




Delay Minimization for NOMA-enabled Mobile Edge Computing in Industrial Internet of Things

Van Dat Tuong , Wonjong Noh , *Senior Member, IEEE*, and Sungrae Cho , *Member, IEEE*

Abstract—Mobile edge computing and nonorthogonal multiple access (NOMA) have been considered as promising technologies that can satisfy rigorous requirements of industrial internet of things systems. However, system dynamics, including channel states and computation task requests, may continuously change NOMA decoding order and computation uploading time, making it difficult to reduce latency using conventional highly complex optimization methods. In this study, we investigate a novel scheme that effectively reduces the average task delay to improve the quality of service for all users by jointly optimizing subchannel assignment (SA), offloading decision (OD), and computation resource allocation (CRA). To deal with the high complexity, the original multi-server problem is first decomposed into multiple single-server problems. Subsequently, each single-server problem is decoupled into CRA and SA/OD subproblems. Using convex optimization, a closed-form solution is derived for the optimal CRA action. Concurrently, the optimal SA/OD action is obtained using a distributed multi-agent deep reinforcement learning algorithm. Simulation results reveal that the proposed scheme significantly outperforms the state-of-the-art schemes. In particular, it reduces the action decision duration by 30 times while achieving a near-optimal performance of up to 97% of the optimum under the exhaustive search scheme.

Index Terms—Delay minimization, industrial internet of things, mobile edge computing, nonorthogonal multiple access, reinforcement learning, resource allocation.

I. INTRODUCTION

THE surge increase in industrial Internet of Things (IoT) and wireless network technologies has led to the rapid development of many computation-intensive and delay-sensitive applications, such as smart manufacturing, industrial surveillance, and predictive maintenance. Concurrently, IoT devices are generally computationally constrained, presenting poor performance when implementing computation-intensive workloads. To support users offloading tasks effectively, mobile edge computing (MEC) was proposed to distribute computing resources and application services from the core network to the edge (access) network [1]. Instead of using expensive and high-latency backhaul links like cloud computing, MEC promotes the implementation of applications on the access network to the maximal extent. Thus, it can significantly

reduce network traffic, network bottleneck, and transmission costs [2]–[5].

Nevertheless, as the number of IoT devices in an industrial network rapidly increases, the limited (access) network capacity becomes a major challenge. For example, when numerous devices upload computation tasks simultaneously, there is frequent network congestion [6]. To deal with this challenge, nonorthogonal multiple access (NOMA) has been proposed for fifth-generation and sixth-generation networks [7]. By allowing users to simultaneously transmit data over the same resource block (RB) and employing successive interference cancellation (SIC) to decode individual user signals, NOMA can accommodate more users than the conventional orthogonal multiple access technique, resulting in spectral efficiency improvement and delay reduction [8], [9].

Motivated by the advantages of NOMA and MEC, this study aims to minimize the average task delay for users in an industrial IoT network, where the main contributions can be summarized as follows:

- A joint optimization problem of subchannel assignment (SA), offloading decision (OD), and computation resource allocation (CRA) was formulated to minimize the average task delay for all users. Different from existing schemes, the formulated problem characterized the interaction between uploading delay and system dynamics, e.g., channel conditions and computation task requests, allowing changing SIC decoding order and task uploading durations at different times. Furthermore, the formulated problem was nontrivial to solve owing to its nonconvex objective function and mixed types of variables.
- Because of the high complexity, multi-server problem was transformed into multiple single-server problems, where users selected the server with the strongest uplink reference signal received power. Subsequently, each single-server problem was decoupled into CRA and SA/OD subproblems, which were solved iteratively. The optimal CRA action was obtained in a closed-form expression using convex optimization, and the optimal SA/OD action was obtained using a novel multi-agent deep reinforcement learning (MA-DRL) algorithm.
- The complexity of the proposed algorithm was polynomial that proved the feasibility for application in practical systems. In addition, numerical results demonstrated that the proposed algorithm reliably converged and presented a near-optimal performance in terms of average task delay. Specifically, compared to existing schemes, the proposed scheme significantly reduced action decision time to approximately 64 ms, which can satisfy the

Manuscript received May 18, 2021; revised September 1, 2021; accepted September 28, 2021. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean Government (Ministry of Science and ICT) under Grants 2017R1D1A1B03036526, 2019R1A2C1090447, and 2020R1F1A1069119. (*Corresponding authors: Wonjong Noh and Sungrae Cho.*)

Van Dat Tuong and Sungrae Cho are with the School of Computer Science and Engineering, Chung-Ang University, Seoul 06974, Republic of Korea. (*email: vdtuong@uclab.re.kr, srcho@cau.ac.kr*)

Wonjong Noh is with the School of Software, Hallym University, Chunchon 24252, Republic of Korea. (*email: wonjong.noh@hallym.ac.kr*)

stringent delay constraint of practical systems.

The remainder of this paper is organized as follows. Section II provides the literature review. In section III, system model and problem formulation are described. Section IV presents our proposed solution. The simulation results are discussed in section V. Finally, section VI concludes the study.

II. LITERATURE REVIEW

In recent years, the effectiveness of a NOMA–MEC system, which utilizes the advantages of both MEC and NOMA, has been demonstrated in various optimization frameworks [10]–[13]. Ding *et al.* [10] revealed that employing NOMA can efficiently reduce latency and energy consumption of MEC offloading. Huynh *et al.* [11] considered a cache-aided NOMA–MEC scheme, aiming to minimize the overall completion latency. A block successive upper-bound minimization method was implemented to solve the joint optimization problem of offloading and caching policy. Wu *et al.* [12] proposed a partial offloading scheme based on NOMA transmission. The overall completion latency was minimized by jointly optimizing computation offloading policy and durations for uploading task and downloading result. Sheng *et al.* [13] investigated a delay minimization scheme under differentiated uploading delay. The optimal offloading policy and resource allocation were determined using convex optimization incorporated with semidefinite relaxation, convex–concave procedure, and matching theory.

In practical systems, hyperparameters, such as channel conditions and computation task requests, frequently change, leading to the challenge of rapidly performing conventional optimization methods within extremely short periods. In this context, machine learning-based solutions have been proposed [14]–[20]. Several studies considered simple schemes with a single server [14]–[16]. Yang *et al.* [14] developed a Q-learning-based framework to reduce energy consumption for cache-aided NOMA–MEC networks. However, because Q-learning relies on a look-up Q-table, it can not adapt to large problems. Considering the advantage of deep neural networks in evaluating Q-values, the authors in [15] investigated the offloading delay minimization for a multi-user NOMA–MEC scenario using Deep Q-network (DQN) algorithm. Chen *et al.* [16] developed a cooperative MA-DRL algorithm to learn the optimal offloading policy, considering each user as an agent. Based on a sharing policy network, each agent selects its own action to minimize a mixed cost of consumed power and latency. This design is not suitable for application in heterogeneous networks where the gap in policy between different agents is significant.

Comparing to single-server schemes in the above, multi-server schemes in [17]–[20] were more practical. Ning *et al.* [17] investigated a distributed offloading framework for 5G-enabled vehicular networks. As a result, spectrum resources were effectively allocated using a MA-DQN algorithm that improves quality of service for all users (V2I and V2R users). Wang *et al.* [18] considered a distributed offloading scheme, where helper nodes with rich computation resources can provide computing services to nearby nodes. An optimization

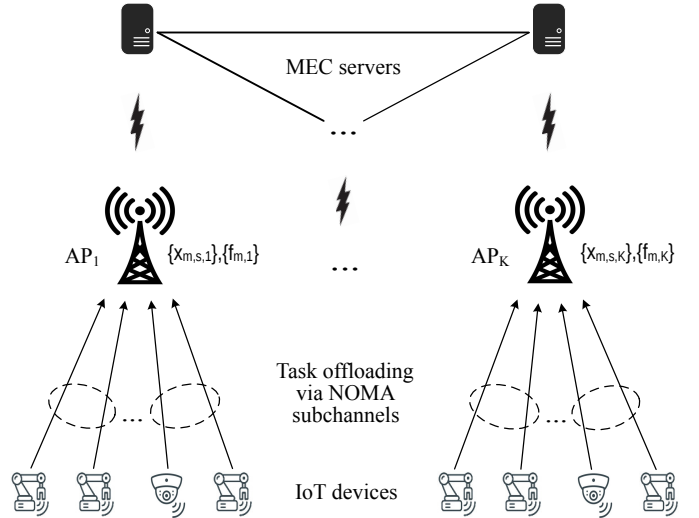


Fig. 1. Computation offloading in multi-carrier multi-server NOMA-enabled mobile edge computing for industrial IoT networks.

problem was formulated to minimize energy and delay cost by jointly optimizing task scheduling and resource allocation. In particular, a MA-Q-learning framework was proposed, which incorporated matching theory for subcarrier allocation and convex optimization for power and computation resource allocations. In [19], the energy and delay cost minimization was conducted for ultra-dense NOMA–MEC networks, where user clustering was first obtained using a matching algorithm, and power and computation resource allocations were subsequently determined using a mean field deep deterministic policy gradient (MF-DDPG) algorithm. Li *et al.* [20] aimed at minimizing total cost for a blockchain-empowered IoT network. They proposed a MA-actor-critic algorithm to learn the offloading policy, where agents cooperated in a hierarchical league framework. Despite of the reliable convergence demonstrated in both schemes [19] and [20], extra communication overhead incurred in their cooperative scenarios may degrade the overall performance.

The proposed scheme in this study is very close to the schemes in [13], [18]–[20]. We summarize the comparisons between the proposed scheme and these state-of-the-art schemes in Table I.

III. SYSTEM MODEL AND PROBLEM FORMULATION

This study considers smart manufacturing systems, e.g., smart factories, which can quickly adapt to various mass productions without changing inherent resources, and thus, improving productivity and minimizing time to market. To do this, they adopt a communication system to interconnect intelligent machines and IoT devices (IoTDs) which implement lots of industrial applications. In particular, this study focuses on product tracking and automated robot, which are delay-sensitive and computing-hungry, aiming to minimize the system service delay. Furthermore, all devices are assumed to have power constraints and be aware of the environment, network information, and control processes in production.

TABLE I
COMPARISONS BETWEEN PROPOSED SCHEME AND STATE-OF-THE-ART SCHEMES

Scheme	Utilized Technique	Advantage	Disadvantage
Sheng <i>et al.</i> [13]	Convex optimization, semidefinite relaxation, and matching theory	Considered differentiated uploading delay	Considered a single-server scheme and remained high complexity
Wang <i>et al.</i> [18]	Convex optimization, matching theory, and MA-Q-learning	Guaranteed convergence	Used look-up Q-tables and remained high complexity
Li <i>et al.</i> [19]	Matching theory and MA-DDPG under a mean field game	Achieved reliable convergence	Generated extra communication overhead
Li <i>et al.</i> [20]	MA-actor-critic under a hierarchical league framework	Achieved reliable convergence	Generated extra communication overhead
Proposed scheme	Convex optimization and MA-DQN	Considered differentiated uploading delay, achieved reliable convergence and low complexity	Did not consider power allocation problem

TABLE II
LIST OF KEY NOTATIONS

Notation	Definition
\mathcal{K}	Set of K access points (APs)
\mathcal{M}	Set of M IoT devices (IoTDs)
\mathcal{M}_k	Set of IoTDs associating with AP k
\mathcal{M}_k^1	Set of offloading IoTDs of AP k
\mathcal{S}	Set of S subchannels (SCs)
F	Computation capacity of each MEC server
f_m	Computation capacity of IoTD m
p_m	Transmit power of IoTD m
\mathcal{T}	Duration of τ timing epochs
$h_{m,s,k}(t)$	Uplink channel gain between IoTD m and AP k on SC s
$d_m(t)$	Task data size in bits of IoTD m at time t
$c_m(t)$	Task workload in CPU-cycles of IoTD m at time t
$\mathbf{x}(t)$	Subchannel assignment (SA) vector at time t
$\mathbf{x}_k(t)$	SA vector of AP k at time t
$x_{m,s,k}(t)$	Assignment of SC s from AP k to IoTD m at time t
$\mathbf{f}(t)$	Computation resource allocation (CRA) vector at time t
$\mathbf{f}_k(t)$	CRA vector of AP k at time t
$f_{m,k}(t)$	CRA of AP k for IoTD m at time t
$R_{m,n,s,k}(t)$	Uplink rate of IoTD m to AP k via SC s under NOMA interference from IoTD n
$R_{m,s,k}(t)$	Uplink rate of IoTD m to AP k via SC s without NOMA interference
$I_{m,n,s,k}(t)$	NOMA interference from IoTD n to IoTD m
$I_{m,s,k}(t)$	Inter-cell interference impacted to IoTD m
$t_m(t)$	Time required to locally execute a task of IoTD m
$t_m^k(t)$	Time required to remotely execute a task of IoTD m at MEC server k
$t_m^u(t)$	Uploading time of IoTD m (solely utilize a SC)
$t_{m,n}^u(t)$	Total uploading time of NOMA paired users m and n
$T_k(t)$	Overall delay for completing tasks of the users associated with AP k

As shown in Fig. 1, the network model consists of multiple access points (APs), each of which is connected to a MEC server. There are lots of IoTDs distributed around each AP. We denote APs, IoTDs, and subchannels (SCs) by $k \in \mathcal{K} = \{1, \dots, K\}$, $m \in \mathcal{M} = \{1, \dots, M\}$, and $s \in \mathcal{S} = \{1, \dots, S\}$, respectively. Each MEC server represents a processor, whose computation capacity is limited to F CPU-cycles per second. The network is assumed to operate over a timing basis, in which time is partitioned into τ timing epochs denoted by $t \in \mathcal{T} = \{1, \dots, \tau\}$. Within each epoch, all network parameters remain unchanged and a computation task may arrive at any user, requiring to execute locally or offload to a MEC server. The binary offloading scheme is considered, in which OD is 1 if a user offloads computation to MEC server and 0 if it executes locally [21]. Key notations used in this study are listed in Table II.

Moreover, we assume that all APs operate in NOMA to improve spectrum efficiency and reduce latency. Utilizing NOMA, multiple IoTDs can transmit data to an AP on the same RB. Specifically, NOMA superimposes signals of different IoTDs before transmitting them via a single RB. As a result, the number of users that can be served concurrently would be increased. When AP receives the superimposed signal, SIC is utilized to decode the actual signals. For uplink NOMA, the SIC decoding order is always from the better user to the worse user [22], in which the worse user's signal is considered as NOMA interference when decoding the better user's signal. Furthermore, if many IoTDs are active on one SC, the SC may become overloaded and the decoding process can fail. To address this problem, similar to [23], [24], we assume that each SC is shared by at most two users.

A. System Dynamics

Users, such as industrial robots, frequently move or change location, varying channel conditions between them and the APs. In addition, workload of each user typically varies. Thus, considering system dynamics such as the variation in channels and computation task requests becomes a necessity. Let $h_{m,s,k}(t)$ denote the channel gain between user m and AP k on SC s , affected by path loss, shadowing loss, antenna gain, and instantaneous fading process. Computation tasks of user m are denoted by $w_m(t) \triangleq \{d_m(t), c_m(t)\}$, where $d_m(t)$ is the data size in bits and $c_m(t)$ is the workload in CPU-cycles. The rapid variation in system dynamics makes it challenging to perform the optimal joint control of SA, OD, and CRA.

We denote SA vector as $\mathbf{x}(t) \triangleq \{x_{m,s,k}(t) | \forall m \in \mathcal{M}, \forall s \in \mathcal{S}, \forall k \in \mathcal{K}\}$, where $x_{m,s,k}(t) \in \{0, 1\}$ represents the assignment of SC s from AP k to user m , i.e., $x_{m,s,k}(t) = 1$ if user m is assigned SC s of AP k , and $x_{m,s,k}(t) = 0$ otherwise. In practice, as the number of users is frequently larger than that of SCs, a user who is assigned a SC should utilize it to offload task. Consequently, $\mathbf{x}(t)$ can represent OD, in which $x_{m,s,k}(t) = 1$ implies that user m offloads computation to the MEC server associated with AP k via SC s , and $\sum_{s,k} x_{m,s,k}(t) = 0$ when user m is assigned no SC, implying that it executes task locally. The CRA vector is denoted as $\mathbf{f}(t) \triangleq \{f_{m,k}(t) | \forall m \in \mathcal{M}, k \in \mathcal{K}\}$, where $f_{m,k}(t) \geq 0$ represents CRA of AP k to user m .

B. Task Execution Delay

1) *Local Execution*: Computation tasks are processed directly using local IoTDS. Therefore, time required to locally execute a task of user m is computed as

$$t_m(t) = \left(1 - \sum_{k \in \mathcal{K}} \sum_{s \in \mathcal{S}} x_{m,s,k}(t)\right)^+ \frac{c_m(t)}{f_m}, \quad (1)$$

where f_m is the computation capacity of IoTDS m in CPU-cycles per second and $(*)^+$ represents a non-negative function, i.e., $(*)^+ = 0$ if $(*) < 0$.

2) *MEC Execution*: In MEC execution, computation tasks are processed by MEC servers, instead of local IoTDS. It is assumed that each user offloads its task to only one MEC server. Time required to remotely execute a task of user m at MEC server k is computed as

$$t_m^k(t) = x_{m,s,k}(t) \frac{c_m(t)}{f_{m,k}(t)}. \quad (2)$$

C. Task Uploading Delay

If a user decides to offload task to a MEC server, there exists a task uploading delay which depends on its uplink rate. Consider the following cases:

- With NOMA interference: If a user m shares a SC s of AP k with a worse user n ($h_{m,s,k}(t) \geq h_{n,s,k}(t)$), who is transmitting data, NOMA decodes the signal of user m considering the signal of user n as NOMA interference. In this case, uplink rate of user m to AP k via SC s is computed as

$$R_{m,n,s,k}(t) = B \log_2 \left(1 + \frac{p_m h_{m,s,k}(t)}{I_{m,s,k}(t) + I_{n,s,k}(t) + \sigma^2}\right), \quad (3)$$

where B is the SC bandwidth, p_m is the transmit power of IoTDS m , σ^2 is the additive white Gaussian noise power spectral density, $I_{m,n,s,k}(t) = p_n h_{n,s,k}(t)$ is NOMA interference, and $I_{m,s,k}(t)$ is the intercell interference, which is calculated as

$$I_{m,s,k}(t) = \sum_{k' \in \mathcal{K} \setminus k} \sum_{o \in \mathcal{M} \setminus m} x_{o,s,k'}(t) p_o h_{o,s,k}(t). \quad (4)$$

- Without NOMA interference: A user m has no NOMA interference in one of the following cases: (i) it solely occupies a SC s , (ii) it shares a SC s with a better user n ($h_{m,s,k}(t) < h_{n,s,k}(t)$), and (iii) it shares a SC s with a worse user n ($h_{m,s,k}(t) \geq h_{n,s,k}(t)$) but user n finishes transmitting or does not transmit its signal. In these cases, uplink rate of user m without NOMA interference is computed as

$$R_{m,s,k}(t) = B \log_2 \left(1 + \frac{p_m h_{m,s,k}(t)}{I_{m,s,k}(t) + \sigma^2}\right). \quad (5)$$

Considering an AP k , if a user m solely utilizes a SC s , its task uploading time is computed as

$$t_m^u(t) = x_{m,s,k}(t) \left(1 - \sum_{n \in \mathcal{M} \setminus m} x_{n,s,k}(t)\right)^+ \frac{d_m(t)}{R_{m,s,k}(t)}. \quad (6)$$

Otherwise, if SC s is shared by a pair of users, e.g., m and n with $h_{m,s,k}(t) \geq h_{n,s,k}(t)$, the signal of user n is decoded without NOMA interference, resulting in a task uploading time computed as

$$t_n^u(t) = x_{m,s,k}(t) x_{n,s,k}(t) \frac{d_n(t)}{R_{n,s,k}(t)}. \quad (7)$$

To precisely compute task uploading time of user m , the following two derivation scenarios are considered:

- Scenario \mathbf{S}_1 ($\frac{d_m(t)}{R_{m,n,s,k}(t)} \leq \frac{d_n(t)}{R_{n,s,k}(t)}$): In this scenario, user m finishes uploading no later than user n ; hence, it experiences NOMA interference throughout its uploading process. Accordingly, task uploading time of user m is computed as

$$t_m^{u,\mathbf{S}_1}(t) = x_{m,s,k}(t) x_{n,s,k}(t) \frac{d_m(t)}{R_{m,n,s,k}(t)}. \quad (8)$$

- Scenario \mathbf{S}_2 ($\frac{d_m(t)}{R_{m,n,s,k}(t)} > \frac{d_n(t)}{R_{n,s,k}(t)}$): During the uploading process of user n , user m experiences NOMA interference. However, after user n finishes uploading, user m uploads its remaining data without NOMA interference. Therefore, task uploading time of user m is computed as

$$t_m^{u,\mathbf{S}_2}(t) = x_{m,s,k}(t) x_{n,s,k}(t) \frac{d'_m(t)}{R_{m,s,k}(t)} + t_n^u(t), \quad (9)$$

where $d'_m(t) = d_m(t) - R_{m,n,s,k}(t) t_n^u(t)$ is the remaining data of user m when user n finishes uploading.

Note that uploading time of user n is computed the same in scenarios \mathbf{S}_1 and \mathbf{S}_2 , i.e., $t_n^{u,\mathbf{S}_1}(t) = t_n^{u,\mathbf{S}_2}(t) = t_n^u(t)$. Thus, total task uploading time of paired users, m and n , is computed as

$$t_{m,n}^u(t) = \begin{cases} t_m^{u,\mathbf{S}_1}(t) + t_n^{u,\mathbf{S}_1}(t) & \text{if scenario } \mathbf{S}_1, \\ t_m^{u,\mathbf{S}_2}(t) + t_n^{u,\mathbf{S}_2}(t) & \text{if scenario } \mathbf{S}_2. \end{cases} \quad (10)$$

D. Problem Formulation

This study aims at minimizing the average task delay of all users across τ offloading epochs. To reduce the high complexity of the multi-server problem, we assume that each user selects its serving AP with the strongest reference signal received power in the uplink. Following this, each AP k is selected by a number of users, forming its user group denoted by \mathcal{M}_k . Because each user selects only one AP with the strongest uplink signal, there is no intersection between user sets. Based on deterministic user sets, the multi-server problem can be divided into distributed single-server problems, each of which is for an AP as follows:

$$\min_{\mathbf{x}_k(t), \mathbf{f}_k(t)} \sum_{t=1}^{\tau} \frac{T_k(t)}{|\mathcal{M}_k|}, \quad (11)$$

$$\text{s.t. } \mathbf{C1} : x_{m,s,k}(t) \in \{0, 1\}, \forall m \in \mathcal{M}_k, \forall s \in \mathcal{S},$$

$$\mathbf{C2} : \sum_{s \in \mathcal{S}} x_{m,s,k}(t) \leq 1, \forall m \in \mathcal{M}_k,$$

$$\mathbf{C3} : \sum_{m \in \mathcal{M}_k} x_{m,s,k}(t) \leq 2, \forall s \in \mathcal{S},$$

$$\mathbf{C4} : f_{m,k}(t) \geq 0, \forall m \in \mathcal{M}_k,$$

$$\mathbf{C5} : \sum_{m \in \mathcal{M}_k} f_{m,k}(t) \leq F,$$

where $\mathbf{x}_k(t) \triangleq \{x_{m,s,k}(t) | \forall m \in \mathcal{M}_k, \forall s \in \mathcal{S}\}$ is the SA of AP k , $\mathbf{f}_k(t) \triangleq \{f_{m,k}(t) | \forall m \in \mathcal{M}_k\}$ is the CRA for associated users of AP k , and $T_k(t)$ is the overall delay for completing tasks of these users. As per the binary offloading scheme, for each user, the local and MEC computations cannot be carried out simultaneously. Based on the derivations of the task execution and uploading delays, the local execution time is 0 (not contribute to the overall delay) if offloading task, and the uploading time and the MEC execution time are both 0 (not contribute to the overall delay) if local computing. Therefore, $T_k(t)$ can be computed as follows:

$$T_k(t) = \sum_{m \in \mathcal{M}_k} \left(t_m(t) + \sum_{s \in \mathcal{S}} \left(t_m^k(t) + t_m^u(t) + \sum_{n \in \mathcal{M}_k | n > m} t_{m,n}^u(t) \right) \right), \quad (12)$$

where $t_m^u(t)$ is the task uploading time of user m who solely utilizes one SC, and $t_{m,n}^u(t)$ is the task uploading time of NOMA paired users m and n . Constraints **C1** and **C2** describe SA action that any user is assigned at most one SC. Constraint **C3** states that any SC is shared by at most two users. Moreover, constraints **C4** and **C5** indicate that the total allocated computation resources do not exceed the computation capacity of each MEC server.

The distributed problem expressed in (11) is a mixed-integer and nonconvex optimization problem because of the binary and numerical variables as well as the nonconvex objective function. In general, this type of a problem is NP-hard, typically requiring exponential time complexity to find the optimal solution [25]. Furthermore, owing to system dynamics, numerous system states are generated over time, leading to the challenge of applying an one-shot optimization solution in practice. To address this problem, we focus on developing a novel efficient solution.

IV. PROPOSED SOLUTION

Considering the large action space including SA $\mathbf{x}_k(t)$ and CRA $\mathbf{f}_k(t)$, we decompose the problem expressed in (11) into CRA and SA subproblems. First, it is transformed into a local CRA subproblem, aiming to optimize CRA for each AP when SA is given and $t_m(t)$, $t_m^u(t)$, and $t_{m,n}^u(t)$ become constants. The CRA subproblem is formulated as follows:

$$\begin{aligned} \min_{\{\mathbf{f}_k(t) | \forall k \in \mathcal{K}\}} \quad & F(\mathbf{f}_k(t)) \triangleq \sum_{t=1}^{\tau} \sum_{m \in \mathcal{M}_k^1} \sum_{s \in \mathcal{S}} t_m^k(t), \quad (13) \\ \text{s.t.} \quad & \mathbf{C4}, \mathbf{C5}, \end{aligned}$$

where $\mathcal{M}_k^1 \subset \mathcal{M}_k$ is the set of offloading users of AP k , determined when SA is given. The subproblem (13) is a typical convex problem, which is solved by Lagrangian dual decomposition using the Karush–Kuhn–Tucker (KKT) conditions. Afterward, under CRA of each AP, the problem expressed in (11) is transformed to the SA subproblem as

$$\begin{aligned} \min_{\{\mathbf{x}_k(t) | \forall k \in \mathcal{K}\}} \quad & \sum_{t=1}^{\tau} \frac{T_k(t)}{|\mathcal{M}_k|}, \quad (14) \\ \text{s.t.} \quad & \mathbf{C1}, \mathbf{C2}, \mathbf{C3}. \end{aligned}$$

A. A Closed-form Solution for CRA

Constraints **C4** and **C5** are convex. In addition, the second-order derivatives of the objective function expressed in (13) are non-negative as follows:

$$\frac{\partial^2 F(\mathbf{f}_k(t))}{\partial f_{m,k}^2(t)} = \frac{2c_m(t)}{f_{m,k}^3(t)} > 0, \forall m \in \mathcal{M}_k^1, \quad (15)$$

$$\frac{\partial^2 F(\mathbf{f}_k(t))}{\partial f_{m,k}(t) \partial f_{n,k}(t)} = 0, \forall m, n \in \mathcal{M}_k^1, m \neq n. \quad (16)$$

Because the Hessian matrix of the objective function expressed in (13) is diagonal with strictly positive elements, it is positive-definite. Therefore, the CRA subproblem is a convex optimization problem. We derive its Lagrangian dual function as

$$\begin{aligned} \mathcal{L}(F, \nu, \kappa) = \sum_{m \in \mathcal{M}_k^1} \frac{c_m(t)}{f_{m,k}(t)} + \nu \left(\sum_{m \in \mathcal{M}_k^1} f_{m,k}(t) - F \right) \\ - \sum_{m \in \mathcal{M}_k^1} \kappa_m f_{m,k}(t), \quad (17) \end{aligned}$$

where ν and $\kappa = \{\kappa_m | \forall m \in \mathcal{M}_k^1\}$ are the Lagrangian multipliers ($\nu \geq 0$, $\kappa_m \geq 0 \forall m \in \mathcal{M}_k^1$). The derivatives of the Lagrangian dual function w. r. t. $\mathbf{f}_k(t)$ are calculated as

$$\frac{\partial \mathcal{L}(F, \nu, \kappa)}{\partial f_{m,k}(t)} = -\frac{c_m(t)}{f_{m,k}^2(t)} + \nu - \kappa_m, \forall m \in \mathcal{M}_k^1. \quad (18)$$

Applying KKT conditions, the optimal values for ν , κ , and $\mathbf{f}_k(t)$ are obtained as follows:

$$\kappa_m^* = 0, \forall m \in \mathcal{M}_k^1, \quad (19)$$

$$\nu^* = \left(\frac{1}{F} \sum_{m \in \mathcal{M}_k^1} \sqrt{c_m(t)} \right)^2, \quad (20)$$

$$f_{m,k}^*(t) = \frac{F \sqrt{c_m(t)}}{\sum_{m \in \mathcal{M}_k^1} \sqrt{c_m(t)}}, \forall m \in \mathcal{M}_k^1. \quad (21)$$

B. MA-DRL Algorithm for SA

The SA subproblem (14) can be characterized as an RL task based on a markov decision process (MDP) model. We propose to train the optimal SA using DRL, an emerging tool in wireless network control. The single-agent DRL model can be used, in which a centralized agent collects related information from all APs, and trains the overall SA action by iteratively interacting with the environment. It has some drawbacks. First, it increases the time required to decide an SA action because the centralized agent must wait for related information from all APs. Second, the sizes of state and action spaces are large as they are proportional to the number of APs and users, resulting in a long time for training the optimal SA action. To tackle these drawbacks, we design a MA-DRL model in which each AP employs a DRL agent to train its own optimal SA action. The state space, action space, and reward function of the MDP model are defined as below.

- 1) *State space*: At time epoch t , a state is formulated for each AP k as

$$\mathbb{S}_k(t) \triangleq \{\mathbf{h}(t), \mathbf{d}_k(t), \mathbf{c}_k(t), \mathbf{x}(t^-)\}, \quad (22)$$

where $\mathbf{h}(t) \triangleq \{h_{m,s,k}(t) | \forall m \in \mathcal{M}, \forall s \in \mathcal{S}, \forall k \in \mathcal{K}\}$ is the channel gain state vector, $\mathbf{d}_k(t) \triangleq \{d_m(t) | \forall m \in \mathcal{M}\}$ and $\mathbf{c}_k(t) \triangleq \{c_m(t) | \forall m \in \mathcal{M}\}$, such that $d_m(t) \leftarrow 0$ and $c_m(t) \leftarrow 0$ for all $m \notin \mathcal{M}_k(t)$, are the vectors of task size and workload states, respectively, and $\mathbf{x}(t^-)$ is the overall SA state vector updated at the beginning of time epoch t . Here, each AP updates its channel and SA states to other APs using a wired link connection.

- 2) *Action space*: At time t , after formulating state $\mathbb{S}_k(t)$, DRL agent of AP k takes an SA action as

$$\mathbf{a}_k(t) \triangleq \mathbf{x}_k(t), \quad (23)$$

where $\mathbf{x}_k(t) = \{x_{m,s,k}(t) | \forall m \in \mathcal{M}_k, \forall s \in \mathcal{S}\}$ is the SA focusing on the users who associate with AP k . Following this, the dimension of the action space is significantly reduced that improves the scalability.

- 3) *Reward function*: The step reward, $r_k(t)$, for each DRL agent k is taken as the negative of the average delay of the users who associate with AP k . Based on the observed state, $\mathbb{S}_k(t)$, and the instant action, $\mathbf{a}_k(t)$, the step reward is computed as

$$r_k(t) \triangleq -\frac{T_k(t)}{|\mathcal{M}_k|}. \quad (24)$$

Specifically, intercell and NOMA interferences and up-link rates of the associated users of AP k are calculated based on $\mathbf{h}(t)$, $\mathbf{x}(t^-)$, and $\mathbf{a}_k(t)$. Here, $\mathbf{x}(t^-)$ and $\mathbf{a}_k(t)$ are combined to determine an immediate overall SA. Moreover, offloading users in \mathcal{M}_k^1 are updated based on $\mathbf{a}_k(t)$. Subsequently, the optimal CRA is calculated using (21). Finally, the step reward is computed, which requires $\mathbf{d}_k(t)$ and $\mathbf{c}_k(t)$ for calculating task execution delay and task offloading delay.

We relax and transform the SA subproblem expressed in (14) to an RL task problem as follows:

$$\max_{\{\mathbf{a}_k(t) | \forall k \in \mathcal{K}\}} \mathbb{E} \left[\sum_{t=1}^{\tau} \gamma^t r_k(t) \right], \quad (25)$$

where γ^t is the discounting factor and constraints **C1–C3** are ensured by checking generated action, $\mathbf{a}_k(t)$, before processing. For each DRL agent in (25), it is trained toward the optimal action that maximizes its long-term reward, i.e., the expectation of the negative of the long-term average delay. According to [26] and [27], state components including $\mathbf{h}(t)$ and $\{\mathbf{d}_k(t), \mathbf{c}_k(t)\}$ can be modelled into Markov chains. To be more specific, they can be specified based on deterministic levels of channel gains and computation task types, respectively. At each epoch, channel state and computation task request jump from one state to another state based on transition probability matrices. Considering a specific industrial network, e.g., a smart factory network, we assume that the transition probability matrices are pre-defined. Following this, we can model the Markov chains offline on a cloud server.

Accordingly, offline training for each agent is implemented on the cloud server based on the modelled Markov chains of channel states and computation task requests. The training result is updated to each AP to make its action online that significantly shortens the action decision time.

Under constraints **C1–C3**, the SA action space for each DRL agent is discrete. Therefore, value-based DRL algorithms, such as DQN and deep SARSA, can be deployed to train SA action. Compared to deep SARSA, DQN is an off-policy algorithm, which may require more time to converge. However, training with DQN frequently provides higher return than with deep SARSA owing to its efficient sampling [28]. In this study, a DQN-based algorithm is investigated to train distributed DRL agents at the APs. The proposed MA-DRL algorithm can be described as follows:

- First, each DRL agent k is initialized with a replay buffer of size D , primary DQN \mathcal{Q}_k , and target DQN $\hat{\mathcal{Q}}_k$ with weight matrices θ_k and $\hat{\theta}_k$, respectively. Primary and target DQNs have the same neural network structure, comprising two fully connected hidden layers implemented together with input and output layers. In addition, each hidden layer has H neurons, and utilizes rectified linear unit (ReLU) as the activation function.
- The training process iterates some steps. It begins by re-ordering users as descending channel gain and resetting initial states, $\mathbb{S}_k(0)$, for all DRL agents. For each decision step t , each DRL agent obtains current state, $\mathbb{S}_k(t)$, and determines its SA action $\mathbf{a}_k(t)$ based on ϵ -greedy strategy. The strategy starts with a reasonably randomized action (probability ϵ is initially large and close to 1) and gradually moves to a strategic action (probability ϵ is decayed in each step until a small value close to 0). Once SA action is decided, the SIC decoding order of all users is updated based on the re-ordered user list and the updated NOMA user pairs. Subsequently, CRA action, $\mathbf{f}_k(t)$, is calculated based on (21). Performing SA action $\mathbf{a}_k(t)$, step reward, $r_k(t)$, and evolved state, $\mathbb{S}_k(t+1)$, are observed. The experience tuple $(\mathbb{S}_k(t), \mathbf{a}_k(t), r_k(t), \mathbb{S}_k(t+1))$ is saved in the replay buffer. As the replay buffer size is fixed, when memory becomes full, a new experience replaces the oldest one. The experiences will be used as local training data.
- In the learning procedure, each DRL agent samples a random mini-batch of C experiences from the replay buffer ($C \ll D$), and feeds into DQNs, where Q-values act as the output. A target Q-value [29] is defined as

$$y_k = r_k(t) + \gamma \mathcal{Q}_k(\mathbb{S}_k(t+1), \mathbf{a}_k(t+1); \theta_k), \quad (26)$$

where $\mathbf{a}_k(t+1) = \arg\max_{\mathbf{a}_k} \hat{\mathcal{Q}}_k(\mathbb{S}_k(t+1), \mathbf{a}_k; \hat{\theta}_k)$. Each DRL agent aims to minimize the following loss function:

$$L(\theta_k) = \mathbb{E} \left[(y_k - \mathcal{Q}_k(\mathbb{S}_k(t), \mathbf{a}_k(t); \theta_k))^2 \right]. \quad (27)$$

The gradient with respect to primary weight, θ_k , of Q-value loss is calculated as follows:

$$\nabla_{\theta_k} L(\theta_k) = \mathbb{E} [y_k - \mathcal{Q}_k(\mathbb{S}_k(t), \mathbf{a}_k(t); \theta_k) \times \nabla_{\theta_k} \mathcal{Q}_k(\mathbb{S}_k(t), \mathbf{a}_k(t); \theta_k)]. \quad (28)$$

The stochastic gradient descent algorithm can be used to minimize the loss function by performing gradient descent to update primary weight matrix θ_k . Target weight matrix is soft updated with learning rate ρ after every G steps as $\hat{\theta}_k \leftarrow \rho\theta_k + (1 - \rho)\hat{\theta}_k$.

- Finally, training process of each DRL agent is complete when it satisfies the stop condition: the maximum number of iterations is reached or the update amount in the primary weight is less than a predefined threshold value. Thereafter, the optimal weight, θ_k^* , of each DRL agent is returned for subsequent exploitation. In the exploitation phase, each AP k can rapidly decide the optimal SA action, $\mathbf{x}_k^*(t) = \arg\max_{\mathbf{a}_k} Q_k(\mathbb{S}_k(t), \mathbf{a}_k; \theta_k^*)$, using primary DQN Q_k once it observes a state, $\mathbb{S}_k(t)$. Afterward, the optimal CRA action is obtained using (21).

Remark 1. Considering the concept drift issue caused by the change in the relationship between input and output data over time, the following points should be accounted. First, instead of using a static data distribution for training, the proposed model employs DRL agents to dynamically collect training data as the experiences through interacting with the environment. Thus, it allows changing the relationship between input and output data over time. Second, the replay memory with a fixed size always updates the most recent experiences for training. Third, although the training process is implemented offline on the cloud, new experiences can be periodically updated from APs to the cloud in the exploitation phase that improves training the model. Based on these observations, the proposed model can effectively address the concept drift issue.

C. Complexity and Feasibility

There are K DRL agents, each corresponding to an AP. As all DRL agents are trained in parallel, the overall complexity is calculated as the complexity for each DRL agent.

- For sorting users based on the channel gains, we utilize the quick sort algorithm with the overall complexity $O(S(M\log(M)))$.
- For selecting SA action $\mathbf{a}_k(t)$, the complexity depends on the neural network structure if $\mathbf{a}_k(t)$ is a strategic action, and it is $O(1)$ if $\mathbf{a}_k(t)$ is a random action. From (22), input size (state vector size) of each DQN is $2(MSK+M)$. Furthermore, because all APs operate in NOMA with at most two users allowed to share a SC, the maximum number of users that each AP can serve is $2S$. This implies that $|\mathcal{M}_k| \leq 2S, \forall k \in \mathcal{K}$. From (23), output size (action space size) is $(2S * S)^2 = 4S^4$ because SA action, $\mathbf{a}_k(t)$, is a binary vector. Considering the sizes of input, output, and hidden layers and ReLU activation in two hidden layers, the complexity of selecting a strategic action $\mathbf{a}_k(t)$ using a DQN is $O(2(MSK+M) * H + H * H + H * 4S^4 + 2H)$, which can be simplified into $O(H(MSK+2S^4+M+H))$. The complexity of computing CRA is $O(2S)$ because the maximum number of offloading users is $2S$.
- For learning procedure, each DRL agent performs gradient descent on Q-value loss. Each learning step is implemented over C samples, and it utilizes both primary and target DQNs to determine Q-value loss. Therefore,

the complexity of each learning step is $O(2CH(MSK+2S^4+M+H))$.

Assuming that training process is implemented over Ω episodes, the overall complexity of the proposed algorithm is computed as $O(\Omega\tau(SM\log M+2CH(MSK+2S^4+M+H)+2S))$. In the online decision phase, output SA action is obtained using primary DQN as $\mathbf{x}_k^*(t) = \arg\max_{\mathbf{a}_k} Q_k(\mathbb{S}_k(t), \mathbf{a}_k; \theta_k^*)$ and CRA action is obtained based on SA action using (21). Therefore, the complexity of an online decision is only $O(SM\log M + H(MSK+2S^4+M+H) + 2S)$. Overall, the complexity of the proposed algorithm is polynomial that proves the feasibility for application to practical systems.

V. SIMULATION RESULTS

A. Parameter Settings

Extensive simulations were performed using Pytorch with Python 3.7.3 on a PC, which was powered by CPU Intel Core i5-8500, GPU Nvidia GTX 1050 Ti, and a memory of 250 GB. The network model represented a factory area of $300 \text{ m} \times 300 \text{ m}$, where 4 APs, each connected to a MEC server, were uniformly distributed. The baseline setting comprised 20 IoTDs, including both industrial robots and surveillance cameras. They were uniformly distributed within the factory area, ensuring a minimum distance of 2 m from each AP. The transmit power of all IoTDs was fixed at 100 mW. Each AP employed three SCs, wherein each SC provided a 2 MHz bandwidth. To make the factory network noisy, we utilized Rayleigh fading with a shadowing deviation of 10 dB, elevated non-line-of-sight path-loss $140.7 + 36.7\log_{10}d(\text{km})$, and Gaussian noise with a noise figure of 9 dB. These network settings may generate some transient communication failures. Furthermore, computation tasks of users were uniformly selected from a set as $\{(100, 0.8); (125, 1.0); (150, 1.2); (175, 1.4); (200, 1.6)\}$ (KB, G-cycle). The architecture of the deep neural network and hyperparameters for DRL model significantly affect to the performance of the algorithm. Therefore, we carefully chose DRL parameters for simulations based on [29], which proposed DQN algorithm with common settings. In addition, we changed some settings for various performance evaluations, e.g., replay memory size D , learning rate ρ , discounting factor γ , and batch size C .

B. Convergence Analysis

When communication failures occur, DRL agents may not know exactly the channel state. Thus, it is a challenge for them to train the optimal policy based on communication failure experiences. In this study, we assume that network condition is good enough with less communication failures. Furthermore, a prioritized replay memory is utilized to filter communication failure experiences in the training process. This helps improve the convergence and the quality of training result. In Fig. 2, we elucidated the convergence of the proposed algorithm under various batch sizes: $C = 16$, $C = 32$, and $C = 64$. For the first 3000 training episodes (each includes $\tau = 300$ epochs), the overall task delay gradually decreased from approximately 300 to 110 s. Subsequently, it fluctuated in a deterministic

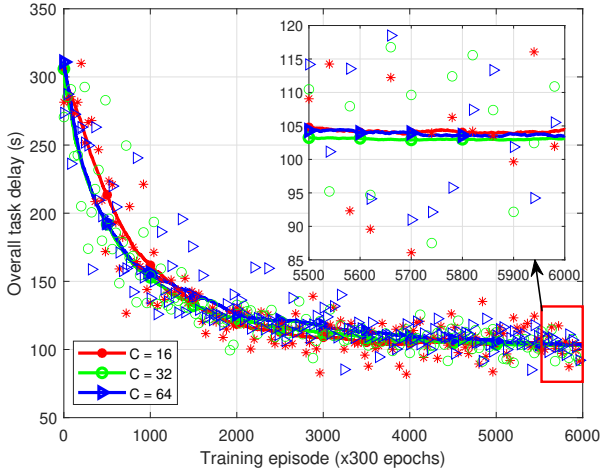


Fig. 2. Convergence of the proposed algorithm in terms of the average delay.

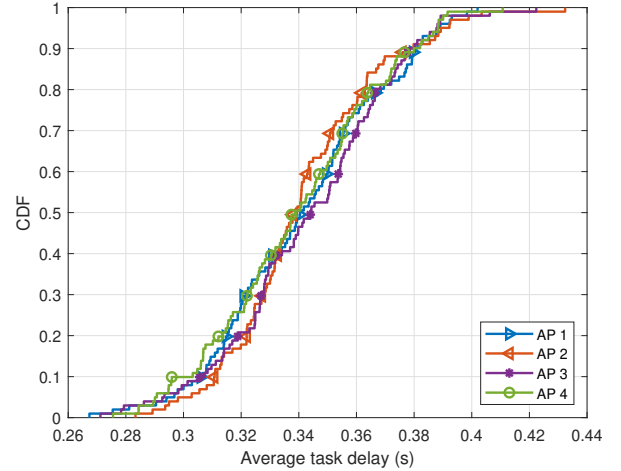


Fig. 4. Cumulative distribution function of the delay for different agents.

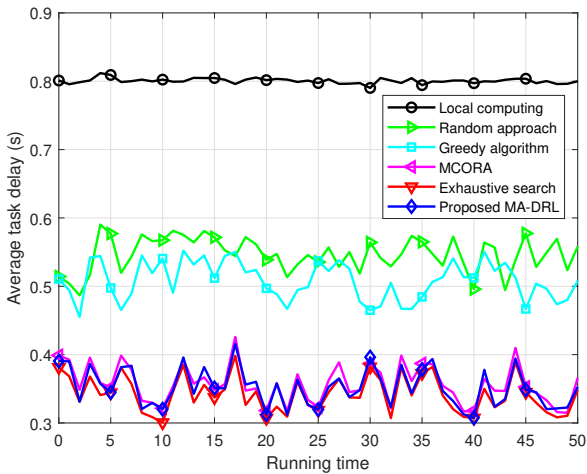


Fig. 3. Delay comparison over 50 consecutive epochs.

range between 90 and 130 s. This fluctuation occurred because the training process was based on system dynamics, including various channels and computation task requests of all users. In addition, the achievable average delay with $C = 32$ was slightly lower and more stable than those with $C = 16$ and $C = 64$. Specifically, comparing to $C = 16$, $C = 32$ can provide more accurate estimation of the gradient. Furthermore, comparing to $C = 64$, $C = 32$ is more noisy, offering a regularizing effect and lower generalization error, and makes it easier to fit one batch worth of training data in GPU memory. Therefore, the DRL model obtained by training with $C = 32$ was used for further performance evaluation.

C. Performance Evaluation

To demonstrate the excellence of the proposed solution, we compared it with several state-of-the-art schemes, including the exhaustive search scheme, a random approach, the local computing, the greedy algorithm, and the matching scheme MCORA [13].

Fig. 3 presents the average task delay comparison over a stochastic duration of 50 consecutive epochs. Along the running time, the average task delay varied in all schemes owing to the system dynamics, i.e., channel conditions and computation task requests. The delay overhead in the MA-DRL, MCORA, and exhaustive search schemes was much smaller than those in local computing, random approach, and greedy algorithm schemes. Furthermore, the MA-DRL algorithm outperformed the matching game-based MCORA algorithm, and it was comparable to the exhaustive search algorithm. Note that the exhaustive search scheme is considered the optimum as it screens over all possible actions to find the action deriving the best service delay.

Fig. 4 shows the cumulative distribution function of the average task delay for all DRL agents. The results were obtained by running simulation within 200 consecutive epochs. It was observed that all agents achieved the average task delay fluctuated between 0.26 and 0.44 s, and the average task delay of APs 1 and 4 was slightly better than that of APs 2 and 3. However, the gap was not significant that demonstrated the fairness among the APs.

Fig. 5 shows the impact of the number of users on the service delay. On increasing the number of users, the average task delay almost remained unchanged in local computing scheme, whereas it gradually increased in other schemes. This is because the number of users affects the delay only when edge computation resources are utilized to assist task execution. In addition, as edge computation resources reach the limit, the delay increases when the number of users increases. From Fig. 5, we also observed that the MA-DRL algorithm outperformed other schemes: MCORA, greedy algorithm, random approach, and local computing, and its performance was close to the optimum under the exhaustive search algorithm. For instance, in the case of 30 users, the average task delay was 0.48 s in the proposed scheme, yielding gains of approximately 96% of the exhaustive search scheme and 2%, 33%, 44%, and 67% over the MCORA, greedy algorithm, random approach, and local computing schemes, respectively.

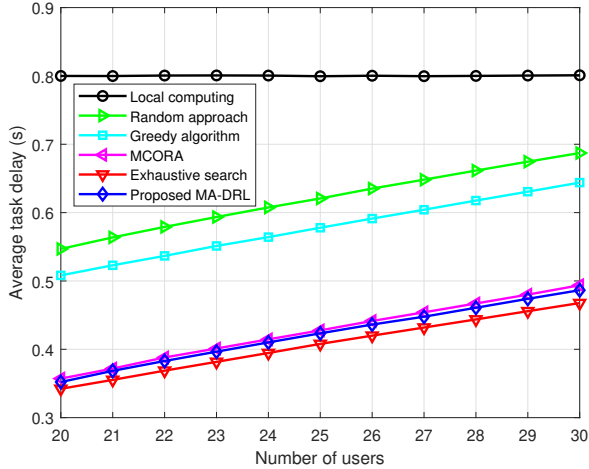


Fig. 5. Delay comparison versus the number of users.

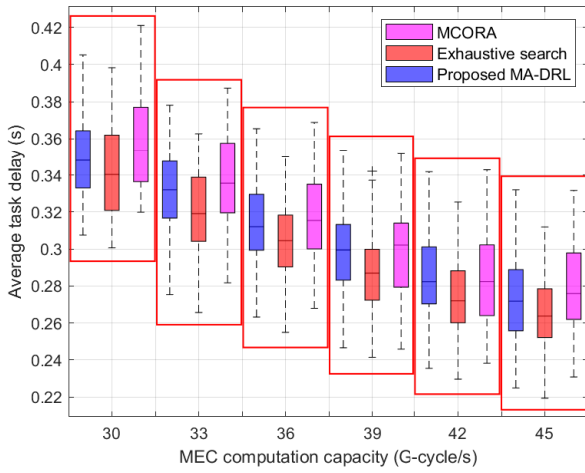


Fig. 6. Delay comparison versus MEC computation capacity.

Fig. 6 reflects the average task delay comparison under different MEC computation capacity. When MEC computation capacity increased, the average task delay decreased in all schemes (MCORA, exhaustive search, and MA-DRL). This is because larger edge computation capacity support faster execution of offloading tasks; hence, reducing latency. In addition, we observed the better performance of the proposed scheme over the MCORA scheme as it was close to the optimal result of the exhaustive search scheme. For instance, when the MEC computation capacity was 45 G-cycle/s, the average task delay in the proposed scheme was 0.272 s, which is approximately 2% smaller and 3% larger than those in MCORA and exhaustive search schemes, respectively.

Moreover, action decision times of different schemes are shown in Table III. It can be seen that the proposed scheme required more time to decide an action than local computing, random approach, and greedy schemes. However, it improved action decision duration by approximately 2 times and 30 times compared to the MCORA and exhaustive search algorithms, respectively. Considering the performance in delay

TABLE III
TIME CONSUMPTION FOR ACTION DECISION

Scheme	Decision time
Local computing approach	0.00003 s
Random approach	0.00193 s
Greedy algorithm	0.00472 s
MCORA [13]	0.11548 s
Exhaustive search algorithm	1.84331 s
Proposed MA-DRL algorithm	0.06396 s

reduction and time complexity, the proposed scheme was found to be the most suitable scheme for industrial systems. Furthermore, action decision time of the proposed scheme was less than 100 ms. Based on [30], the proposed scheme can satisfy delay constraints of a smart manufacturing system.

VI. CONCLUSIONS

In this study, a novel NOMA-enabled computation offloading scheme in industrial IoT systems was investigated to optimize SA, OD, and CRA, aiming to reduce the service latency. We considered system dynamics as various channel variations and computation task requests. To reduce the complexity, the multi-server problem was divided into multiple single-server problems, each was subsequently decoupled into CRA and SA/OD subproblems, which were solved iteratively. The optimal CRA action was obtained in a closed-form expression using convex optimization. Concurrently, the optimal SA/OD action was obtained using a distributed MA-DRL algorithm. Simulation results demonstrated a reliable convergence and superior performance of the proposed scheme over other schemes: local computing, random approach, greedy algorithm, and MCORA algorithm. Specifically, the proposed scheme achieved up to 97% of the optimal performance under the exhaustive search scheme. The smart factory network is characterized by the noise medium owing to blocking regions, metallic reflection surfaces, and unwanted electromagnetic signal. As a future work, we will further study the effect of the noise issue on the performance of the proposed scheme under various network settings using fading models, path loss models, line-of-sight probability, blocker density, multi-path delay spread, and mobility.

REFERENCES

- [1] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile Edge Computing — A key technology towards 5G," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.
- [2] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. S. Quek, "Offloading in Mobile Edge Computing: Task Allocation and Computational Frequency Scaling," *IEEE Trans. Commun.*, vol. 65, no. 8, pp. 3571–3584, 2017.
- [3] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Joint service placement and request routing in multi-cell mobile edge computing networks," in *Proc. IEEE INFOCOM*, 2019, pp. 10–18.
- [4] M. Mukherjee, S. Kumar, M. Shojafar, Q. Zhang, and C. X. Mavroumoustakis, "Joint Task Offloading and Resource Allocation for Delay-Sensitive Fog Networks," in *Proc. IEEE ICC*, 2019, pp. 1–7.
- [5] S. Xiao et al., "System delay optimization for Mobile Edge Computing," *Future Gener. Comput. Syst.*, vol. 109, pp. 17–28, 2020.
- [6] N. Javaid, A. Sher, H. Nasir, and N. Guizani, "Intelligence in IoT-Based 5G Networks: Opportunities and Challenges," *IEEE Commun. Mag.*, vol. 56, no. 10, pp. 94–100, 2018.

- [7] Z. Ding et al., "A Survey on Non-Orthogonal Multiple Access for 5G Networks: Research Challenges and Future Trends," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 10, pp. 2181–2195, 2017.
- [8] P. Wang, J. Xiao, and P. Li, "Comparison of orthogonal and non-orthogonal approaches to future wireless cellular systems," *IEEE Veh. Technol. Mag.*, vol. 1, no. 3, pp. 4–11, 2006.
- [9] V. D. Tuong, T. P. Truong, T.-V. Nguyen, W. Noh, and S. Cho, "Partial Computation Offloading in NOMA-Assisted Mobile Edge Computing Systems Using Deep Reinforcement Learning," *IEEE Internet Things J.*, vol. 8, no. 17, pp. 13 196–13 208, 2021.
- [10] Z. Ding, P. Fan, and H. V. Poor, "Impact of Non-Orthogonal Multiple Access on the Offloading of Mobile Edge Computing," *IEEE Trans. Commun.*, vol. 67, no. 1, pp. 375–390, 2019.
- [11] L. N. T. Huynh et al., "Joint Computational Offloading and Data-Content Caching in NOMA-MEC Networks," *IEEE Access*, vol. 9, pp. 12 943–12 954, 2021.
- [12] Y. Wu et al., "Delay-Minimization Nonorthogonal Multiple Access Enabled Multi-User Mobile Edge Computation Offloading," *IEEE J. Sel. Top. Signal Process.*, vol. 13, no. 3, pp. 392–407, 2019.
- [13] M. Sheng et al., "Delay-Aware Computation Offloading in NOMA MEC Under Differentiated Uploading Delay," *IEEE Trans. Wireless Commun.*, vol. 19, no. 4, pp. 2813–2826, 2020.
- [14] Z. Yang, Y. Liu, Y. Chen, and N. Al-Dhahir, "Cache-Aided NOMA Mobile Edge Computing: A Reinforcement Learning Approach," *IEEE Trans. Wireless Commun.*, vol. 19, no. 10, pp. 6899–6915, 2020.
- [15] P. Yang et al., "Latency Optimization for Multi-user NOMA-MEC Offloading Using Reinforcement Learning," in *Proc. 28th Wireless Opt. Commun. Conf. (WOCC)*. IEEE, 2019, pp. 1–5.
- [16] Z. Chen et al., "NOMA-based Multi-User Mobile Edge Computation Offloading via Cooperative Multi-Agent Deep Reinforcement Learning," *IEEE Trans. Cognitive Commun. Netw.*, 2021. Early access.
- [17] Z. Ning et al., "When deep reinforcement learning meets 5G-enabled vehicular networks: A distributed offloading framework for traffic big data," *IEEE Trans. Ind. Informat.*, vol. 16, no. 2, pp. 1352–1361, 2020.
- [18] K. Wang et al., "Online Task Scheduling and Resource Allocation for Intelligent NOMA-Based Industrial Internet of Things," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 5, pp. 803–815, 2020.
- [19] L. Li et al., "Resource Allocation for NOMA-MEC Systems in Ultra-Dense Networks: A Learning Aided Mean-Field Game Approach," *IEEE Trans. Wireless Commun.*, vol. 20, no. 3, pp. 1487–1500, 2021.
- [20] Z. Li et al., "NOMA-Enabled Cooperative Computation Offloading for Blockchain-Empowered Internet of Things: A Learning Approach," *IEEE Internet Things J.*, vol. 8, no. 4, pp. 2364–2378, 2021.
- [21] T. T. Nguyen et al., "Joint Data Compression and Computation Offloading in Hierarchical Fog-Cloud Systems," *IEEE Trans. Wireless Commun.*, vol. 19, no. 1, pp. 293–309, 2020.
- [22] Z. Yang et al., "A General Power Allocation Scheme to Guarantee Quality of Service in Downlink and Uplink NOMA Systems," *IEEE Trans. Wireless Commun.*, vol. 15, no. 11, pp. 7244–7257, 2016.
- [23] Y. Sun et al., "Optimal Joint Power and Subcarrier Allocation for Full-Duplex Multicarrier Non-Orthogonal Multiple Access Systems," *IEEE Trans. Commun.*, vol. 65, no. 3, pp. 1077–1091, 2017.
- [24] H. V. Nguyen et al., "Joint Power Control and User Association for NOMA-Based Full-Duplex Systems," *IEEE Trans. Commun.*, vol. 67, no. 11, pp. 8037–8055, 2019.
- [25] Y. Pochet and L. A. Wolsey, *Production Planning by Mixed Integer Programming*. Springer Science & Business Media, New York, NY, USA, 2006.
- [26] Y. He, N. Zhao, and H. Yin, "Integrated Networking, Caching, and Computing for Connected Vehicles: A Deep Reinforcement Learning Approach," *IEEE Trans. Veh. Technol.*, vol. 67, no. 1, pp. 44–55, 2018.
- [27] A. Bonadio, F. Chiti, and R. Fantacci, "Performance Analysis of an Edge Computing SaaS System for Mobile Users," *IEEE Trans. Veh. Technol.*, vol. 69, no. 2, pp. 2049–2057, 2020.
- [28] V. D. Tuong, N.-N. Dao, W. Noh, and S. Cho, "Deep Reinforcement Learning-Based Hierarchical Time Division Duplexing Control for Dense Wireless and Mobile Networks," *IEEE Trans. Wireless Commun.*, 2021. Early access.
- [29] H. V. Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-Learning," in *Proc. 30th AAAI Conf. Artif. Intell.*, vol. 30, no. 1, Phoenix, Arizona, USA, 2016, pp. 2094–2100.
- [30] J. Cheng, W. Chen, F. Tao, and C.-L. Lin, "Industrial IoT in 5G environment towards smart manufacturing," *J. Ind. Informat. Integr.*, vol. 10, pp. 10–19, 2018.



Van Dat Tuong received the B.S. degree in mechatronics from Hanoi University of Science and Technology, Hanoi, Vietnam, in 2012, and the M.S. degree in system software, from Chung-Ang University, Seoul, South Korea, in 2021, where he is currently pursuing the Ph.D. degree in big data. From 2012 to 2018, he was a Software Engineer with the Viettel Software Center, Viettel Telecom and the Samsung Vietnam Mobile Center (SVMC), Samsung Electronics Vietnam, Hanoi, Vietnam. He was a recipient of the Global Korea Scholarship sponsored by the Korean Government, from 2018 to 2021. His research interests include machine learning, game theory, convex optimization, and their applications in wireless networking and ubiquitous computing.



Wonjong Noh received the B.S., M.S., and Ph.D. degrees from the Department of Electronics Engineering, Korea University, Seoul, South Korea, in 1998, 2000, and 2005, respectively. From 2005 to 2007, he conducted the postdoctoral research with Purdue University, West Lafayette, IN, USA, and University of California at Irvine, Irvine, CA, USA. From 2008 to 2015, he was a Principal Research Engineer with the Samsung Advanced Institute of Technology (SAIT), Samsung Electronics, South Korea. After that, he worked as an Assistant Professor with the Department of Electronics and Communication Engineering, Gyeonggi University of Science and Technology, South Korea, and since 2019, he has been worked as an Associate Professor with the School of Software, Hallym University, Chuncheon, South Korea. He received the Government Postdoctoral Fellowship from the Ministry of Information and Communication, South Korea, in 2005. He was also a recipient of the Samsung Best Paper Gold Award in 2010, the Samsung Patent Bronze Award in 2011, and the Samsung Technology Award in 2013. His current research interests include fundamental analysis and evaluations on machine learning-based 5G and 6G wireless communications and networks.



Sungrae Cho is a Professor with the School of Computer Sciences and Engineering, Chung-Ang University (CAU), Seoul. Prior to joining CAU, he was an Assistant Professor with the Department of Computer Sciences, Georgia Southern University, Statesboro, GA, USA, from 2003 to 2006, and a Senior Member of Technical Staff with the Samsung Advanced Institute of Technology (SAIT), Kiheung, South Korea, in 2003. From 1994 to 1996, he was a Research Staff Member with Electronics and Telecommunications Research Institute (ETRI), Daejeon, South Korea. From 2012 to 2013, he held a Visiting Professorship with the National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA. He received the B.S. and M.S. degrees in electronics engineering from Korea University, Seoul, South Korea, in 1992 and 1994, respectively, and the Ph.D. degree in electrical and computer engineering from Georgia Institute of Technology, Atlanta, GA, USA, in 2002. His current research interests include wireless networking, ubiquitous computing, and ICT convergence. He has been a Subject Editor of IET Electronics Letter since 2018, and was an Area Editor of Ad Hoc Networks Journal (Elsevier) from 2012 to 2017. He has served numerous international conferences as an Organizing Committee Chair, such as IEEE SECON, ICOIN, ICTC, ICUFN, TridentCom, and IEEE MASS, and as a Program Committee Member, such as IEEE ICC, GLOBECOM, VTC, MobiApps, SENSORNETS, and WINSYS.