Collaborate Edge and Cloud Computing With Distributed Deep Learning for Smart City Internet of Things

Huaming Wu, *Member, IEEE*, Ziru Zhang, Chang Guan, Katinka Wolter, *Associate Member, IEEE*, and Minxian Xu, *Member, IEEE*

***Abstract*—City Internet-of-Things (IoT) applications are becoming increasingly complicated and thus require large amounts of computational resources and strict latency requirements. Mobile cloud computing (MCC) is an effective way to alleviate the limitation of computation capacity by offloading complex tasks from mobile devices (MDs) to central clouds. Besides, mobile-edge computing (MEC) is a promising technology to reduce latency during data transmission and save energy by providing services in a timely manner. However, it is still difficult to solve the task offloading challenges in heterogeneous cloud computing environments, where edge clouds and central clouds work collaboratively to satisfy the requirements of city IoT applications. In this article, we consider the heterogeneity of edge and central cloud servers in the offloading destination selection. To jointly optimize the system utility and the bandwidth allocation for each MD, we establish a hybrid offloading model, including the collaboration of MCC and MEC. A distributed deep learning-driven task offloading (DDTO) algorithm is proposed to generate near-optimal offloading decisions over the MDs, edge cloud server, and central cloud server. Experimental results demonstrate the accuracy of the DDTO algorithm, which can effectively and efficiently generate near-optimal offloading decisions in the edge and cloud computing environments. Furthermore, it achieves high performance and greatly reduces the computational complexity when compared with other offloading schemes that neglect the collaboration of heterogeneous clouds. More precisely, the DDTO scheme can improve computational performance by 63%, compared with the local-only scheme.**

***Index Terms*—City Internet of Things (IoT), distributed deep learning, mobile cloud computing (MCC), mobile-edge computing (MEC), task offloading.**

Manuscript received February 29, 2020; revised April 25, 2020; accepted May 19, 2020. Date of publication May 25, 2020; date of current version September 15, 2020. This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFC0809800, in part by the National Natural Science Foundation of China under Grant 61801325, in part by the Natural Science Foundation of Tianjin City under Grant 18JCQNJC00600, and in part by the Huawei Innovation Research Program under Grant HO2018085138. *(Ziru Zhang and Chang Guan contributed equally to this work.) (Corresponding author: Huaming Wu.)*

Huaming Wu is with the Center for Applied Mathematics, Tianjin University, Tianjin 300072, China (e-mail: whming@tju.edu.cn).

Ziru Zhang and Chang Guan are with the School of Mathematics, Tianjin University, Tianjin 300072, China.

Katinka Wolter is with Institut für Informatik, Freie Universität Berlin, 14195 Berlin, Germany (e-mail: katinka.wolter@fu-berlin.de).

Minxian Xu is with the Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518055, China (e-mail: mx.xu@siat.ac.cn).

Digital Object Identifier 10.1109/JIOT.2020.2996784

I. Introduction

With the fast development of mobile networks and the widespread application of city Internet of Things (IoT) in various fields (e.g., smart transportation, smart home, and smart manufacturing), the demand for mobile devices (MDs) is increasing drastically. However, MDs, such as smartphones, tablet computers, unmanned aerial vehicles (UAVs), and wearable devices, are usually constrained by limited resources, e.g., CPU computing power, storage space, energy capacity, and environmental awareness. Complex computing tasks, e.g., optical character recognition (OCR), face recognition (FR), and augmented reality (AR), are inefficient to be handled locally. Furthermore, a diversity of city IoT applications, such as delay-sensitive and delay-tolerant applications can cause a variety of different computation and communication costs.

To alleviate the limitations of mobile computation capacity, one effective way is to offload complex compute tasks from the MDs to a central cloud. By taking advantage of the rich virtual resources and the fast processing speed of the cloud servers, we can lower the pressure on MDs in handling tasks locally. Considerable attention has been paid to mobile cloud computing (MCC), which has emerged as a solution to offload task workloads to computation-rich cloud data centers. However, due to the limitations of the centralized service mode and access bandwidth, this approach still faces many challenges, such as high latency, low bandwidth, and network congestion.

Always offloading the tasks to the central cloud server, however, is not suitable, especially for those tasks that are data concentrated and latency sensitive [1]. Recently, mobile-edge computing (MEC) has emerged as a novel computing paradigm that harnesses computing resources in the proximity of IoT devices. It has attracted extensive interest [2]. In MEC, IoT devices are connected to edge servers instead of directly to cloud servers. Around 29 billion IoT devices are estimated to be connected to the Internet by 2022 [3]. Due to their proximity to mobile users, the communication cost for task offloading will become very small, which can greatly reduce the latency of network operations and service delivery, and further meet the requirements of the ultrahigh bandwidth and ultralow latency of future networks [4], [5]. However, the computing power of edge cloud servers is relatively low and cannot efficiently satisfy the requirements of the

city IoT applications while central cloud servers have sufficient computing power.

To better serve IoT users with diverse requirements, heterogeneous clouds composed of edge clouds and central clouds should be jointly exploited to meet stringent delay requirements. This will make task execution faster, cheaper, and more stable. However, if all computing tasks are only offloaded to the edge or central cloud server, the wireless link between the IoT devices and the MCC or MEC servers can be congested and the latency of the computation can be unacceptable. In fact, MCC and MEC can cooperate in terms of computing, storage, and communication facilities since they are complementary to each other. To reduce the overall cost of delay and energy consumption, how to make real-time offloading decisions becomes the most significant challenge.

Due to the rapid changes in channel conditions, the number of users and other system parameters, offloading decisions and resource allocation need to be completed within a few milliseconds. In practical scenarios, however, large-scale offloading decision making is often involved since the total number of possible decisions increases exponentially with the number of users and tasks, and it is very difficult to enumerate all possible decisions. Conventional task offloading techniques usually apply some heuristic algorithms, which involve difficult to solve and complex problems that require a large amount of computation, and additional computation is also needed to execute the offloading decisions. In addition, the best solution can usually not be calculated in the face of complex workflows with correlations and only local optimal solutions can be given. Considering the absence of the optimal decisions for each task workload of the users, deep learning becomes a promising method due to its ability to provide solutions based on labeled data. Deep learning-driven approaches can facilitate offloading decision making, dynamic resource allocation, and content caching as they benefit from the growth in volumes of communication and computation for emerging city IoT applications. However, how to customize deep learning techniques for task offloading in IoT is still unknown.

To minimize the weighted sum of the task completion delay and energy consumption while maintaining the Quality of Service (QoS) for MDs, more intelligent technologies and effective parallel algorithms are required to address such complicated offloading scenarios limited by high dimensionality. In this article, the motivation of designing a distributed deep learning-driven task offloading (DDTO) algorithm is to find a way to proceed with optimal learning in the MEC and MCC heterogeneous environments and to further solve computationally expensive problems in offloading decision making. Considering the characteristics of the abundant computing resources in MCC and the low transmission delay in MEC comprehensively, we integrate MCC and MEC for task offloading.

The contributions of this article can be summarized as follows.

1) We formalize the MCC and MEC hybrid task placement problem as a multiobjective optimization problem. To jointly minimize the system utility and the bandwidth allocation for each MD, we propose an effective and efficient offloading framework with intelligent decision-making capabilities.
2) We design a DDTO algorithm, where multiple parallel deep neural networks (DNNs) are adopted to effectively and efficiently generate offloading decisions over the MDs, edge cloud server, and central cloud server.
3) We conduct experiments in distinct situations to evaluate the effectiveness of DDTO. When compared with several offloading schemes without the cooperation of MEC and MCC, our proposed DDTO algorithm can achieve superior performance.

The remainder of this article is organized as follows. In Section II, we review the related work. The system model and problem formulation are described in Section III. The proposed algorithm based on deep learning to generate the optimal binary offloading decisions is presented in Section IV. The numerical and comparison results are shown in Section V. Finally, this article is concluded in Section VI.

II. Related Work

MCC and MEC have become important solutions to satisfy the requirements of applications running on IoT devices, especially for latency-sensitive applications and those running on energy-constrained IoT devices. A significant number of offloading decision schemes in MCC and MEC are provided in the literature, which can be classified as follows.

A. Markov-Based Offloading Decisions

The Markov decision process is a well-known discrete-time mathematical framework applied for modeling decision making with uncertainty. It models a system based on Markov chains during the time which experiences the transition from one state to another according to certain probabilistic rules.

Numerous stochastic offloading schemes via modeling the task offloading procedure as Markov decision processes have been proposed in the literature to help them in making better offloading decisions [6]. Several queueing models were applied in [7] and [8] to mitigate the weighted sum of power usage and performance expressed in different metrics. Various offloading decision policies have been taken into account, where arriving tasks are either processed locally in the MDs or offloaded to the remote cloud via a WLAN or cellular network. Moreover, a Markov-based offloading strategy was developed, which solved the problem of where to offload the tasks based on an $M/G/1$-FCFS queue model [7]. The offloading approach proposed in [8] supports two delayed offloading policies, a partial offloading model where jobs can leave the slow offloading to be executed locally, and a full offloading model, where jobs can be offloaded directly via the cellular network. Alasmari *et al.* [6] proposed a mobile-edge offloading method based on a Markov decision process to generate offloading decisions, which used a numbering scheme (1, 2, and 3) to denote executing the tasks in local devices, at the edge and cloud, respectively. Considering the clock frequency configuration, transmission power allocation, channel rate scheduling, and offloading strategy selection, a

distributed algorithm was derived in [11], where an $M/M/n$ queue model was also used to optimize the offloading decision.

B. Graph-Based Offloading Decisions

It is important to note that city IoT applications can be viewed as heterogeneous workflows with a different number of tasks and data flows. To make offloading decisions based on optimizing the response time or energy consumption, many research efforts have been devoted to computation partitioning in mobile computing. Automatic application partitioning has attracted more and more attention.

The offloading operation can be modeled via a cost graph, where finding the optimal solution for offloading is equivalent to finding the constrained shortest path in this graph [12], [32], [33]. Zhang and Wen [15] modeled a mobile application as a general topology, which consists of a set of fine-grained tasks. Each task within the application can be either executed on the MD or on the cloud. By using arbitrary topographical consumption graphs, Wu *et al.* [16] proposed a graph-cut-based partitioning algorithm, which determines whether the parts of the tasks run locally or offload to the cloud server. The decision engine in this proposal is placed at the MD aiming at finding a group of tasks for offloading, by which the execution time of a mobile application and energy consumption of an MD are reduced.

Preferably, the graph partitioning between IoT devices and cloud/edge servers should be dynamic and the offloading decisions should be made adaptively at runtime. However, only homogeneous resources are considered in these studies, and unlike them, we consider the edge and cloud computing to be heterogeneous environments to support the IoT applications running on diverse devices in a better manner.

C. Optimization-Based Offloading Decisions

A diversity of platforms and algorithms have been proposed to solve the problems of offloading binary decisions for MCC and MEC.

An offloading platform named MAPCloud was proposed in [17], which consists of a local cloud and a common cloud. MAPCloud determined the optimal location of tasks according to multiple QoS factors of users. El Haber *et al.* [18] proposed a successive convex approximation method, which approximately optimizes the computational cost and figures out the energy-efficient task offloading strategy mathematically. A computational offloading algorithm based on the NSGA-III is presented in [20], where big data methods have been used for IoT-enabled cloud-edge computing. In addition, computation offloading game theory has been discussed in [21], which proposed C-SGA and F-SGA algorithms to solve the problem.

Energy-efficient task offloading algorithms in MEC or MCC based on the Lyapunov optimization theory have been widely investigated [22], [23]. The authors derived adaptive offloading decision algorithms when taking advantage of Lyapunov optimization techniques. The algorithm determined when and on which network, and where to perform each application task (i.e., IoT device, edge server, or cloud server) such that the overall energy consumption is minimized while guaranteeing the average queue length.

Many optimization-based algorithms, e.g., traversal or linear programming, can only obtain results after multiple iterations, which often involve too many complex calculation operations, e.g., matrix inversion and singular value decomposition, resulting in high running time cost in offloading decision making. Moreover, these optimization methods that only take advantage of MEC or MCC struggle to balance the complexity and optimality. Thus, it is necessary to develop an algorithm that can be used for real-time offloading decisions with MEC and MCC collaboration.

D. Deep Learning-Based Offloading Decisions

Deep learning is very promising for solving complicated real-world scenarios, e.g., Internet of Vehicles (IoV) [29], UAVs [30], and industrial IoT [31]. Recently, deep learning-driven offloading schemes play an increasingly important role in dealing with task offloading decisions for MEC and/or MCC, i.e., intelligent offloading [27].

A model-free reinforcement learning offloading mechanism was proposed in [25], which uses a gaming framework and reaches 87.87% payoff compared to the optimal condition. In order to solve the offloading decision problem in the MEC environment, a distributed deep learning-based offloading algorithm has been proposed in [26], where parallel computing is utilized to speed up the computation. Apart from that, Min *et al.* [28] proposed a reinforcement learning-based solution to solve the task offload decision of IoT devices with energy-harvesting functions, which enables IoT devices to optimize the offloading strategy without knowing the MEC model, energy consumption model, and delay model.

Many existing deep learning-based offloading schemes, however, optimize all system parameters simultaneously, which will eventually identify infeasible solutions as the optimal offloading decision. Moreover, the heterogeneity of the servers is still ignored in the selection of the offloading destination and the definition of the convergence in these works is not clear. Inspired by recent advantages of deep learning in handling offloading decision problems with large search spaces, we take advantage of parallel computing of DNNs, meanwhile, the convergence of the deep learning-based decision algorithm is clearly defined and improved. In addition, the heterogeneity of servers and devices is also considered in the MEC and MCC environments. Once the IoT environment changes, the use of deep learning methods requires new labeled data, and the offloading decision for complex tasks required by different services should have long-term programming and continuous learning capabilities to meet the requirements of city IoT applications.

E. Qualitative Comparison

As listed in Table I, we identify and compare key elements of related work with ours in terms of their offloading modes, architectural properties, and decision objectives. To summarize, the literature above only concentrates on local devices

TABLE I
COMPARISON OF DIFFERENT OFFLOADING DECISION SCHEMES

Categories	Offloading Schemes	Offloading Mode	Task Number	Architectural Properties			Decision Objectives		
				MCC	MEC	MCC & MEC	Time	Energy	Weighted
Markov-based offloading decisions	Scheme in [6]	Partial	Multiple	×	✓	×	✓	✓	×
	Scheme in [7]	Partial	Multiple	✓	×	×	✓	✓	✓
	Scheme in [8]	Partial	Multiple	✓	×	×	✓	✓	✓
	MCOWA [9]	Partial	Multiple	×	✓	×	✓	✓	×
	Unit-slot [10]	Partial	Single	✓	×	×	✓	×	×
	Scheme in [11]	Partial	Multiple	×	✓	×	✓	✓	×
Graph-based offloading decisions	K-M-LARAC [12]	Partial	Multiple	✓	×	×	✓	✓	×
	Scheme in [13]	Partial	Multiple	×	✓	×	×	✓	×
	Scheme in [14]	Partial	Multiple	✓	×	×	✓	×	×
	One-climb policy [15]	Partial	Multiple	✓	×	×	×	✓	×
	MCOP [16]	Partial	Multiple	✓	×	×	✓	✓	✓
Optimization-based offloading decisions	MAPCloud [17]	Partial	Multiple	✓	×	×	✓	✓	×
	Scheme in [18]	Partial	Single	×	✓	×	×	✓	×
	SDTO [19]	Full	Multiple	×	✓	×	✓	✓	×
	COM [20]	Partial	Multiple	✓	×	×	✓	✓	×
	F-SGA&C-SGA [21]	Partial	Multiple	×	✓	×	✓	×	×
	EEDOA [22]	Full	Multiple	×	✓	×	×	✓	×
	Scheme in [23]	Partial	Multiple	✓	×	×	✓	✓	×
	HGPCA [24]	Full	Multiple	×	✓	×	×	✓	×
Deep learning-based offloading decisions	Scheme in [25]	Partial	Multiple	×	✓	×	×	✓	×
	DDLO [26]	Partial	Multiple	×	✓	×	×	×	✓
	DROO [27]	Partial	Multiple	×	✓	×	×	✓	×
	DRLO [28]	Partial	Multiple	×	✓	×	×	✓	×
	Scheme in [29]	Partial	Multiple	×	✓	×	×	✓	×
	Scheme in [30]	Partial	Multiple	×	✓	×	✓	✓	✓
	MCCG [31]	Partial	Multiple	×	×	✓	✓	✓	×
	Proposed DDTO scheme	Partial	Multiple	✓	✓	✓	✓	✓	✓

TABLE II
COMPARISON OF MCC AND MEC [36]

Technical Aspect	MCC	MEC
Deployment	centralized	distributed
Distance to IoT device	high	low
Latency	high	low
Delay jitter	high	low
Computational power	ample	limited
Storage capacity	ample	limited
Mobility support	ample	limited
Privacy protect	low	high

and edge clouds or ignores the possibly high dimensions of the problem.

In fact, there can be multiple offloading destinations and targets for task placement [34]. Due to the different speeds of heterogeneous cloud servers, offloading the same application to different places may complete a different amount of computation within the same time interval. It may incur different communication costs due to the specific connectivity and cloud availability [35].

As shown in Table II, compared to cloud servers, edge servers are closer to the MDs and thus have lower latency. However, the edge server has low computing power as compared to the cloud server, which has relatively sufficient computing power [36]. Therefore, MEC can be treated as an extension of the traditional MCC, but not an alternative to MCC.

Few recent studies have focused on identifying and addressing important challenges of task offloading in heterogeneous edge and cloud computing environments, where edge clouds and central clouds work collaboratively to satisfy the city IoT application requirements. Here, we consider the heterogeneity of different edge and cloud servers in the offloading destination selection. To jointly optimize the system utility and the bandwidth allocation for each MD, we establish a hybrid offloading model with the collaboration of MCC and MEC. In addition, a distributed deep learning-driven algorithm is proposed to generate optimal offloading decisions for heterogeneous clouds. To the best of our knowledge, this article is the first that adopts deep learning with the collaboration of MCC and MEC for heterogeneous servers.

III. SYSTEM MODEL AND PROBLEM FORMULATION

In this section, we consider a framework of hybrid task offloading with heterogeneous clouds, in which MDs can execute their workflows locally or completely/partially offload them to the central cloud and/or to the edge cloud for execution.

A. System Model

Fig. 1 presents an overview of our system model. We consider one edge cloud, one central cloud, and multiple MDs, where each MD can choose to offload its computation tasks

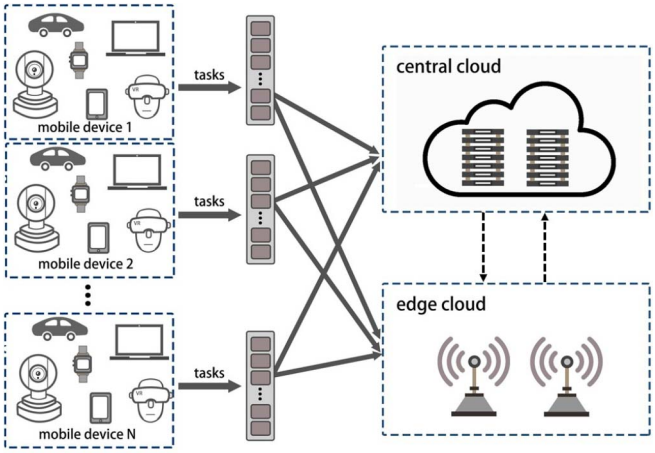


Fig. 1. System model of task offloading with heterogeneous clouds,

either to the edge cloud server or to the central cloud server. We aim at effectively integrating heterogeneous computing resources in the MEC and MCC collaborative computing environment, where the edge cloud and the central cloud can be interconnected. The mobile application is divided into multiple tasks by the application partitioning algorithm. The tasks can then be offloaded to cloud servers. Offloading is performed according to the complexity of the tasks and the present network environment, i.e., offloading the compute-intensive task to the central cloud server and offloading the data-intensive task to the edge cloud server, thereby alleviating the difficulties of load bottlenecks, delays, and ensuring fault tolerance.

The system model consists of one central cloud server, one edge cloud server, one wireless access point (AP), multiple MDs, denoted by a set $\mathcal{N} = \{1, 2, \ldots, N\}$, and some independent computational tasks, denoted by a set $\mathcal{M} = \{1, 2, \ldots, M\}$. We denote the size of the mth task of the nth MD by $w_{(n,m)}$ since each MD has several tasks to cope with. In addition, each MD can either execute their tasks locally or offload them to the cloud servers for further execution. Once the decision is taken that a task will be offloaded to the cloud server, it can be offloaded either to the central cloud server or to the edge cloud server. We define two binary variables named $x^1_{(n,m)}$ and $x^2_{(n,m)}$ to represent the offloading decisions.

On the one hand, $x^1_{(n,m)} \in \{0, 1\}$ stands for the offloading decision for the mth task, which is measured as

$$x^1_{(n,m)} = \begin{cases} 1, \text{ if task is executed on the } n\text{th MD} \\ 0, \text{ if task is offloaded to the cloud server} \end{cases} \tag{1}$$

where $x^1_{(n,m)} = 1$ denotes the mth task is processed locally on the nth MD, and $x^1_{(n,m)} = 0$ indicates the mth task is offloaded to the cloud server.

On the other hand, once the mth task is decided to be offloaded to the cloud server, we further define $x^2_{(n,m)} \in \{0, 1\}$ to represent the offloading destination selection for the mth task, which is measured as

$$x^2_{(n,m)} = \begin{cases} 1, \text{ if offloaded to edge cloud } \& \ x^1_{(n,m)} = 0 \\ 0, \text{ if offloaded to central cloud } \& \ x^1_{(n,m)} = 0 \end{cases} \tag{2}$$

where $x^2_{(n,m)} = 1$ indicates that the mth task is offloaded to the edge cloud server, and $x^2_{(n,m)} = 0$ denotes that the mth task is offloaded to the central cloud server.

For convenience, all parameters used in this article are listed in Table III. The detailed operations of local computing, edge cloud computing, and central cloud computing models are illustrated as follows, respectively.

1) Local Computing Model: We first introduce the local computing model when the MDs decide to execute their tasks locally. Due to limited resources such as battery capacity, MDs can only perform fundamental tasks.

Let $c_{(n,m)}$ denote the total CPU cycles of computing the mth task of the nth MD. Considering that the CPU cycles are proportional to the workloads, which is given by

$$c_{(n,m)} = \delta w_{(n,m)} \tag{3}$$

where δ denotes the positive coefficient of proportionality.

The energy used while executing the mth task at the nth local device can be expressed as

$$El_{(n,m)} = \theta_l c_{(n,m)} = \theta_l \delta w_{(n,m)} \tag{4}$$

where θ_l denotes the energy consumption on the local device per unit of workloads.

The execution time of the local device can be calculated as

$$Tl_{(n,m)} = \frac{c_{(n,m)}}{f_l} = \frac{\delta w_{(n,m)}}{f_l} \tag{5}$$

where f_l denotes the task processing rate of the MDs.

Therefore, the total computation time of the nth MD can be derived as

$$Tl_{(n)} = \sum_{m=1}^{M} x^1_{(n,m)} Tl_{(n,m)}. \tag{6}$$

2) Edge Cloud Computing Model: Edge cloud servers are close to the MDs and communicate with them via different wireless communication technologies, such as Bluetooth or WiFi. Edge cloud servers provide a low-latency computing service to MDs because they form a local area network (LAN) with the MDs.

The transmission time for offloading the workload to the edge cloud server via the AP can be given by

$$Tt_{(n,m)} = \frac{w_{(n,m)}}{b_n} \tag{7}$$

where b_n denotes the bandwidth of the nth MD.

The energy consumption for the transmission can be expressed as

$$Et_{(n,m)} = \sigma w_{(n,m)} \tag{8}$$

where σ denotes the positive coefficient of proportionality.

After transmitting the tasks to the edge cloud, they will be executed by the edge cloud server. The completion delay of the whole progress can be formulated as

$$Te_{(n,m)} = Tt_{(n,m)} + \frac{c_{(n,m)}}{f_e} \tag{9}$$

where f_e indicates the task processing rate of the edge cloud server. Then, the total time delay of the nth MD can be

TABLE III
SUMMARY OF NOTATIONS

Notation	Description
$w_{(n,m)}$	The m^{th} task workload of the n^{th} MD
δ	The positive proportion coefficient
$c_{(n,m)}$	The total CPU cycles of computing the m^{th} task of the n^{th} MD
$x^1_{(n,m)}$	$x^1_{(n,m)} = 1$ if computing the m^{th} task at local, $x^1_{(n,m)} = 0$ if computing the m^{th} task at the cloud
$x^2_{(n,m)}$	When $x^1_{(n,m)} = 0$, $x^2_{(n,m)} = 0$ if computing the task at the central cloud, $x^2_{(n,m)} = 1$ if computing the task at the edge cloud
$El_{(n,m)}$	The energy consumption of the m^{th} task of the n^{th} MD
θ_l	The local device energy consumption per unit of workloads
$Tl_{(n,m)}$	The execution time of the n^{th} MD
f_l	The task processing rate of the MD
$Tl_{(n)}$	The total execution time of the n^{th} MD
$Tt_{(n,m)}$	The transmission time of offloading the m^{th} task to the edge cloud server via the access point
b_n	The bandwidth of the n^{th} MD
B	The total available bandwidth of all users
$Te_{(n,m)}$	The time delay of offloading the m^{th} task of the n^{th} MD to the edge cloud server
f_e	The task processing rate of the edge cloud server
$Te_{(n)}$	The total time delay of offloading all the tasks of the n^{th} MD to the edge cloud server
$Ee_{(n,m)}$	The energy consumption of offloading the m^{th} tasks of the n^{th} MD to the edge cloud server
θ_e	The edge cloud energy consumption per unit of workloads
$Tc_{(n,m)}$	The time delay of offloading the m^{th} task of the n^{th} MD to the central cloud server
f_c	The task processing rate of the central cloud server
$Ec_{(n,m)}$	The energy consumption of offloading the m^{th} task of the n^{th} MD to the central cloud server
θ_c	The central cloud energy consumption per unit of workloads
$Tc_{(n)}$	The total time delay of offloading all the tasks of the n^{th} MD to the central cloud server

formulated as

$$Te_{(n)} = \sum_{m=1}^{M}\left(1 - x^1_{(n,m)}\right)x^2_{(n,m)}Te_{(n,m)}. \tag{10}$$

The energy consumption during all steps can be computed as

$$Ee_{(n,m)} = Et_{(n,m)} + \theta_e c_{(n,m)} \tag{11}$$

where θ_e denotes the energy consumption per unit of workload at the edge cloud server.

3) Central Cloud Computing Model: Central cloud servers can provide the most powerful computing capacity and can be a private cloud or public cloud offered by cloud service providers.

Similarly to the edge cloud computing model, we assume that the transmission time and the energy consumption from the local device to the central cloud server are approximately equal to $Tt_{(n,m)}$ and $Et_{(n,m)}$, respectively. Then, the total execution time and the energy consumption can be given by

$$Tc_{(n,m)} = Tt_{(n,m)} + \frac{c_{(n,m)}}{f_c} \tag{12}$$

$$Ec_{(n,m)} = Et_{(n,m)} + \theta_c c_{(n,m)} \tag{13}$$

where θ_c denotes the energy consumption per unit of workload of the central cloud. f_c denotes the task processing rate of the central cloud server.

In general, the computing power of MDs, edge cloud server, and cloud server satisfy the following: $f_l < f_e < f_c$, which means that the central cloud server has the strongest computing power, followed by the edge cloud server, and then the MDs [37].

Therefore, the total execution time of the nth MD can be derived as

$$Tc_{(n)} = \sum_{m=1}^{M}\left(1 - x^1_{(n,m)}\right)\left(1 - x^2_{(n,m)}\right)Tc_{(n,m)}. \tag{14}$$

B. Problem Formulation

In order to minimize both, the execution time of all the tasks and the energy consumption of MDs, we introduce a function $Q(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{b})$, which is the weighted sum of the execution time and the energy consumption. The weighted sum is related to the workload, the offloading decision, and the bandwidth allocated to the task.

The total energy consumption consumed in the whole hybrid offloading model can be expressed by

$$E = \sum_{n=1}^{N}\sum_{m=1}^{M}\Big[x^1_{(n,m)}El_{(n,m)} + \left(1 - x^1_{(n,m)}\right) \times \left(x^2_{(n,m)}Ee_{(n,m)} + \left(1 - x^2_{(n,m)}\right)Ec_{(n,m)}\right)\Big]. \tag{15}$$

Meanwhile, the total execution time required to execute all the tasks can be given by

$$T = \sum_{n=1}^{N}\max\left\{Tl_{(n,m)}, Te_{(n,m)}, Tc_{(n,m)}\right\}. \tag{16}$$

Then, the function $Q(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{b})$ can be calculated as

$$Q(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{b}) = \psi \times E + (1 - \psi) \times T \tag{17}$$

where $\boldsymbol{w} = \{w_{(n,m)} | n \in \mathcal{N}, m \in \mathcal{M}\}$, $\boldsymbol{b} = \{b_n | n \in \mathcal{N}\}$, and $\boldsymbol{x} = \{x^1_{(n,m)}, x^2_{(n,m)} | n \in \mathcal{N}, m \in \mathcal{M}\}$. The parameter ψ with $0 \leq \psi \leq 1$ is a weighting parameter that represents the relative significance of the energy consumption and the

execution time, by which the weighted cost model can be adjusted according to the users' requirements. To focus more on improving the performance, ψ should be less than 0.5; to focus more on reducing the energy consumption, ψ should be greater than 0.5. We only consider the execution time in the case $\psi = 0$, and we only consider the energy consumption at MDs when $\psi = 1$.

Next, we formulate an optimization problem (P) to minimize $Q(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{b})$ by jointly optimizing offloading decisions and bandwidth allocation, which is expressed as

$$(P): Q(\boldsymbol{w}) = \min_{\boldsymbol{x},\boldsymbol{b}} \quad Q(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{b}) \tag{18a}$$

$$\text{s.t.:} \quad b_n \geq 0 \quad \forall n \in \mathcal{N} \tag{18b}$$

$$\sum_{n=1}^{N} b_n \leq B \tag{18c}$$

$$x^1_{(n,m)}, \quad x^2_{(n,m)} \in \{0, 1\} \tag{18d}$$

where B denotes the total available bandwidth of N users. The constraint in (18b) indicates that the allocation of the bandwidth should not be negative. In addition, the sum of b_n cannot exceed the maximum bandwidth B, which is given in (18c). The binary offloading decisions $x^1_{(n,m)}$ and $x^2_{(n,m)}$ are defined in (1) and (2), respectively. Studies on efficiently solving the bandwidth allocation problem have been shown in [38] and [39], where the bandwidth allocation is a convex problem that can be solved by an optimizer. Here, we just consider the given workloads $\boldsymbol{w}$ and the offloading decision $\boldsymbol{x}$ to optimize the function $Q(\boldsymbol{w}, \boldsymbol{x})$.

This is a mixed-integer programming (MIP) problem with high-dimensional state space. In order to tackle such a complex problem, one needs to find an optimal offloading decision in MEC and MCC heterogeneous environments. In this problem, there are a total of 3^{NM} possible offloading decisions to select from. Due to the exponentially large search space, the optimization problem is difficult to be solved in conventional ways such as with heuristic search algorithms. To solve the problem (P) in an effective and efficient way, we will, in the next section, introduce a deep learning-driven algorithm to generate offloading decisions.

IV. DDTO Algorithm

In this section, we propose a DDTO algorithm for the MCC and MEC hybrid offloading model, which is based on multiple parallel DNNs. The architecture of the DDTO algorithm is as depicted in Fig. 2.

When all users' task workloads are given by $\boldsymbol{w} = [w_{(1,1)}, w_{(1,2)}, \ldots, w_{(N,M)}]$, our target is to figure out the optimal offloading decision $\boldsymbol{x} = [x^1_{(1,1)}, x^2_{(1,2)}, x^1_{(1,2)}, \ldots, x^1_{(N,M)}, x^2_{(N,M)}]$. We assume that all users have the same number of tasks because the application can be divided into multiple tasks and the workloads of extra tasks can be treated as zeros. Furthermore, since we cannot get the optimal decisions directly, it is an unsupervised learning problem that is difficult to solve. Therefore, we propose a method to obtain the offloading decisions and turn it into a supervised learning problem.

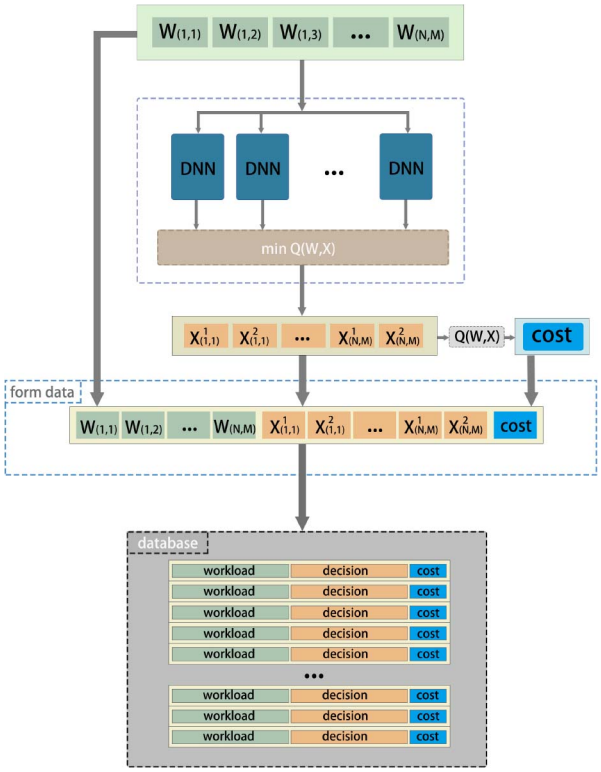


Fig. 2. Procedure of the DDTO algorithm.

We regard the workloads $\boldsymbol{w}$ as the input to the neural networks and the optimal offloading decisions $\boldsymbol{x}$ as our output. Importantly, we store $\boldsymbol{w}$, the best decisions $\boldsymbol{x}$, and the minimum value of the function $Q(\boldsymbol{w}, \boldsymbol{x})$ into a database together. We then use these labeled data to train our multiple parallel DNNs and generate new data to replace the old data in the database. Thus, we can update the database and train the DNNs to solve the NP-hard problem well.

A. Offloading Decision Generation

In this section, we propose a method to obtain the approximate optimal offloading decisions. The mean-square-error (MSE) function is applied to obtain the optimal offloading decision by minimizing the loss function in deep learning. The MSE function is formulated by

$$\text{MSE} = \frac{1}{n} \sum_{t=1}^{n} \left| \text{logits}_t - \text{outputs}_t \right|^2 \tag{19}$$

where *logits* and *outputs* denote the label and the predicted value, respectively. As each element of the decision is binary, the logits can only be 0 or 1. It is straight forward to prove that if the output is larger than $1/2$, the logits will be 1, otherwise 0. In this way, the MSE function is minimized, which means the precision of the model is the highest.

As depicted in Fig. 3, the generation process of the offloading decisions can be expressed as follows: when the inputs $\boldsymbol{w}$ are given, we first use the DNNs to get the outputs. Then, we use the method described above to generate the offloading decision as our logits. This is how we create the labeled database and then the problem can be solved by deep learning.

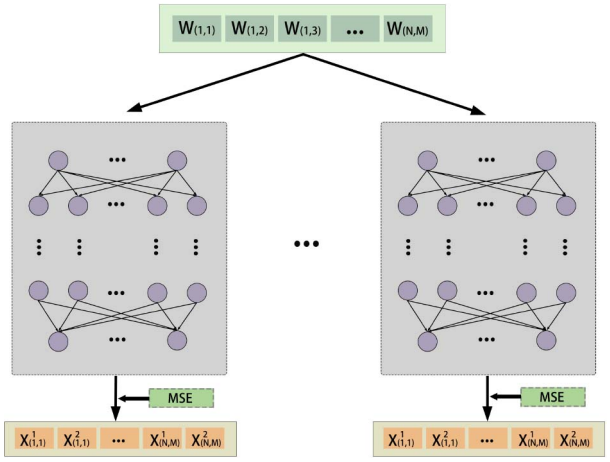


Fig. 3. Process of generating the offloading decisions.

B. Training

We decide to train S parallel DNNs to solve the optimization problem. Each of the DNNs consists of one input layer, two hidden layers, and one output layer.

Once the best offloading decision $\boldsymbol{x}^*$ is obtained, we save the workloads $\boldsymbol{w}$, the best offloading decision $\boldsymbol{x}^*$, and the value of the function $Q(\boldsymbol{w}, \boldsymbol{x}^*)$ together in a database. The size of the database is limited and can be set to an arbitrary value. The database works in a round-robin fashion, i.e., when the database is full, the oldest data will be abandoned and new data will replace the old one. In addition, the labeled data from the structure can be used to train all DNNs. One issue we considered here is that it will take too long if all DNNs are trained by all labeled data from the database. Therefore, relay technology is used in this part [40]. More specifically, the database is shared by all DNNs and each of them can extract a batch of data randomly from the database to train the neural network. As the database is constantly updated and the newly generated data will be more precise than the older one, the efficiency will be improved by this database.

This is a classification problem, we determine to perform the cross-entropy as the loss function, which is given by

$$L(\lambda_k) = -\boldsymbol{x}^T \log f_{\lambda_k}(\boldsymbol{w}) - (1-\boldsymbol{x})^T \log\big(1 - f_{\lambda_k}(\boldsymbol{w})\big) \quad (20)$$

where λ_k is the parameter value of the DNN. We employ the gradient descent method to minimize the cross-entropy loss, and then update the parameters of all DNNs.

C. Testing

We generate offloading decisions as our logits and update the database structure continuously, thus we define the convergence as the process of approaching toward a defined value, i.e., the extremum. More specifically, we denote the minimum value of the function $Q(\boldsymbol{w}, \boldsymbol{x})$ during the process of generating offloading decisions as Q_1. When we randomly select a batch of data from the database and repeat the previous procedure, we obtain another optimal value of $Q(\boldsymbol{w}, \boldsymbol{x})$, denoted as Q_2. Then, the ratio R_1 can be formulated as

$$R_1 = \frac{\min(Q_1, Q_2)}{\max(Q_1, Q_2)} \quad (21)$$

Algorithm 1: DDTO Algorithm

Input: Workloads $\boldsymbol{w}$ of local MDs
Output: Optimal offloading decisions

Initialization:
  Initialize S DNNs with random parameter λ_j
  Empty the database
for $i = 1, 2, \ldots, N$ **do**
  Replicate i^{th} offloading decision candidate $\boldsymbol{x}_i$ from the i^{th} DNN
  Select the optimal offloading decision $\boldsymbol{x}^*$ by minimizing $Q(\boldsymbol{w}, \boldsymbol{x})$ and calculate $Q(\boldsymbol{w}, \boldsymbol{x^*})$ as Q^*
  if *database is not full* **then**
    Store (w_i, x^*, Q^*) into the database
  else
    Discard the oldest data and save the new one
  end
end
for $j = 1, 2, \ldots, S$ **do**
  Randomly choose a batch of data from database
  Train the DNNs and update the parameter λ_j
end

where the ratio R_1 can be interpreted as the convergence of the DDTO algorithm.

We cannot treat the extremum as the true minimum or maximum value, so we decide to enumerate all cases to find the true minimum value of $Q(\boldsymbol{w}, \boldsymbol{x})$ expressed as Q_1^* and compare the result with our optimal value Q_2^*. We find Q_1^* by using a time-consuming greedy algorithm, where we enumerate all offloading decision combinations and identify the true optimal one. Then, we compute a ratio of the minimum value to the optimal value. We defined it as R_2, which can be given by

$$R_2 = \frac{Q_1^*}{Q_2^*} \quad (22)$$

where $0 < R_2 \leq 1$ indicates how close the solution found by our algorithm comes to the true optimal solution achieved by the greedy algorithm. When $R_2 = 1$, it means that we have found the true optimal solution and we call it *relative optimality*.

The whole progress of the DDTO algorithm for the MCC and MEC hybrid offloading model is displayed in Algorithm 1. The database structure is initially empty and multiple DNNs are initialized with random parameter values λ_k. The proposed DDTO framework learns from the past offloading experiences in MEC and MCC heterogeneous environments and then automatically adjusts the parameters to generate near-optimal offloading decisions. In this way, it eliminates the need for solving complex MIP problems and then avoids the curse of dimensionality with a high-dimensional search space. The DDTO algorithm only needs to choose from a few candidate offloading decisions each time and thus the computational complexity will not increase dramatically with the growth in the numbers of users and tasks. Good convergence performance can be achieved because of the high diversity in the generated offloading decisions.

TABLE IV
EVALUATION PARAMETERS

Evaluation Parameters	Values
The number of users or mobile devices	$N = 3$
The number of independent computational tasks	$M = 3$
The local energy consumption per unit	$\theta_l = 3$ J/MB
The edge cloud energy consumption per unit	$\theta_e = 1.5$ J/MB
The central cloud energy consumption per unit	$\theta_c = 1$ J/MB
The processing rate of the local device	$f_l = 100$ MHz
The processing rate of the edge cloud server	$f_e = 800$ MHz
The processing rate of the central cloud server	$f_c = 1200$ MHz
The input workloads of all tasks	$[0 - 30]$ MB
The bandwidth limit	$b_n = 50$ Mbps
The weighting parameter of time and energy	$\psi = 0.5$

V. PERFORMANCE EVALUATION

In this section, we demonstrate the experimental results of our proposed DDTO algorithm for solving the problem (P) and evaluate the performance of different offloading strategies.

A. Parameter Setting

In our experiments, the proposed DDTO algorithm and other offloading decision algorithms are implemented and evaluated in Python using the ML library Tensorflow. An edge and cloud computing heterogeneous environment is built. We set the number of users or MDs $N = 3$, and each of them has $M = 3$ independent computation tasks at the same time.

Apart from that, we set the parameter $\theta_c = 1$, the central cloud energy consumption per unit of workloads, since the energy consumption at the central cloud server is the lowest. Then, we set the edge cloud energy consumption per unit of workloads $\theta_e = 1.5$ J/MB and the local energy consumption per unit of workloads $\theta_l = 3$ J/MB, respectively [41].

There are two kinds of energy profilers that can be used to estimate the energy consumption of MDs, namely, software and hardware monitors. Although the measurement results provided by the former are not as accurate as those provided by the latter, they are more convenient to use and the result is still reasonable [16]. Similarly, we set the task processing rate $f_l = 1$ and the processing rates of the edge cloud server and the central cloud server are $f_e = 800$ MHz and $f_c = 1200$ MHz, respectively. The parameters satisfy: $f_l < f_e < f_c$. In addition, we suppose that the input workloads of all tasks are randomly distributed between 0 and 30 MB. In all simulations, the set value of the bandwidth limit b_n of each user is 50 Mb/s. The weighting parameter ψ is set to 0.5, indicating that our focus is on both, balancing performance and reducing power consumption. We train DNNs using batches of size 500 of the labeled data from the database. The summary of our evaluation parameters and their respective values are presented in Table IV.

B. Convergence Performance

We demonstrate the convergence of the DDTO algorithm in distinct situations, where it converges to the optimal solution under a wide range of parameter settings. The convergence performance of the DDTO algorithm is analyzed on the basis of the number of DNNs, the size of the database, and the learning rate, respectively.

We observe from Fig. 4(a) that R_1 converges to 1 as the learning step increases when $S \geq 2$. According to the definition of R_1, when $S = 1$, the neural network cannot be trained very well and the result will not converge. However, when $S \geq 2$, the convergence performs very well after 10 000 learning steps. It illustrates that as the number of DNNs increases, the function of the DDTO algorithm will be improved. However, the computation time will also increase. Therefore, we set the number of DNNs to $S = 6$ as a compromise between the best configurations to optimize the time consumption of running the code and the performance of the convergence.

As depicted in Fig. 4(b), the ratio R_1 increases with the learning step. Importantly, when the size of the database equals 1400, the convergence performs best. This is because the data will be updated at a low rate when the size of the database is too large. Meanwhile, the data that are randomly selected from the database will not be acceptable when it is too small. Therefore, the size of the database has a considerable influence on the gain ratio because it alters the speed of updates.

As shown in Fig. 4(c), the best performance is achieved when $\alpha_0 = 0.01$. We analyze that when α_0 is too small, the convergence rate is low. Simultaneously, when α_0 is too large, it will converge to another extremum, thus the ratio R_1 will be very low. Therefore, the learning rate α_0 also has an influence on the convergence.

C. Performance of the Relative Optimality R_2

We demonstrate that the relative optimality, the ratio R_2, is affected by the number of DNNs, the learning rate, and the size of the database, respectively.

Fig. 5(a) depicts the impact of the number of DNNs on the relative optimality R_2. It can be seen that the value of R_2 is increasing when the DNN number increases and the value of R_2 reaches its maximum value approximately when the number of DNNs is larger than 5. We selected $S = 6$ before, thus the figure indicates that this was a good choice.

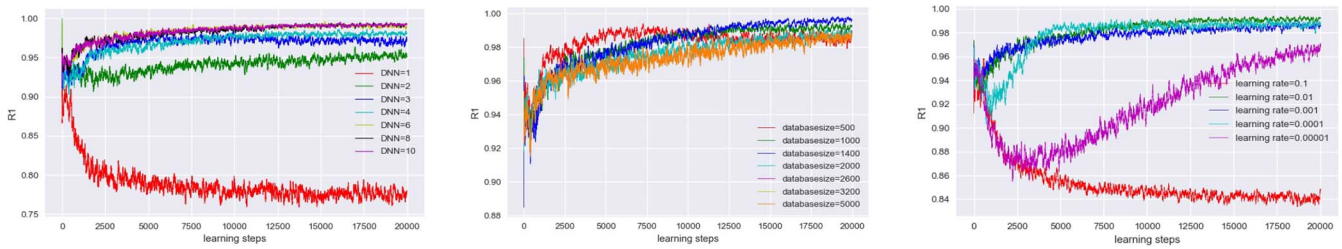
Fig. 5(b) shows the performance of the relative optimality R_2 under different sizes of the database. The value of R_2 is fluctuating when the size is increasing. It is visible that when the size of the database reaches 1400, R_2 gets the vertex, which is greater than 0.925. We previously chose 1400 as the size of the database, and this figure supports this selection.

Fig. 5(c) shows the performance of the relative optimality R_2 with different learning rates. The value of R_2 is up to maximum when the learning rate is 0.01. It indicates that the previous choice is the optimal one since when the learning rate is larger or smaller, R_2 is decreasing.

Fig. 5(a)–(c) demonstrates that the former selection of the DNN number, the size of the database, and the learning rate are all reasonable. In addition, the value of R_2 exceeds 0.92 under our choice.

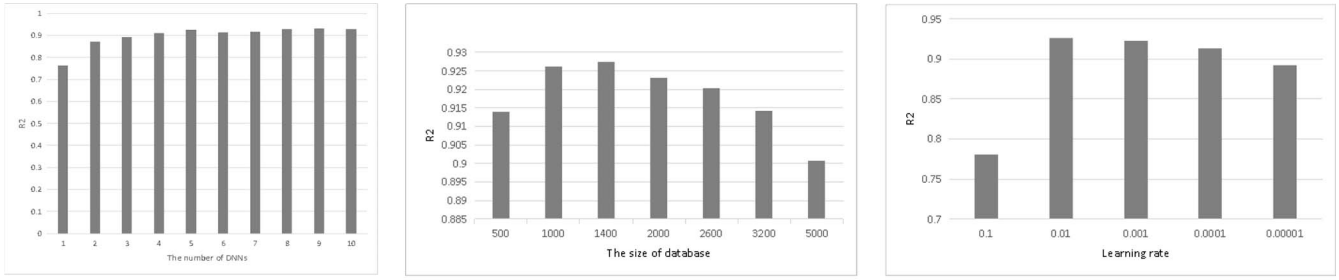
D. Comparison Analysis

To gain some insights and analyze the efficiency of the proposed DDTO algorithm, the following state-of-the-art offloading decision methods are implemented for comparisons.



(a) (b) (c)

Fig. 4. Impact of the (a) number of DNNs, (b) size of the database, and (c) learning rate on R_1.



(a) (b) (c)

Fig. 5. Impact of the (a) number of DNNs, (b) size of the database, and (c) learning rate on R_2.

1) *Local-Only Scheme (i.e., Zero Offloading Scheme):* In this method, all tasks of workflows are executed locally on their respective MDs, and hence, no parallel execution of tasks can be performed for workflows. Here, the offloading decisions $x^1_{(n,m)}$ will be 1. The results of this method can be used as a benchmark to analyze the gain of different types of task offloading techniques.
2) *Edge-Only Scheme (i.e., Edge Cloud-Only Offloading Scheme):* In this method, all tasks of workflows are fully offloaded to the edge cloud server for execution [42]. The offloading decisions $x^1_{(n,m)}$ and $x^2_{(n,m)}$ will be 0 and 1, respectively.
3) *Central-Only Scheme (i.e., Central Cloud-Only Offloading Scheme:)* In this method, all tasks of workflows are fully offloaded to the central cloud server for further processing [23]. The offloading decisions $x^1_{(n,m)}$ and $x^2_{(n,m)}$ will be 0 and 0, respectively.
4) *Local and Central Scheme (i.e., Local Execution and Central Cloud Partial Offloading Scheme):* In this method, some tasks of workflows are processed locally on the MDs, while some of them are offloaded to the central cloud server for further processing [26].
5) *Our Algorithm (i.e., Proposed DDTO Scheme:)* In this method, we adopt the proposed DDTO algorithm to generate optimal offloading decisions over the MDs, the edge cloud server, and the central cloud server.

The comparison of the results of different offloading schemes is shown in Fig. 6. It can be seen that the optimal decisions of the proposed DDTO algorithm perform very well since the relative optimality, the ratio R_2, of the *DDTO* scheme exceeds 0.93, which is much higher than for any of the other four schemes. For example, the R_2-value of the *local-only* scheme is only about 0.3, and the *local and central* scheme

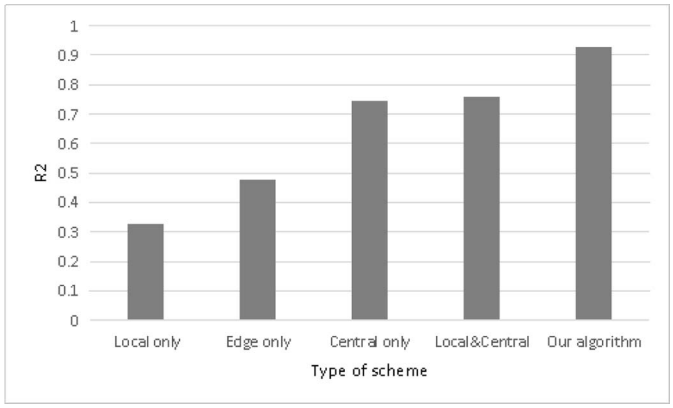


Fig. 6. Comparison with several offloading schemes.

is approximately 0.75. This is because unlike the *edge-only* and the *central-only* schemes, the *DDTO* scheme dynamically offloads tasks according to the heterogeneous computing environment, such as task workloads, communication data, and network conditions. Especially when the network bandwidth is very low, offloading tasks to the edge/cloud server may not be beneficial. Therefore, the proposed DDTO scheme can achieve near-optimal offloading decisions in edge and cloud computing heterogeneous environments.

VI. Conclusion

In contrast to conventional distributed deep learning approaches, we have proposed a DDTO algorithm with heterogeneous clouds, i.e., the central cloud and the edge cloud, which optimizes the weighted sum of the energy consumption and the execution time in the MCC and MEC hybrid offloading model. This is achieved by generating and storing

the offloading decisions with the workloads and system consumption together in a database and then training and updating multiple parallel DNNs with a batch of labeled data.

The DDTO algorithm determines whether a task should be executed at a local device or whether it should be offloaded to the clouds, and if it should be offloaded to the clouds the algorithm determines, whether to offload the task to the central cloud or to the edge cloud. The numerical results demonstrate the accuracy of the DDTO algorithm and in comparison with several previously known schemes, our results are significantly better. In future studies, we will consider more factors in the hybrid offloading model to further improve the capability of our algorithm in handling realistic mobile offloading scenarios. Moreover, we will build a platform that can evaluate the performance of the DDTO algorithm during the actual task offloading progress.

Offloading decisions in MEC and MCC are becoming more intelligent with the emergence of innovative technologies and paradigms, such as fog-aided wireless networks, blockchain, and artificial intelligence [43], [44]. To meet more stringent requirements for security and environmental adaptability, we plan to use the blockchain and metalearning techniques for intelligent offloading in the future.

In view of the single point of failure, data privacy and security problems faced by the current centralized IoT systems, we will develop a blockchain-based decentralized offloading scheme, to address the challenge of data loss or privacy disclosure that may occur in the process of task offloading, effectively promote data intelligence across devices, and ensure data integrity. When the environment of the IoT system changes, such as the performance of the edge server or the bandwidth, deep learning-based methods have to train from scratch. To solve the problem of poor portability, we also introduce metalearning to ensure that the offloading decision model can quickly adapt to the new environment by learning the initial parameters of the neural network in a different environment.

REFERENCES

[1] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Netw. Appl.*, vol. 18, no. 1, pp. 129–140, 2013.

[2] R. Mahmud, R. Kotagiri, and R. Buyya, "Fog computing: A taxonomy, survey and future directions," in *Internet of Everything*. Singapore: Springer, 2018, pp. 103–130.

[3] B. Varghese and R. Buyya, "Next generation cloud computing: New trends and research directions," *Future Gener. Comput. Syst.*, vol. 79, pp. 849–861, Feb. 2018.

[4] D. Yu *et al.*, "Implementing abstract MAC layer in dynamic networks," *IEEE Trans. Mobile Comput.*, early access, Feb. 4, 2020, doi: 10.1109/TMC.2020.2971599.

[5] K. Yu, Y. Wang, J. Yu, D. Yu, X. Cheng, and Z. Shan, "Localized and distributed link scheduling algorithms in IoT under Rayleigh fading," *Comput. Netw.*, vol. 151, pp. 232–244, Mar. 2019.

[6] K. R. Alasmari, R. C. Green, and M. Alam, "Mobile edge offloading using Markov decision processes," in *Proc. Int. Conf. Edge Comput.*, 2018, pp. 80–90.

[7] H. Wu, W. Knottenbelt, and K. Wolter, "Analysis of the energy-response time tradeoff for mobile cloud offloading using combined metrics," in *Proc. IEEE 27th Int. Teletraffic Congr. (ITC)*, 2015, pp. 134–142.

[8] H. Wu and K. Wolter, "Stochastic analysis of delayed mobile offloading in heterogeneous networks," *IEEE Trans. Mobile Comput.*, vol. 17, no. 2, pp. 461–474, Feb. 2018.

[9] K. Peng *et al.*, "An energy-and cost-aware computation offloading method for workflow applications in mobile edge computing," *EURASIP J. Wireless Commun. Netw.*, vol. 2019, no. 1, p. 207, Aug. 2019.

[10] Y. Nan, W. Li, W. Bao, F. C. Delicato, P. F. Pires, and A. Y. Zomaya, "A dynamic tradeoff data processing framework for delay-sensitive applications in cloud of things systems," *J. Parallel Distrib. Comput.*, vol. 112, pp. 53–66, Feb. 2018.

[11] Q. Wang, S. Guo, J. Liu, and Y. Yang, "Energy-efficient computation offloading and resource allocation for delay-sensitive mobile edge computing," *Sustain. Comput. Informat. Syst.*, vol. 21, pp. 154–164, Mar. 2019.

[12] V. Haghighi and N. S. Moayedian, "An offloading strategy in mobile cloud computing considering energy and delay constraints," *IEEE Access*, vol. 6, pp. 11849–11861, 2018.

[13] L. Dong, M. N. Satpute, J. Shan, B. Liu, Y. Yu, and T. Yan, "Computation offloading for mobile-edge computing with multi-user," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2019, pp. 841–850.

[14] J. Barrameda and N. Samaan, "A novel statistical cost model and an algorithm for efficient application offloading to clouds," *IEEE Trans. Cloud Comput.*, vol. 6, no. 3, pp. 598–611, Jul.–Sep. 2018.

[15] W. Zhang and Y. Wen, "Energy-efficient task execution for application as a general topology in mobile cloud computing," *IEEE Trans. Cloud Comput.*, vol. 6, no. 3, pp. 708–719, Jul.–Sep. 2018.

[16] H. Wu, W. J. Knottenbelt, and K. Wolter, "An efficient application partitioning algorithm in mobile environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 7, pp. 1464–1480, Jul. 2019.

[17] M. R. Rahimi, N. Venkatasubramanian, S. Mehrotra, and A. V. Vasilakos, "MAPCloud: Mobile applications on an elastic and scalable 2-tier cloud architecture," in *Proc. IEEE/ACM 5th Int. Conf. Utility Cloud Comput.*, 2012, pp. 83–90.

[18] E. El Haber, T. M. Nguyen, D. Ebrahimi, and C. Assi, "Computational cost and energy efficient task offloading in hierarchical edge-clouds," in *Proc. IEEE 29th Annu. Int. Symp. Pers. Indoor Mobile Radio Commun. (PIMRC)*, 2018, pp. 1–6.

[19] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 587–597, Mar. 2018.

[20] X. Xu *et al.*, "A computation offloading method over big data for IoT-enabled cloud-edge computing," *Future Gener. Comput. Syst.*, vol. 95, pp. 522–533, Jun. 2019.

[21] M. Li, Q. Wu, J. Zhu, R. Zheng, and M. Zhang, "A computing offloading game for mobile devices and edge cloud servers," *Wireless Commun. Mobile Comput.*, vol. 2018, pp. 1–10, Dec. 2018.

[22] Y. Chen, N. Zhang, Y. Zhang, X. Chen, W. Wu, and X. S. Shen, "Energy efficient dynamic offloading in mobile edge computing for Internet of Things," *IEEE Trans. Cloud Comput.*, early access, Feb. 11, 2019, doi: 10.1109/TCC.2019.2898657.

[23] H. Wu, Y. Sun, and K. Wolter, "Energy-efficient decision making for mobile cloud offloading," *IEEE Trans. Cloud Comput.*, early access, Jan. 4, 2018, doi: 10.1109/TCC.2018.2789446.

[24] F. Guo, H. Zhang, H. Ji, X. Li, and V. C. M. Leung, "An efficient computation offloading management scheme in the densely deployed small cell networks with mobile edge computing," *IEEE/ACM Trans. Netw.*, vol. 26, no. 6, pp. 2651–2664, Dec. 2018.

[25] T. Q. Dinh, Q. D. La, T. Q. S. Quek, and H. Shin, "Learning for computation offloading in mobile edge computing," *IEEE Trans. Commun.*, vol. 66, no. 12, pp. 6353–6367, Dec. 2018.

[26] L. Huang, X. Feng, A. Feng, Y. Huang, and L. P. Qian, "Distributed deep learning-based offloading for mobile edge computing networks," *Mobile Netw. Appl.*, to be published.

[27] L. Huang, S. Bi, and Y. J. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Trans. Mobile Comput.*, early access, Jul. 24, 2019, doi: 10.1109/TMC.2019.2928811.

[28] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu, and W. Zhuang, "Learning-based computation offloading for IoT devices with energy harvesting," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1930–1941, Feb. 2019.

[29] Z. Ning *et al.*, "Deep reinforcement learning for intelligent Internet of Vehicles: An energy-efficient computational offloading scheme," *IEEE Trans. Cogn. Commun. Netw.*, vol. 5, no. 4, pp. 1060–1072, Dec. 2019.

[30] M.-A. Messous, S.-M. Senouci, H. Sedjelmaci, and S. Cherkaoui, "A game theory based efficient computation offloading in an UAV network," *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 4964–4974, May 2019.

[31] Z. Hong, W. Chen, H. Huang, S. Guo, and Z. Zheng, "Multi-hop cooperative computation offloading for industrial IoT–edge–cloud computing environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 12, pp. 2759–2774, Dec. 2019.
[32] D. Yu, L. Ning, Y. Zou, J. Yu, X. Cheng, and F. C. Lau, "Distributed spanner construction with physical interference: Constant stretch and linear sparseness," *IEEE/ACM Trans. Netw.*, vol. 25, no. 4, pp. 2138–2151, Aug. 2017.
[33] Q. Hua *et al.*, "Faster parallel core maintenance algorithms in dynamic graphs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1287–1300, Jun. 2020.
[34] M. Goudarzi, H. Wu, M. S. Palaniswami, and R. Buyya, "An application placement technique for concurrent IoT applications in edge and fog computing environments," *IEEE Trans. Mobile Comput.*, early access, Jan. 15, 2020, doi: 10.1109/TMC.2020.2967041.
[35] D. Yu *et al.*, "Stable local broadcast in multihop wireless networks under SINR," *IEEE/ACM Trans. Netw.*, vol. 26, no. 3, pp. 1278–1291, Jun. 2018.
[36] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, 4th Quart., 2017.
[37] F. Wang, J. Xu, X. Wang, and S. Cui, "Joint offloading and computing optimization in wireless powered mobile-edge computing systems," *IEEE Trans. Wireless Commun.*, vol. 17, no. 3, pp. 1784–1797, Mar. 2018.
[38] H. Zhang, H. Liu, J. Cheng, and V. C. M. Leung, "Downlink energy efficiency of power allocation and wireless backhaul bandwidth allocation in heterogeneous small cell networks," *IEEE Trans. Commun.*, vol. 66, no. 4, pp. 1705–1716, Apr. 2018.
[39] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Trans. Wireless Commun.*, vol. 16, no. 3, pp. 1397–1411, Mar. 2017.
[40] D. Horgan *et al.*, "Distributed prioritized experience replay," 2018. [Online]. Available: arXiv:1803.00933.
[41] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Trans. Commun.*, vol. 64, no. 10, pp. 4268–4282, Oct. 2016.
[42] Z. Zhang, Z. Hong, W. Chen, Z. Zheng, and X. Chen, "Joint computation offloading and coin loaning for blockchain-empowered mobile-edge computing," *IEEE Internet Things J.*, vol. 6, no. 6, pp. 9934–9950, Dec. 2019.
[43] S. Singh, I. Chana, and R. Buyya, "STAR: SLA-aware autonomic management of cloud resources," *IEEE Trans. Cloud Comput.*, early access, Jan. 5, 2017, doi: 10.1109/TCC.2017.2648788.
[44] S. S. Gill *et al.*, "Transformative effects of IoT, blockchain and artificial intelligence on cloud computing: Evolution, vision, trends and open challenges," *Internet Things*, vol. 8, Dec. 2019, Art. no. 100118.

Huaming Wu (Member, IEEE) received the B.E. and M.S. degrees in electrical engineering from Harbin Institute of Technology, Harbin, China, in 2009 and 2011, respectively, and the Ph.D. degree (Highest Hons.) in computer science from Freie Universität Berlin, Berlin, Germany, in 2015.

He is currently an Associate Professor with the Center for Applied Mathematics, Tianjin University, Tianjin, China. His research interests include model-based evaluation, wireless and mobile network systems, mobile cloud computing, and deep learning.

Ziru Zhang is currently pursuing the bachelor's degree with the School of Mathematics, Tianjin University, Tianjin, China.

His research interests include distributed deep learning and edge computing.

Chang Guan is currently pursuing the bachelor's degree with the School of Mathematics, Tianjin University, Tianjin, China.

His research interests include distributed deep learning and edge computing.

Katinka Wolter (Associate Member, IEEE) received the Ph.D. degree from Technische Universität Berlin, Berlin, Germany, in 1999.

In 2012, she joined Freie Universität Berlin, Berlin, as a Professor for dependable systems. She has been an Assistant Professor with Humboldt-University Berlin, Berlin, and a Lecturer with Newcastle University, Newcastle upon Tyne, U.K. Her research interests are model-based evaluation and improvement of dependability, security, and performance of distributed systems and networks.

Minxian Xu (Member, IEEE) received the B.Sc. and M.Sc. degrees in software engineering from the University of Electronic Science and Technology of China, Chengdu, China, in 2012 and 2015, respectively, and the Ph.D. degree from the University of Melbourne, Melbourne, VIC, Australia, in 2019.

He is currently an Assistant Professor with Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Beijing, China. His research interests include resource scheduling and optimization in cloud computing.