Deep Learning-Assisted Online Task Offloading for Latency Minimization in Heterogeneous Mobile Edge

Yu Liu, Yingling Mao, Zhenhua Liu, and Yuanyuan Yang, *Fellow, IEEE*

Abstract—With the proliferation of smart devices in recent years, many applications requiring high computing capability and low latency have emerged. Edge computing is one of the promising paradigms to support such applications. Due to the high volatility of edge environments, e.g., frequent movements of mobile devices, varying task sizes, and time-variant channel conditions, we have to make the offloading and resource management decisions on the fly. This paper formulates and studies the problem of online task offloading and resource management in heterogeneous mobile edge environments. The goal of the problem is to minimize the overall system latency. We prove that the problem is NP-hard. Moreover, traditional algorithms needing long decision-making times are insufficient to support applications with high volatility. This paper proposes a deep learning-assisted online algorithm that can make fast decisions. In particular, we design an offline solver for the proposed problem and use a deep neural network to emulate the solver. We conduct extensive simulations to evaluate the proposed approach. Results show that the proposed approach is around $50,000\times$ and $500\times$ faster than the commercial Gurobi solver for the optimal solution and the proposed offline approximation solver, respectively. Moreover, the overall latency under the proposed approach is near-optimal.

Index Terms—Online Task Offloading, Mobile Edge Computing, High Volatility

1 Introduction

The number of Internet of Things (IoT) devices has been increasing rapidly over the past years, from 8.5 million in 2019 to 13.1 million (projected value) in 2023 [1]. Enormous data generated from IoT devices coupled with the development of information technologies, such as machine learning, have facilitated many novel applications, e.g., autopilot, object detection, and virtual reality gaming [2]. Such applications require low response latency, e.g., single-digit millisecond latency, extensive data, and high computing power [3]. Cloud computing has enormous benefits in providing high computing capability and low cost. However, the long distances between cloud servers and end-users incur high propagation latency [4], and there is congestion in the core network during peak hours [5]. Therefore, cloud computing is insufficient to support applications requiring low response latency, and there is a push to perform computing near end users. On the other hand, end mobile devices are insufficient for the applications because of their low computing capability and limited battery capacity [6]. Mobile Edge Computing (MEC), drawing lots of attention, is one of the promising computing paradigms to support such applications because edge computing devices are close to end-users and have sufficient computing capability.

- *Y. Liu, Y. Mao, and Y. Yang are with the Department of Electrical and Computer Engineering, Stony Brook University, Stony Brook, NY, 11794. Z. Liu is with the Department of Applied Mathematics and Statistics, Stony Brook University, Stony Brook, NY, 11794. E-mail: {yu.liu.3,yingling.mao, zhenhua.liu, yuanyuan.yang}@stonybrook.edu*
- *This work was supported in part by the National Science Foundation under grant numbers CCF-1730291, CCF-2046444, CNS-2146909, CNS-2106027, and CNS-2214980.*

This paper considers the online task offloading and resource management problem in heterogeneous mobile edge environments. The system runs in discrete time. At the beginning of each time slot, we observe three sets of system states, i.e., input data sizes, task sizes, and channel conditions, and make four decisions: offloading, computing resource management, base station selection, and bandwidth allocation decisions. Since we consider applications requiring low latency, decisions must be made immediately, e.g., in several milliseconds. System states and decisions at each time slot jointly determine the overall latency. Some papers consider the corresponding static problems where the system states are invariant [7], [8], [9], [10], yet they are insufficient for highly volatile systems considered in this paper. The algorithms proposed in these papers do not have polynomial time complexity. Consequently, the system states may have changed before the algorithms finish their computation. In addition, we can hardly accelerate them because they are iterative algorithms. Therefore, an algorithm with extremely low decision-making time is needed to deal with highly volatile systems.

Motivated by [11], we adopt an online deep reinforcement learning method to tackle the proposed problem. The deep reinforcement learning approach uses deep neural networks to make decisions based on the current system states, and the forward passes of these networks can be accelerated through parallel computing. Despite the advantages of making decisions using deep reinforcement learning methods, it is challenging to solve the proposed problem. The problem of collectively offloading tasks, selecting base stations, and allocating bandwidth and computing resources is NP-hard. There are contradictions between the latencies of different devices, and we must make decisions thoughtfully

to minimize the overall latency. In addition, the proposed problem is more complicated than that in the literature, and the algorithm used in the literature cannot solve the proposed problem efficiently (see Section 7.2 for details). Therefore, we must design a novel online reinforcement learning approach to solve the proposed problem.

Our main contributions are listed as follows:

- We formulate the online task offloading and resource management problem in heterogeneous edge environments (*OTORM*) where system states are highly volatile. We prove the formulated problem is NP-hard.
- We decompose the problem into two disjoint subproblems: joint task offloading and computing resource allocation (JCM) problem and joint base station selecting and bandwidth allocation (JPM) problem. By eliminating the computing resource allocation variables in JCM and the bandwidth allocation variables in JPM, we convert JCM and JPM to their equivalent problems, P1 and P2, respectively.
- We design an offline solver for P1 and P2 and prove the solver has an approximation ratio of $2.62/(1-8\lambda)$, where λ is a tunable parameter.
- We design an online deep reinforcement learning approach for the proposed problem. The approach learns from past decisions under different system states. For fast convergence and better performance, the approach leverages both previous decisions made by itself and that of the offline solver.
- We conduct extensive real-world data-driven simulations to evaluate the proposed approach. Simulation results show that the proposed approach outperforms popular baselines. Furthermore, the approach can make decisions swiftly, e.g., around one millisecond on average. In addition, the proposed approach is near-optimal, achieving 1.016 times the optimal latency on average.

The remainder of this paper is organized as follows. Section 2 discusses related works. Section 3 formulates and simplifies the problem. Section 4 analyzes and simplifies the proposed problem. Section 5 proposes the online deep reinforcement approach. Section 6 states the offline approximation solver used by the approach. Section 7 shows the performance evaluation results. Section 8 concludes this paper.

2 Related Works

Some papers considering related offline offloading problems are as follows. In [8], Jošilo *et al.* consider wireless and computing resource allocation for computation offloading in edge computing and design a game theoretic-based algorithm with a constant approximation ratio. In [9], Jošilo *et al.* consider offloading tasks to network slices and allocating wireless and computing resources and propose a game-theoretic-based algorithm similar to the method in [8]. Both [8] and [9] do not consider fronthaul links, where congestion may occur. Moreover, in [10], Yu *et al.* consider the service function chain placement, routing, and resource management problem in network edge and provide a constant factor approximation algorithm using an approach similar to methods in [8], [9]. The algorithms designed in [8], [9], [10] may need exponential time to converge. In addition, the algorithms are iterative-based and can hardly be accelerated. Therefore, these algorithms take a long time to make decisions and cannot handle online problems with high volatility. There are also some papers allowing partial task offloading. In [12], Pavlos *et al.* consider a heterogeneous multi-MEC system considering both the risk-aware behavior of the individual users and the risk of failure of the computing resource and propose a game-theoretic based algorithm with low complexity. In [13], Pavlos *et al.* focus on the problem of risk-aware data offloading in multi-server multi-access and propose a non-cooperative game-theoretic-based distributed low-complexity algorithm.

Some other papers consider online task offloading problems with different settings and goals. In [14], Gang *et al.* design a collaborative task offloading mechanism for mobile edge computing systems with the goal of maximizing social welfare. In [15], Ni *et al.* focus on the caching problem in mobile edge computing systems with stationary system states and design a Lyapunov optimization-based algorithm. In addition, a similar Lyapunov optimization-based approach is used in [16] to minimize the time average cost of mobile edge devices. In [17], Qi *et al.* focus on minimizing energy consumption in mobile edge computing systems and propose a Lyapunov optimization-based algorithm similar to that in [15]. In [18], Zhi *et al.* consider a computation offloading problem in edge computing to maximize the revenue of edge servers. In [19], Ye *et al.* focus on lowering the energy consumption of a small cell base station on/off switching problem and propose a deep reinforcement learning-based algorithm. In [20], Panagiotis *et al.* consider the problem of two unmanned aerial vehicles of different operators allocating energy to users and formulate the problem as a generalized colonel blotto game. In addition, the authors proposed a reinforcement learning algorithm to schedule energy.

A closely related paper is [11], which uses a deep reinforcement learning-based approach to solve online computation offloading problems. However, there are significant differences between [11] and this paper. First, they consider different problems. We consider a system with heterogeneous edge servers, while [11] assumes only one edge server. In addition, [11] does not consider computational resource management, but we take it into consideration. Therefore, the problem considered in this paper is much more complicated than that of [11]. From the perspective of the technical approach, there are also significant differences. The approach in [11] makes decisions after it trains and updates its neural networks, while we make decisions at the beginning of each time slot for low decision-making time. The approach in [11] only learns from the best decision among randomly generated decisions, which may not be effective for more complex problems (as discussed in Section 7). Therefore, we design an offline solver, and our approach takes advantage of the offline solver. Although the approach used in this paper is not a standard reinforcement learning method like Q-learning or Actor-Critic methods, we **follow** [11] and call it a reinforcement learning-based method.

$\mathcal{D} = \{D_1, \cdots, D_I\}$	set of mobile WDs
$\mathcal{B} = \{B_0, \cdots, B_K\}$	set of base stations
$\mathcal{S} = \{S_1, \cdots, S_N\}$	set of edge servers
$d_{i,t}, i \in [I], t \in [T]$	input data size (bit) of D_i at slot t
$\mathbf{d}_t$	set $\{d_{i,t} \mid i \in [I]\}$
$f_{i,t}, i \in [I]$	job size (FLOP) of D_i at time slot t
$\mathbf{f}_t$	set $\{f_{i,t} \mid i \in [I]\}$
$W_k^F, k \in [K]$	fronthaul bandwidth (Hz) of B_k
$W_k^U, k \in [K]$	uplink bandwidth (Hz) of B_k
$F_n, n \in [N]$	computing capability of S_n (FLOP/s)
$\sigma_{i,n}$	suitability between D_i and S_n
$x_{i,k,t}$	base station selection decision
$\mathbf{x}_t$	$\{x_{i,k,t} \mid i \in [I], k \in [K] \cup \{0\}\}$
$\phi_{i,k,t}^U$	uplink resource management decision
$\Phi_t^U, t \in [T]$	$\{\psi_{i,k,t}^U \mid i \in [I], k \in [K] \cup \{0\}\}$
$\phi_{i,k,t}^F$	fronthaul resource management decision
$\Phi_t^F, t \in [T]$	$\{\psi_{i,k,t}^F \mid i \in [I], k \in [K] \cup \{0\}\}$
$\Phi_t, t \in [T]$	$\Phi_t^F \cup \Phi_t^U$
$T_{i,t}^U$ & $T_{i,t}^F$	uplink&fronthaul latency of D_i
$T_{i,t}^C = T_{i,t}^U + T_{i,t}^F$	communication latency of D_i
$y_{i,n,t}$	task offloading decision
$\mathbf{y}_t, t \in [T]$	$\{y_{i,n,t} \mid i \in [I], n \in [N]\}$
$\psi_{i,n,t}$	computing resource management decision
$\Psi_t, t \in [T]$	$\{\psi_{i,n,t} \mid i \in [I], n \in [N]\}$
$T_{i,t}^P$	processing latency of D_i
$h_{i,k,t}^U$	uplink channel condition (bps/Hz)
$\mathbf{h}_t$	$\{h_{i,k,t}^U \mid i \in [I], k \in [K] \cup \{0\}\}$
$\mathbf{h}_t$, $\mathbf{f}_t$ and $\mathbf{d}_t$	system states at time slot t

TABLE 1: Important Notations

3 System Model and Problem Formulation

This section describes the heterogeneous mobile edge computing system and formulates the online task offloading and resource management problem (*OTORM*). Some important notations are listed in Table 1. We introduce the edge computing system in Section 3.1 and formulate the *OTORM* problem in Section 3.2.

3.1 System Model

We consider a mobile edge computing system that operates in slotted time, i.e., $t \in \{1, 2, \cdots\}$. The mobile edge computing system consists of edge computing servers, base stations, and mobile Wireless Devices (WDs). A network topology diagram of the mobile edge computing system is shown in Figure 1.

There are N edge servers in the system, and we use $\mathcal{S} = \{S_1, S_2, \cdots, S_N\}$ to denote the collection of all edge servers. Since the cost of edge data centers per scale decreases as the scale increases, we assume that there is one edge server room for cost efficiency. The model proposed in this paper can handle the case that there are multiple edge server rooms. There are $K + 1$ base stations, and the collection of all base stations is denoted by

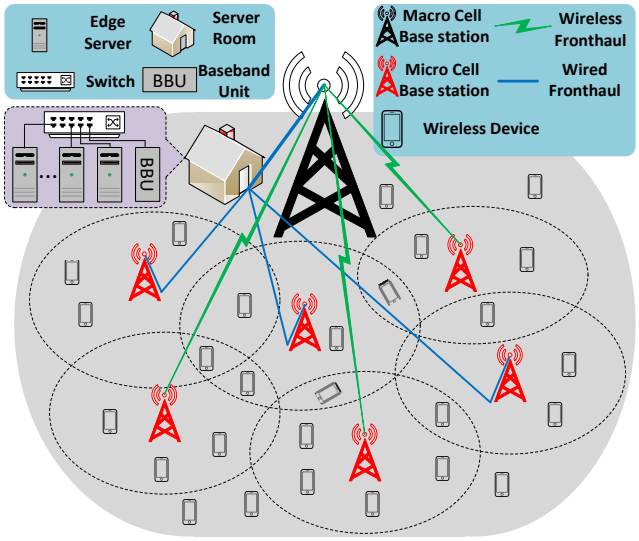


Fig. 1: An example of the mobile edge computing system. The macro base station covers the gray area. The dashed ellipses are the areas covered by the micro base stations. Some mobile devices can connect to more than one micro base station, while others may not be covered by any micro base stations. Some micro base stations use wired fronthaul links, while others have wireless fronthaul connections. Servers in the edge server room are heterogeneous.

$\mathcal{B} = \{B_0, B_1, B_2, \cdots, B_K\}$. In particular, B_0 is a macro cell base station, and $\{B_1, B_2, \cdots, B_K\}$ are micro cell base stations. The macro base station uses a relatively low frequency compared with micro base stations, e.g., 700 MHz for macro base stations and 3.5 GHz micro base stations [21], where low-frequency signals attenuate slower than high-frequency signals and can go through barriers like walls and windows. Typically, a macro base station can cover a few kilometers, and a micro base station covers a range of around a hundred meters. There are I wireless devices (WDs) in the system, and we use $\mathcal{D} = \{D_1, D_2, \cdots, D_I\}$ to represent the set of all WDs. All WDs can access the macro base station B_0, while each micro base station only covers a subset of $\mathcal{D}$. Base stations and WDs are located in a two-dimensional space. Since we consider an online system, the locations of WDs are time-varying.

The macro base station B_0 covers an area, e.g., the dashed ellipse in Figure 1, which is the area considered by the system. Each micro base station covers a smaller area, e.g., the dashed ellipses in Figure 1. At time slot t, wireless device D_i can access to base station B_k if it is within the coverage area of B_k. The macro base station connects to an edge server room via optical fiber, where the baseband unit (BBU) locates. The connections between base stations and the edge server room are so-called fronthaul links [22]. There are two types of fronthaul links, namely wireless fronthaul and wired fronthaul [23]. Wireless fronthaul links can be millimeter wave (mmWave) radio or free-space optical communication [24], and wired fronthaul links can be optical fibers or coaxial cables [25]. For each $k \in \{0\} \cup [K]$ [1], we use W_k^F to denote the bandwidth of

1. For any positive integer n, $[n]$ represents set $\{1, 2, \cdots, n\}$.

the fronthaul link between base station B_k and the edge server room. In addition, there is a link for each base station communicating with mobile wireless devices for uploading tasks' input data. To distinguish it from the fronthaul link, we refer to links between mobile wireless devices and base stations as uplinks. In particular, we use W_k^U to denote the uplink bandwidth of base station B_k.

At the beginning of each time slot t, D_i generates a task with an input data size of $d_{i,t}$ bits, and it takes $f_{i,t}$ floating-point operations (FLOPs) to complete the task. There are N edge servers in the edge server room. Edge server $n \in [N]$ has a computing capability denoted by F_n (FLOP per second). The edge servers are equipped with different computing devices such as CPU and GPU. Since the edge servers are heterogeneous, each edge server is more suitable for running tasks of specific WDs. For each $i \in [I]$ and $n \in [N]$, there is a suitability parameter $\sigma_{i,n} \in [0,1]$ for wireless device D_i and edge server S_n, where the more significant $\sigma_{i,n}$ is better. The approach proposed in this paper can handle the case that $\sigma_{i,n}$ varies over time. The suitability parameter is widely adopted in the literature, e.g., [8], [9], [10].

3.2 Problem Formulation

In this section, we formulate the *OTORM* problem. The goal of the system is to minimize the overall latency, where there are two types of latency, namely communication latency and processing latency.

3.2.1 Communication Latency

At the beginning of each time slot t, the system controller has to select a base station for each wireless device $D_i \in \mathcal{D}$. We use $x_{i,k,t} \in \{0,1\}$ to represent whether wireless device D_i chooses base station B_k for offloading its task at time slot t. For tasks that can be split and executed on multiple servers, we consider each task as multiple independent fine-grained tasks. Each WD has to select exactly one base station at each time slot. Therefore, we have the constraint as follows:

$$\sum_{k=0}^{K} x_{i,k,t} = 1, \quad i \in [I], t \in [T]. \tag{1}$$

Let $\mathbf{x}_t$ be the set of base station selection variables at time slot t, i.e., $\mathbf{x}_t = \{x_{i,k,t} | i \in [I], k \in [K] \cup \{0\}\}$. In addition, we use variable $\phi_{i,k,t}^U \in [0,1]$ to represent the proportion of base station B_k's uplink bandwidth resource allocated to D_i. For example, the system may use the orthogonal frequency division multiplexing (OFDMA) multi-access technique, and allocating bandwidth resources corresponds to allocating resource units. Since the total bandwidth resource that B_k allocates to WDs can not exceed W_k^U, the total uplink bandwidth of B_k, we have the following constraint:

$$\sum_{i=1}^{I} \phi_{i,k,t}^U \leq 1, \quad k \in [K] \cup \{0\}, t \in [T]. \tag{2}$$

We use Φ_t^U to denote the set of all uplink bandwidth management variables at slot t, i.e., $\Phi_t^U = \{\phi_{i,k,t}^U | i \in [I], k \in [K] \cup \{0\}\}$. Similarly, the fronthaul links of base stations are shared by WDs, and we use variable $\phi_{i,k,t}^F \in [0,1]$ to represent the proportion of B_k's fronthaul bandwidth allocated to D_i. We have the following constraint:

$$\sum_{i=1}^{I} \phi_{i,k,t}^F \leq 1 \quad k \in [K] \cup \{0\}, t \in [T]. \tag{3}$$

Let Φ_t^F be the collection of fronthaul link bandwidth management variables at slot t, i.e., $\Phi_t^F = \{\phi_{i,k,t}^F | i \in [I], k \in [K] \cup \{0\}\}$. Moreover, we use Φ_t to denote the collection of all bandwidth management decisions, i.e., $\Phi_t = \Phi_t^F \cup \Phi_t^U$.

The latency experienced by D_i over the uplink is as follows:

$$T_{i,t}^U = \sum_{k=0}^{K} x_{i,k,t} \frac{d_{i,t}}{W_k^U \phi_{i,k,t}^U h_{i,k,t}^U}, \tag{4}$$

where $h_{i,k,t}^U$ is the achievable data rate per Hz (bit/(s·Hz)) of the uplink channel between B_k and D_i at time slot t. $h_{i,k,t}^U$ is a function of the locations of WDs, and some real-world data of achievable data rates can be found in [26]. Note that $h_{i,k,t}^U = 0$ if the location D_i is not in the coverage area of B_k. We use $\mathbf{h}_t$ to denote the collection of uplink channel conditions at time slot t, i.e., $\mathbf{h}_t = \{h_{i,k,t}^U | i \in [I], k \in [K] \cup \{0\}\}$. $\mathbf{h}_t$ varies over time, and we observe it at the beginning of each time slot. Similar to (4) for uplink latency, the latency experienced by D_i over the fronthaul link is as follows:

$$T_{i,t}^F = \sum_{k=0}^{K} x_{i,k,t} \frac{d_{i,t}}{W_k^F \phi_{i,k,t}^F h_k^F}, \tag{5}$$

where h_k^F is the achievable data rate per Hz (bit/(s·Hz)) of the fronthaul link of B_k. Parameter h_k^F is fixed and known. Then, the communication latency of D_i at time slot t, denoted by $T_{i,t}^C$, is as follows:

$$T_{i,t}^C = \sum_{k=0}^{K} x_{i,k,t} \frac{d_{i,t}}{W_k^U \phi_{i,k,t}^U h_{i,k,t}^U} + \sum_{k=0}^{K} x_{i,k,t} \frac{d_{i,t}}{W_k^F \phi_{i,k,t}^F h_k^F}. \tag{6}$$

Note that there is an abuse of notation in (4), (5), and (6), where we let $x_{i,k,t} \cdot v = 0$ if $x_{i,k,t} = 0$ regardless of what v is. Then, the overall communication latency of the system at time slot t is a function of $\mathbf{x}_t$ and Φ_t as follows:

$$T_t^C(\mathbf{x}_t, \Phi_t) = \sum_{i=1}^{I} T_{i,t}^C(\mathbf{x}_t, \Phi_t). \tag{7}$$

In our formulation, we assume that the download time of processed outcomes is negligible because the output data are usually small and the download link is powerful. If the downloading time is significant and cannot be ignored, we can add a download time term to (6), and the proposed algorithm can handle the new formulation.

3.2.2 Processing Latency

At the beginning of each time slot $t \in [T]$, the system controller chooses an edge server for each WD. We use variable $y_{i,n,t} \in \{0,1\}$ to represent whether wireless device D_i computes its task on server S_n at slot t. A WD can only offload its task to one edge server at each time slot t. Thus, we have the following constraint:

$$\sum_{n=1}^{N} y_{i,n,t} = 1, \quad i \in [I], t \in [T]. \tag{8}$$

We use $\mathbf{y}_t$ to denote the collection of task-offloading variables, i.e., $\mathbf{y}_t = \{y_{i,n,t} | i \in [I], n \in [N]\}$. In addition, we use variable $\psi_{i,n,t}$ to represent the proportion of S_n's computing capability allocated to D_i at time slot t. Similar to (2) and (3), there is a constraint for variable $\psi_{i,n,t}$ as follows:

$$\sum_{i=1}^{I} \psi_{i,n,t} \leq 1, \quad n \in [N], t \in [T]. \tag{9}$$

Let Ψ_t be the collection of computing resource management variables at time slot t, i.e., $\Psi_t \triangleq \{\psi_{i,n,t} | i \in [I], n \in [N]\}$. Then, the processing latency experienced by D_i at time slot t, denoted by $T_{i,t}^P$, is as follows:

$$T_{i,t}^P = \sum_{n=1}^{N} y_{i,n,t} \frac{f_{i,t}}{F_n \psi_{i,n,t} \sigma_{i,n}}. \tag{10}$$

The above formula for computing processing latency is widely adopted in the literature [7], [8], [9], [10], where $f_{i,t}$ is the task size of D_i at slot t, $F_n \psi_{i,n,t}$ is the amount of computing capability allocated to D_i, and $\sigma_{i,n}$ is the suitability of running D_i's task on S_n. Then, similar to (6) and (7), the overall processing latency of the system at time slot t is a function of $\mathbf{y}_t$ and Ψ_t as follows:

$$T_t^P(\mathbf{y}_t, \Psi_t) = \sum_{i=1}^{I} T_{i,t}^P = \sum_{i=1}^{I} \sum_{n=1}^{N} y_{i,n,t} \frac{f_{i,t}}{F_n \psi_{i,n,t} \sigma_{i,n}}. \tag{11}$$

Similar to that in (6), there is an abuse of notation in (11) where we let $y_{i,n,t} \cdot v = 0$ as long as $y_{i,n,t} = 0$.

3.2.3 Mathematical Expression

Next, we state mathematically state the formulated problem as follows.

$$\begin{aligned}
\min_{\substack{\mathbf{x}_t, \mathbf{y}_t, \\ \Phi_t, \Psi_t}} \quad & \sum_t \left(T_t^C(\mathbf{x}_t, \Phi_t) + T_t^P(\mathbf{y}_t, \Psi_t) \right) \\
\text{s.t.} \quad & x_{i,k,t} \in \{0,1\}, \quad i \in [I], k \in [K] \cup \{0\}, t \in [T] \\
& \phi_{i,k,t}^U \in [0,1], \quad i \in [I], k \in [K] \cup \{0\}, t \in [T] \\
& \phi_{i,k,t}^F \in [0,1], \quad i \in [I], k \in [K] \cup \{0\}, t \in [T] \\
& y_{i,n,t} \in \{0,1\}, \quad i \in [I], n \in [N], t \in [T] \\
& \psi_{i,n,t} \in [0,1], \quad i \in [I], n \in [N], t \in [T] \\
& (1), (2), (3), (8), \text{ and } (9).
\end{aligned} \tag{OTORM}$$

The system operates in an online manner. We observe the current system states at the beginning of each time slot t, i.e., channel conditions $\mathbf{h}_t$, task sizes $\mathbf{f}_t$, and input data sizes $\mathbf{d}_t$, and make control decisions $(\mathbf{x}_t, \mathbf{x}_t, \Phi_t, \Psi_t)$ immediately. Let $\mathbb{H}$, $\mathbb{F}$, and $\mathbb{D}$ be the feasible space of $\mathbf{h}_t$, $\mathbf{f}_t$, and $\mathbf{d}_t$, respectively, i.e, $\mathbf{h}_t \in \mathbb{H}$, $\mathbf{f}_t \in \mathbb{F}$, and $\mathbf{d}_t \in \mathbb{D}$. In addition, $\mathbb{H} \subseteq \mathbb{R}^{I*(K+1)}$, $\mathbb{F} \subseteq \mathbb{R}^I$, and $\mathbb{D} \subseteq \mathbb{R}^I$, where $\mathbb{R}$ is the set of real numbers.

4 Problem Analysis and Simplification

In this section, we analyze and simplify the problem we have to solve at each time slot, i.e., minimizing $(T_t^C(\mathbf{x}_t, \Phi_t) + T_t^P(\mathbf{y}_t, \Psi_t))$ over $(\mathbf{x}_t, \Phi_t, \mathbf{y}_t, \Psi_t)$ subjecting to corresponding constraints. First, from objective function $(T_t^C(\mathbf{x}_t, \Phi_t) + T_t^P(\mathbf{y}_t, \Psi_t))$, the communication delay is merely determined by $(\mathbf{x}_t, \Phi_t)$, and the processing delay is a function on variable $(\mathbf{y}_t, \Psi_t))$. Second, there is no coupling between $(\mathbf{x}_t, \Phi_t)$ and $(\mathbf{y}_t, \Psi_t)$ in the constraints. Therefore, we can decompose the problem of minimizing $(T_t^C(\mathbf{x}_t, \Phi_t) + T_t^P(\mathbf{y}_t, \Psi_t))$ into two subproblems. We refer to the first problem of minimizing $T_t^C(\mathbf{x}_t, \Phi_t)$ over $(\mathbf{x}_t, \Phi_t)$ as the Joint base station selection and bandwidth management problem for Communication latency Minimization (JCM), and refer to the second problem of minimizing $T_t^P(\mathbf{y}_t, \Psi_t)$ over $(\mathbf{y}_t, \Psi_t)$ as the Joint task offloading and computing resource management problem for Processing latency Minimization (JPM). Although we can minimize $(T_t^C(\mathbf{x}_t, \Phi_t) + T_t^P(\mathbf{y}_t, \Psi_t))$ as a whole, dividing it into two minor problems can help us to develop a more efficient approach. In particular, we can solve the two subproblems in parallel, speeding up the decision-making time. In addition, since we make decisions using DNNs, solving the two independent subproblems as a whole may cause interference between them, leading to performance degradation.

4.1 Optimal Bandwidth Management for JCM

We consider simplifying the JCM problem by deriving the closed form optimal bandwidth variable Φ_t under any given $\mathbf{x}_t$. First, we formally state the JCM problem as follows:

$$\begin{aligned}
\min_{\mathbf{x}_t \Phi_t} \quad & T_t^C(\mathbf{x}_t, \Phi_t) \\
\text{s.t.} \quad & x_{i,k,t} \in \{0,1\}, \quad i \in [I], k \in [K] \cup \{0\} \\
& \phi_{i,k,t}^U \in [0,1], \quad i \in [I], k \in [K] \cup \{0\} \\
& \phi_{i,k,t}^F \in [0,1], \quad i \in [I], k \in [K] \\
& \sum_{k=0}^{K} x_{i,k,t} = 1, \quad i \in [I] \\
& \sum_{i=1}^{I} \phi_{i,k,t}^U \leq 1, \quad k \in [K] \cup \{0\} \\
& \sum_{i=1}^{I} \phi_{i,k,t}^F \leq 1, \quad k \in [K] \cup \{0\}.
\end{aligned} \tag{JCM}$$

If $\mathbf{x}_t$ is given, JCM is a convex problem, and we can derive the optimal Φ_t under $\mathbf{x}_t$ by exploiting the KKT conditions. For each $k \in [K] \cup \{0\}$, we know the WDs that choose base station B_k if $\mathbf{x}_t$ is given. We use $\mathcal{I}_k(\mathbf{x}_t)$ to denote the set of indexes of WDs that choose B_k under decision $\mathbf{x}_t$, i.e., $i \in \mathcal{I}_k(\mathbf{x}_t)$ if $x_{i,k,t} = 1$. We use $\Phi_t(\mathbf{x}_t)$ to denote the optimal Φ_t under $\mathbf{x}_t$, and $\Phi_t(\mathbf{x}_t)$ is shown in Lemma 1.

Lemma 1. For any feasible $\mathbf{x}_t$, the optimal Φ_t under $\mathbf{x}_t$, denoted by $\Phi_t(\mathbf{x}_t)$, is as follows:

$$\phi_{i,k,t}^U = \begin{cases} \frac{\sqrt{d_{i,t}/h_{i,k,t}^U}}{\sum_{j \in \mathcal{I}_k(\mathbf{x}_t)} \sqrt{d_{j,t}/h_{j,k,t}^U}}, & \text{if } i \in \mathcal{I}_k(\mathbf{x}_t), k \in [K] \cup \{0\} \\ 0, & \text{otherwise.} \end{cases}$$

$$\phi_{i,k,t}^F = \begin{cases} \frac{\sqrt{d_{i,t}/h_k^F}}{\sum_{j \in \mathcal{I}_k(\mathbf{x}_t)} \sqrt{d_{j,t}/h_k^F}}, & \text{if } i \in \mathcal{I}_k(\mathbf{x}_t), k \in [K] \cup \{0\} \\ 0, & \text{otherwise.} \end{cases}$$

The main idea of the proof is to exploit the KKT conditions, and then we can derive the equations in Lemma 1.

Since the process is standard, we omit the proof. Substituting $\Phi_t(\mathbf{x}_t)$ into JCM, JCM is equivalent to the following minimization problem over variable $\mathbf{x}_t$.

$$\begin{aligned}\min_{\mathbf{x}_t} \quad & \sum_{k=0}^{K} \frac{1}{W_k^F}\Big(\sum_{i=1}^{I} x_{i,k,t}\sqrt{\frac{d_{i,t}}{h_k^F}}\Big)^2 + \\ & \sum_{k=0}^{K} \frac{1}{W_k^U}\Big(\sum_{i=1}^{I} x_{i,k,t}\sqrt{\frac{d_{i,t}}{h_{i,k,t}^U}}\Big)^2 \\ \text{s.t.} \quad & x_{i,k,t} \in \{0,1\},\ i \in [I], k \in [K]\cup\{0\} \\ & \sum_{k=0}^{K} x_{i,k,t} = 1,\ i \in [I].\end{aligned} \tag{P1}$$

Although variable Φ_t is eliminated, P1 is still NP-hard as shown in Theorem 1.

Theorem 1. *P1 is NP-hard.*

Proof. We prove that P1 is NP-hard by reducing the set partition problem to a degenerated version of P1. The degenerated version of P1 is denoted by P1*. Under P1*, there are only two base stations, and the uplink bandwidths are infinite. In addition, the two base stations have the same fronthaul bandwidth. Therefore, P1* is partitioning all WDs to two disjoint sets, $\mathcal{I}_0$ and $\mathcal{I}_1$, with the goal of minimizing $(\sum_{i\in\mathcal{I}_0} p_i)^2 + (\sum_{i\in\mathcal{I}_1} p_i)^2$, where $p_i = \sqrt{d_{i,t}} > 0$ and $\mathcal{I}_0 \cup \mathcal{I}_1 = [I]$. The set partition problem is the task of deciding whether set $S = \{p_1, p_2, \cdots, p_I\}$ can be partitioned to two sets, S_1 and S_2, such that $\sum_{p\in S_0} p = \sum_{p\in S_1} p$.

First, we show that if the minimum objective value of P1* is $(\sum i \in [I] p_i)^2/2$ under partition $\mathcal{I}_0$ and $\mathcal{I}_1$, S can be partitioned to two sets such that $\sum_{p\in S_0} p = \sum_{p\in S_1} p$. Let $P_{\text{sum}} = \sum_{i\in[I]} p_i$ and $z = \sum_{i\in\mathcal{I}_0} p_i$. Then, $(\sum i \in [I] p_i)^2/2$ is the minimum objective value of P1* means $(P_{\text{sum}} - z)^2 + z^2 = P_{\text{sum}}^2/2$. Since $P_{\text{sum}} > 0$ and $z \geq 0$, $(P_{\text{sum}} - z)^2 + z^2 = P_{\text{sum}}^2/2$ holds if and only if $z = P/2$. Then, we can partition S to $S_0 = \{p_i | i \in \mathcal{I}_0\}$ and $S_1 = \{p_i | i \in \mathcal{I}_1\}$ such that $\sum_{p\in S_0} p = \sum_{p\in S_1} p$. Similarly, we have if the minimum objective value of P1* is not $(\sum i \in [I] p_i)^2/2$, S can not be partitioned to sets S_1 and S_2 such that $\sum_{p\in S_0} p = \sum_{p\in S_1} p$. That is, we can solve the set partition problem if P1* can be solved. Since the set partition problem is NP-hard, P1* is NP-hard. □

4.2 Optimal Computing Resource Management for JPM

Similar to Section 4.1, we derive the closed-form optimal computing resource management variables Ψ_t under any given $\mathbf{y}_t$ in this section. We first formally state the JPM problem as follows:

$$\begin{aligned}\min_{\mathbf{y}_t,\Psi_t} \quad & T_t^P(\mathbf{y}_t, \Psi_t) \\ \text{s.t.} \quad & y_{i,n,t} \in \{0,1\}, \qquad i \in [I], n \in [N], \\ & \psi_{i,n,t} \in [0,1], \qquad i \in [I], n \in [N], \\ & \sum_{n=1}^{N} y_{i,n,t} = 1, \qquad i \in [I], \\ & \sum_{i=1}^{I} \psi_{i,n,t} \leq 1, \qquad n \in [N].\end{aligned} \tag{JPM}$$

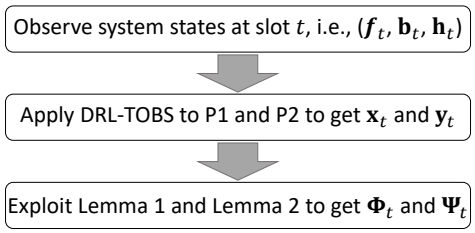


Fig. 2: Scheme for solving *OTORM* at each slot.

If $\mathbf{y}_t$ is given, JPM is convex, and we can derive the optimal Ψ_t under $\mathbf{y}_t$ by exploiting the KKT conditions. We know the WDs that compute their tasks on S_n if $\mathbf{y}_y$ is given. We use $\mathcal{I}_n(\mathbf{y}_t)$ to denote the set of indexes of WDs offloading their task to S_n, i.e., $i \in \mathcal{I}_n(\mathbf{y}_t)$ if $y_{i,n,t} = 1$. In addition, we use $\Psi(\mathbf{y}_t)$ to represent the optimal Ψ under $\mathbf{y}_t$. Then, we have Lemma 2 as follows.

Lemma 2. For any feasible offloading decision $\mathbf{y}_t$, the optimal computing resource management decision Ψ_t under $\mathbf{y}_t$, denoted by $\Psi(\mathbf{y}_t)$, is as follows:

$$\psi_{i,n,t} = \begin{cases} \dfrac{\sqrt{f_{i,t}/\sigma_{i,n}}}{\sum_{j\in\mathcal{I}_n(\mathbf{y}_t)}\sqrt{f_{j,t}/\sigma_{j,n}}}, & \text{if } i \in \mathcal{I}_n(\mathbf{y}_t), k \in [K] \\ 0, & \text{otherwise.} \end{cases}$$

The proof of Lemma 2 is omitted, since the proof is simply an application of KKT conditions. Then, by substituting $\Psi_t(\mathbf{y}_t)$ into JPM, JPM is equivalent to the following minimization problem over variable $\mathbf{y}_t$.

$$\begin{aligned}\min_{\mathbf{y}_t} \quad & \sum_{n=1}^{N} \frac{1}{F_n}\Big(\sum_{i=1}^{I} y_{i,n,t}\sqrt{\frac{f_{i,t}}{\sigma_{i,n}}}\Big)^2 \\ \text{s.t.} \quad & y_{i,n,t} \in \{0,1\}, \qquad i \in [I], n \in [N] \\ & \sum_{n=1}^{N} y_{i,n,t} = 1, \qquad i \in [I].\end{aligned} \tag{P2}$$

Theorem 2. *P2 is NP-Hard.*

The proof of Theorem 2 is similar to that of Theorem 1, and we omit it.

5 Online Scheme Design for *OTORM*

In this section, we focus on designing a scheme for solving *OTORM* in an online manner. A diagram of the scheme is shown in Figure 2. At the beginning of each time slot t, we observe the current system states, i.e., input data sizes $\mathbf{d}_t$, task sizes $\mathbf{f}_t$, and uplink channel conditions $\mathbf{h}_t$, and make four sets of decisions, namely $\mathbf{x}_t, \mathbf{y}_t, \Phi_t, \Psi_t$. We have to decide the decision variables in a short time. Otherwise, high decision-making time will cause degeneration of experience. From Section 4, we can get the closed form optimal Φ_t and Ψ_t once $\mathbf{x}_t$ and $\mathbf{y}_t$ are given. Therefore, the core of the scheme is solving P1 and P2 to get $\mathbf{x}_t$ and $\mathbf{y}_t$, respectively.

Since P1 and P2 are NP-hard, there is no algorithm can solve them in polynomial time. In this section, we design an algorithm named Deep Reinforcement Leaning based Task Offloading and Base station Selection algorithm (*DRL-TOBS*), for solving P1 and P2 efficiently. A Diagram for

the *DRL-TOBS* approach is shown in Figure 3. While the proposed approach is not a classical reinforcement learning method, it shares some features with RL. Specifically, in our approach, the offline solver is a part of the environment of RL, and the agent is represented by the DNNs. At each time slot, the environment generates a new state and updates the datasets. In contrast to the classical RL approach, our method does not have a reward at each time slot but instead relies on the datasets as a substitute. In other words, while the agent in classical RL updates its policy based on the reward, our approach updates its policy (the DNNs) based on the datasets.

The overall idea of the *DRL-TOBS* approach is as follows. *DRL-TOBS* uses two DNNs to make decision $(\mathbf{x}_t, \mathbf{y}_t)$, similar to the actor-network of the actor-critic RL algorithm. Since the outputs of the DNNs are continuous variables and $(\mathbf{x}_t, \mathbf{y}_t)$ are binary, we then discretize the continuous outputs to multiple binary candidates and select the best one among the candidates to perform. After performing the decisions and before the beginning of the next slot, we use an offline solver to get another decision $(\bar{\mathbf{x}}_t, \bar{\mathbf{y}}_t)$. *DRL-TOBS* then compares the decisions made by the offline solver and the performed decisions and stores the better decisions in datasets. Lastly, we randomly select samples from the datasets to train the DNNs. Next, we will introduce the modules in Figure 3 in detail.

5.1 Deep Neural Networks

From the universal approximation theorem, it is possible to use neural networks with hidden layers to approximate solvers of P1 and P2. At the beginning of each time slot t, we observe the current system states, i.e., $\mathbf{d}_t, \mathbf{f}_t$ and $\mathbf{h}_t$, and pass them to the neural networks, i.e., DNN-1 and DNN-2 for P1 and P2, respectively.

a) DNN-1 for P1:

The inputs of DNN-1 are $\mathbf{h}_t$ and $\mathbf{d}_t$. The input layer has $I(K+2)$ neurons. Then, the input layer is connected to hidden layers. Since we consider an online scenario, the training and inference processes of DNN-1 have to be fast. Compared with the sigmoid function, the ReLU function and its derivative are easy to compute. Therefore, we use the ReLU function as the activation function of hidden layers.

Next, we consider the output layer of the neural network. We set the number of neurons in the output layer to be $I(K+1)$, where each output approximates a decision variable $x_{i,k,t}$. There is a constraint for $x_{i,k,t}$, i.e., $\sum_{k=0}^{K} x_{i,k,t} = 1$ for $i \in [I]$. We embed the constraint in the output layer, as shown in Figure 4. To be more specific, the neurons in the output layers are partitioned into I groups of the same size. The outputs of the i-th group approximate variables related to D_i, i.e., $x_{i,k,t}$ for $k \in [K] \cup \{0\}$. Then, neurons of each group use a softmax activation function. Use $z_{i,k}$ to denote the input of the k-th neuron in the i-th group. Under softmax activation functions, the output of the k-th neuron in the i-th group, denoted by $\tilde{x}_{i,k}$, is as follows,

$$\tilde{x}_{i,k} = \frac{e^{z_{i,k}}}{\sum_{k'=0}^{K} e^{z_{i,k'}}}. \tag{12}$$

Therefore, we have $\sum_{k=0}^{K} \tilde{x}_{i,k,t} = 1$ for $i \in [I]$. The physical meaning of $\tilde{x}_{i,k,t}$ is the probability of D_i choosing base station B_k, and $\sum_{k=0}^{K} \tilde{x}_{i,k,t} = 1$ means each WD has to select one base station.

The loss function of the network is the cross entropy between output $\tilde{\mathbf{x}}_\tau = \{\tilde{x}_{i,k,\tau}, i \in [I], k \in [K] \cup \{0\}\}$ and the corresponding target decision in the dataset denoted by $\mathbf{x}_\tau$. To be more specific, the loss function of DNN-1 is as follows:

$$Loss_1(\tilde{\mathbf{x}}_\tau, \mathbf{x}_\tau) = \sum_{i=1}^{I} \sum_{k=0}^{K} -x_{i,k,\tau} \log(\tilde{x}_{i,k,\tau}). \tag{13}$$

a) DNN-2 for P2:

DNN-2 for P2 is similar to DNN-1 for P1. At each time slot t, the input of DNN-2 is $\mathbf{f}_t = \{f_{i,t} | i \in [I]\}$. That is, the input layer has I neurons. Then, we pass the inputs to hidden layers, where the hidden layers use the ReLU function as their activation function for high efficiency. The output layer has $I \times N$ outputs approximating $y_{i,n,t}$ for $i \in [I]$, $n \in [N]$. Similar to DNN-1, we embed constraint $\sum_{n=1}^{N} y_{i,n,t} = 1, i \in [I]$ to the output layer. In particular, the output neurons are partitioned into I groups of the same size, and neurons in each group use the softmax function as the activation function. Use $z_{i,n}$ to denote the input of the n-th neuron in the i-th group. Then, the output of the n-th neuron in the i-th group, denoted by $\tilde{y}_{i,n}$, is as follows,

$$\tilde{y}_{i,n} = \frac{e^{z_{i,n}}}{\sum_{n'=1}^{K} e^{z_{i,n'}}}. \tag{14}$$

We have $\sum_{n=1}^{N} \tilde{y}_{i,n} = 1$ for $i \in [N]$, and $\tilde{y}_{i,n}$ represents the probability of D_i computing its task on server S_n. The loss function of the network is the cross entropy between output $\tilde{\mathbf{y}}_\tau = \{\tilde{y}_{i,n,\tau}, i \in [I], k \in [N]\}$ and the corresponding decision in the dataset denoted by $\mathbf{y}_\tau$, as follows:

$$Loss_2(\tilde{\mathbf{y}}, \mathbf{y}) = \sum_{i=1}^{I} \sum_{n=1}^{N} -y_{i,n} \log(\tilde{y}_{i,n}). \tag{15}$$

The time complexities of a forward pass of the DNNs are polynomial with respect to the number of neurons in each layer. Moreover, the process primarily involves matrix multiplications, which can be parallelized efficiently using GPUs.

5.2 Discretization and Performing Decision

From (12) and (14), the outputs of DNN-1 and DNN-2 are real numbers between 0 and 1. We then discretize the outputs to feasible binary decision variables. For each $i \in [N]$, let k_i^* be a realization of random variable $K_{i,t}$ where the probability mass function (PMF) of $K_{i,t}$ is $\mathbb{P}(K_{i,t} = k) = \tilde{x}_{i,k,t}, k \in [K] \cup \{0\}$. Then, we set $x_{i,k_i^*,t} = 1$ and set $x_{i,k,t} = 0$ for $k \neq k_i^*$. Similarly, let n_i^* be a realization of random variable $N_{i,t}$ where the probability mass function (PMF) of $N_{i,t}$ is $\mathbb{P}(N_{i,t} = n) = \tilde{y}_{i,n,t}, n \in [N]$. We set $y_{i,n_i^*,t} = 1$ and set $y_{i,n,t} = 0$ for $n \neq N_i^*$. In addition to the above discretization method, D_i can choose the base station with the largest $\tilde{x}_{i,k,t}$ and choose the edge server with the largest $\tilde{y}_{i,n,t}$. We generate a number of feasible solutions for P1 and P2. We then compare the solutions for P1 and use $\hat{\mathbf{x}}_t$ to denote the solution with the lowest communication latency. Similarly, we compare the solutions for P2 and use $\hat{\mathbf{y}}_t$ to denote the solution with the lowest processing latency. The time complexity of the discretization process

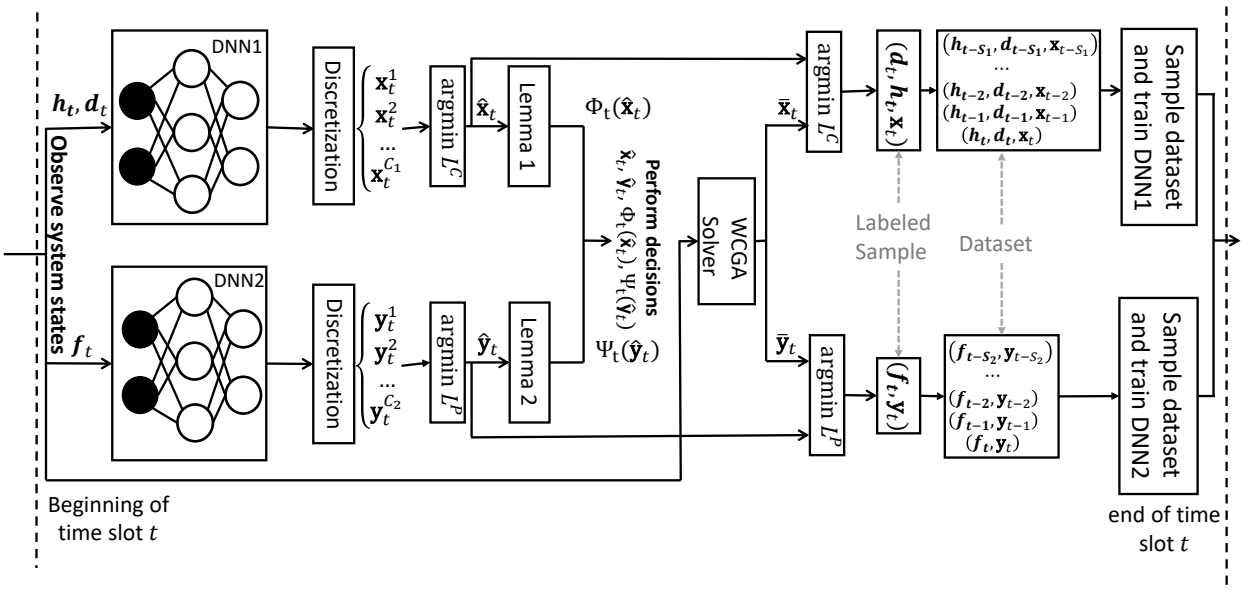


Fig. 3: Diagram of the *DRL-TOBS* Approach.

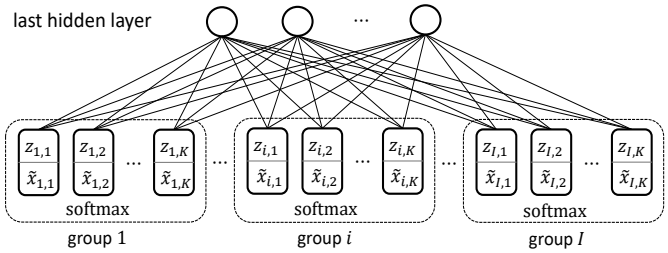


Fig. 4: Embedding Constraint (1) to the Output Layer

is polynomial to I, N, K and the number of candidate decisions. Next, referring to Lemma 1 and Lemma 2, we can get the corresponding closed-form resource management decisions, i.e., $\Phi(\hat{\mathbf{x}})$ and $\Psi(\hat{\mathbf{y}})$. Then, we perform decision $\{\hat{\mathbf{x}}, \Phi(\hat{\mathbf{x}}), \hat{\mathbf{y}}_t, \Psi(\hat{\mathbf{y}})\}$.

After making decision $\{\hat{\mathbf{x}}, \Phi(\hat{\mathbf{x}}), \hat{\mathbf{y}}_t, \Psi(\hat{\mathbf{y}})\}$ in a short time, we can improve the neural networks using the remaining time in time slot t. In the following, we consider the processes of generating data samples and training the neural networks.

5.3 Solver For P1 and P2

After performing the decision of time slot t and before the beginning of the next time slot, we use a solver to solve P1 and P2 to get better decisions, which are used for training the neural networks.

Some commercial solvers for P1 and P2 are available, e.g., Gurobi [27], MOSEK [28], and GLPK [29]. The advantage of the solvers is that they can return the optimal solution, while the disadvantage is that it may take a relatively long time to get the optimal solution. Although we can use multi-slots, e.g., δ, to solve P1 and P2 and train the neural networks every δ time slots, it will lead to a longer convergence time for the neural networks. For example, the Gurobi solver may take up to ten minutes to solve P2 under $I = 120$ and $N = 16$, and *WCGA* proposed in Section 6 takes less than one second to get an approximation solution with high precision. In real-world systems, the system states have high volatility, e.g., change for each second. Therefore, designing a solver to generate data samples for DNN-1 and DNN-2 more frequently is necessary. We propose an algorithm named *WCGA* in Section 6 that is more efficient at solving P1 and P2 in order to decrease the convergence time of DNN-1 and DNN-2. The *WCGA* algorithm sacrifices precision in order to solve P1 and P2 more efficiently. The details of the *WCGA* solver are in Section 6.

5.4 Datasets, Sampling and Training

Once the solver for P1 and P2 returns solutions $\bar{\mathbf{x}}_t$ and $\bar{\mathbf{y}}_t$, we compare them with the previous decisions made by DNN-1 and DNN-2, i.e., $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{y}}_t$, and use $\mathbf{x}_t$ and $\mathbf{y}_t$ to denote the better solutions for P1 and P2, respectively. After we have $\mathbf{x}_t$, let $((\mathbf{h}_t, \mathbf{d}_t), \mathbf{x}_t)$ be an data sample for DNN-1, where $(\mathbf{h}_t, \mathbf{d}_t)$ is the input and $\mathbf{x}_t$ is the target. Similarly, after we have $\mathbf{y}_t$, let $(\mathbf{f}_t, \mathbf{y}_t)$ be an data sample for DNN-2. The datasets for DNN-1 and DNN-2 have sizes of S_1 and S_2, respectively. If a dataset is full, the latest data sample will replace the oldest one.

Similar to [11], after we update the datasets, we randomly sample a batch of samples from each dataset and use the sampled batches to train their corresponding neural networks. We use stochastic gradient descent (SGD) with momentum as the optimizer for training the neural networks. Updating the DNN parameters has a polynomial time complexity that depends on the number of neurons at each layer and the number of data samples. This process primarily involves matrix operations, which can be highly parallelized using GPUs.

5.5 The *DRL-TOBS* Algorithm

We then combine the components of the *DRL-TOBS* algorithm and state the algorithm as a whole. Since P1 and P2 are independent, the *DRL-TOBS* algorithm can solve P1 and P2 in parallel. Next, we formally state the *DRL-TOBS* algorithm

Algorithm 1: The *DRL-TOBS* algorithm for P1(P2).

Input: W_k^F,W_k^U,F_n, $\sigma_{i,n}$
Output: A solution for P1 (P2): $\bar{\mathbf{x}}$ ($\bar{\mathbf{y}}$, respectively)

Initialization: Randomly choose parameters for DNN-1 (DNN-2, respectively);
for $t = \{1, 2, \cdots, \}$ **do**
  Observer current system states, i.e., $\mathbf{d}_t = \{d_{i,t} | i \in [I]\}$, $\mathbf{f}_t = \{f_{i,t} | i \in [I]\}$, and $\mathbf{h}_t = \{h_{i,k,t}^U | i \in [I], k \in [K] \cup \{0\}\}$;
  Feed $\mathbf{d}_t \cup \mathbf{h}_t$ $(\mathbf{f}_t)$ to DNN-1 (DNN-2, respectively) and get $\tilde{\mathbf{x}}_t$ ($\tilde{\mathbf{y}}_t$, respectively);
  Discretize $\tilde{\mathbf{x}}_t$ ($\tilde{\mathbf{y}}_t$) to C_1 (C_2, respectively) feasible candidates, i.e., $\{\mathbf{x}_t^1, \mathbf{x}_t^2, \cdots, \mathbf{x}_t^{C_1}\}$ ($\{\mathbf{y}_t^1, \mathbf{y}_t^2, \cdots, \mathbf{y}_t^{C_2}\}$, respectively);
  Choose the best candidate, denoted by $\hat{\mathbf{x}}_t$ ($\hat{\mathbf{y}}_t$), among $\{\mathbf{x}_t^1, \mathbf{x}_t^2, \cdots, \mathbf{x}_t^{C_1}\}$ ($\{\mathbf{y}_t^1, \mathbf{y}_t^2, \cdots, \mathbf{y}_t^{C_2}\}$, respectively) and calculate $\Phi(\hat{\mathbf{x}}_t)$ ($\Psi(\hat{\mathbf{y}}_t)$, respectively);
  Perform decision $\hat{\mathbf{x}}_t, \Phi(\hat{\mathbf{x}}_t)$ $(\hat{\mathbf{y}}_t, \Psi(\hat{\mathbf{y}}_t))$ for JCM (JPM, respectively);
  if $t\%\delta_1 == 0$ ($t\%\delta_2 == 0$) **then**
    Apply *WCGA* to P1(P2) to get $\bar{\mathbf{x}}_t$ ($\bar{\mathbf{y}}_t$, respectively);
    Save the latest sample to dataset;
    Randomly sample a batch of data samples for DNN-1 (DNN-2, respectively);
    Train DNN-1 (DNN-2, respectively) and update its parameters;
  end
end

in Algorithm 1. We train DNN-1 and DNN-2 for every δ_1 and δ_2 time slots, respectively.

6 Algorithm Design for P1 and P2

Although P1 and P2 are two different problems, they are special cases of a problem named Weighted Congestion Problem (*WCP*).

We first state the weighted congestion problem (*WCP*). Under the *WCP* problem, there are I players and R types of resources. $\mathcal{R}$ is the set of all resources. Each player $i \in [I]$ has to choose a decision from feasible set $\mathcal{D}$, where $\mathcal{D}$ has a finite number of elements. We use $\mathbf{d}_i \in \mathcal{D}$ to denote the decision made by player i and let $\mathbf{D} = (\mathbf{d}_1, \mathbf{d}_2, \cdots, \mathbf{d}_I)$ be the collection of decisions of all players. There is a parameter m_r for each resource $r \in \mathcal{R}$. For each decision $\mathbf{d} \in \mathcal{D}$, there is a set $\mathcal{R}(\mathbf{d}) \in \mathcal{R}$ associated with it, which means that player i uses resources in set $\mathcal{R}(\mathbf{d})$. In addition, there is a parameter $p_{i,r}$ for player i and resource r. Let $\mathcal{I}_r(\mathbf{D})$ to denote the set of players that chooses resource r, i.e., $i \in \mathcal{I}_r(\mathbf{D})$ if and only if $r \in \mathcal{R}(\mathbf{d}_i)$. Each resource $r \in \mathcal{R}$ has a congestion value $p_r(\mathbf{D})$ which is a function of decision $\mathbf{D}$ as follows:

$$p_r(\mathbf{D}) = \sum_{i \in \mathcal{I}_r(\mathbf{D})} p_{i,r}. \tag{16}$$

The cost function of player i is

$$f_i(\mathbf{D}) = \sum_{r \in \mathcal{R}(\mathbf{d}_i)} m_r p_{i,r} p_r(\mathbf{D}).$$

The goal of the system is to minimize

$$f(\mathbf{D}) = \sum_{i \in [I]} \sum_{r \in \mathcal{R}(d_i)} m_r p_{i,r} p_r(\mathbf{D}).$$

All the parameters, i.e., $p_{i,r}$ and m_r, are non-negative. We formally state *WCP* as an optimization problem as follows.

$$\begin{aligned} \min_{\mathbf{d}_i, i \in [I]} \quad & f(\mathbf{D}) = \sum_{i \in [I]} \sum_{r \in \mathcal{R}(\mathbf{d}_i)} m_r p_{i,r} p_r(\mathbf{D}) \\ \text{s.t.} \quad & \mathbf{d}_i \in \mathcal{D}, \qquad i \in [I]. \end{aligned} \tag{WCP}$$

6.1 Interpreting P1 and P2 as *WCP*

In what follows, we show P1 and P2 can be expressed as the *WCP* problem.

a) Interpreting P1:

First, we express P1 as *WCP*. Let the I WDs of P1 be the I players of *WCP*. Let $\mathcal{R} = \{r_k^U, r_k^F | k \in [K] \cup \{0\}\}$ be the $R = 2(K+1)$ types of resources where r_k^U and r_k^F are the uplink and fronthaul link of base station k, respectively. For $k \in [K] \cup \{0\}$, let $m_r = 1/W_k^U$ if r is r_k^U, and let $m_r = 1/W_k^F$ if r represents r_k^F. There are $K+1$ decisions in feasible set $\mathcal{D}$, i.e., $\mathcal{D} = \{B_0, B_1 \cdots, B_K\}$ representing the base stations that users can choose. $\mathcal{R}(\mathbf{d}) = \{r_k^U, r_k^F\}$ if $\mathbf{d}$ is base station B_k. Let $p_{i,r} = \sqrt{d_{i,t}/h_k^F}$ if r is r_k^F, and $p_{i,r} = \sqrt{d_{i,t}/h_{i,k,t}^U}$ if r is r_k^U.

For each $i \in [I]$, constraint (1) is equivalent to each player i can only choose one base station. That is, there is a bijection between feasible solutions of (1) and that of *WCP*. For each $i \in [I]$, the communication latency experienced by D_i is

$$\begin{aligned} T_{i,t}^C =& \sum_{k=0}^{K} \frac{x_{i,k,t}}{W_k^F} \sqrt{\frac{d_{i,t}}{h_k^F}} \left(\sum_{j=1}^{I} x_{j,k,t} \sqrt{\frac{d_{j,t}}{h_k^F}} \right) \\ &+ \sum_{k=0}^{K} \frac{x_{i,k,t}}{W_k^U} \sqrt{\frac{d_{i,t}}{h_{i,k,t}^U}} \left(\sum_{j=1}^{I} x_{j,k,t} \sqrt{\frac{d_{j,t}}{h_{i,k,t}^U}} \right) \\ =& \sum_{r \in \mathcal{R}(\mathbf{d}_i)} m_r p_{i,r} p_r(\mathbf{D}). \end{aligned}$$

Summing the above equation up for $i \in [I]$, for each feasible $\mathbf{x}_t$ and its corresponding $\mathbf{D}$, we have the objective value of P1 and that of *WCP* are equal. Therefore, P1 is a special case of *WCP*.

b) Interpreting P2:

Next, we express P2 as *WCP*. Let the I WDs of P1 be the I players of *WCP*. Let $\mathcal{R} = \{r_1, r_2, \cdots, r_N\}$ be the $R = N$ types of resources where r_n represents the computing resource of edge server S_n. In addition, we have $m_r = 1/F_n$ if $r = r_n$. There are N decisions in feasible set $\mathcal{D}$, i.e., $\mathcal{D} = \{S_1, S_2 \cdots, S_N\}$ representing the edge servers that users can choose. For each $\mathbf{d} \in \mathcal{D}$, there is only one element in $\mathcal{R}(\mathbf{d})$. In particular, $\mathcal{R}(\mathbf{d}) = \{r_n\}$ if $\mathbf{d}$ represents S_n. Moreover, $p_{i,r} = \sqrt{f_{i,t}/\sigma_{i,n}}$ if $r = r_n$. Similar to that for P1, there is a bijection between feasible solutions of (2) and that of *WCP*. In addition, for any $\mathbf{y}_t$ and its corresponding $\mathbf{D}$, the objective value of P2 and that of *WCP* are equal. That is, P2 is a special case of the *WCP* problem.

Algorithm 2: *WCGA*(λ)

Input: $I, \mathcal{R}, \{p_{i,r} | i \in [I], r \in \mathcal{R}\}, \mathcal{D}, \mathcal{R}(\mathbf{d}_i)$ for $\mathbf{d}_i \in \mathcal{D}, m_r$ for $r \in \mathcal{R}$.

Output: A feasible solution to *WCP*: $\hat{\mathbf{D}}$

Initialization: choose $\mathbf{d}_i$ from $\mathcal{D}$ randomly for $i \in [I]$;

while $\{\exists i \in \mathcal{I}, (1-\lambda) f_i(\mathbf{D}) > \min_{\bar{\mathbf{d}}_i \in \mathcal{D}} f_i(\bar{\mathbf{d}}_i, \mathbf{D}_{-i})\}$ **do**

  $i := \arg\max_{j \in \mathcal{I}} \Big\{ f_j(\mathbf{D}) - \min_{\bar{\mathbf{d}}_j \in \mathcal{D}} f_i(\bar{\mathbf{d}}_i, \mathbf{D}_{-i}) \Big\}$;

  $\hat{\mathbf{d}}_i := \arg\min_{\bar{\mathbf{d}}_i \in \mathcal{D}} f_i(\bar{\mathbf{d}}_i, \mathbf{D}_{-i})$;

  $\mathbf{D} := (\hat{\mathbf{d}}_i, \mathbf{D}_{-i})$;

end

Return $\hat{\mathbf{D}} := \mathbf{D}$;

6.2 Algorithm Design

Since P1 and P2 can be expressed as the *WCP* problem, in this section, we focus on designing an algorithm for *WCP*. We can solve *WCP* by a game theoretic-based approach shown in Algorithm 2.

Then, we state the algorithm designed for solving *WCP*, named the Weighted Congestion Game-based Algorithm (WCGA), in Algorithm 2. The *WCGA* algorithm has a tunable parameter λ. Next, we show the performance of *WCGA*.

Theorem 3. *For any $\lambda < 0.125$, WCGA(λ) will generate a decision $\mathbf{D}$ with $f(\mathbf{D}) \leq \frac{2.62}{1-8\lambda} f(\mathbf{D}^*)$ in at most $\frac{I}{\lambda} \log(\frac{P_{\max}}{P_{\min}})$ iterations where $P_{\max}$ and $P_{\min}$ are the maximum and the minimum values of potential function $P(\mathbf{D})$.*

The proof of Theorem 3 is found in the our technical report [30]. Note that each iteration takes $O(|\mathcal{D}|)$ steps. In addition, if $\lambda = 0$, we have the following theorem.

Theorem 4. *WCGA(0) will terminate to a decision $\mathbf{D}$ with $f(\mathbf{D}) \leq 2.62 f(\mathbf{D}^*)$ in finite time steps.*

The proof of Theorem 4 is similar to that of Theorem 4 and Theorem 5 in [10]. Consequently, we omit it.

Theorem 3 shows that *WCGA*(λ) can generate a solution with a constant approximation ratio in polynomial iterations. Theorem 4 shows that *WCGA*(0) has an approximation ratio smaller than that of *WCGA*(λ) but has no guarantee about convergence time. Nevertheless, experiments show that *WCGA*(0) converges faster than the Gurobi solver, where *WCGA*(0) sacrifices precision for low time complexity. The simulations in Section 7 use *WCGA*(0) to generates data samples for P1 and P2 for better performance.

7 Numerical Evaluation

In this section, we conduct extensive simulations to evaluate the *DRL-TOBS* approach. We implement our simulations using Python 3.10 on a desktop computer with 32GB RAM, Ryzen 7 2700X Eight-Core Processor, GEFORCE GTX 1080 Ti GPU, and Windows 10 operating system.

7.1 Simulation Settings

We simulate a relatively large-scale system with more than 100 wireless devices. We set $F_n, n \in [N]$, computing capabilities of edge servers, as real-world computing capabilities of EC2 instances [31] as shown in Table 2. In particular, the unit of $F_n, n \in [N]$ is Giga floating-point operations per second (GFLOP/s), and there are $N = 16$ servers in the edge server room, i.e., two of each type in Table 2. In [32], some real-world task sizes are in the range between 65 MFLOPs and 250 MFLOPs. Therefore, the task sizes of each time slot are randomly drawn from the range between 65 and 250 MFLOPs. Similar to [10], we draw parameter $\sigma_{i,n}$ randomly from the range from 0.5 to 1.

Type	GFLOP/s	Type	GFLOP/s
c3.4xlarge	358.4	c3.8xlarge	716.8
c4.2xlarge	371.2	c4.4xlarge	742.4
cc2.8xlarge	665.6	d2.2xlarge	307.2
d2.4xlarge	614.4	g2.8xlarge	665.6

TABLE 2: Computing Capabilities of Servers.

We assume there are ten 5G base stations in the system. In particular, there are nine micro base stations and one macro base station. We assume the base stations are operated by AT&T using low-land n5 and mid-band n77. We assume the macro base station uses band n5, which has an uplink bandwidth of 45 MHz, i.e., $W_0^U = 45MHz$. The micro base stations use band n77 of bandwidth 100 MHz, i.e., $W_k^U = 100MHz$. We assume all wireless mobile devices can get access to the macro base station, and each mobile device can only connect to five micro base stations. From [26], LTE uplink spectrum efficiency can be up to 50 bps/Hz. The channel condition $h_{i,k,t}^U$ is a function of the communication distance and other parameters, which increases as the communication distance decreases. As the wireless devices walk randomly, the communication distances change randomly, and the channel condition varies accordingly. We abstract the random walk model by manipulating $h_{i,k,t}^U$ directly. At the very beginning, if D_i is covered by B_k, we draw $h_{i,k,0}^U$ randomly from the range between 15 and 50. Since $h_{i,k,t}^U$ changes over time, we set $h_{i,k,t}^U = h_{i,k,t-1}^U \cdot (1 + e_{i,k,t})$, where $e_{i,k,t}$ is randomly drawn from normal distribution $N(0, 0.01)$. From [33], the spectrum efficiency of optical fiber communication can reach higher than ten bps/Hz, and we set h_k^F as ten. The bandwidth of the fronthaul links is set to 1GHz [34]. In addition, input data length $d_{i,t}$ is randomly drawn from the range between 1 to 5 megabits, similar to that in [10].

We set the number of hidden layers of both DNN-1 and DNN-2 as three. For DNN-1, the numbers of neurons of the first, second, and third hidden layers are $I \cdot K$, I, and I, respectively. Similarly, the number of neurons of the first, second, and third hidden layers of DNN-2 are $I \cdot N$, I, and I, respectively. Note that, we can replace the deep neural networks with other structures, e.g., Transformer Networks, RNN, and Pointer Networks. We set the duration of each time slot to be one second to simulate a highly volatile system. Since commercial solver Gurobi takes up to ten minutes to solve P1 and P2, we use the *WCGA* solver for generating data samples. The *WCGA*(0) takes less than one second to solve P1 and P2 on average. Thus, we set $\delta_1^A = 1$ and $\delta_1^B = 1$. That is, we update the two neural networks every time slot. Other parameters are stated in detail in the following.

7.2 Convergence of DNN-1 and DNN-2

In this section, we evaluate the convergence performance of the *DRL-TOBS* approach. We use *WCGA*(0) as the solver for P1 and P2 at each time slot. In addition to the loss functions defined in (13) and (15), we measure the performance of *DRL-TOBS* using normalized communication latency and normalized processing latency for P1 and P2, respectively. In particular, the normalized communication latency for P1 (P2) equals the ratio of communication latency (processing latency, respectively) under *DRL-TOBS* to the optimal communication latency (processing latency, respectively). We set the learning rate of DNN-1 and DNN-2 to 0.01. In addition, we apply the DROO algorithm [11] to the problems.

The normalized communication latency under *DRL-TOBS* is shown in Figure 5. The black line in Figure 5 shows the moving average normalized communication latency with a step size equaling 20 slots. The silver area in Figure 5 is the range of the normalized communication latency over every 20 slots. Figure 5 shows that *DRL-TOBS* converges in 60 slots, and the converged communication latency is near-optimal (less than 1.02 times the optimal communication latency on average). In addition, the red line in Figure 5 is the loss of DNN-1 over time. The normalized communication latency of P1 decreases as the loss of DNN-1 (13) decreases, which shows that the loss function defined in (13) can properly represent the objective value of P1. In addition, the dark blue line in Figure 5 shows that the DROO algorithm [11] is not capable of solving P1, and it is necessary to design *DRL-TOBS*.

The normalized processing latency of *DRL-TOBS* is shown in Figure 6. Similar to that of Figure 5, the black line in Figure 6 is the moving average of the normalized processing latency under a step size of 20 time slots, and the silver area shows the range of the normalized processing latency over every 20 slots. Figure 6 shows that *DRL-TOBS* converges to a near-optimal approximation ratio (less than 1.02 on average) in a relatively short time (around 60 time slots). In addition, the red line in Figure 6 shows the loss of DNN-2 over time. The loss of DNN-2 decreases as the normalized processing latency of P2 declines, which shows that minimizing the loss function defined in (15) is equivalent to minimizing the objective value of P2 to a certain extent. In addition, the normalized latency under the DROO algorithm does not converge to near-optimal.

Next, we use a single DNN for the *OTORM* problem to simulate the case that *OTORM* can not be partitioned into two subproblems. In particular, $(\mathbf{f}_t, \mathbf{h}_t, \mathbf{d}_t)$ are the inputs of the DNN. As shown in Figure 7, the normalized latency consisting both communication and computing latencies can converge to near optimal. In addition, the loss of the DNN can also converge but have a higher loss than that of DNN-1 and DNN-2.

7.3 Performance against System Fluctuations

In this section, we evaluate the performance of *DRL-TOBS* under system fluctuations. Specifically, we focus on a scenario where users can join and leave the system. We set the number of WDs to 180. After the convergence of DNN-1 and DNN-2, we randomly shut 20 WDs down at time slot

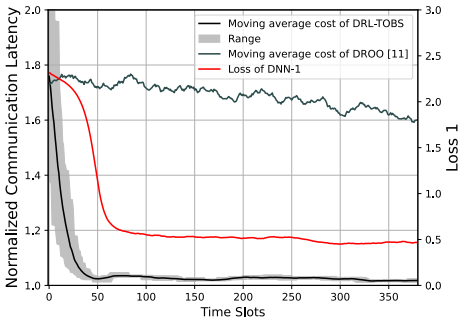


Fig. 5: Convergence of DNN-1

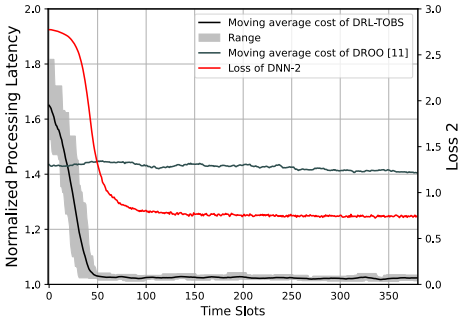


Fig. 6: Convergence of DNN-2

$t = 200$, and we then reboot the 20 WDs at time slot $t = 300$. We set $d_{i,t}$ and $f_{i,t}$ to 0 if WD i is down at time slot t.

We first consider the performance of *DRL-TOBS* for P1 against system fluctuations. The normalized communication latency and loss of DNN-1 are shown in Figure 8. In particular, the dashed gray line shows the normalized processing latency under *DRL-TOBS* using the loss function considering both inactive and active WDs. The solid black line indicates the normalized processing latency using the loss function only considering active WDs. Similarly, the solid red line is the loss of DNN-1 using the loss function considering only active WDs, while the dashed orange line uses the loss function considering all WDs. As we can see from Figure 8, when WDs leave the system at time slot $t = 200$, the normalized communication latency does not have significant volatility, and *DRL-TOBS* performs as well as that of stationary situations. However, when WDs join the system at time slot $t = 300$, the normalized communication latency increases significantly under *DRL-TOBS* using the loss function considering all WDs. It takes around ten time slots for the normalized communication latency to converge again. In contrast, under *DRL-TOBS* using the loss function considering only active WDs, the normalized communication latency remains near-optimal when WDs join the system. *DRL-TOBS* using the loss function considering all WDs performs worse than that considering only active WDs because *WCGA* offloads all inactive WDs via one base station. And *DRL-TOBS* tends to offload the WDs via the same base station when the inactive WDs rejoin the system, which causes congestion.

Next, we consider the performance of *DRL-TOBS* for P2 against system fluctuations. As shown in Figure 9, when the loss function of DNN-2 considers only active WDs, *DRL-TOBS* has near-optimal normalized processing latency when

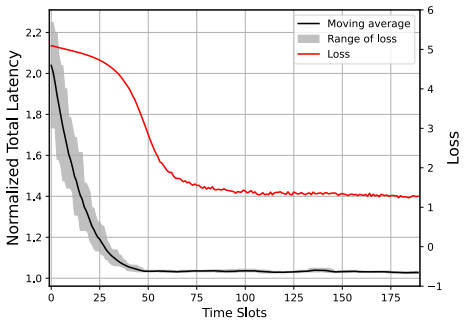


Fig. 7: Convergence of a Single DNN Approach

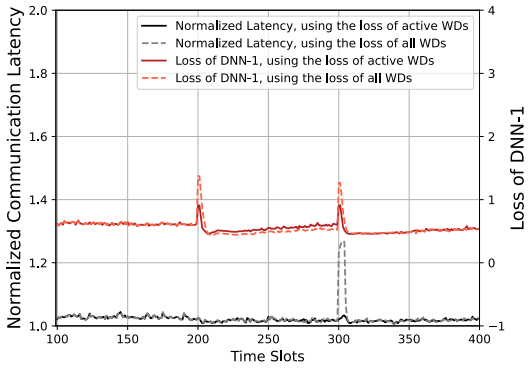


Fig. 8: Performance of *DRL-TOBS* for P1 against system fluctuations

WDs leave and rejoin the system. When the loss function of DNN-2 considers all WDs, *DRL-TOBS* has high processing latency after inactive WDs rejoin the system. The reason is that *WCGA* offloads all inactive WDs to one edge server, and *DRL-TOBS* tends to keep them in the same edge server after rejoining the system, causing congestion.

7.4 Performance Comparison with Baselines

In this section, we compare the performance of *DRL-TOBS* with two baselines. The first baseline is the MCMC, where MCMC represents the Markov chain Monte Carlo method. To be more specific, the MCMC algorithm chooses $\mathbf{x}$ and $\mathbf{y}$ randomly at the beginning. Then, WDs take turns to randomly generate a decision and move to the new decision with a probability, where the probability is related to the objective values of the old and new decisions. The second baseline is called ROPT, which is similar to that used in [10]. In particular, ROPT chooses a base station and an edge server for each device randomly and uses the optimal resource management decisions, i.e., optimal Φ_t and Ψ_t under randomly selected $\mathbf{x}_t$ and $\mathbf{y}_t$. Details of the two baselines can be found in the source code.

We first compare the average system latency (sum of processing and communication latency) of the proposed algorithm with that of MCMC, ROPT, and the optimal latency, under different numbers of WDs. As shown in Figure 10 (the unit is a millisecond), the proposed algorithm outperforms the two baselines used in the literature. In addition, the proposed algorithm is near optimal, which can achieve $1.016\times$ the optimal latency on average. The latency under each algorithm increases as the number of

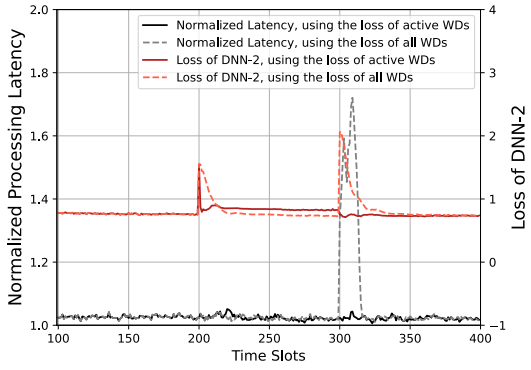


Fig. 9: Performance of *DRL-TOBS* for P2 against system fluctuations

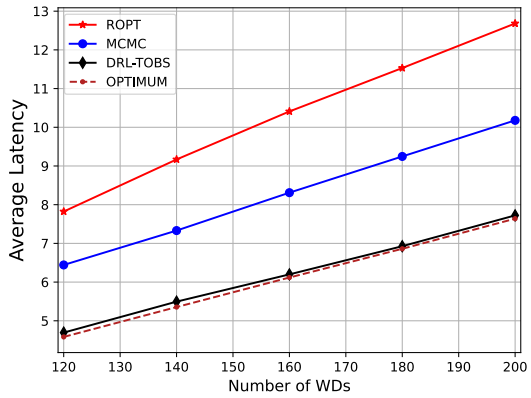


Fig. 10: Average system latency under *DRL-TOBS* v.s. the number of WDs

WDs increases due to system congestion. Then, we consider the time complexity of *DRL-TOBS* with that of MCMC, ROPT, and the Gurobi solver. We solve P1 and P2 in parallel, and each time listed in Table 3 is the maximum of corresponding times for P1 and P2. As we can see from Table 3, *WCGA* takes around one millisecond to make a decision that is significantly lower than the MCMC method and the Gurobi solver, while the Gurobi solver takes approximately 1 minute to make a decision. In addition, the training time is in milliseconds. The time complexities of the MCMC method and the Gurobi solver are much longer than a time slot, and the approximation ratio of the ROPT baseline is relatively large. On the contrary, the proposed algorithm has a small approximation ratio and a low time complexity, which can handle the online task offloading and resource management problem with high volatility. Subsequently, we demonstrate the scalability of our proposed algorithm. Figure 10 illustrates the approximation ratio and time complexity of the algorithm under various system scales, i.e., the number of WDs. Our algorithm maintains a consistent approximation ratio and time complexity as the system scale increases. The time complexity remains stable because the forward pass of the deep neural networks can be executed in parallel.

8 Conclusion

In this paper, we formulated and studied the online task offloading and resource management problem in mobile

# of WDs	120	140	160	180	200
PORT	6.9e-03	8.0e-03	8.1e-03	8.8e-03	9.6e-03
MCMC	7.2e+00	9.7e+00	1.0e+01	1.1e+01	1.3e+01
DRL-TOBS	1.8e-03	1.5e-03	1.3e-03	1.3e-03	1.3e-03
WCGA	4.0e-01	5.6e-01	7.7e-01	9.6e-01	1.1e+00
Gurobi	5.5e+01	6.4e+01	4.8e+01	6.7e+01	7.5e+01
Training	3.7e-03	3.9e-03	3.7e-03	4.0e-03	4.0e-03

TABLE 3: Comparison of Time Complexities (second)

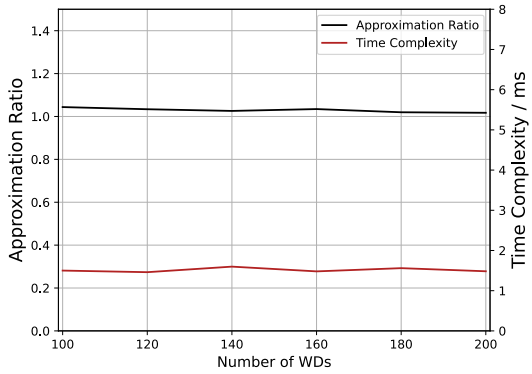


Fig. 11: Scalability of *DRL-TOBS*

edge environments. We proved the problem is NP-hard and proposed a deep reinforcement learning-based approach named *DRL-TOBS*. In particular, we first designed an offline solver named *WCGA* for the formulated problem. The *DRL-TOBS* approach uses deep neural networks to approximate the *WCGA* solver. The *DRL-TOBS* approach observes system states and makes decisions at the beginning of each time slot. Then, the *DRL-TOBS* approach uses the remaining time of each time slot to improve the deep neural networks. We conducted extensive simulations to evaluate the proposed approach. Simulation results show *DRL-TOBS* can converge and is robust to system fluctuations. The decision-making time of *DRL-TOBS* is around one millisecond, which is 5e+04× and 5e+02× faster than an optimal solution solver and the *WCGA* approximation solver, respectively. In addition, the average system latency of *DRL-TOBS* is near-optimal.

REFERENCES

[1] A. Holst, "Number of iot connected devices worldwide 2019–2030," *Statistica*, 2021.

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[3] G. Cui, Q. He, F. Chen, Y. Zhang, H. Jin, and Y. Yang, "Interference-aware game-theoretic device allocation for mobile edge computing," *IEEE Transactions on Mobile Computing*, 2021.

[4] X. Shang, Y. Huang, Z. Liu, and Y. Yang, "Reducing the service function chain backup cost over the edge and cloud by a self-adapting scheme," *IEEE Transactions on Mobile Computing*, vol. 21, no. 8, pp. 2994–3008, 2022.

[5] T. Lawrence, "Evening internet 'rush-hour' affects broadband users - news - gadgets & tech," in *The Independent*, 2013.

[6] B. Yang, X. Cao, J. Bassey, X. Li, and L. Qian, "Computation offloading in multi-access edge computing: A multi-task learning approach," *IEEE Transactions on Mobile Computing*, vol. 20, no. 9, pp. 2745–2762, 2021.

[7] S. Jošilo and G. Dán, "Joint management of wireless and computing resources for computation offloading in mobile edge clouds," *IEEE Transactions on Cloud Computing*, vol. 9, no. 4, pp. 1507–1520, 2021.

[8] S. Jošilo and G. Dán, "Wireless and computing resource allocation for selfish computation offloading in edge computing," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 2467–2475.

[9] S. Jošilo and G. Dán, "Joint wireless and edge computing resource management with dynamic network slice selection," *IEEE/ACM Transactions on Networking*, pp. 1–14, 2022.

[10] Y. Liu, X. Shang, and Y. Yang, "Joint sfc deployment and resource management in heterogeneous edge for latency minimization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 8, pp. 2131–2143, 2021.

[11] L. Huang, S. Bi, and Y.-J. A. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Transactions on Mobile Computing*, vol. 19, no. 11, pp. 2581–2593, 2020.

[12] P. A. Apostolopoulos, G. Fragkos, E. E. Tsiropoulou, and S. Papavassiliou, "Data offloading in uav-assisted multi-access edge computing systems under resource uncertainty," *IEEE Transactions on Mobile Computing*, vol. 22, no. 1, pp. 175–190, 2023.

[13] P. A. Apostolopoulos, E. E. Tsiropoulou, and S. Papavassiliou, "Risk-aware data offloading in multi-server multi-access edge computing environment," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1405–1418, 2020.

[14] G. Li and J. Cai, "An online incentive mechanism for collaborative task offloading in mobile edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 624–636, 2019.

[15] N. Zhang, S. Guo, Y. Dong, and D. Liu, "Joint task offloading and data caching in mobile edge computing networks," *Computer Networks*, vol. 182, p. 107446, 2020.

[16] Y. Chen, N. Zhang, Y. Zhang, X. Chen, W. Wu, and X. S. Shen, "Toffee: Task offloading and frequency scaling for energy efficiency of mobile devices in mobile edge computing," *IEEE Transactions on Cloud Computing*, vol. 9, no. 4, pp. 1634–1644, 2021.

[17] Q. Zhang, L. Gui, F. Hou, J. Chen, S. Zhu, and F. Tian, "Dynamic task offloading and resource allocation for mobile-edge computing in dense cloud ran," *IEEE Internet of Things Journal*, vol. 7, no. 4, pp. 3282–3299, 2020.

[18] Z. Ma, S. Zhang, Z. Chen, T. Han, Z. Qian, M. Xiao, N. Chen, J. Wu, and S. Lu, "Towards revenue-driven multi-user online task offloading in edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 5, pp. 1185–1198, 2021.

[19] J. Ye and Y.-J. A. Zhang, "Drag: Deep reinforcement learning based base station activation in heterogeneous networks," *IEEE Transactions on Mobile Computing*, vol. 19, no. 9, pp. 2076–2087, 2020.

[20] P. Charatsaris, M. Diamanti, E. E. Tsiropoulou, and S. Papavassiliou, "Competitive energy allocation for aerial computation offloading: A colonel blotto game," in *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, 2022, pp. 970–975.

[21] D. Wisely, N. Wang, and R. Tafazolli, "Capacity and costs for 5g networks in dense urban areas," *IET Communications*, vol. 12, no. 19, pp. 2502–2510, 2018.

[22] A. de la Oliva, J. A. Hernandez, D. Larrabeiti, and A. Azcorra, "An overview of the cpri specification and its application to c-ran-based lte scenarios," *IEEE Communications Magazine*, vol. 54, no. 2, pp. 152–159, 2016.

[23] G. Kalfas, C. Vagionas, A. Antonopoulos, E. Kartsakli, A. Mesodiakaki, S. Papaioannou, P. Maniotis, J. S. Vardakas, C. Verikoukis, and N. Pleros, "Next generation fiber-wireless fronthaul for 5g mmwave networks," *IEEE Communications Magazine*, vol. 57, no. 3, pp. 138–144, 2019.

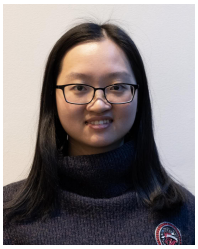
[24] J. Spácil, J. Bohata, D.-N. Nguyen, M. Mazánek, and S. Zvánovec, "Effect of erbium-doped fiber amplifier loss compensation on 5g new radio millimeter-wave seamless transmission over analog fiber and free space optical fronthaul at 60 ghz," *Optical Engineering*, vol. 61, no. 6, p. 066104, 2022.

[25] D. Acatauassu, M. Licá, A. Ohashi, A. L. P. Fernandes, M. Freitas, J. C. Costa, E. Medeiros, I. Almeida, and A. M. Cavalcante, "An efficient fronthaul scheme based on coaxial cables for 5g centralized radio access networks," *IEEE Transactions on Communications*, vol. 69, no. 2, pp. 1343–1357, 2020.

[26] Y. Huo, X. Dong, and W. Xu, "5g cellular user equipment: From

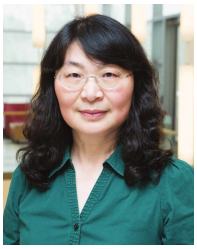
theory to practical hardware design," *IEEE Access*, vol. 5, pp. 13 992–14 010, 2017.
[27] B. Bixby, "The gurobi optimizer," *Transp. Re-search Part B*, vol. 41, no. 2, pp. 159–178, 2007.
[28] M. ApS, "Mosek optimization suite," 2019.
[29] A. Makhorin, "Glpk (gnu linear programming kit)," *http://www.gnu. org/s/glpk/glpk. html*, 2008.
[30] "Technical report." [Online]. Available: https://www.dropbox.com/sh/d52565m18xhw5q8/AAC63CP0Wpad8mGfPUuve-nxa?dl=0
[31] J. Emeras, S. Varrette, V. Plugaru, and P. Bouvry, "Amazon elastic compute cloud (ec2) versus in-house hpc platform: A cost analysis," *IEEE Transactions on Cloud Computing*, vol. 7, no. 2, pp. 456–468, 2019.
[32] G. Huang, S. Liu, L. Van der Maaten, and K. Q. Weinberger, "Condensenet: An efficient densenet using learned group convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2752–2761.
[33] G. P. Agrawal, "Optical communication: its history and recent progress," *Optics in our time*, pp. 177–199, 2016.
[34] J. Bohata, M. Komanec, J. Spáčil, Z. Ghassemlooy, S. Zvánovec, and R. Slavík, "24-26 ghz radio-over-fiber and free-space optics for fifth-generation systems," *Opt. Lett.*, vol. 43, no. 5, pp. 1035–1038, Mar 2018. [Online]. Available: http://opg.optica.org/ol/abstract.cfm?URI=ol-43-5-1035

Yu Liu received his B. Eng. degree with honor in Telecommunication Engineering from Xidian University, Xi'an, China. He is now pursuing his Ph.D. degree in Computer Engineering at Stony Brook University. His research interests are in online algorithms, distributed storage, cloud computing, edge computing and data center networks, with focus on placement and resource management of virtual network functions and reliability of service function chains.



Yingling Mao received the B.S. degree in Mathematics and Applied Mathematics in lZhiyuan College from Shanghai Jiao Tong University, Shanghai, China, in 2018. She is currently working toward the Ph.D degree in the Department of Electrical and Computer Engineering, Stony Brook University. Her research interests include network function virtualization, software-defined network, cloud computing.

Zhenhua Liu is currently assistant professor in the Department of Applied Mathematics and Statistics, also affiliated with Department of Computer Science and Smart Energy Technology Cluster, since August 2014. During the year 2014-2015, he is on leave for the ITRI-Rosenfeld Fellowship in the Energy and Environmental Technology Division at Lawrence Berkeley National Laboratory. Dr. Liu received his Ph.D. degree in Computer Science at the California Institute of Technology, where he was co-advised by Prof. Adam Wierman and Prof. Steven Low. Before Caltech, he received an M.S. degree of Computer Science Technology in 2009 and a B.E. degree of Measurement control in 2006, both from Tsinghua University with honor, as well as a B.S. degree of Economics from Peking University in 2009.



Yuanyuan Yang received the BEng and MS degrees in computer science and engineering from Tsinghua University, Beijing, China, and the MSE and Ph.D. degrees in computer science from Johns Hopkins University, Baltimore, Maryland. She is a SUNY Distinguished Professor of computer engineering and computer science at Stony Brook University, New York, and is currently on leave at the National Science Foundation as a Program Director. Her research interests include edge computing, data center networks, cloud computing and wireless networks. She has published more than 460 papers in major journals and refereed conference proceedings and holds seven US patents in these areas. She is currently the Editor-in-Chief for IEEE Transactions on Cloud Computing and an Associate Editor for IEEE Transactions on Parallel and Distributed Systems and ACM Computing Surveys. She has served as an Associate Editor-in-Chief for IEEE Transactions on Cloud Computing, Associate Editor-in-Chief and Associated Editor for IEEE Transactions on Computers, and Associate Editor for IEEE Transactions on Parallel and Distributed Systems. She has also served as a general chair, program chair, or vice chair for several major conferences and a program committee member for numerous conferences. She is an IEEE Fellow.