile latency of per-layer is generally higher at the back-end, which means there is a unique opportunity for computation splitting in the middle of the CNN between the mobile device and the cloud [8]. However, existing work on collaborative inference lets each mobile device independently makes offloading decisions according to the network connection, ignoring the acceleration chances by aggregating and scheduling workloads at the edge site. We have an important observation that the average inference time can be reduced if we process tasks on GPU by batches. We conduct experiments on Nvidia GTX1080 GPU by using three typical CNNs, i.e., AlexNet, GoogLeNet, and LeNet. The average inference time under different batch sizes is shown in

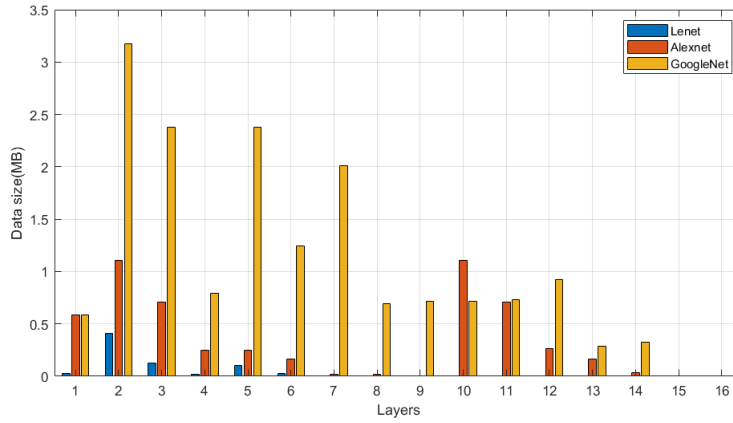


Figure 3.1: The intermediate output of CNN (Lenet,Alexnet and GoogleNet) layers by caffe2 framework

Figure 3.2, where we can observe that the average inference time shows as a convex function of the batch size. This phenomenon motivates us to batch tasks of the same CNN layers at the edge site.

To fully exploit the benefit of workload batching at GPU, we propose to jointly consider the CNN offloading and workload batching, with the objective of minimizing the average inference time of a given number of tasks. Specifically, we consider a set of mobile devices within the service range of an edge computing site, as shown in Figure 3.3. Each mobile device generates some CNN inference tasks and offloads the computation of some CNN layers to the edge site with a stable communication network. In order to illustrate the discrepancy memory size of each layer in the CNN model, we obtain the intermediate data of layers for three CNN models (Lenet, ALEXnet, GoogLeNet) by using the caffe2 framework, as shown in Figure 3.1. The size of the output of each layer sharply increases then decreases while computation generally increases through the execution of the CNN network.

At the edge site, there are several servers equipped with GPU, and each server batches the offloaded tasks containing the same CNN layers. Given a set of CNN inference tasks, we design an optimal algorithm that decides which layers should be offloaded to the edge for each task, by jointly considering the computational capability of mobile devices, network bandwidth, and task batching at GPU. With



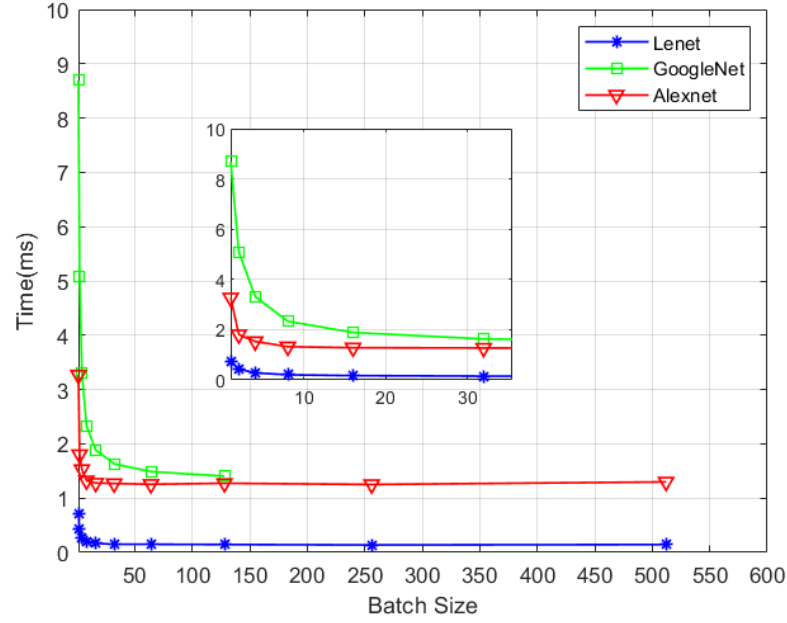


Figure 3.2: Inference time *vs.* Batch size.

model splitting between mobile device and edge, the hybrid computation resources can contribute comprehensive utilization towards low-latency DNN inference. Furthermore, we consider a more practical scenario that mobile users request inference services in an online manner, i.e., users submit their requests at different times. An online algorithm with low complexity has been proposed. The main contributions of this chapter are summarized as follows:

- We propose to jointly consider CNN offloading and workload batching to minimize the average inference time.
- We design a near-optimal algorithm to solve the problem of minimizing average inference time, given the number of mobile users.
- An online algorithm has been proposed to handle the scenario that users submit their requests at different times.
- Extensive trace-driven simulations have been conducted and the results show that our proposal significantly outperforms existing approaches.

The rest of this chapter is organized as follows. In Section 3.3, we introduce the system model and the background is introduced in Section 3.2. An optimal

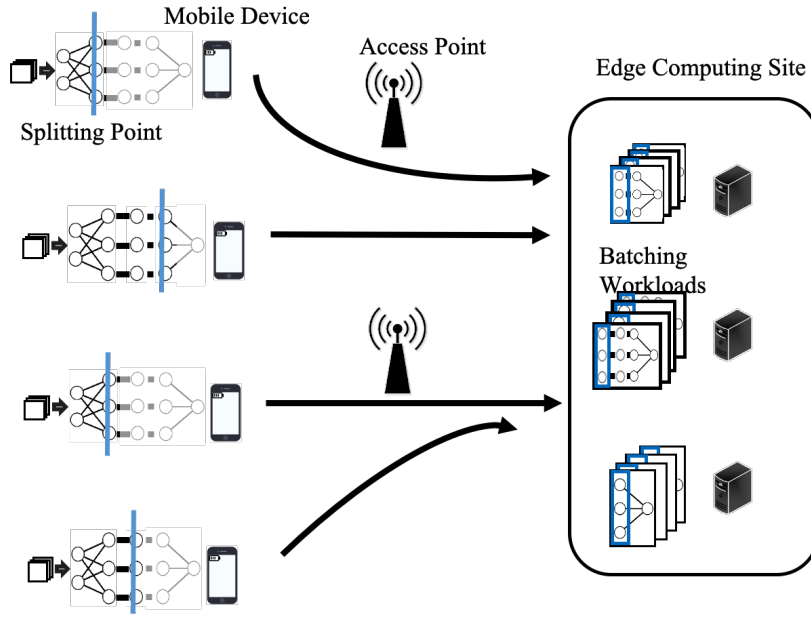


Figure 3.3: System overview of cooperation of mobile devices for fast CNN inference

algorithm given all CNN inference tasks is presented in Section 3.4. The online algorithm design is in Section 3.5. Section 3.6 shows the experimental results. Finally, Section 3.7 concludes the chapter.

## 3.2 Background

Previous research work focuses on offloading computation and resource allocation from the mobile to cloud, applications can benefit from the resource-rich infrastructure by delegating computation task execution to remote servers or edge servers.

Edge computing provides computation and storage resources at the edge of the mobile network. A critical research challenge is to decide how to offload computation tasks to the edge computing sites to save energy or accelerate task processing. Zhang *et al.* [85] investigate the joint offloading decision, wireless and computational resources allocation problem with the aiming to energy harvesting. From the mobile operator's perspective, the objective is to gain much revenue by maximizing the amount of offloaded tasks with lower energy expenditure and low

delays. In [86], Chen *et al.* investigate the computation offloading policies for a representative mobile user in an ultra-dense sliced RAN, where an offloading task can be offloaded to multiple base stations. Guo *et al.* [87] propose an energy-efficient joint offloading and resource scheduling (EDors) policy to save energy consumption and reduce application completion time.

In [88], Thanh Quang Dinh *et al.* investigate a multi-user multi-edge-nodes computation offloading problem as a non-cooperative exact potential game (EPG) due to the limitation of communication and computation resource when many mobile users (MUs) offload to the same edge node simultaneously, in a time slot. Each MU selfishly maximizes its computation resource (e.g., process central processor unit (CPU) cycles) and reduce its energy consumption. For a practical scenario, they propose a model-free reinforcement learning offloading mechanism to help MUs maximize their long-term utilities by learning the long-term offloading strategies. While in [89], Jošilo *et al.* regard mobile devices as selfish in a dense wireless network where the independent mobile device can choose to offload their computations via multiple access points or the base station with the aims to minimize the computation costs. To solve the problems, they prove the existence of pure strategy Nash equilibria and offer an efficient decentralized algorithm to find an equilibrium.

Much CNN architecture optimization work has been done to execute CNN-based applications on mobile devices efficiently and energy-efficient. Forrest *et al.* [90] show a smaller CNN architecture that achieves AlexNet-level accuracy with 50x fewer parameters. Smaller CNNs require less computation and become more feasible to deploy on a mobile device with limited resources. In order to employ CNN to mobile devices and compact networks, the authors propose an approach for transforming convolutional layers of a CNN which includes  $m$  convolutional blocks to quantify the CNN parameter via hardware optimization [91]. In [92], Leyuan Wang *et al.* provides an end-to-end solution that is employed to execute the CNN model inference on the integrated GPUs at the edge site by

optimizing vision-specific operators on integrated GPUs from multiple vendors. Additionally, machine learning-based scheduling search schemes are used to optimize computation-intensive operators. In [93], Cuervo *et al.* propose an offloading scheme to enable energy saving with minimal burden on the program. Motivated by the computation throughput might not well match the memory bandwidth provided by a Field-programmable Gate Array (FPGA) platform, Zhang *et al.* [94] introduce an analytical design policy for any CNN by using the roof-line model. Kang *et al.* [8] propose a fine-grained layer-level computation partitioning strategy that finds that the data and computation variation of a DNN has significant latency and energy advantage. However, they focus on how to partition the DNN model without considering the resource utilization of the cloud. Odessa [95] has been proposed to enable offloading by considering the processing latency and data requirements of part of the function, without global consideration of the application. In [96], Fowers *et al.* present NPUs that satisfy the requirement for the execution of the DNNs-based model with low latency, high throughput, and high efficiency by batching. Xu *et al.* [97] bridge the gap that exists between Moore’s-law-based CMOS scaling and the scaling of CNNs for the inference at edge site by considering the architecture and algorithms innovations. Compared with the existing work, we jointly consider the CNN splitting between the mobile device and edge server, and workload batching at the edge server, with the objective of minimizing the average inference time of a given number of tasks. It shows that the average inference time can be reduced if we process tasks on GPU by batches, according to our experimental study. We believe that this kind of collaborative way between mobile devices and edge can bring low-latency DNN inference.

### 3.3 System Model

We consider a number of  $N$  mobile devices running a CNN-based mobile application. Each mobile device has some tasks of inferring input data (e.g., an image,

a video, or a voice record) using a well trained CNN model. The computation capability of the  $i$ -th mobile device is denoted by  $c_i$ , which can be the CPU cycles per second. All mobile devices are within the service range of an edge site, which maintains  $K$  servers dedicated to CNN inference computation. We suppose that the CNN model is stored by mobile devices and the edge site. The data transmission speed between the  $i$ -th mobile device and the edge site is denoted by  $b_i$ .

Following the model in [8], each CNN inference computation can be divided into two parts, which can be assigned to the mobile device and the edge site, respectively. As shown in Figure 3.3, the mobile device runs the inference computation over the first a few layers of the CNN and then transmits the intermediate results to the edge site, which finally conducts the computation over the rest layers. Although such a kind of CNN splitting approach is promising, it is challenging to decide which layers should be offloaded to the edge site. We suppose that there are  $K$  different CNN splitting schemes, each of which is handled by a dedicated server at the edge site. In the  $k$ -th splitting scheme, the computation loads (i.e., CPU cycles) on the  $i$ -th mobile device is denoted by  $s_i^k$ , and the amount of corresponding intermediate data is  $d_i^k$ .

Each edge server can batch the workloads and process them using GPU. The relationship between average processing time and batching workloads can be expressed as a function  $g_k(\cdot)$ . Thanks to the batching features of GPUs, the function  $g_k(\cdot)$  is convex and non-increasing.

The task completion time (TCT) of each mobile device contains three parts, i.e., the local processing time, data transmission time, and edge processing time. Our objective is to choose a splitting scheme for each mobile device so that the total task completion time is minimized.

### 3.4 Near-Optimal algorithm design

In this section, we study to formulate the problem and to solve it using linearization techniques. We first define a binary variable  $x_i^k$  for the selection of DNN splitting schemes as follows.

$$x_i^k = \begin{cases} 1, & \text{if device } i \in N \text{ chooses the } k\text{-th splitting scheme;} \\ 0, & \text{otherwise.} \end{cases}$$

Since each mobile device can choose only one splitting scheme, we have the following constraint for variable  $x_i^k$ :

$$\sum_{1 \leq k \leq K} x_i^k = 1, \forall 1 \leq i \leq N. \quad (3.1)$$

Therefore, the local processing time  $T_i^{local}$  of the  $i$ -th mobile device can be calculated by:

$$T_i^{local} = \sum_{1 \leq k \leq K} x_i^k \frac{s_i^k}{c_i}, \forall 1 \leq i \leq N. \quad (3.2)$$

After the local processing, the intermediate data are transmitted to the edge site. Given the transmission speed of  $b_i$ , the network transmission time can be calculated by:

$$T_i^{net} = \sum_{1 \leq k \leq K} x_i^k \frac{d_i^k}{b_i}, \forall 1 \leq i \leq N. \quad (3.3)$$

The edge site assigns the intermediate data from mobile devices choosing the  $k$ -th scheme to the dedicated server, which is referred to as the  $k$ -type server in our model. The total workloads received by the  $k$ -type server are:

$$D_k = \sum_{1 \leq i \leq N} x_i^k d_i^k, \forall 1 \leq k \leq K. \quad (3.4)$$

The  $k$ -type server batches the received workloads and processes them as a whole using GPU. The processing time of mobile device  $i \in N$  can be expressed

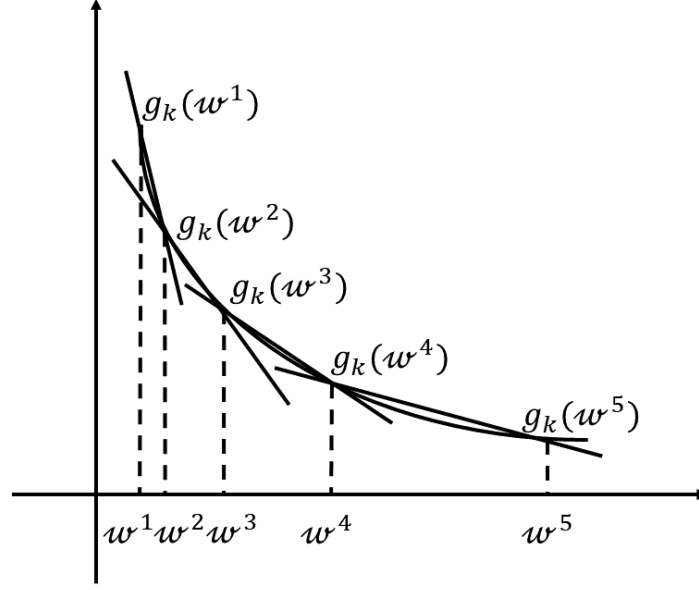


Figure 3.4: Approximation of nonlinear constraints.

by:

$$T_i^{server} = \sum_{1 \leq k \leq K} x_i^k g_k(D_k), \forall 1 \leq i \leq N. \quad (3.5)$$

Since the final results of CNN reference are usually with small sizes, we ignore their transmission time back to mobile devices. The task completion time (TCT) of mobile device  $i \in$  can be calculated by:

$$T_i = T_i^{local} + T_i^{net} + T_i^{server}, \forall 1 \leq i \leq N. \quad (3.6)$$

Therefore, the problem of minimizing the average TCT of  $N$  mobile users can be formulated as:

$$\begin{aligned} & \min \sum_{1 \leq i \leq N} T_i / N \\ & \text{subject to: (3.1) -- (3.6).} \end{aligned}$$

By carefully examining the above formulation, we find that it is a mixed-integer nonlinear programming problem, which is NP-hard in general. The main difficulty is the processing time  $T_i^{server}$  that is nonlinear due to function  $g_k(\cdot)$ . Our

idea is to use several linear constraints to replace the nonlinear function  $g_k(\cdot)$  [98]. Specifically, we find a set of values  $\{w^1, w^2, \dots, w^L\}$  in the domain of  $g_k(\cdot)$  and define following lines:

$$f_k^l(w) = \alpha_k^l(w - w^l) + g_k(w^l), \forall 1 \leq k \leq K, 1 \leq l \leq L, \quad (3.7)$$

where  $w$  is the variable and  $\alpha_k^l$  is the slope of  $l$ -th line that can be expressed as:

$$\alpha_k^l = \frac{g_k(w^l) - g_k(w^{l-1})}{w^l - w^{l-1}}, \forall 1 \leq k \leq K, 1 \leq l \leq L. \quad (3.8)$$

As shown in Figure 3.4, the constraint (3.5) can be replaced by:

$$T_i^{server} = \sum_{1 \leq k \leq K} x_i^k \hat{g}_k(D_k), \forall 1 \leq i \leq N; \quad (3.9)$$

$$\hat{g}_k(D_k) \geq f_k^l(D_k), \forall 1 \leq k \leq K, 1 \leq l \leq L, \quad (3.10)$$

where  $\hat{g}_k(D_k)$  is an auxiliary variable. Although we successfully replace the nonlinear function  $g_k(\cdot)$  with a number of linear constraints, it is still difficult to handle the constraint (3.9) because of the multiplication of two variables. Next, we define a new variable  $y_i^k$  as follows:

$$y_i^k = x_i^k \hat{g}_k(D_k), \forall 1 \leq i \leq N, 1 \leq k \leq K, \quad (3.11)$$

so that the constraint (3.9) becomes

$$T_i^{server} = \sum_{1 \leq k \leq K} y_i^k, \forall 1 \leq i \leq N, 1 \leq k \leq K. \quad (3.12)$$



The constraint (3.11) can be equivalently replaced by:

$$0 \leq y_i^k \leq \hat{g}_k(D_k), \forall 1 \leq i \leq N, 1 \leq k \leq K; \quad (3.13)$$

$$\begin{aligned} \hat{g}_k(D_k) - (1 - x_i^k) \hat{g}_k^{max}(D_k) &\leq y_i^k \leq x_i^k \hat{g}_k^{max}(D_k), \\ \forall 1 \leq i \leq N, 1 \leq k \leq K. \end{aligned} \quad (3.14)$$

Finally, the target problem can be approximated by the following one:

$$\begin{aligned} &\min \sum_{i \in N} T_i / N \\ &\text{subject to: (3.1) -- (3.4), (3.6), (3.7), (3.8), (3.10), (3.12) -- (3.14).} \end{aligned}$$

All constraints of the above formulation are linear. Although it still uses integer variables, this problem can be quickly solved by software (e.g., CPLEX).

## 3.5 Online algorithm design

In practice, mobile users generate CNN inference tasks at different times. When the edge site receives an inference task, it needs to decide the CNN splitting scheme according to the computational capability of the mobile device, quality of network connection, and current workloads at the edge servers. An intuitive approach is to calculate the expected task completion time under different splitting schemes, assuming this request is immediately processed by the corresponding GPU. After that, the splitting scheme with the minimum expected task completion time is selected. Unfortunately, this simple approach misses the chances of batching workloads.

Therefore, we propose an online algorithm by considering the workload batching. Specifically, when the edge site receives a request, it still calculates the expected task completion time, but under an assumption of future arriving rate, which can be estimated according to historical records. Specifically, we let  $r_k$  denote the estimated task arriving rate at the  $k$ -th server responsible for the

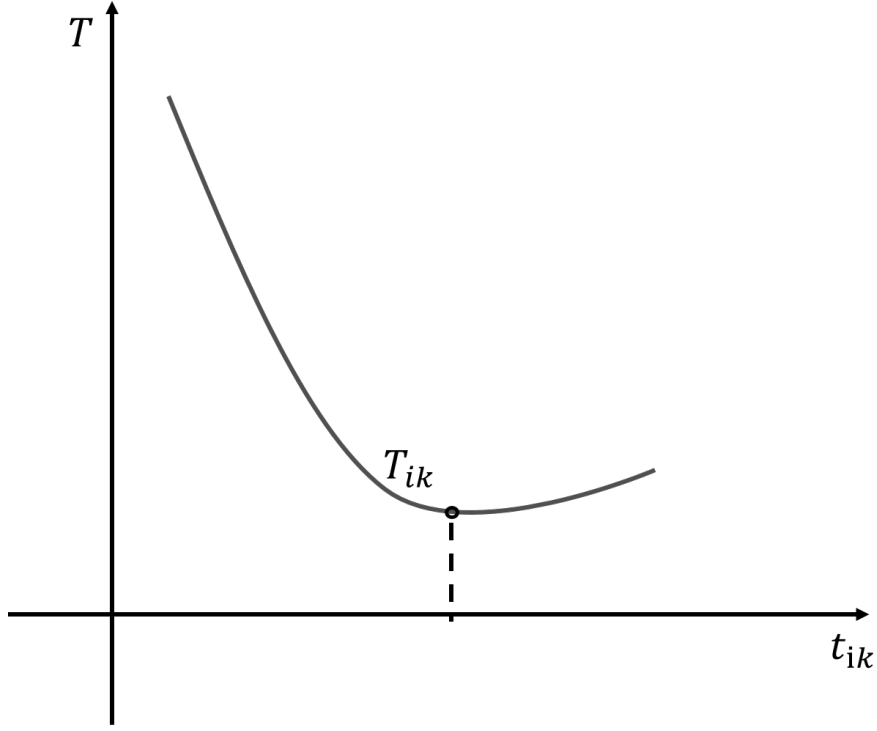


Figure 3.5: The expected task processing time under different  $t_{ik}$ .

workloads of the  $k$ -th CNN splitting scheme. Since the arrived task  $i$  will not be immediately processed, we let  $t_{ik}$  denote its waiting time. The expected task processing time under the  $k$ -th CNN splitting scheme can be calculated by:

$$T_{ik} = \frac{s_i^k}{c_i} + \frac{d_i^k}{b_i} + g_k(D_k + r_k \cdot t_{ik}) + t_{ik}. \quad (3.15)$$

We can observe that (3.15) is a convex function of  $t_{ik}$ , as shown in Figure 3.5. The optimal waiting time  $t_{ik}^* = \arg \min T_{ik}$  can be calculated by solving the following equation:

$$\frac{\partial T_{ik}}{\partial t_{ik}} = 0. \quad (3.16)$$

The pseudo-codes of the proposed online algorithm is shown in Algorithm 1. For each arrived CNN inference task  $i$ , we first randomly select a subset of servers and maintain them in set  $K' \subset K$ . For each server  $k \in K'$ , we calculate the optimal waiting time  $t_{ik}^*$  and associated task processing time  $T_{ik}^*$ , according to (3.16). After that, we choose the server  $k^*$  leading to the minimum  $T_{ik}^*$  and send the task  $i$  to the waiting queue of the server. If  $t_{ik}^* = 0$ , the server  $k^*$  batches

---

**Algorithm 1** Online Algorithm

---

```
1: for each task  $i$  do
2:   Randomly select a subset  $K' \subset K$  of servers;
3:   for each server  $k \in K'$  do
4:     Calculate the optimal waiting time  $t_{ik}^*$  according to (3.16);
5:     Calculate the corresponding task processing time  $T_{ik}^*$ ;
6:   end for
7:   Select the server  $k^*$  with the minimum  $T_{ik}^*$ ;
8:   Send the task  $i$  to the server  $k^*$ ;
9:   if  $t_{ik}^* = 0$  then
10:    Process waiting tasks at server  $k^*$  immediately;
11:   end if
12: end for
```

---

its waiting tasks and processes them immediately. Intuitively, we should evaluate the expected task processing time on all servers. However, it may lead to high overhead due to the operations searching for optimal  $t_{ik}^*$ . To reduce computational complexity, we randomly select a subset of servers in our online algorithm design.

## 3.6 Performance Evaluation

In this section, extensive simulations are conducted to evaluate the performance of the proposed algorithms.

### 3.6.1 Experimental Setup

The simulation algorithm programmed by Python is executed at a desktop PC which is equipped with a quad-core 2.8 GHz Intel processor and 16 GB RAM. Firstly, we assume that there are 12 servers equipped with GPUs at the edge computing site. We consider a number of mobile users whose computational capability is randomly distributed according to a Gaussian distribution. For the sake of simplifying, we use the standard GoogLeNet as an example. We compare the proposed algorithms, which is referred to as COB (CNN Offloading with Batch-ing), with the other three schemes. *Local-only*: All CNN inference tasks generated by mobile devices are executed by themselves. *Remote-only*: Mobile devices send

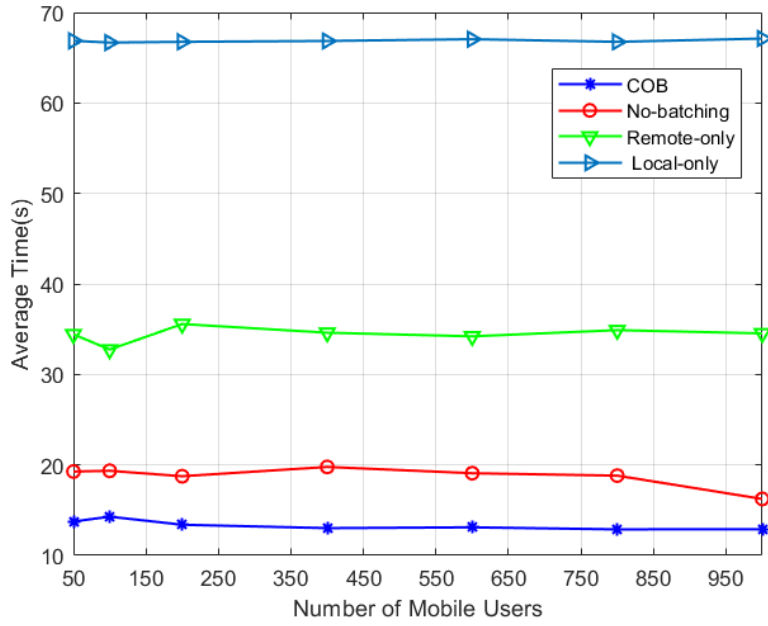


Figure 3.6: The average task completion time under different number of mobile users.

all CNN inference tasks to the edge site for processing. *No-batching*: Each node independently decides how to split the CNN model and the edge servers process inference tasks one by one, without batching.

### 3.6.2 Experiments

The average task completion time under the different number of users is shown in Figure 3.6. In this experiment, we explore the impact of the number of mobile users by setting the number of users from 50 to 1000 with a fixing bandwidth of 10 Mbps. We can observe that our proposed algorithm always outperforms others with the growth of mobile users. We also show the splitting schemes adopted by the optimal solution under the different numbers of mobile users. Furthermore, we count up the percentage of mobile users who chose the different splitting layers with different computational capabilities. As shown in Table. 3.1, most of the users choose to split the CNN model at the first a few layers. As the number of mobile users grows, more users choose the third and fourth layers.

We show the percentage of users who offload their tasks to the edge site under

Table 3.1: The percentage of mobile users choosing different splitting layers.

Users Layers	20	40	60	80	100	120
1	0.00	0.00	0.00	0.00	0.00	0.00
2	20%	20%	13.3%	15%	36%	18.3%
3	20%	60%	40%	22.5%	34%	44.16%
4	25%	17.5%	11.67%	28.75%	7%	20.83%
5	15%	30%	20%	11.25%	13%	12.5%
6	20%	10%	6.7%	15%	6%	0.00%
7	0.00%	12.5%	8.3%	7.5%	4%	4.17%

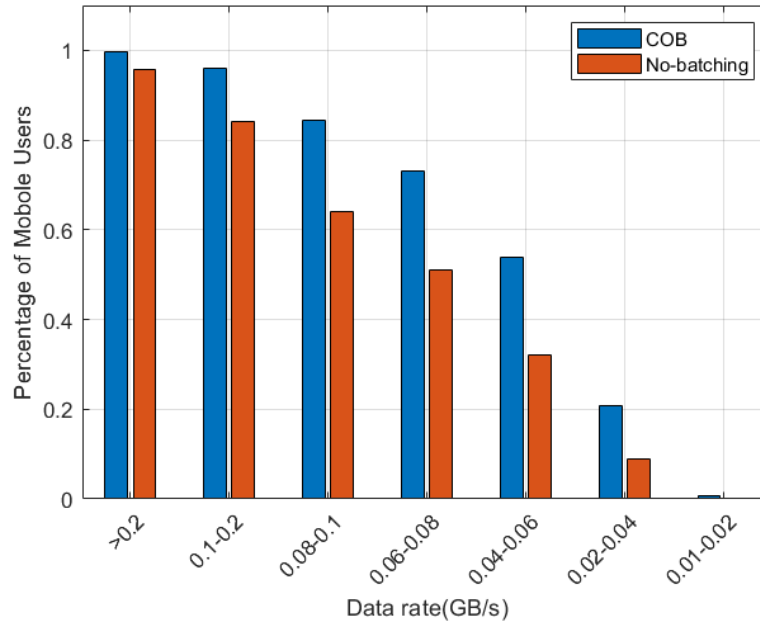


Figure 3.7: The percentage of mobile users choosing splitting under different network bandwidth.

different network quality. In the experiment, we assume there are 120 mobile users and each mobile user has some computation tasks. We set the data rate from 10 Mbps to 200Mbps for each mobile user. As shown in Figure 3.7, more users choose to offload their tasks when the network transmission rate is higher. That is because edge servers have strong computational capability and they can accelerate inference tasks when the network connection is good. Compared with No-batching, Our proposal is better to motivate users to offload their tasks.

Then, we explore the impact of computation resources of mobile devices on average TCT. We assume the mobile user will allocate some dedicated computa-

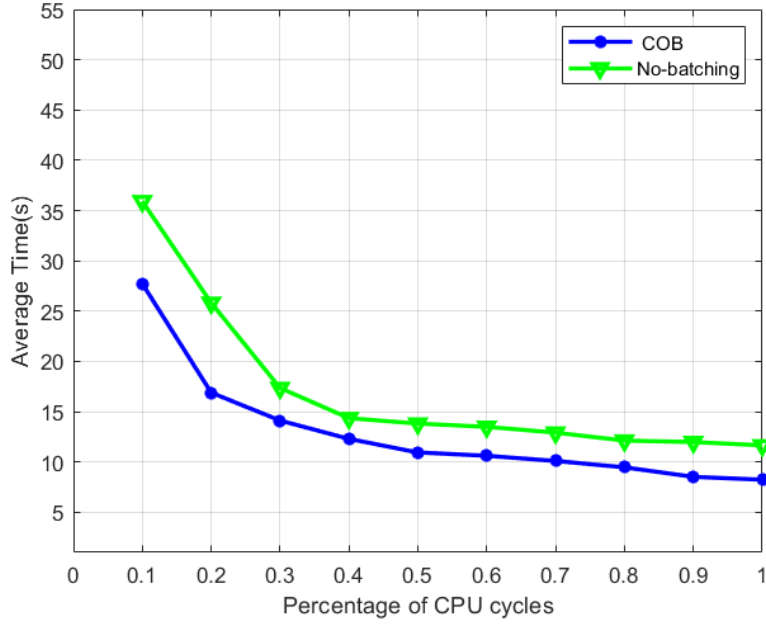


Figure 3.8: The average tasks completion time under different mobile computing capability.

tion resource for this kind of task with denoted by the percentage of CPU cycles from 10 % to 100 %. And mobile devices have the CPU capability from 1.2 GHz to 2.2 GHz with the Gaussian distribution. As shown in Figure 3.8, the average TCT decreases as mobile devices have the stronger computational capability. Our proposed algorithm with batching always outperforms the scheme without batching.

Next, we explore the impact of different network quality by setting the bandwidth from 10 Mbps to 1 Gb. In this context, there are 120 mobile users with their CNN-based tasks that might offload to the edge server through the communication network. In Figure 3.9 and Figure 3.10 the both two illustrate that the worse network makes TCT increase exponentially. That is because a worse network means high latency which impedes offloading task, the total TCT for user increases indirectly. Finally, we evaluate the performance of our proposed online algorithm under the different numbers of users. In Figure 3.11, as the increase in the number of users from 50 to 1000, we observe that our online algorithm always outperforms others.

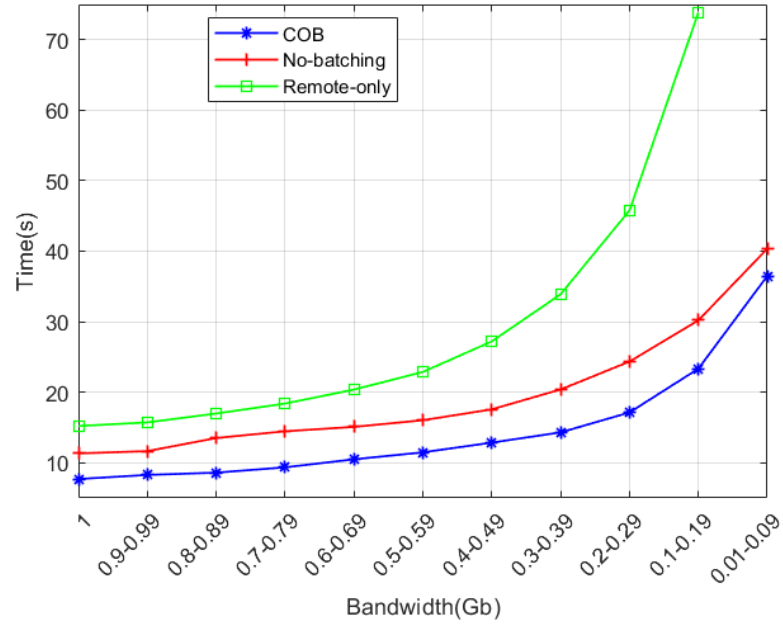


Figure 3.9: The average tasks completion time under different network bandwidth.

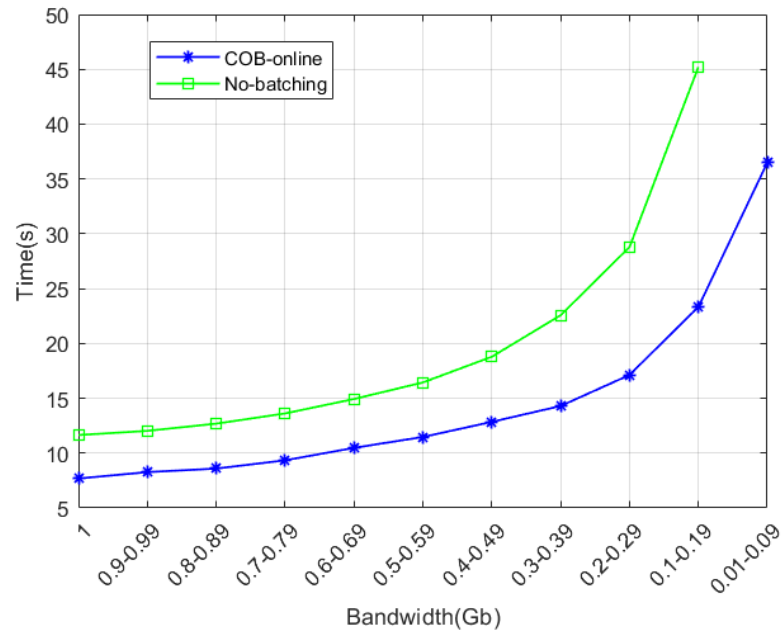


Figure 3.10: The average tasks completion time under different network bandwidth based for online scenario.

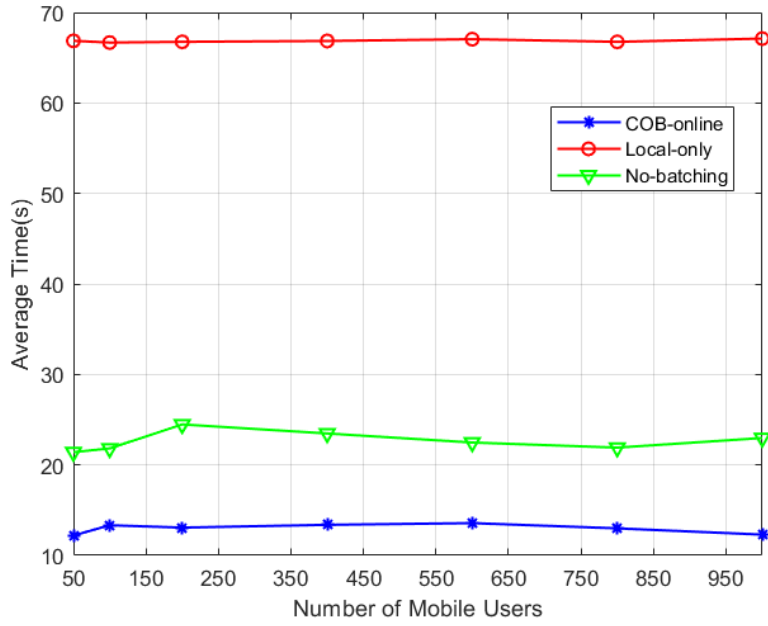


Figure 3.11: The average task completion time of online settings under different number of users.

### 3.7 Summary

In this chapter, based on our observation that batching multiple users' data will be more efficient than handling user's data individually, we propose an edge computing-based co-operational computation offloading approach for fast CNN inference by jointly considering mobile device capability, network bandwidth, and batch workloads. Different from existing works, we exploit the benefit of batching inference workloads at GPUs and design an optimal algorithm to decide the offloading choices for a number of mobile users. Moreover, we design an online algorithm to handle the scenario that mobile users submit their inference tasks at different times. The simulation results show that our proposed algorithms significantly outperform existing works.



# Chapter 4

## Cooperation of Mobile Devices for Fast Inference of Deep Learning Application

### 4.1 Introduction

The algorithmic breakthrough of deep learning (i.e., Deep Neural Network) in the past decades has attracted wide interests in developing artificial intelligence (AI) mobile applications to conduct language translation, object recognition, health monitoring, and malware detection [99]. The deep learning-based intelligent services provided by these mobile applications enable people to enjoy a more convenient as well as smarter mobile life. Although today's mobile devices become much more powerful than ever with greater computing capability and longer battery life, it should be noticed that not everyone is able to be equipped with the newest powerful mobile devices. Therefore, significant heterogeneity (of available storage, CPUs, and batteries) exists between people's mobile devices.

A natural way to tackle this challenge is to employ cloud computing by offloading the computation tasks to remote servers (aka the cloud). For example, when the local mobile device needs to recognize the objects in a picture, it needs

to upload this picture to the cloud and waits for a remote response of the final recognition results [100]. However, there are two major concerns about this kind of cloud computing-based method. First, data transmission would consume a great amount of network bandwidth. The traffic loads will get heavier as the users accumulate, and eventually degrade the cloud's Quality-of-Service (QoS). Second, the transmission latency between the local mobile devices and the remote cloud would be long [101]. In some emergent scenarios, users might expect near real-time response, while the transmission latency will be a big problem. Local offloading is promising to address these challenges. Many mobile devices now are able to collaborate on computing tasks by sharing their resources (e.g., CPU, GPU, and memory), which motivates us to aggregate workloads of mobile devices to effectively accelerate the deep learning inference process. That is because GPUs prefer batched workloads, which can reduce the overhead of GPU memory accesses [102]. We conduct experiments by running two typical CNN models, MNIST and CIFAR, on Nvidia TX2 GPU. The inference time per image under different batch sizes is shown in Figure 4.1. We observe that inference time decreases with the growth of batch size. Motivated by this fact, we claim that batching multiple users' data will be more efficient than dealing with user data individually.

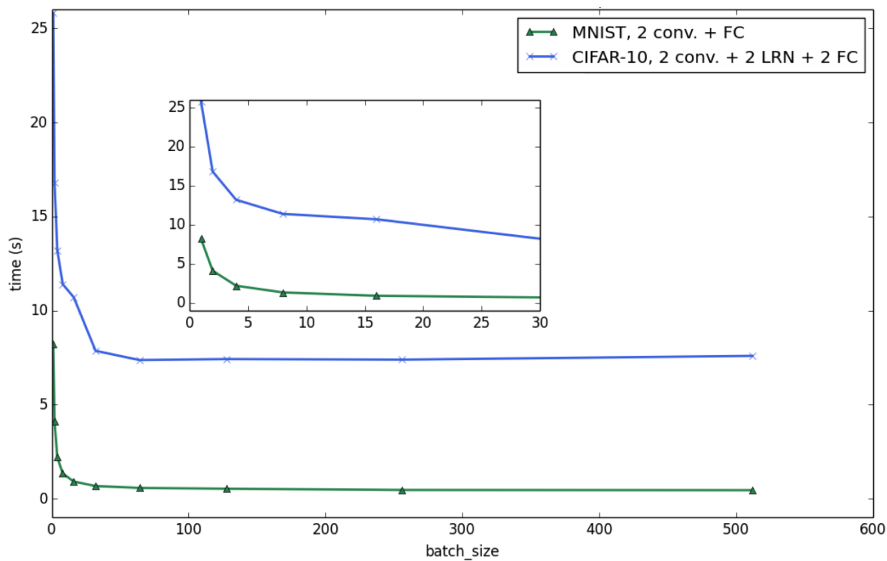


Figure 4.1: Inference time *vs.* Batch size.

Motivated by the above facts, in this chapter, we propose to employ local of-

floading to enable collaborative inference among local mobile devices. We first model the connections among mobile devices regarding their mobility. In this scenario, some virtual central controllers managed by network operators are entrusted by mobile devices which can be used to collect the mobile device task features, and assist in D2D-enabled MEC by pooling and sharing the resources among each other [103]. In other words, the mobile device can obtain the global information (network status, connection requirement, etc.) through the virtual central controller. Hence, a central controlling entity in the network (the base station) is used to provide information and response to requests for such information for D2D communication underlying cellular communication [104]. Mobile devices are responsible for publishing this information (because the central server does not save files), let the central controller know what tasks they want to process.

After the local link connections between mobile devices are established, the transmission latency on each link is assumed to remain constant yet various from each other. In what we show later the practical inference procedure is near real-time, mobile devices, therefore, are reasonably regarded as staying static until they receive the computation results. As shown in Figure 4.2, non-aggregation nodes transfer their data to aggregation node. After inference computing, the aggregation nodes send inference results to non-aggregation nodes.

Our main concern naturally focuses on minimizing the total cost of machine learning inference among all devices by considering the tasks processed on GPU by batches, which contains communication cost and computation cost. This problem can be formulated as a non-linear programming problem that is NP-hard in general. To solve the problem, we propose a heuristic algorithm based on partial swarm optimization (PSO) which is a versatile population-based stochastic optimization technique. This algorithm can quickly converge to a good solution and can adapt to the changing environments by adjusting collaboration decisions. We also design a distributed algorithm to address the challenge of collecting global network information and running the centralized algorithm. The contributions of

our work are summarized as follows:

- We propose a novel collaborative inference design to accelerate mobile deep learning applications and formulate the problem of minimizing the total processing cost of a number of mobile devices.
- We design a heuristics algorithm based on PSO to efficiently minimize the total time costs for collaborative inference, with a dynamic procedure of selecting the optimal computing nodes from local mobile devices.
- We propose a distributed algorithm that each mobile device uses its one-hop information to make offloading decisions.
- We conduct extensive simulations to evaluate the performances of our proposed algorithm and demonstrate its comprehensive advantages to existing solutions.

The rest of this chapter is organized as follows. In Section 2, we introduce the system model and formulate the problem. The algorithm design is presented in Section 3. Section 4 gives the distributed algorithm design. Section 5 shows the simulation results, followed by related work in Section 6. Finally, Section 7 summarizes this chapter.

## 4.2 Background

Much research work about efficiency in machine learning and cooperation in the mobile cloud has been done. In [105], Sharan Chetlur *et al.* improve the performance by 36% for convolution neural networks on the Caffe framework and reduce memory consumption. In [106], Alfredo *et al.* do any analysis on the accuracy, power consumption, inference time, memory footprint by experiments on a framework named Caffe. In [107], Han *et al.* benchmark the layer-wise speed up on CPU, GPU, and mobile GPU by deep compression for networks. In [108],

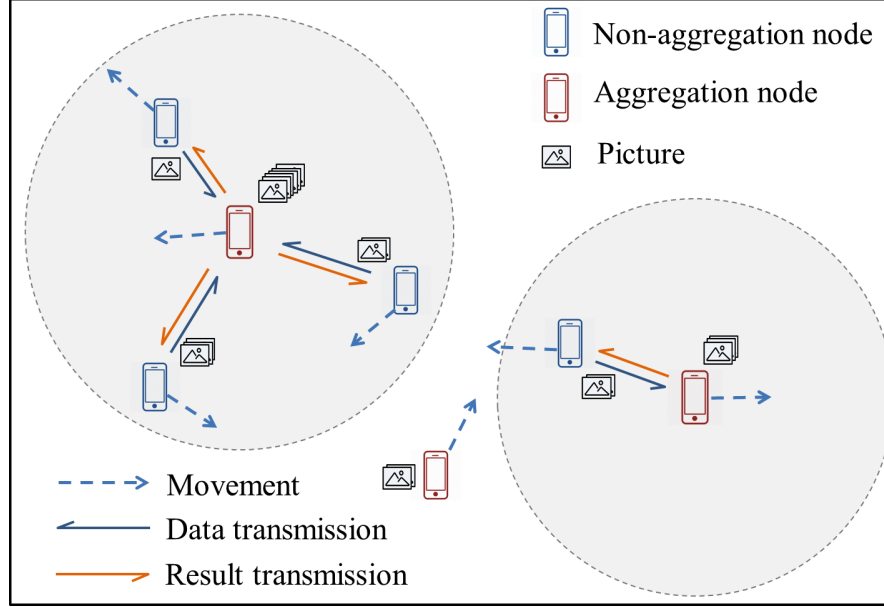


Figure 4.2: System overview of D2D-based computation offloading for CNN inference

Tang *et al.* propose a client-architecture where the training process is implemented in a server, and then the mobile device downloads the trained predictor from the server to make transmission decisions. The aforementioned works focus on improving the inference performance by optimizing the architecture of the DNN model and utilizing the potential of hardware (i.e., CPU, GPU, etc). In [109], the lab established a SmartLab with 40 Android devices which cloud provides an open testbed to facilitate research and smartphone applications can be deployed massively. In [110], Heyi *et al.* propose a back-end general architecture that is able to require crowd-sourcing for mobile applications. And they adopt the Microsoft Azure cloud computing platform to deploy their back-end. In [111], Yao *et al.* introduce a mobile cloud service framework based on crowdsourcing that meets mobile users' requirements by sensing their context information and provide corresponding services to each of the users. In [112], Considering a dynamic network in which mobile devices may join and leave the network at any time, Ke *et al.* merge crowdsourcing into an existing mobile cloud framework where data acquisition and processing can be conducted. In [113], Fan *et al* propose a novel privacy-aware and trustworthy data aggregation protocol based on malicious behavior like submitting

data to damage the fog system for mobile sensing. The above works mainly focus on the optimization of offloading between mobile devices and servers. Different from the existing works, we exploit to employ local offloading to enable collaborative inference among local mobile devices. In addition, we found that batching can reduce the average DNN inference time. Hence, in our D2D work, we jointly considering the cooperation among the devices and the batching mechanism when the device work as a server to improve the inference performance.

### 4.3 System Model

We consider a set  $N$  of mobile devices running an application powered by deep neural networks (DNNs). The DNN model has been well trained and installed on mobile devices. Each mobile device  $i \in N$  has an amount of  $w_i$  data to process using equipped mobile GPU. These mobile devices can connect with each other using direct links, e.g., Bluetooth and WiFi-Direct. The neighbors of device  $i$  are included in set  $N_i$ . The communication delay between two nodes  $i$  and  $j$  is denoted by  $d_{ij}$ . As we have shown that batching ML workloads on GPU can effectively reduce the average processing time, multiple devices can aggregate their workloads on a single one. We define a binary variable  $x_i$  to indicate whether mobile device  $i \in N$  is an aggregator.

$$x_i = \begin{cases} 1, & \text{if node } i \text{ is an aggregator} \\ 0, & \text{otherwise.} \end{cases}$$

Note that some devices process only their own data, without receiving workloads from other nodes. We also treat them as aggregators at  $x_i = 1$ . Each non-aggregator may connect to multiple aggregators. We define a variable  $y_{ij}$  to indicate the portion of workloads offloaded from device  $i$  to  $j$ . Since aggregators do not offload their workloads to others, we have  $\sum_{j \in N_i} y_{ij} = 0$ . For each non-aggregation

node, we have  $\sum_{j \in N_i} y_{ij} = 1$ . In summary,

$$y_{ji} \leq x_i, \forall i \in N, j \in N; \quad (4.1)$$

$$x_i + \sum_{j \in N} y_{ij} = 1, \forall i \in N. \quad (4.2)$$

We define an auxiliary binary variable  $z_{ij}$  to indicate whether device  $i$  offloads data to device  $j$ , which is constrained by:

$$y_{ij} \leq z_{ij}, \forall i \in N, j \in N; \quad (4.3)$$

$$z_{ij} \leq x_i, \forall i \in N, j \in N. \quad (4.4)$$

We define a total cost  $T_i$  of device  $i$  as the sum of its computation and communication cost, i.e.,

$$T_i = 2 \sum_{j \in N_i} d_{ij} z_{ij} + f\left(w_i + \sum_{j \in N_i} y_{ji} w_j\right), \quad (4.5)$$

where  $f(\cdot)$  is a non-decreasing function that describes the relationship between GPU processing time and the number of workloads. The neighbors of device  $i$  are maintained in set  $N_i$ .

The first term in the above equation indicates the round-trip delay of sending data and receiving inference results. The second item denotes the inference time with the workloads which include two parts: original data and received data from neighbors. With an objective of minimizing the total cost among all devices, our studied problem can be formulated as:

$$\begin{aligned} & \min \sum_{i \in N} T_i \\ & \text{subject to: (4.1), (4.2), (4.3) and (4.4).} \end{aligned}$$

Unfortunately, the above formulation is a nonlinear mixed-integer programming problem that is NP-hard in general. We will design an efficient heuristic algorithm to solve the problem in the next section.

Table 4.1: Variables and symbols

Notations	Description
$\vec{X}_i$	The present solution
$\vec{v}_i$	A group of randomly generated feasible break-reconnect information
$\vec{p}_d$	The local optimum in $d^{th}$ loop
$p_g$	The global optimum in all previous $\vec{p}_d$
$w$	The inertia weight
$c_1, c_2$	Acceleration constants, which are used to adjust step
$h_1, h_2$	Two random functions, whose field is $[0,1]$ , which are used to increase search randomness
$\vec{V}$	The set of $\vec{v}_i$
$n_i$	The non-aggregation node which should removed from $m_i$
$m_i$	Denotes the aggregation node which connected to $n_i$
$r_i$	Denotes the other aggregation node
$F$	The fitness for swarm
$f$	Describes the relationship between GPU processing time and workloads
$P$	Particle swarm, it is equal to the dimensions of $\vec{V}$

## 4.4 Centralized Algorithm Design

In this section, we propose a heuristic algorithm based on particle swarm optimization (PSO), which is motivated by the phenomenon of bird predating. The key idea of PSO is to iteratively improve a candidate solution with regard to a given measure of quality [114]. The solution obtained by the PSO may not be theoretically optimal, but it can quickly generate a solution with satisfying performance in practice.

First, we introduce two key variables  $\vec{v}_i$  and  $\vec{X}_i$  for the PSO procedure. The  $\vec{v}_i$  is a group of randomly generated feasible break-reconnect information which consists of  $n_i$ ,  $m_i$  and  $r_i$ , represented by (4.6).

$$[n_i, m_i, r_i] \in \vec{v}_i, \quad (4.6)$$

where  $n_i$  denotes the non-aggregator that should be removed from  $m_i$ ,  $m_i$  denotes the aggregator connected to  $n_i$ , and  $r_i$  denotes the other aggregators. A simple example is shown in Figure 4.3, where the  $i^{th}$  solution has two aggregators 2 and 3.



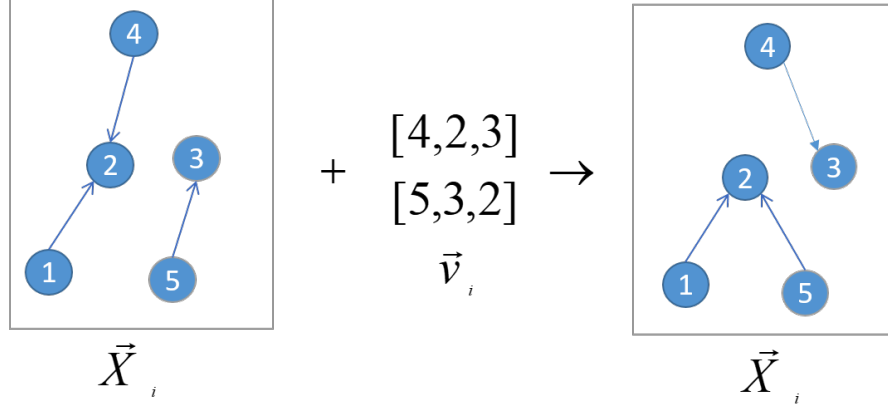


Figure 4.3: a simple illustration for line 7 of Algorithm 2

In addition,  $R$  represents a set of operations generated randomly in each iteration. The other major notations used in this algorithm are summarized in Table 4.1. We use  $\vec{X}_i$  to represent the  $i^{th}$  solution. According to update  $\vec{v}_i$ , the  $\vec{X}_i$  gets its update. Parameters  $c_1$  and  $c_2$  are used to adjust the maximum step of each iteration. In addition,  $h_1$  and  $h_2$  are two random numbers which contribute to search randomness.

---

**Algorithm 2** Implementation of PSO Algorithm

---

```

1: Input:  $\vec{V}$ ,  $P$ ;
2:  $d = 1$ ;
3: while  $d \leq Loop$  do
4:   for  $k$  in  $P$  do
5:     for  $i$  in  $N$  do
6:        $\vec{v}_i \leftarrow w * \vec{R} + c_1 h_1 \otimes (\vec{p}_d - \vec{X}_i) + c_2 h_2 \otimes (p_g - \vec{X}_i)$ ;
7:        $\vec{X}_i \leftarrow \vec{X}_i + \vec{v}_i$ ;
8:     end for
9:      $F \leftarrow Fitness(k)$ ;
10:    if  $\vec{p}_d > F$  then
11:       $\vec{p}_d \leftarrow F$ ;
12:    end if
13:  end for
14:  if  $p_g > \vec{p}_d$  then
15:     $p_g \leftarrow \vec{p}_d$ ;
16:  end if
17:   $d = d + 1$ ;
18: end while
19: Output:  $p_g$ ;

```

---

Then, we introduce the process of the algorithm. The operation for particle

made by the break-reconnect information  $\vec{v}_i$  is to remove the connectivity between the  $n_i$  and  $m_i$ , and then to create a new connection for  $n_i$  with the aggregators in  $r_i$ .  $P$  is particle swarm, which is equal to the dimensions of  $\vec{V}$ . After input  $\vec{V}$  and  $P$ , we initialize  $\vec{p}_d$  as  $p_0$ , which denotes the local optimum related to best solution in  $P$  before the loop start. Next, all kinds of structures of  $\vec{v}_i$  should be considered. In the first place, we should consider when the  $n_i$  change from a general device to an aggregator. The  $n_i$  needs to be disconnected without establishing any connection with others. Under this case, we set  $c_i = 0$ . Similarly, there is also a case where  $r_i$  is in  $\vec{v}_i$ , which means that the  $n_i$  who is the aggregator in the old particle should be converted into the non-aggregator in the new particle, and establish a connection with the aggregator  $m_i$ . At this time, if the  $n_i$  acts as an aggregator in the old particle without connection with other non-aggregators, then we just create a connection to the aggregator  $m_i$  in our algorithm. In the loop,  $m_i$ ,  $r_i$ ,  $\vec{v}_i$  and  $\vec{X}_i$  are updated in each iteration. In addition, the  $fitness(k)$  can be calculated according to (4.5) and  $f(\cdot)$ . Once get the fitness value  $F$ , the local optimum and global optimum can be determined by line 11 of algorithm and line 15 of algorithm.

If the  $n_i$  is selected as the aggregator in the old particle, we need to look for the computation which meets the condition that can be connected with other devices in new particles. In addition, through analysis, it can be found that  $m_i$  and  $r_i$  are not equal to 0 at the same time. Finally, the output  $p_g$  is the global optimum we seek.

## 4.5 Distributed Algorithm Design

In practice, it is difficult to collect global network information and run the centralized algorithm. To address this challenge, we design a distributed algorithm with a practical assumption that each node has only information of one-hop neighbors. The pseudo-codes are shown in Algorithm 3. For each node  $i$ , it first

estimates the local processing time  $T_i^l$  without offloading. Then, it assumes to offload workloads to neighbors and calculates the time  $T_i^o$  by solving a simplified optimization problem. Because the function  $f(\cdot)$  is nonlinear, we can apply the technique proposed in the last section to obtain the results. After that, we compare the values of  $T_i^l$  and  $T_i^o$ . If  $T_i^l \leq T_i^o$ , node  $i$  becomes an aggregation node by setting  $\bar{x}_i = 1$ . Otherwise, it attempts to offload data to neighbors with  $\bar{x}_i = 0$ . Then, node  $i$  broadcast the value of  $\bar{x}_i$  to all neighbors, while receiving  $\bar{x}_j$  from them.

If node  $i$  is not an aggregation node and at least one neighbor works under aggregation mode, node  $i$  solve an updated optimization problem by considering a subset of neighbors  $N'_i$  inline 12. Otherwise, node  $i$  becomes an aggregation node and receive workloads from neighbors.

---

**Algorithm 3** Distributed Algorithm

---

```

1: for each node  $i \in N$  do
2:   calculate the local processing time  $T_i^l = f_i(w_i)$ 
3:   find the  $\bar{y}_{ij}$  by solving the local optimization problem  $\min T_i^o = \sum_{j \in N_i} z_{ij} [2d_{ij} + f_j(w_j + w_i \bar{y}_{ij})]$ 
4:   if  $T_i^l \leq T_i^o$  then
5:      $\bar{x}_i = 1$  //local processing
6:   else
7:      $\bar{x}_i = 0$  //offloading
8:   end if
9:   broadcast  $\bar{x}_i$  to neighbors and receive  $\bar{x}_j$  from neighbors  $j \in N_i$ 
10:  maintain neighbors with  $\bar{x}_j = 1$  in set  $N'_i$ 
11:  if  $\bar{x}_i = 0$  and at least one neighbor is aggregation node then
12:    find the  $\bar{y}'_{ij}$  by solving the local optimization problem  $\min \sum_{j \in N'_i} [2d_{ij} + f_j(w_j + w_i \bar{y}'_{ij})]$ 
13:    send workloads  $w_i \bar{y}'_{ij}$  to neighbor  $j$  and wait for results
14:  else
15:    receive workloads from neighbors, conduct inference and return results
16:  end if
17: end for

```

---

## 4.6 Performance Evaluation

### 4.6.1 Simulation settings

In this subsection, we conduct extensive simulations to evaluate the performance of the PSO-based algorithm. We compare our approach with the optimal solution implemented by Gurobi that is state-of-the-art [115]. In the evaluation, assuming there are no more than 30 users in our experiment environment, because of the limitation of the license of Gurobi. The simulation codes are written in Python, and they run on DELL, whose CPU Core is i5@2.30GHZ and memory 16GB.

In the beginning, we set the population size as 30 in the PSO-based algorithm. In other words, there are 30 initial random solutions. The weight  $w$  and  $Loop$  are set as 0.5 and 50, respectively. And the cost of transmission delay and computation delay is defined by 1 and 1.75, respectively.

### 4.6.2 Simulation results

In this subsection, we show the experimental results of our proposed algorithms. We first evaluate the performance of the proposed PSO algorithm, compared with Random and Optimal.

In Figure 4.4, the blue line denotes the optimal solutions generated by Gurobi, with 30 nodes in the environment. The red curve represents the converging process of our algorithm. When the iteration times exceeds 30, the green line tends to converge.

It can be seen that with the growth of the number of nodes  $N$  which ranges from 2 to 30, the minimum time of PSO always resides between Optimal and NonCollaborative, in Figure 4.5 . Analogous observations can be drawn from Figure 4.6, the number of aggregators of Random generated randomly at the beginning of initialization. Compared with Random, the amount of aggregators of Optimal and PSO is almost the same. In other words, our PSO algorithm is

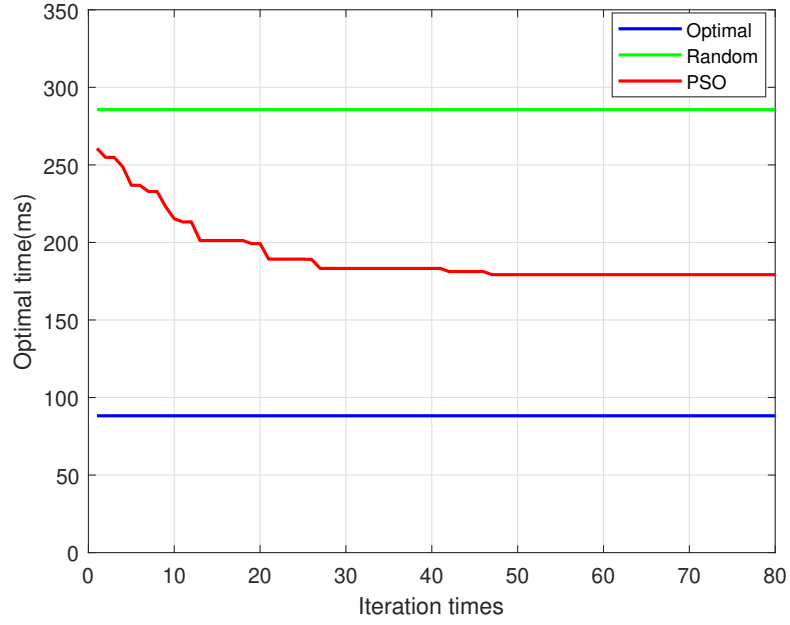


Figure 4.4: Performance analysis comparison of the PSO algorithm and Random Condition.

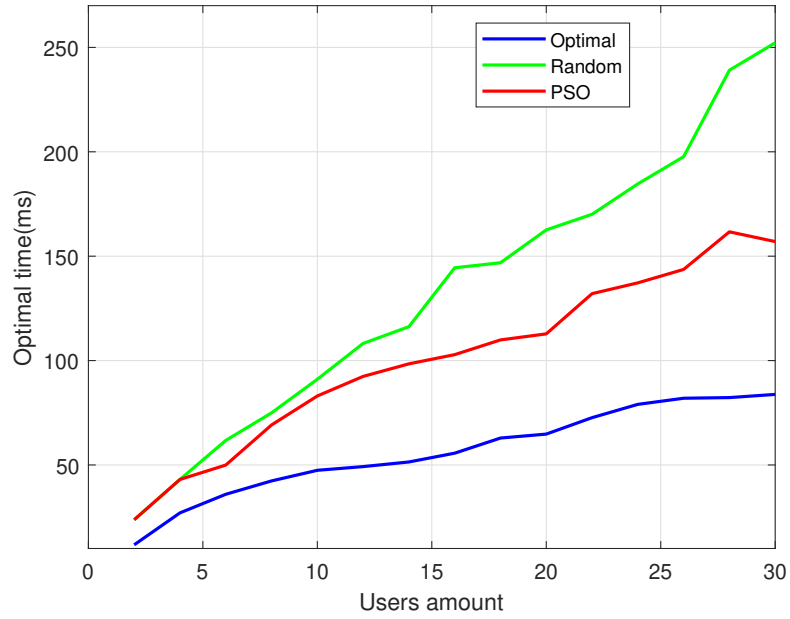


Figure 4.5: With the growth of the number of users  $N$ , which ranging from 2 to 30, the minimum time of PSO resides between Random and Optimal.

able to find the same amount aggregators as Optimal does.

Moreover, we compare our Distributed Algorithm with Optimal and NonCollaborative conditions as shown in Figure 4.7.

In Figure 4.8, compared with Optimal, the number of aggregation nodes of

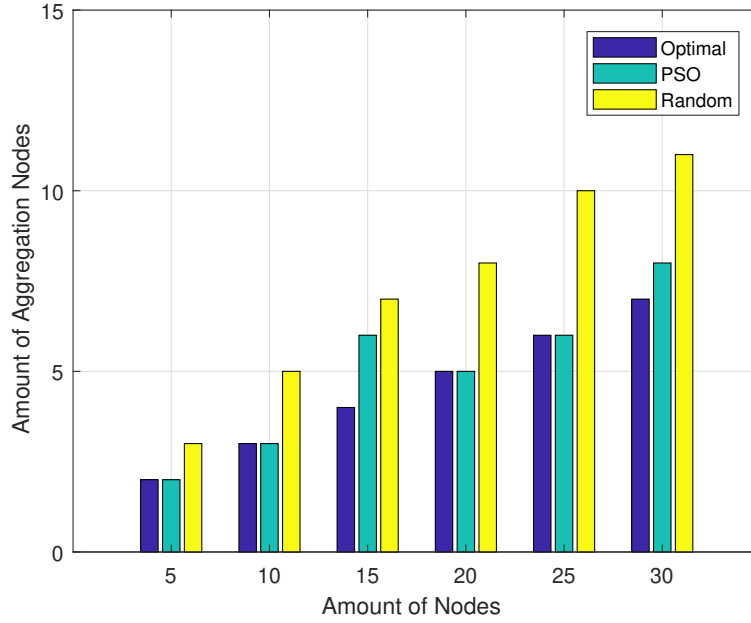


Figure 4.6: Number of aggregation nodes selected over Optimal, PSO and Random.

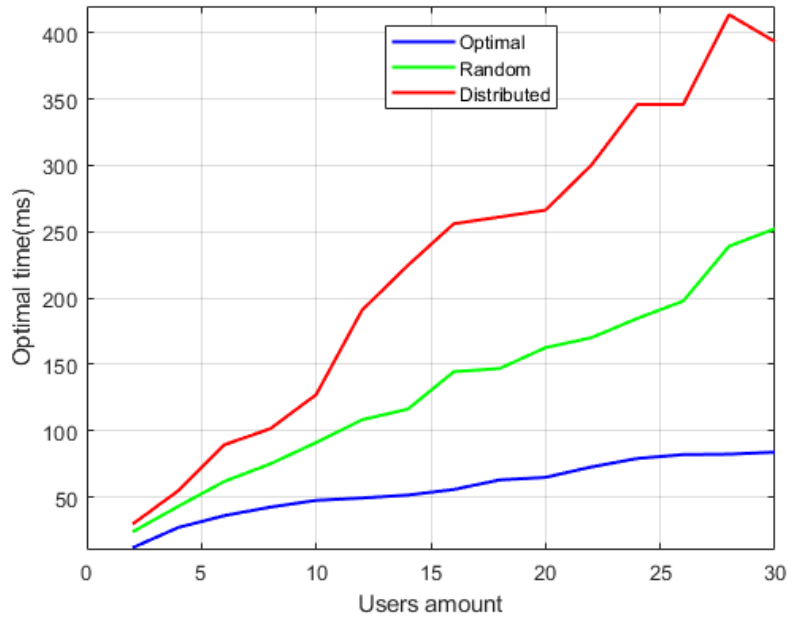


Figure 4.7: With the growth of the number of users  $N$ , which ranging from 2 to 30, the minimum time of Distributed Algorithm resides between NonCollaborative and Optimal.

the Distributed Algorithm is almost always keeping consistent when the number of nodes doesn't exceed 14. When exceeds 14, the Distributed Algorithm will select more aggregation nodes that are almost double than optimal. Because

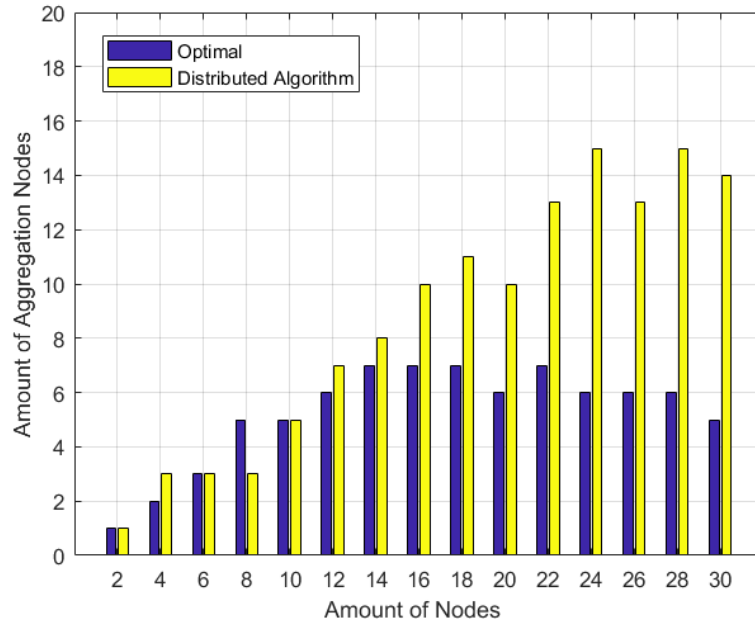


Figure 4.8: Number of aggregation nodes selected over Distributed Algorithm and Optimal.

the Distributed Algorithm makes nodes send data separably when finding the aggregation nodes, which is more applicable to real-life than the nodes must take out all of their data.

## 4.7 Summary

This chapter presents a model for transmission to implement local cooperation among mobile devices motivated by the inference process of machine learning, which can be used to guide the mobile users to choose which approach to handling their data for the goal of improving system performance efficiency. To address the challenge, we design a heuristics global optimization algorithm PSO and a distributed algorithm with a practical assumption that each node has only information on one-hop neighbors.

By performance evaluation, we find that the collaborative inference scheme can reduce global dealing time in the given field compared with handling the data which is affected by the high transmission latency between mobile devices and cloud. As a global optimization random search algorithm, the particle swarm op-

timization algorithm has the characteristics of fast convergence and high precision. In future work, the experiments will be conducted on mobile devices, instead of running simulations, which should be more practical.



# Chapter 5

## Deep Reinforcement Learning based Task Scheduling for Energy Efficiency Edge Computing

### 5.1 Introduction

As a key technology of enabling Artificial Intelligence (AI) applications in the 5G era, Deep Neural Networks (DNNs) have quickly attracted widespread attention. Deep learning technology provides a wide field of capabilities that drive many aspects of user experience including ranking posts, content understanding, object detection and tracking for augmented and virtual reality, speech and text translations [116]. Unfortunately, it remains challenging to run intensive-computation-oriented and low-latency-demanding DNN-based tasks on mobile devices due to the limited computation resources. What's worse, traditional cloud-based DNN inference is heavily hindered by the significant wide-area network latency, leading to poor real-time performance as well as low quality of user experience.

Mobile Edge Computing (MEC) is a new paradigm that evolves to alleviate latency, bandwidth, cost, and quality-of-service (QoS) concerns of cloud-assisted applications as billions of mobile devices are integrated into the Internet. In order

to minimize users' network bandwidth and improve response time (transmission time and executing time), executing inference on the edge makes certain deep learning services possible [116].

However, the energy consumption of servers is becoming a major concern due to the increased requirements of modern computing-intensive applications [9]. The energy efficiency and energy proportionality (EP) of edge server vary in different time and space, thanks to the configuration and workload of the edge server are different [11].

For example, the small- and medium-sized servers are not always keeping a high-load operating state which still consumes much energy when they are idle. For small- and medium-sized edge servers, they are not always in a high-load operating state, and thus the servers still consume a lot of energy when they are idle. The server utilization of small scale is much lower than the large scale's as well as their power use effectiveness [12]. The power consumption of an edge server under a low or idle workload account for more than 20% of the power at 100% workload for a typical server. The number of small- and medium-sized edge server accounts for more than 95% of the installed servers [117].

In addition, the incoming requirements might be sporadic and the workloads of the server fluctuate on an hourly, daily, or weekly basis. In [15], Luiz *et al.* finds that real systems attain peak efficiency at peak utilization but quickly lose efficiency as utilization drops because they are unable to reduce power consumption proportionately. They believe that an ideal energy-proportional (EP) system should always use energy in proportion to the work done by maintaining the peak efficiency, even at a reduced load.

Many research works have been done to improve the energy efficiency of cloud data server by virtual machine scheduling in a visualized environment [11, 118]. However, the works mainly focus on the energy efficiency of cloud servers by virtualization technology which is not suitable for the MEC context, thanks to the conventional communication networks (wireless or wired networks) and di-

verse computing capacity among edge servers. Meanwhile, researchers have been studying sleep (ON/OFF) strategies to address the energy-efficiency problem with promising results [119].

In this chapter, we investigate the energy efficiency (ON/OFF) problem with the tasks migration in the MEC environment, according to the status of edge servers (idle or low-load). We exploit the trade-off among the switching ON/OFF (SO2) the idle or low-load servers to save energy with their tasks migrated to the high-load server by considering their status of workload.

Different from the existing works, we not only exploit the switching ON/OFF strategies, but the migration portion of tasks of the inactive edge nodes. The goal is to fully minimize the energy cost by dynamically switching ON/OFF the edge nodes by considering the status of workload. Meanwhile, this practical scenario that the edge nodes continue to receive tasks from consumers is considered, due to the workloads vary in space and time.

In this dynamic context, the main challenge is that we have no knowledge of incoming tasks that can be formulated into a sequential decision-making problem, and conventional task scheduling algorithms can't solve the dynamic tasks migration problem efficiently. Since deep reinforcement learning (DRL) is being able to tackle a wide range of complex decision-making tasks by integrating deep learning with reinforcement learning [16], we exploit to address this problem by using a learning-based method. To solve the reinforcement learning problem, we formulate it as an optimization problem that can be learned by conventional learning methods. Finally, we conduct extensive simulations to evaluate the proposal.

This chapter makes the following contributions.

- We propose a learning-based task migration scheduling to improve the energy efficiency of the edge server by switching ON/OFF with considering the traffic status in the MEC environment.
- A deep reinforcement learning-based algorithm has been proposed to address the scenario where workloads vary in time and space.

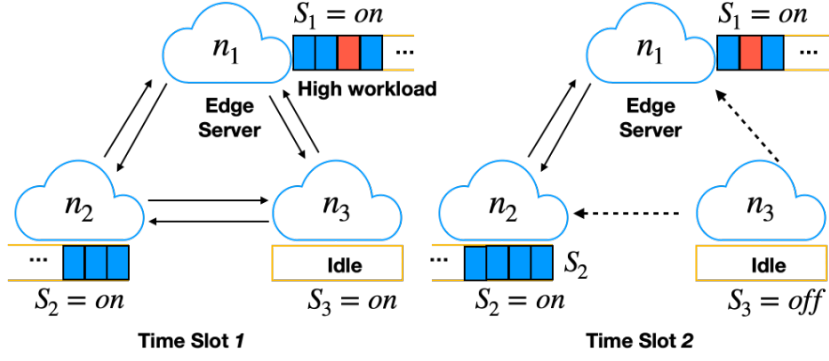


Figure 5.1: Model Overview

- Extensive simulations have been conducted to validate the proposal and the results show that our proposal significantly improves the energy-efficiency.

The rest of this chapter is organized as follows. In Section 5.3, we mainly introduce the system model and give the formulations. The dynamic scenario is designed in Section 5.4. Section 5.4.2 shows the reinforcement learning approach called E3RL. In Section 5.5, simulation experiment is conduct to validate our propose. Some important related work is reviewed in Section 5.2. Finally, Section 5.6 concludes the chapter.

## 5.2 Related Work

In this section, we are mainly discussing the researches that are related to our work. Luiz *et al* [120] believe that energy management has been become a significant issue for servers and data center operations, focusing on reducing all energy-related costs (aka, capital, operating expenses, and environmental impacts). In [121], Wu *et al.* propose Dynamo-a power management system at data center scale that can monitors the entire power hierarchy and make orchestra control decisions to use provisioned data center power safely and efficiently, because there is no real at-scale solution for data center-wide power monitoring in the literature. In [122], Dallal *et al.* investigate the influence of novel features (the emergence at the hypervisor software level of virtual bridging functions, dynamically migrate virtual machines across virtualization servers and more efficient

exploitation of the large path diversity) in data center network optimization by providing a comprehensive mathematical formulation and a repeated matching heuristic for its resolution.

In [118], Jiang *et al.* first explore the power and energy characteristics of different hypervisors in order to emulate realistic multi-tenant cloud environments by using computation-intensive, memory-intensive, and mixed Web server-database workloads. Based on their experiments, they summarize the power and energy characteristics of hypervisors from four aspects: 1. Hypervisors indicate diverse power and energy consumption with the same workload run on the same hardware; 2. Although mainstream hypervisors have different energy efficiencies aligned with different workload types and workload levels, no single hypervisor outperforms the other hypervisors on all platforms in terms of power or energy consumption; 3, It doesn't show more power-efficiency than conventional virtualization technology for container virtualization in terms of implementation and maintenance; 4, It still brings high energy consumption when server completes computation tasks with a long execution time, sometimes, although the ARM64 server has low power consumption. Xu *et al.* [123] show that these state-of-the-art virtualization designs noticeably increase the demand of CPU resources when handling network transactions, generating excessive interrupt requests with ceaseless context switching, which in turn, increases energy consumption via real-world measurement on both Xen- and KVM-based platforms. Even when a physical machine is in an idle state, its VM's network transactions will incur nontrivial energy consumption. According to the analysis in [118,123], it shows that idle server in a cloud environment still brings nontrivial energy consumption. And these works just focus on the testing of energy consumption of VM's in the cloud without energy resources optimization. In [124], the authors apply a deep reinforcement learning approach to address resource management problems in systems and networking which are ubiquitous. Different from existing works, we exploit the switching ON/OFF strategy for edge nodes to achieve energy saving with computing tasks migrated among edge nodes

Table 5.1: Notations

Notations	Description
$E_i(t)$	The energy cost of node $i$ at time slot $t$
$x_i(t)$	Switching ON/OFF at time slot $t$
$z_{i,j}(t)$	The percentage migrated from edge node $i$ transmitted to edge node $j$ at time slot $t$
$D_i(t)$	The magnitude of tasks at edge node $i$ at time slot $t$
$P_i(t)$	The switching cost of edge node $i$ at time slot $t$
$f(\cdot)$	The relationship between energy consumption and workloads

considering the status of the workload.

## 5.3 System Model and Formulation

In this section, we mainly describe the system model as shown in Figure 5.1.

### 5.3.1 Network Model

Assuming that there is a densely deployed edge nodes network, in which tasks are non-uniformly distributed in the coverage area of this network. We consider a time period  $T = \{1, 2, \dots, |T|\}$  that is divided into multiple discrete time slots in which tasks migration are scheduled by time-slot decisions for each time slot  $t \in T$ . Meanwhile, we have no information about the distribution of the traffic for the edge node in each time  $t \in T$ . In this chapter, we assume that the  $D_i(t)$  denote the magnitude of tasks at the edge node  $i$  at time slot  $t$ , and  $i \in N$ , where  $N$  denotes the set of edge nodes.

### 5.3.2 Power Model

Thanks to the lower utilization of smaller edge servers and lower power use effectiveness, some idle or low workload server should be shut down to save power. Hence, the tasks at the edge server with low-load operating can be migrated to the

edge server with a high-load operating state through a wired network. Meanwhile, we think the energy cost of the operation of switching-on or switching-off for a sever should be considered because it takes time for a server to restore service. we denote the active state of the edge server as a binary variable  $x_i(t)$  when  $x_i(t) = 1$  at time slot  $t$ . Otherwise,  $x_i(t) = 0$ .

$$x_i(t) = \begin{cases} 1, & \text{switch-on} \\ 0, & \text{switch-off} \end{cases}$$

In addition, we define a variable  $z_{i,j}(t)$  to represent the portion of workloads migrated from edge node  $i$  to  $j$ . In order to migrate all tasks on the switching-off node, we have the following constraint:

$$1 - x_i(t) \leq \sum_j z_{i,j}(t) \quad (5.1)$$

$$0 \leq \sum_j z_{i,j}(t) \leq 1 \quad (5.2)$$

Considering the limitation of bandwidth, we have the following constraint:

$$\sum_{i,j} z_{i,j}(t) D_i(t) \leq B \quad (5.3)$$

where  $B$  denotes the bandwidth between two edge nodes  $i$  and  $j$ . For the sake of simplicity, we use  $c_i(t)$  to denote the state changing which can be expressed as:

$$c_i(t) = \begin{cases} 0, & \text{otherwise;} \\ 1, & x_i(t) - x_i(t-1) = 1; \end{cases}$$

We define total energy cost  $E_i$  of node  $i$  as the sum of its computation cost without communication cost due to the negligible energy cost,

$$E_i(t) = f(D_i(t) + \sum_j z_{j,i}(t)D_j(t)) + c_i(t)P_i(t) \quad (5.4)$$

where,  $f(\cdot)$  is a non-linear function, which denotes the relationship between the magnitude of workloads and energy cost, due to the energy consumption is tightly related to the task workloads for each task. For example, the energy consumption increases with the number of processes used by the task for a computation-intensive task [125].

In existing work, the operational power of the edge node is considered to be a constant. Meanwhile,  $P_i(t)$  denotes the cost of the switching on or off of the node  $i$  at time slot  $t$ . The energy cost (5.4) includes the tasks processing power and switching power. Most existing works that study the switching problem of base stations with the assumption that the switching energy cost of an edge node is a constant just exists when the inactive state turns into an active state. In practice, the switching energy cost exists whenever the state of an edge node changes. Hence, in order to optimize the long-term energy efficiency, we exploit to use  $E_{avg}$  to denote the long-term time-averaged energy cost budget over a time span of  $T$  time slots, which is expressed by:

$$E_{avg} = \lim_{T \rightarrow \infty} \frac{1}{T} \sum^T \sum^N E_i(t) \quad (5.5)$$

Therefore, the problem of maximize the long-term time energy saving can be formulated as:

$$\begin{aligned} & \min E - E^{avg} \\ & \text{subject to: (5.1 – 5.4).} \end{aligned}$$



## 5.4 Reinforcement Learning-based Methods

To solve the aforementioned objective function, it remains challenging due to the non-linear of the formulation and the status of the edge server varies in space and time. Conventional heuristic algorithms can solve this kind of problem without flexibility, hence, we exploit using a learning-based method (Deep Reinforcement Learning) to address this problem since learning focuses on the analysis of data for decision making. In addition, deep reinforcement learning is a paradigm that learns by trial-and-error to maximize the reward and uses deep representations to reduce the dimensions (state space and action space). Next, we will explain how to use the learning-based method to solve the aforementioned problem in detail.

### 5.4.1 Markov Decision Process Description

From the edge node's perspective, we formulate the scenario as a finite Markov Decision Process (MDP) with a discrete-time step. Meanwhile, the time interval between the two contiguous steps is one hour. With MDP property, the current state transition probabilities and rewards only rely on the state of the environment and the action taken by the agent at the former step.

First, we define the main component of switching ON/OFF decisions as an agent and the rules, rewards mechanism as the environment. At time step  $t$ , we find that the system state  $s_t$  mainly composes of the information about the number of tasks, and the energy cost. According to the above information, the agent is asked to choose action  $a_t$  (switching ON/OFF ( $x_i(t)$ ), and migration percentage ( $z_{i,j}(t)$ )) to minimize the energy consumption of the processed tasks during this time interval. After the action executed, we observe the new system  $s_{t+1}$  and the action  $a_{t+1}$  for time step  $t + 1$  will be chose. MDP gives a mathematical architecture to model decision making in issue with random and partly under the control of decision-makers. We apply a five-tuple  $(S, A, P(. , .), r(. , .), \gamma)$  to describe the MDP process, where  $S$  denotes the system states,  $A$  denotes a finite

set of actions,  $P(.,.)$  stands for the state transition probability,  $r(.,.)$  stands for the immediate reward from the environment after executing the action and  $\gamma$  denotes the discount rate. Specially,  $\gamma \in [0, 1)$ . More details of MDP are described as follow:

**1. State Space:**

We denote a vector  $s_t = (D_i(t), E(t), N_{t-23}, \dots, N_t)$  as the state at time step  $t$  for one edge node. Where,  $E(t)$  denotes the energy consumption of edge nodes at time step  $t$ ;  $D_i(t)$  denotes the number of unperformed tasks for the edge node at time step  $t$ ;  $(N_{t-23}, \dots, N_t)$  stands for the number of arriving tasks for past 24-hours. At time step  $t$ , the edge node has the total number of tasks and for each task, there will be three choices (like  $z_{i,j}(t)$ , and  $x_i(t)$ ).

**2. Action Space:** Given the state  $s_t$ , the action  $a_t = \{x_i(t), z_{i,j}(t)\}$  stands for the processing behavior of each task for the agent at time  $t$ . At each time step, the edge node processes  $D_i(t)$  tasks and the action space is  $2^{D_i(t)}$ . It will be an important and difficult problem to explore the action space, each action affects a state which means the number of state space is very huge. To limit the size of the action space, we propose a pre-screening method before learning. For time step  $t$ , we regard action space as  $a_t = \{x_i(t), z_{i,j}(t)\}$ .

**3. Reward Function :**

From the edge node's perspective, our goal is to maximize the energy-saving, so the reward at time  $t$  is crafted as the energy consumption with switching strategy subtracting the normal energy consumption (without switching ON/OFF strategy)  $E_t$  since we need to maximize the total reward, with the following representation by:

$$E_t - E_{avg}(t) \tag{5.6}$$

Hence, the cumulative reward over time keeps with the energy-saving with switching ON/OFF strategy.

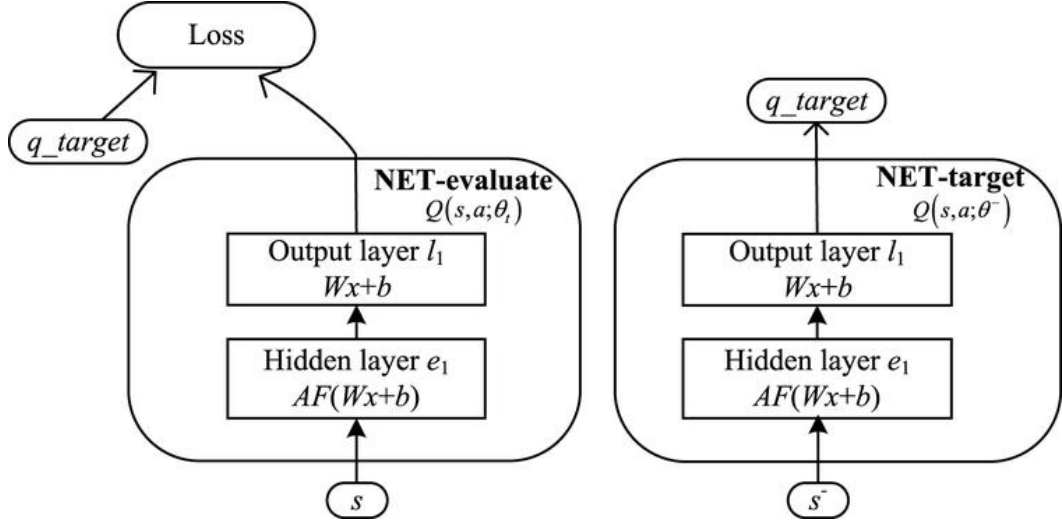


Figure 5.2: Schematic of DQN

#### 4. Action-value Function:

The quality of dynamic scheme is assessed by total discounted future conditional expectation rewards for the horizon  $K$  steps expressed as follow:

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi \left[ \sum_{k=0}^K \gamma^k \cdot r_{t+k} \middle| s_t = s, a_t = a \right] \quad (5.7)$$

where,  $Q^\pi(s_t, a_t)$  denotes the total discounted future conditional expectation rewards accumulated by performing a sequences of actions, eventually. Policy  $\pi$  makes it gain maximum total reward after executing a sequence of actions corresponding to the current state. In the most general setting, a policy means a decision-making strategy where the agent adaptively chooses actions based on the history of observations. The goal of the scheduling problem aims at finding an optimal  $Q^\pi$  to maximum the action-value function expressed as

$$Q^*(s, a) = \max_{\pi} Q^\pi(s_t, a_t) \quad (5.8)$$

where,  $Q^*(s, a)$  denotes the optimal action-value function.

### 5.4.2 Deep Reinforcement Learning Design

In this subsection, we mainly introduce the reinforcement learning method called Deep Q-learning. Given the diverse actions and states in time and space, there are infinite  $\{state, action\}$  pairs which make it intractable to store them in tabular form and address the problem by legacy methods(aka, Q-learning and SARAS [16, 126]). Thanks to the arriving rate fluctuate in a quasi-periodic way and the regular pattern, as usual, it is viable to predict the coming trends from the past arriving rate. To address this issue, we propose to employ a neural network to stand for the policy  $\pi$ , where the neural network can uniformly approximate continuous function [127]. The details of the proposed approach are described below.

#### 1. Deep Q-learning Description

In this chapter, we denote the parameters of Q-network as  $\theta$  (this is, weights), where  $\theta$  denotes the parameters which mainly contains weights and bias. For the sake of simplicity, we apply  $\delta$  to denote the temporal difference error based on Bellman Equation [128] between target Q-value  $r + \gamma Q^\pi(a', s'; \theta^-)$  and prediction action-value  $Q^\pi(s, a)$  at iteration  $i$  which can be calculated by

$$\delta_i = Q^\pi(s, a; \theta_i) - (r + \gamma Q^\pi(a', s'; \theta_i^-)) \quad (5.9)$$

Where,  $s'$  denotes the transferred state after executing the action  $a$  at state  $s$  and  $a'$  is executed action, according to policy  $\pi$ . Moreover,  $\delta$  must be zero in expectation for true optimal Q-function. Where,  $\theta^-$  is the parameter of target network in Fig.5.2. Hence, the loss function at  $i$ -th iteration can be expressed as:

$$L_i(\theta_i) = \begin{cases} r_i, & \text{if Epoch terminates at step } i + 1 ; \\ \delta_i^2, & \text{otherwise.} \end{cases}$$

The loss function is minimized by updating the process of  $\theta$  based gradient

descent approach can be expressed as:

$$\theta_i \leftarrow \theta_i - \eta \nabla L_i(\theta_i) \quad (5.10)$$

Where,  $\eta$  is the learning rate.

## 2. Training Process

First, we present the policy as a prediction network which takes as input the  $\vec{d}_t$ , and the outputs is a probability distribution corresponding to all possible actions. The epoch terminates when all tasks processed. Moreover, from line 3 to line 5, the algorithm mainly initializes the parameters which include the initial  $Q(s, a, \theta)$ ,  $s$ ,  $a$  and network parameter  $\theta$ . Then, the network parameters are updated in the loop for each epoch. When the loop starts, the algorithm first chooses an action based on  $\epsilon$ -greedy search where the schedule is randomly selected with  $\epsilon$  probability. After that, we obtain the reward and the transmission state after executing actions. Next, the transitions  $(s_t, a_t, s_{t+1}, a_{t+1})$  stored in replay memory  $D$  are employed to update the parameters  $\theta$ . Because the algorithm is model-free, it addresses the reinforcement learning task by sampling from  $D$ , without explicitly estimating the reward and transition probability. Furthermore, it learns the greedy policy  $a = \operatorname{argmax}_a Q(s, a'; \theta)$  with following a behavior distribution that ensures deep exploration of the state space. Generally, the behavior distribution is selected by an  $\epsilon$ -greedy policy which means to selects a random action with probability, otherwise follows the greedy policy with probability  $1 - \epsilon$ . Hereafter, we calculate the temporal difference error between its one-step approximation and the action value according to (5.9). With the gradient policy, parameters  $\theta$  are updated in line 12. Finally, the parameters of the target network are updated by the parameters of the prediction network every  $C$  step.

---

**Algorithm 4** SO2 Algorithm

---

```

1: Input:  $(s_t, a_t)$ ;
2: Output: weighted sum  $E - E_{avg}$  at all time instants;
3:  $Q(s, a, \theta) = 0$ ,  $s = s_0$ ,  $a = \pi(s)$ ;
4: Randomly initialize action-value function parameters  $\theta$ ;
5: Initialize target action-value function parameters  $\theta^- = \theta$ 
6: for Epoch=1, 2, ... do
7:   for  $t = 1, 2, \dots, T$  do
8:     Select  $a_t$  with uniform probability  $\epsilon$ 
      Otherwise select  $a_t = \operatorname{argmax}_a Q^\pi(s, a; \theta)$  ;
9:     Observe reward  $r$  and state  $s'$  after executing actions  $a_t$ ;
10:     $s' \leftarrow s, a' \leftarrow a$  ;
11:    Store transition  $(s_t, a_t, s_{t+1}, a_{t+1})$  in  $D$  ;
12:    Perform a gradient descent step on  $\theta_i^2$ 
      with respect to  $\theta \leftarrow \theta - \eta \nabla L_t(\theta_i)$ ;
13:    Reset  $Q^\pi(a', s'; \theta_i^-) = Q^\pi(s, a; \theta_i)$  every  $C$  steps;
14:   end for
15: end for

```

---

## 5.5 Evaluation

In this section, we perform simulation experiments to validate our proposed switching ON/OFF schemes.

### 5.5.1 Settings

Before setting, we build on the following two essential assumptions:

- The edge server energy consumption can be approximated by Multiple Linear Regression, using a linear function of the basic energy consumption, the number of users, and the data transmission rate.
- The energy expended by a system that changes the server status from  $t$  to  $t+1$  in 1 unit of time (here 1 hour), can be approximated again by regression Using a linear function of the absolute change.

The number of tasks and the data transmission speed will fluctuate randomly to simulate a real server. This leads to randomness in energy consumption and the agent has to understand how much energy it has to spend or saved to the server so as not to deteriorate the server's performance. In order to implement

our simulation experiment, we first set the number of tasks given to a Bernoulli process [124]. We construct the SO2 system with the reinforcement learning environment [124] by using a fully connected two hidden layer neural networks with 64 and 32 neurons, respectively. We update the Q-network parameters by using Adam algorithm [129] with a learning rate of 0.0001. The training and testing are programmed with the Tensorflow framework, executed on Ubuntu 18.04 with Nvidia GTX 1080. Here, discount factor is set  $\gamma = 0.9$  and Epsilon greedy exploration rate is set  $\epsilon = 0.3$ . In addition, reply memory is set  $\mathbf{D} = 3000$ . The training batch size is set 128 and the epoch is set 500.

### 5.5.2 Analysis and Discussion

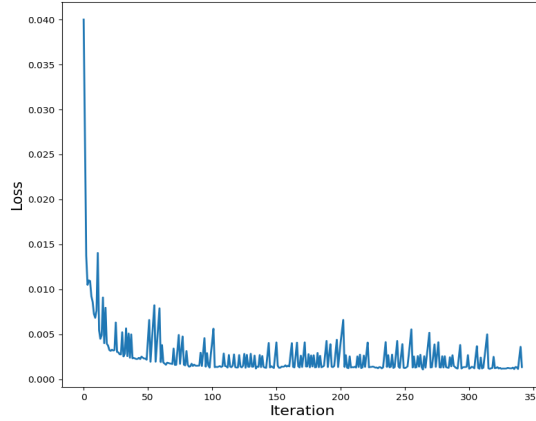


Figure 5.3: The loss during training process

In Figure 5.3, it plots the trend of training loss for deep Q-learning, after about 500 epochs, the fluctuation of loss is very tiny which show the trend of convergence.

In Figure 5.4, we test the energy-saving with the SO2 proposal. We can find that the maximum percentage of energy saved with SO2 can reach almost 70%, while the minimum percentage just reach 0%. We believe that the status of workloads has a significant affect on the decisions of SO2 since each edge server provides full coverage and service during peak traffic time, while offers redundant resources when the traffic workload is low. The results show that our proposal

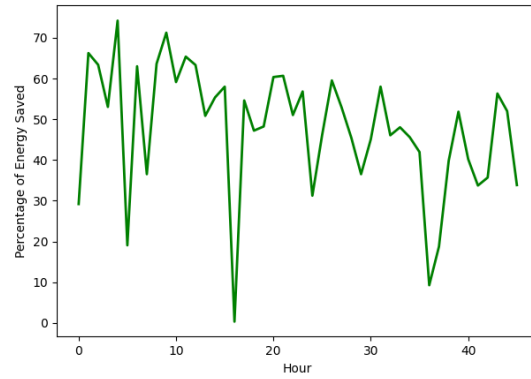


Figure 5.4: The percentage of energy saved compared with Non-SO2 at different time slot (hour)

can make very good energy saving.

## 5.6 Summary

This chapter proposes a switching ON/OFF strategy for edge nodes to achieve energy saving with computing tasks migrated among edge nodes considering the status of the workload. Due to the workloads vary in time and space, we exploit to apply a learning-based approach to address the sequential decisions making problem. The goal is to fully exploit to minimize the energy cost by switching ON/OFF edge nodes with tasks migration. The results show that the switching ON/OFF strategy can save energy compared with the general situations.



# Chapter 6

## Conclusions

To fulfill the high-performance requirement of the fast DNN-based inference service, mobile devices and edge nodes must fully utilize their computation and storage resources to achieve low DNN execution latency. THE existing DNN inference framework seeks to reduce the latency by utilizing heterogeneous hardware resources such as GPU, by offloading the tasks to the cloud or the edge site, by partitioning the DNN model so that it can be processed cooperatively between mobile devices and the cloud. Unfortunately, their performance gets bounded by single-server and individually processing the requirements offloaded by mobile devices. In addition, we find that batching tasks on GPU can significantly reduce average inference time on GPUs. Hence, to maximize the utilization of computation resources (edge sites and mobile devices) in the efficient deep learning service, we exploit to formulate the resource allocation issues as optimization problems.

In Chapter 3, instead of simply offloading the whole task to the edge, we focus on partial offloading, i.e., the mobile devices conduct inference computation over a few layers of CNN models and then send the intermediate results to the edge computing site, which completes the computation of rest layers. We design an algorithm that jointly considers the tasks on all mobile devices and the corresponding batching benefit on the edge site with the objective of minimizing the average inference time of a given number of tasks, different from existing work

on the collaborative inference that lets each mobile device independently make offloading decisions. Moreover, we design an online algorithm to handle the scenario that mobile users submit their inference tasks at different times. The simulation results show that our proposed algorithms significantly outperform existing work.

In Chapter 4, in order to further improve the efficiency of communication in mobile edge computing, we propose to employ local offloading to enable collaborative inference among local mobile devices due to the local offloading like fog computing and mobile edge computing is developed. We investigate to employ partial swarm optimization (PSO) which is a versatile population-based stochastic optimization technique, to help design our collaborative inference scheme.

By performance evaluation, we find that the collaborative inference scheme can reduce global dealing time in the given field compared with handling the data which is affected by the high transmission latency between mobile devices and the cloud. As a global optimization random search algorithm, the particle swarm optimization algorithm has the characteristics of fast convergence and high precision.

In Chapter 5, we propose a switching ON/OFF strategy for edge nodes to achieve energy saving with computing tasks migrated among edge nodes considering the status of the workload. Due to the workloads vary in time and space, we exploit to apply a learning-based approach to address the sequential decisions making problem. The goal is to fully exploit to minimize the energy cost by switching ON/OFF edge nodes with tasks migration. The results show that the switching ON/OFF strategy can save energy compared with the general situations. Finally, we hope these works can inspire blooming studies on the related topics of DNN inference acceleration and efficient edge computing.

# Acknowledgment

In the process of finishing this dissertation, I have got a great range of assistance and support. I would like to thank all the people who helped me during the Ph.D. period.

First, I wish to express my sincere appreciation to my supervisor, Prof. Peng Li. He provided me plenty of professional guidance throughout my doctoral study of three years, especially in the topic selection and the paper writing with great continuous patience.

His passionate attitude toward research always inspired me. He convincingly inspires us to make more improvements in paper writing and presentation. Without his persistent help, the goal of this project would not have been realized.

And I gratefully acknowledge the constructive comments and suggestions from the other review committee members, Prof.Miyazaki, Prof.Lei Jing, and Prof.Anh T.Pham. I have revised my dissertation very carefully by taking all the comments and suggestions into account. Much appreciation for these comments that have much improved the quality of this dissertation. Many thanks for the time and the patience of all referees.

I really appreciate so much that Dr. Xiaofei Luo provided me much help in my doctoral period. As co-author for some papers, he gave me much advice on thinking and we two usually discuss problems together.

Also, really thank all the current lab members, Tao Liu, Changhong Zou, Zhuotao Lian, Fahao Chen, Yawen Liu, Xinyi Zhu, Jinxiao Zhao. Many thanks to these guys who take care of each other in life, Huakun Huang, Lingjun Zhao, Jianbo Xu, Zhaoyang Han, Kungan Zeng, ever lived in the same mansion. I did not think it was possible to have a Ph.D. and so much fun at the same time!

Finally, I wish to acknowledge the support and great love of my family: my grandmother who taught and given me the opportunity of education, my parents who always encouraged and stand with me, my older sisters who offer me much help in life. Thank you to my girlfriend, Huajie Zhang, who provided me with unfailing support and continuous encouragement in life. Last but not least, love for my dears.

Thanks for all your encouragement! Best wishes to you!

# References

- [1] S. Park, K. Bong, D. Shin, J. Lee, S. Choi, and H.-J. Yoo, “4.6 a1. 93tops/w scalable deep learning/inference processor with tetra-parallel mimd architecture for big-data applications,” in *2015 IEEE International Solid-State Circuits Conference-(ISSCC) Digest of Technical Papers*. IEEE, 2015, pp. 1–3.
- [2] E. Meskar, T. D. Todd, D. Zhao, and G. Karakostas, “Energy efficient offloading for competing users on a shared communication channel,” in *2015 IEEE International Conference on Communications (ICC)*. IEEE, 2015, pp. 3192–3197.
- [3] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang, “Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks,” *IEEE access*, vol. 4, pp. 5896–5907, 2016.
- [4] S. Venugopal, M. Gazzetti, Y. Gkoufas, and K. Katrinis, “Shadow puppets: Cloud-level accurate {AI} inference at the speed and economy of edge,” in *{USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.
- [5] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, “Mobile edge computing—a key technology towards 5g,” *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.
- [6] B. P. Rimal, D. Pham Van, and M. Maier, “Mobile-edge computing versus centralized cloud computing over a converged fiwi access network,” *IEEE*

- Transactions on Network and Service Management*, vol. 14, no. 3, pp. 498–513, 2017.
- [7] E. Li, Z. Zhou, and 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.
- [8] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, “Neurosurgeon: Collaborative intelligence between the cloud and mobile edge,” in *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1. ACM, 2017, pp. 615–629.
- [9] D. Fernández-Cerero, A. Fernández-Montes, and F. Velasco, “Productive efficiency of energy-aware data centers,” *Energies*, vol. 11, no. 8, p. 2053, 2018.
- [10] E. Ahvar, A.-C. Orgerie, and A. Lebre, “Estimating energy consumption of cloud, fog and edge computing infrastructures,” *IEEE Transactions on Sustainable Computing*, 2019.
- [11] C. Jiang, Y. Wang, D. Ou, Y. Qiu, Y. Li, J. Wan, B. Luo, W. Shi, and C. Cerin, “Ease: Energy efficiency and proportionality aware virtual machine scheduling,” in *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2018, pp. 65–68.
- [12] Y. Qiu, C. Jiang, Y. Wang, D. Ou, Y. Li, and J. Wan, “Energy aware virtual machine scheduling in data centers,” *Energies*, vol. 12, no. 4, p. 646, 2019.
- [13] D. Mittal, U. N. Kar, and D. K. Sanyal, “A novel matching theory-based framework for computation offloading in device-to-device communication,” in *2017 14th IEEE India Council International Conference (INDICON)*, Dec 2017, pp. 1–6.

- [14] C. Xu, M. Wang, X. Chen, L. Zhong, and L. A. Grieco, “Optimal information centric caching in 5g device-to-device communications,” *IEEE Transactions on Mobile Computing*, vol. 17, no. 9, pp. 2114–2126, 2018.
- [15] L. A. Barroso and U. Hölzle, “The case for energy-proportional computing,” *Computer*, vol. 40, no. 12, pp. 33–37, 2007.
- [16] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [17] T. Mohammed, C. Joe-Wong, R. Babbar, and M. D. Francesco, “Distributed inference acceleration with adaptive dnn partitioning and offloading,” in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 854–863.
- [18] J. Lehtomäki, I. Suliman, J. Vartiainen, M. Bennis, A. Taparugssanagorn, and K. Umebayashi, “Direct communication between terminals in infrastructure based networks,” in *Proc. ICT Mobile Wireless Commun. Summit*. Citeseer, 2008, pp. 1–8.
- [19] A. L. Samuel, “Some studies in machine learning using the game of checkers,” *IBM Journal of research and development*, vol. 3, no. 3, pp. 210–229, 1959.
- [20] U. S. Shanthamallu, A. Spanias, C. Tepedelenlioglu, and M. Stanley, “A brief survey of machine learning methods and their sensor and iot applications,” in *2017 8th International Conference on Information, Intelligence, Systems & Applications (IISA)*. IEEE, 2017, pp. 1–8.
- [21] C. Hewitt and H. Gunes, “Cnn-based facial affect analysis on mobile devices,” *arXiv preprint arXiv:1807.08775*, 2018.
- [22] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, “A survey on the edge computing for the internet of things,” *IEEE access*, vol. 6, pp. 6900–6919, 2017.

- 
- [23] B. Gu and V. Sheng, “A robust regularization path algorithm for  $\nu$ -support vector classification,” *IEEE Transactions on neural networks and learning systems*, vol. 28, no. 5, pp. 1241–1248, 2016.
- [24] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, “Edge intelligence: Paving the last mile of artificial intelligence with edge computing,” *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.
- [25] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [26] K. Noda, Y. Yamaguchi, K. Nakadai, H. G. Okuno, and T. Ogata, “Audio-visual speech recognition using deep learning,” *Applied Intelligence*, vol. 42, no. 4, pp. 722–737, 2015.
- [27] H. Li, K. Ota, and M. Dong, “Learning iot in edge: Deep learning for the internet of things with edge computing,” *IEEE network*, vol. 32, no. 1, pp. 96–101, 2018.
- [28] L. Deng, “A tutorial survey of architectures, algorithms, and applications for deep learning,” *APSIPA Transactions on Signal and Information Processing*, vol. 3, 2014.
- [29] A. Maas, Z. Xie, D. Jurafsky, and A. Y. Ng, “Lexicon-free conversational speech recognition with neural networks,” in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2015, pp. 345–354.
- [30] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen *et al.*, “Deep speech 2: End-to-end speech recognition in english and mandarin,” in *International conference on machine learning*, 2016, pp. 173–182.



- [31] G. Hu, Y. Yang, D. Yi, J. Kittler, W. Christmas, S. Z. Li, and T. Hospedales, “When face recognition meets with deep learning: an evaluation of convolutional neural networks for face recognition,” in *Proceedings of the IEEE international conference on computer vision workshops*, 2015, pp. 142–150.
- [32] J. Liu, Y. Deng, T. Bai, Z. Wei, and C. Huang, “Targeting ultimate accuracy: Face recognition via deep embedding,” *arXiv preprint arXiv:1506.07310*, 2015.
- [33] J. Fu, H. Zheng, and T. Mei, “Look closer to see better: Recurrent attention convolutional neural network for fine-grained image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4438–4446.
- [34] Y. Tian, P. Luo, X. Wang, and X. Tang, “Deep learning strong parts for pedestrian detection,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1904–1912.
- [35] Q. Chu, W. Ouyang, H. Li, X. Wang, B. Liu, and N. Yu, “Online multi-object tracking using cnn-based single object tracker with spatial-temporal attention mechanism,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 4836–4845.
- [36] M. Assefi, M. Wittie, and A. Knight, “Impact of network performance on cloud speech recognition,” in *2015 24th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2015, pp. 1–6.
- [37] Y. Lu, “Artificial intelligence: a survey on evolution, models, applications and future trends,” *Journal of Management Analytics*, vol. 6, no. 1, pp. 1–29, 2019.
- [38] Y. Wang, B. Widrow, L. A. Zadeh, N. Howard, S. Wood, V. C. Bhavsar, G. Budin, C. Chan, R. A. Fiorini, M. L. Gavrilova *et al.*, “Cognitive intelligence: Deep learning, thinking, and reasoning by brain-inspired sys-

- tems,” *International Journal of Cognitive Informatics and Natural Intelligence (IJCINI)*, vol. 10, no. 4, pp. 1–20, 2016.
- [39] L. Deng, D. Yu *et al.*, “Deep learning: methods and applications,” *Foundations and Trends® in Signal Processing*, vol. 7, no. 3–4, pp. 197–387, 2014.
- [40] I. Arel, D. C. Rose, and T. P. Karnowski, “Deep machine learning-a new frontier in artificial intelligence research [research frontier],” *IEEE computational intelligence magazine*, vol. 5, no. 4, pp. 13–18, 2010.
- [41] Z. M. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, “State-of-the-art deep learning: Evolving machine intelligence toward tomorrow’ s intelligent network traffic control systems,” *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2432–2455, 2017.
- [42] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
- [43] A. Kumar, A. Balasubramanian, S. Venkataraman, and A. Akella, “Accelerating deep learning inference via freezing,” in *11th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 19)*, 2019.
- [44] A. S. Rekhi, B. Zimmer, N. Nedovic, N. Liu, R. Venkatesan, M. Wang, B. Khailany, W. J. Dally, and C. T. Gray, “Analog/mixed-signal hardware error modeling for deep learning inference,” in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.
- [45] D. Lungu, J. Gerrand, L. Yang, C. Layton, and R. Stewart, “Apache spark accelerated deep learning inference for large scale satellite image analytics,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2020.
- [46] T. Bolton and L. Zanna, “Applications of deep learning to ocean data inference and subgrid parameterization,” *Journal of Advances in Modeling Earth Systems*, vol. 11, no. 1, pp. 376–399, 2019.

- [47] W. Hua, Y. Zhou, C. De Sa, Z. Zhang, and G. E. Suh, “Boosting the performance of cnn accelerators with dynamic fine-grained channel gating,” in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 139–150.
- [48] S. Sharify, A. D. Lascorz, M. Mahmoud, M. Nikolic, K. Siu, D. M. Stuart, Z. Poulos, and A. Moshovos, “Laconic deep learning inference acceleration,” in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 304–317.
- [49] S. Gudaparthi, S. Narayanan, R. Balasubramonian, E. Giacomini, H. Kam-balasubramanyam, and P.-E. Gaillardon, “Wire-aware architecture and dataflow for cnn accelerators,” in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 1–13.
- [50] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, “Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach,” in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 754–768.
- [51] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, “Understanding error propagation in deep learning neural network (dnn) accelerators and applications,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–12.
- [52] L. Pentecost, M. Donato, B. Reagen, U. Gupta, S. Ma, G.-Y. Wei, and D. Brooks, “Maxnvm: Maximizing dnn storage density and inference efficiency with sparse encoding and error mitigation,” in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 769–781.

- 
- [53] E. Elghoneimy, O. Bouhali, and H. Alnuweiri, "Resource allocation and scheduling in cloud computing," in *2012 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2012, pp. 309–314.
- [54] T. Ma, Y. Chu, L. Zhao, and O. Ankhbayar, "Resource allocation and scheduling in cloud computing: policy and algorithm," *IETE Technical review*, vol. 31, no. 1, pp. 4–16, 2014.
- [55] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future generation computer systems*, vol. 28, no. 5, pp. 755–768, 2012.
- [56] A. J. Younge, G. Von Laszewski, L. Wang, S. Lopez-Alarcon, and W. Carithers, "Efficient resource management for cloud computing environments," in *International Conference on Green Computing*. IEEE, 2010, pp. 357–364.
- [57] B. Xu, C. Zhao, E. Hu, and B. Hu, "Job scheduling algorithm based on berger model in cloud environment," *Advances in Engineering Software*, vol. 42, no. 7, pp. 419–425, 2011.
- [58] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, 2012, pp. 13–16.
- [59] C. Wang, C. Liang, F. R. Yu, Q. Chen, and L. Tang, "Computation offloading and resource allocation in wireless cellular networks with mobile edge computing," *IEEE Transactions on Wireless Communications*, vol. 16, no. 8, pp. 4924–4938, 2017.
- [60] S. Yu, X. Wang, and R. Langar, "Computation offloading for mobile edge computing: A deep learning approach," in *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*. IEEE, 2017, pp. 1–6.
-

- [61] K. Zhang, Y. Zhu, S. Leng, Y. He, S. Maharjan, and Y. Zhang, “Deep learning empowered task offloading for mobile edge computing in urban informatics,” *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 7635–7647, 2019.
- [62] Y. Mao, J. Zhang, and K. B. Letaief, “Dynamic computation offloading for mobile-edge computing with energy harvesting devices,” *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, 2016.
- [63] Y. Hao, M. Chen, L. Hu, M. S. Hossain, and A. Ghoneim, “Energy efficient task caching and offloading for mobile edge computing,” *IEEE Access*, vol. 6, pp. 11 365–11 373, 2018.
- [64] J. Zhang, X. Hu, Z. Ning, E. C.-H. Ngai, L. Zhou, J. Wei, J. Cheng, and B. Hu, “Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks,” *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2633–2645, 2017.
- [65] P. Zhao, H. Tian, C. Qin, and G. Nie, “Energy-saving offloading by jointly allocating radio and computational resources for mobile edge computing,” *IEEE Access*, vol. 5, pp. 11 255–11 268, 2017.
- [66] X. Chen, L. Pu, L. Gao, W. Wu, and D. Wu, “Exploiting massive d2d collaboration for energy-efficient mobile edge computing,” *IEEE Wireless communications*, vol. 24, no. 4, pp. 64–71, 2017.
- [67] A. Ndikumana, N. H. Tran, T. M. Ho, Z. Han, W. Saad, D. Niyato, and C. S. Hong, “Joint communication, computation, caching, and control in big data multi-access edge computing,” *IEEE Transactions on Mobile Computing*, 2019.
- [68] F. Wang, J. Xu, X. Wang, and S. Cui, “Joint offloading and computing optimization in wireless powered mobile-edge computing systems,” *IEEE*

- Transactions on Wireless Communications*, vol. 17, no. 3, pp. 1784–1797, 2017.
- [69] T. X. Tran and D. Pompili, “Joint task offloading and resource allocation for multi-server mobile-edge computing networks,” *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 856–868, 2018.
- [70] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, “Performance optimization in mobile-edge computing via deep reinforcement learning,” in *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*. IEEE, 2018, pp. 1–6.
- [71] C. You, K. Huang, H. Chae, and B.-H. Kim, “Energy-efficient resource allocation for mobile-edge computation offloading,” *IEEE Transactions on Wireless Communications*, vol. 16, no. 3, pp. 1397–1411, 2016.
- [72] G. Qiao, S. Leng, and Y. Zhang, “Online learning and optimization for computation offloading in d2d edge computing and networks,” *Mobile Networks and Applications*, pp. 1–12, 2019.
- [73] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. Quek, “Offloading in mobile edge computing: Task allocation and computational frequency scaling,” *IEEE Transactions on Communications*, vol. 65, no. 8, pp. 3571–3584, 2017.
- [74] W. Li, T. Yang, F. C. Delicato, P. F. Pires, Z. Tari, S. U. Khan, and A. Y. Zomaya, “On enabling sustainable edge computing with renewable energy resources,” *IEEE Communications Magazine*, vol. 56, no. 5, pp. 94–101, 2018.
- [75] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, “Quantized convolutional neural networks for mobile devices,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4820–4828.

- [76] S. S. Ogden and T. Guo, “{MODI}: Mobile deep inference made efficient by edge computing,” in *{USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.
- [77] E. Li, L. Zeng, Z. Zhou, and X. Chen, “Edge ai: On-demand accelerating deep neural network inference via edge computing,” *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 447–457, 2020.
- [78] X. Lin, Y. Rivenson, N. T. Yardimci, M. Veli, Y. Luo, M. Jarrahi, and A. Ozcan, “All-optical machine learning using diffractive deep neural networks,” *Science*, vol. 361, no. 6406, pp. 1004–1008, 2018.
- [79] A. Kaplan and M. Haenlein, “Siri, siri, in my hand: Who’s the fairest in the land? on the interpretations, illustrations, and implications of artificial intelligence,” *Business Horizons*, vol. 62, no. 1, pp. 15–25, 2019.
- [80] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, “Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning,” *IEEE Internet of Things Journal*, 2018.
- [81] L. Lin, X. Liao, H. Jin, and P. Li, “Computation offloading toward edge computing,” *Proceedings of the IEEE*, 2019.
- [82] P. Li, X. Wu, W. Shen, W. Tong, and S. Guo, “Collaboration of heterogeneous unmanned vehicles for smart cities,” *IEEE Network*, vol. 33, no. 4, pp. 133–137, 2019.
- [83] P. Li, T. Miyazaki, K. Wang, S. Guo, and W. Zhuang, “Vehicle-assist resilient information and network system for disaster management,” *IEEE Transactions on Emerging Topics in Computing*, vol. 5, no. 3, pp. 438–448, 2017.
- [84] M. Satyanarayanan, “The emergence of edge computing,” *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

- 
- [85] Z. Zhang, F. R. Yu, F. Fu, Q. Yan, and Z. Wang, “Joint offloading and resource allocation in mobile edge computing systems: An actor-critic approach,” in *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2018, pp. 1–6.
- [86] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, “Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning,” *IEEE Internet of Things Journal*, 2018.
- [87] S. Guo, B. Xiao, Y. Yang, and Y. Yang, “Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing,” in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.
- [88] T. Q. Dinh, Q. D. La, T. Q. Quek, and H. Shin, “Learning for computation offloading in mobile edge computing,” *IEEE Transactions on Communications*, vol. 66, no. 12, pp. 6353–6367, 2018.
- [89] S. Jošilo and G. Dán, “Selfish computation offloading for mobile cloud computing in dense wireless networks,” *arXiv preprint arXiv:1604.05460*, 2016.
- [90] 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 < 0.5 mb model size,” *arXiv preprint arXiv:1602.07360*, 2016.
- [91] K.-h. Kim, Y. Kim, I. Kim, H.-K. Kim, W. Nam, S. Boo, M. Sung, D. Yeo, R. Wooju, T. Jang *et al.*, “Method and device for transforming cnn layers to optimize cnn parameter quantization to be used for mobile devices or compact networks with high precision via hardware optimization,” Jun. 18 2019, uS Patent 10,325,352.
- [92] L. Wang, Z. Chen, Y. Liu, Y. Wang, L. Zheng, M. Li, and Y. Wang, “A unified optimization approach for cnn model inference on integrated gpus,”



- in *Proceedings of the 48th International Conference on Parallel Processing*, 2019, pp. 1–10.
- [93] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, “Maui: making smartphones last longer with code offload,” in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 49–62.
- [94] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, “Optimizing fpga-based accelerator design for deep convolutional neural networks,” in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2015, pp. 161–170.
- [95] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, “Odessa: enabling interactive perception applications on mobile devices,” in *Proceedings of the 9th international conference on Mobile systems, applications, and services*. ACM, 2011, pp. 43–56.
- [96] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi *et al.*, “A configurable cloud-scale dnn processor for real-time ai,” in *Proceedings of the 45th Annual International Symposium on Computer Architecture*. IEEE Press, 2018, pp. 1–14.
- [97] X. Xu, Y. Ding, S. X. Hu, M. Niemier, J. Cong, Y. Hu, and Y. Shi, “Scaling for edge inference of deep neural networks,” *Nature Electronics*, vol. 1, no. 4, p. 216, 2018.
- [98] P. Li, S. Guo, W. Zhuang, and B. Ye, “On efficient resource allocation for cognitive and cooperative communications,” *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 2, pp. 264–273, 2013.

- 
- [99] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic, “Deep learning applications and challenges in big data analytics,” *Journal of Big Data*, vol. 2, no. 1, p. 1, 2015.
- [100] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, “A survey on mobile edge computing: The communication perspective,” *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.
- [101] P. Mach and Z. Becvar, “Mobile edge computing: A survey on architecture and computation offloading,” *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.
- [102] C. Wu, B. Yang, W. Zhu, and Y. Zhang, “Toward high mobile gpu performance through collaborative workload offloading,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 2, pp. 435–449, 2017.
- [103] H. Xing, L. Liu, J. Xu, and A. Nallanathan, “Joint task assignment and resource allocation for d2d-enabled mobile-edge computing,” *IEEE Transactions on Communications*, vol. 67, no. 6, pp. 4193–4207, 2019.
- [104] Wikipedia, “Peer-to-peer,” <https://en.wikipedia.org/wiki/Peer-to-peer>, february 5, 2021.
- [105] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, “cudnn: Efficient primitives for deep learning,” *arXiv preprint arXiv:1410.0759*, 2014.
- [106] A. Canziani, A. Paszke, and E. Culurciello, “An analysis of deep neural network models for practical applications,” *arXiv preprint arXiv:1605.07678*, 2016.
- [107] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” *arXiv preprint arXiv:1510.00149*, 2015.
-

- [108] Z. Tang, S. Guo, P. Li, T. Miyazaki, H. Jin, and X. Liao, “Energy-efficient transmission scheduling in mobile phones using machine learning and participatory sensing,” *IEEE Transactions on Vehicular Technology*, vol. 64, no. 7, pp. 3167–3176, 2014.
- [109] G. Chatzimilioudis, A. Konstantinidis, C. Laoudias, and D. Zeinalipour-Yazti, “Crowdsourcing with smartphones,” *IEEE Internet Computing*, vol. 16, no. 5, pp. 36–44, 2012.
- [110] M. Heyi and C. Rossi, “On the evaluation of cloud web services for crowdsourcing mobile applications,” in *2016 2nd International Conference on Cloud Computing Technologies and Applications (CloudTech)*. IEEE, 2016, pp. 57–62.
- [111] D. Yao, C. Yu, L. Yang, and H. Jin, “Using crowdsourcing to provide qos for mobile cloud computing,” *IEEE Transactions on Cloud Computing*, 2015.
- [112] H. Ke, P. Li, and S. Guo, “Crowdsourcing on mobile cloud: cost minimization of joint data acquisition and processing,” in *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2014, pp. 358–362.
- [113] J. Fan, Q. Li, and G. Cao, “Privacy-aware and trustworthy data aggregation in mobile sensing,” in *2015 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2015, pp. 31–39.
- [114] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Proceedings of ICNN’95-International Conference on Neural Networks*, vol. 4. IEEE, 1995, pp. 1942–1948.
- [115] “GUROBI,” <https://www.gurobi.com/products/gurobi-optimizer/gurobi-features-and-benefits/>.
- [116] C. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia, T. Leyvand, H. Lu, Y. Lu, L. Qiao, B. Reagen,

- J. Spisak, F. Sun, A. Tulloch, P. Vajda, X. Wang, Y. Wang, B. Wasti, Y. Wu, R. Xian, S. Yoo, and P. Zhang, “Machine learning at facebook: Understanding inference at the edge,” in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 331–344.
- [117] S. Zafar, S. A. Chaudhry, and S. Kiran, “Adaptive trintree: Green data center networks through resource consolidation, selective connectedness and energy proportional computing,” *Energies*, vol. 9, no. 10, p. 797, 2016.
- [118] C. Jiang, Y. Wang, D. Ou, Y. Li, J. Zhang, J. Wan, B. Luo, and W. Shi, “Energy efficiency comparison of hypervisors,” *Sustainable Computing: Informatics and Systems*, vol. 22, pp. 311–321, 2019.
- [119] M. Miozzo, L. Giupponi, M. Rossi, and P. Dini, “Switch-on/off policies for energy harvesting small cells through distributed q-learning,” in *2017 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, 2017, pp. 1–6.
- [120] L. A. Barroso and U. Hölzle, “The case for energy-proportional computing,” *Computer*, vol. 40, no. 12, pp. 33–37, 2007.
- [121] Q. Wu, Q. Deng, L. Ganesh, C.-H. Hsu, Y. Jin, S. Kumar, B. Li, J. Meza, and Y. J. Song, “Dynamo: Facebook’s data center-wide power management system,” in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA ’16. IEEE Press, 2016, p. 469-480. [Online]. Available: <https://doi.org/10.1109/ISCA.2016.48>
- [122] D. Belabed, S. Secci, G. Pujolle, and D. Medhi, “Striking a balance between traffic engineering and energy efficiency in virtual machine placement,” *IEEE Transactions on Network and Service Management*, vol. 12, no. 2, pp. 202–216, 2015.

- [123] C. Xu, Z. Zhao, H. Wang, R. Shea, and J. Liu, “Energy efficiency of cloud virtual machines: From traffic pattern and cpu affinity perspectives,” *IEEE Systems Journal*, vol. 11, no. 2, pp. 835–845, 2015.
- [124] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, “Resource management with deep reinforcement learning,” in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. ACM, 2016, pp. 50–56.
- [125] Z. Zhang and S. Fu, “Characterizing power and energy usage in cloud computing systems,” in *2011 IEEE Third International Conference on Cloud Computing Technology and Science*. IEEE, 2011, pp. 146–153.
- [126] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.
- [127] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.
- [128] R. Bellman, “Dynamic programming,” *Science*, vol. 153, no. 3731, pp. 34–37, 1966.
- [129] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.