DQN-based Resource Allocation and Offloading (DQN-RAO) Algorithm for Multi-User Multi-Server MEC Network

Hoa Tran-Dang
Dept. of IT convergence engineering
Kumoh National Institute of Technology
Gumi-si, South of Korea
{hoa.tran-dang}@kumoh.ac.kr

Dong-Seong Kim
Dept. of IT convergence engineering
Kumoh National Institute of Technology
Gumi-si, South of Korea
dskim@kumoh.ac.kr

***Abstract*—This paper proposes a DQN-based resource allocation and offloading (DQN-RAO) Algorithm designed to minimize the average task delay in Multi-User Multi-Server Mobile Edge Computing (MEC) networks. We address the inherent complexity of the joint task-server assignment and resource allocation problem through a two-phase decomposition. First, a Deep Q-Network (DQN) learns the optimal global strategy for the discrete task offloading assignment. Second, a convex optimization model efficiently calculates the optimal CPU frequency allocation for the determined assignment. Comparative simulation results demonstrate that the DQN-RAO algorithm consistently achieves the lowest average task delay, significantly surpassing both the random offloading algorithm (ROA) and the greedy offloading algorithm (GOA) across varying server capacities and task complexities.**

***Index Terms*—Mobile Edge Computing (MEC), Task Offloading, Resource Optimization.**

I. Introduction

Mobile Edge Computing (MEC) has emerged as a transformative paradigm in modern wireless networks, enabling resource-constrained wireless devices to offload computation-intensive tasks to nearby edge servers in order to reduce latency and energy consumption [1], [2]. With the rapid proliferation of delay-sensitive applications such as augmented reality, real-time video analytics, autonomous perception, and smart city intelligence, MEC systems must increasingly accommodate large numbers of users generating substantial volumes of computation tasks. As the density of mobile users continues to rise and tasks become more heterogeneous and dynamic, the optimal coordination of task offloading and edge server resource management has become a central design challenge for next-generation MEC networks.

A considerable body of work has studied the multi-user multi-server offloading problem using a range of algorithmic methodologies. Classical optimization-based approaches typically pose the problem as a mixed-integer nonlinear program (MINLP), coupling discrete task–server assignment decisions with continuous CPU allocation variables. Techniques based on convex relaxation, branch-and-bound search, dual decomposition, and meta-heuristics provide strong theoretical performance guarantees and can account for full offloading requirements and MEC capacity constraints [3]–[5]. However, these optimization methods often suffer from poor scalability due to their exponential or high polynomial complexity, making them unsuitable for highly dynamic or large-scale MEC deployments where real-time decision-making is essential.

To improve adaptability under dynamic conditions, recent research has increasingly explored deep reinforcement learning (DRL) for MEC offloading and resource coordination [6]. Approaches built upon Deep Q-Networks (DQN), actor–critic methods, and ensemble architectures have shown the ability to learn effective policies without explicit system modeling, achieving notable reductions in task delay in stochastic, time-varying environments [7]. Despite these advantages, DRL techniques typically incur substantial training cost, may exhibit limited interpretability, and often struggle with convergence as the discrete action space grows with the number of users and servers [8]. This training instability, combined with the heavy computational overhead of deep models, reduces their practicality for real-time, delay-critical MEC systems.

Other algorithmic strategies have sought to balance computational tractability and solution quality. Heuristic algorithms—such as greedy and marginal-cost-based assignments—offer simplicity and rapid execution [9], enabling fast adaptation to changing conditions. However, their lack of awareness of nonlinear MEC dynamics can lead to suboptimal decisions, particularly when server capacity constraints are tight. Game-theoretic frameworks introduce distributed decision-making principles and can regulate server load through user competition, but decentralization may cause convergence to inefficient equilibria due to limited global coordination [10]. Hybrid learning approaches using LSTM models or Markov decision processes have also been proposed to manage multi-objective trade-offs between delay, energy, and reliability. Nonetheless, these methods often overlook task heterogeneity or become computationally prohibitive in large-scale networks.

Despite these advances, existing approaches still face fundamental limitations. Optimization-based methods guarantee

optimality but scale poorly; heuristic and game-theoretic approaches are fast but may fail under complex constraints; DRL strategies are adaptive but costly to train and difficult to stabilize. As a result, no current solution simultaneously achieves scalability, interpretability, computational efficiency, and robust delay performance in capacity-constrained MEC environments. This longstanding trade-off underscores the need for a new framework that combines analytical rigor with practical real-time efficiency.

Motivated by these challenges, this paper proposes a DQN-based Resource Allocation and Offloading (DQN-RAO) framework for multi-user multi-server MEC networks. The key idea is to decompose the otherwise intractable joint offloading and resource allocation problem into two coordinated phases. In the first phase, a Deep Q-Network is employed to learn task–server assignment decisions that satisfy server-capacity constraints and adapt to dynamic wireless environments. In the second phase, given the learned task assignments, the optimal CPU allocation across tasks is determined analytically using convex optimization. This decomposition not only reduces the complexity of the DRL component—by isolating the decision space to server assignment only—but also preserves optimality in resource allocation through closed-form or deterministic solutions. The proposed DQN-RAO framework therefore achieves a balance between learning-based adaptability and analytical efficiency, enabling real-time task offloading under realistic MEC constraints.

II. System Model and Problem Formulation

A. Network Model

We consider a multi-user multi-server mobile edge computing (MEC) network, as illustrated in Fig. 1, comprising M wireless devices (WDs) and N edge servers (ESs), where typically $N < M$. The WDs are indexed by $m \in \{1, \ldots, M\}$ and the ESs by $n \in \{1, \ldots, N\}$. Time is slotted, and in each slot every WD generates a single computation task that must be fully offloaded to one ES for execution, with no task partitioning or hybrid local–edge processing allowed.

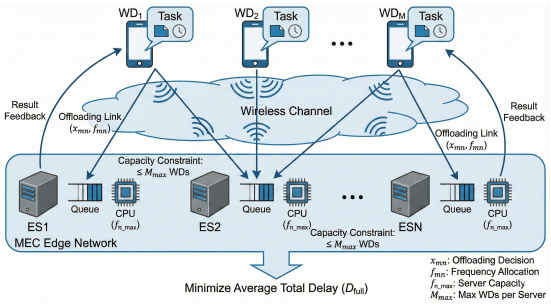

Fig. 1. Architecture of multi-user multi-server MEC network.

Each computation task is characterized by two heterogeneous parameters. The input data size S_m (in bits) represents the payload that WD m must transmit to its serving ES. The computational complexity C_m (in CPU cycles) specifies the execution workload required to complete the task once received at the server. These parameters vary across devices due to the diversity of application workloads in practical MEC systems.

Task offloading occurs over an uplink multiple-access channel with total bandwidth B shared among the WDs. The wireless channel between WD m and ES n is represented by the channel gain h_{mn}, which accounts for path loss, shadowing, and fading effects. Each WD transmits with a fixed power P_m during the offloading phase. On the computation side, ES n is equipped with a processing capability represented by the maximum CPU frequency F_n (in cycles per second), which must be allocated among its associated tasks. To capture practical processing, memory, and connection-management constraints, each ES can concurrently support at most $M_n^{\max}$ tasks within a given slot.

Two sets of decision variables determine the offloading and resource allocation outcomes. The binary association variable x_{mn} indicates whether WD m offloads its task to ES n, where $x_{mn} = 1$ when ES n is selected and $x_{mn} = 0$ otherwise. Since full offloading is enforced, each WD must select exactly one ES. The continuous variable f_{mn} denotes the CPU frequency allocated by ES n to the task of WD m. By definition, $f_{mn} > 0$ only when $x_{mn} = 1$, ensuring that computation resources are provided exclusively to offloaded tasks.

B. Delay Model

1) Uplink Rate (bits/s):

$$R_{mn}(t) = B \log_2 \left(1 + \frac{P_m h_{mn}(t)}{\sigma^2}\right) \tag{1}$$

where B is the system bandwidth, P_m is the transmit power of WD m, h_{mn} is the channel gain and σ^2 is the noise power.

2) Transmission Delay:

$$T_{mn}^{\text{tx}} = \frac{S_m}{R_{mn}} \tag{2}$$

3) Computation Delay at Edge Server:

$$T_{mn}^{\text{comp}} = \frac{C_m}{f_{mn}}. \tag{3}$$

4) Total Delay:

$$T_{mn} = T_{mn}^{\text{tx}} + T_{mn}^{\text{comp}} \tag{4}$$

The average delay across all WDs is

$$D^{\text{full}} = \frac{1}{M} \sum_{m=1}^{M} \sum_{n=1}^{N} x_{mn} T_{mn}. \tag{5}$$

C. Optimal Full-Offloading Optimization Problem

The optimal full-offloading problem is formulated as:

$$\underset{\{x_{mn},f_{mn}\}}{\text{minimize}} \quad D^{\text{full}} = \frac{1}{M}\sum_{m=1}^{M}\sum_{n=1}^{N} x_{mn}\left(\frac{S_m}{R_{mn}} + \frac{C_m}{f_{mn}}\right) \tag{6}$$

$$\text{subject to} \quad \sum_{n=1}^{N} x_{mn} = 1, \quad \forall m, \tag{7}$$

$$\sum_{m=1}^{M} f_{mn} \le F_n, \tag{8}$$

$$\sum_{m=1}^{M} x_{mn} \le M_{\max}, \tag{9}$$

$$x_{mn} \in \{0,1\}, \quad f_{mn} \ge 0, \quad \forall m,n. \tag{10}$$

Problem (6)–(10) is a mixed-integer nonlinear program (MINLP), since it involves both binary variables and nonlinear delay terms. The optimal full-offloading problem is challenging due to its mixed-integer nonlinear nature, with both combinatorial assignment variables and nonlinear coupling in resource allocation. The exponential growth of feasible assignments makes direct optimization intractable as system size increases. Nonlinear delay functions tightly couple scheduling with resource allocation, meaning that assigning a device to a server affects not just delay, but the feasible CPU distribution for all tasks. These factors together render standard exact solvers slow and limit the practicality of many heuristic and learning-based approaches for real-time, large-scale edge computing systems.

III. DQN-RAO Algorithm Design

To overcome the computational intractability of the mixed-integer nonlinear optimization problem in (6)–(10), we propose a decomposition-based solution DQN-RAO as illustrated in Fig.2.

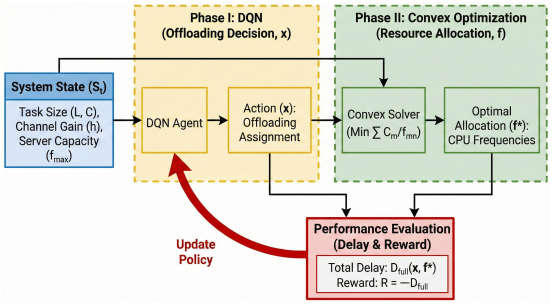

Fig. 2. The illustration of DQN-RAO Algorithm

The key idea is to separate the binary offloading decision from the continuous CPU allocation. In the first phase, a DQN is trained to determine an efficient server-selection matrix $\mathbf{X} = [x_{mn}]$ under dynamic wireless and computational conditions. In the second phase, given $\mathbf{X}$, the optimal CPU frequency allocation $\mathbf{F} = [f_{mn}]$ is computed by solving a convex minimization problem. This two-stage procedure allows scalable and near-optimal operation in large multi-user multi-server MEC networks.

A. Phase I: DQN-Based Offloading Decision Learning

The process of assigning tasks to edge servers is formulated as a Markov Decision Process (MDP). At decision step t, the system state is defined as

$$s_t = \Big(\{h_{mn}(t)\}_{m,n}, \{S_m\}_m, \{C_m\}_m, \{M_n^{\max} - u_n(t)\}_n\Big), \tag{11}$$

where $u_n(t)$ is the number of tasks already assigned to ES n before making decision t.

The action at step t is the selected server index for the current device:

$$a_t \in \{1,\dots,N\}, \tag{12}$$

which determines the assignment $x_{ma_t} = 1$.

The reward is defined to favor low latency while discouraging overload of any ES:

$$r_t = -\left(T_{ma_t}^{\text{tx}} + \frac{C_m}{F_{a_t}/(u_{a_t}(t)+1)}\right) - \lambda \cdot \mathbf{1}\left[u_{a_t}(t)+1 > M_{a_t}^{\max}\right], \tag{13}$$

where λ is a large penalty coefficient.

A Deep Q-Network parameterized by θ is trained to approximate the optimal action-value function

$$Q_\theta(s_t,a_t) \approx \mathbb{E}\left[\sum_{\tau=t}^{T} \gamma^{\tau-t} r_\tau \mid s_t, a_t\right], \tag{14}$$

where γ is the discount factor. The learned policy selects

$$a_t^* = \arg\max_a Q_\theta(s_t,a). \tag{15}$$

After M steps, a complete server association matrix $\mathbf{X}$ is obtained.

B. Phase II: Optimal CPU Frequency Allocation

Given the offloading decisions $\mathbf{X}$, the computation resource allocation reduces to a convex optimization problem. For each ES n, define the set of assigned devices

$$\mathcal{M}_n = \{m : x_{mn} = 1\}. \tag{16}$$

For ES n, the local CPU allocation problem is

$$\underset{\{f_{mn}\}}{\text{minimize}} \quad \sum_{m\in\mathcal{M}_n} \frac{C_m}{f_{mn}} \tag{17}$$

$$\text{subject to} \quad \sum_{m\in\mathcal{M}_n} f_{mn} \le F_n, \tag{18}$$

$$f_{mn} > 0, \quad \forall m \in \mathcal{M}_n. \tag{19}$$

Using KKT conditions, the optimal CPU frequency allocation at ES n is:

$$f_{mn}^* = F_n \frac{\sqrt{C_m}}{\sum_{j\in\mathcal{M}_n}\sqrt{C_j}}, \quad \forall m \in \mathcal{M}_n. \tag{20}$$

Algorithm 1: DQN-RAO Algorithm

Initialize DQN parameters θ, replay memory $\mathcal{M}$
for each episode **do**
  Observe initial system state s_1
  for $t = 1$ to M **do**
    Select action a_t using ϵ-greedy:

$$a_t = \begin{cases} \text{random action}, & \text{with prob. } \epsilon, \\ \arg\max_a Q_\theta(s_t, a), & \text{otherwise} \end{cases}$$

    Assign WD m to ES a_t: $x_{ma_t} = 1$
    Compute reward r_t
    Observe next state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{M}$
    Sample minibatch from $\mathcal{M}$
    Update θ via gradient descent:

$$L(\theta) = \left(r_t + \gamma \max_{a'} Q_\theta(s_{t+1}, a') - Q_\theta(s_t, a_t)\right)^2$$

  end for
  Obtain complete assignment matrix $\mathbf{X}$
  for each ES n **do**
    Solve convex CPU allocation for f^*_{mn}:

$$f^*_{mn} = F_n \frac{\sqrt{C_m}}{\sum_{j \in \mathcal{M}_n} \sqrt{C_j}}$$

  end for
  Evaluate overall delay D^{full}
end for

Once f^*_{mn} is computed for all m and n, the total delay is obtained from

$$T_{mn} = T^{\text{tx}}_{mn} + \frac{C_m}{f^*_{mn}}, \tag{21}$$

and the averaged total delay D^{full} is computed accordingly.

The integrated DQN-RAO solution is summarized in Algorithm 1.

IV. Numerical Simulation Results and Evaluation

A. Simulation Environment

The simulation models a representative MEC network with $M = 9$ wireless devices (WDs) and $N = 3$ edge servers (ESs), where each server can serve at most $M_{\max} = 3$ devices concurrently. The computation tasks are heterogeneous: input data size $S_m \sim \mathcal{U}[200, 500]$ kb and CPU cycles $C_m \sim \mathcal{U}[5 \times 10^8, 2 \times 10^9]$, covering lightweight to computation-intensive workloads. The uplink bandwidth is $B = 1$ MHz, and each WD transmits with power $P_m = 0.1$ W. Channel gains $h_{mn} \sim \mathcal{U}[0.5, 2.0]$ capture path loss and fading, and noise power is $\sigma^2 = 10^{-9}$ W/Hz. Edge servers have heterogeneous CPU frequencies $F_n \sim \mathcal{U}[5 \times 10^9, 9 \times 10^9]$ cycles/s.

The DQN in the first phase of DQN-RAO is implemented as a fully connected feed-forward network. The key configuration parameters are summarized in Table I.

TABLE I
DQN Configuration for Offloading Decision Learning

Parameter	Value
Hidden layers	2 (128, 64 neurons)
Activation function	ReLU
Output layer	N Q-values (servers)
Replay memory size	10^5 transitions
Minibatch size	64
Optimizer	Adam, LR = 1×10^{-3}
Discount factor γ	0.95
Exploration strategy	ϵ-greedy, ϵ decay $1.0 \rightarrow 0.05$ over 2000 episodes
Target network update	Every 100 steps
Training episodes	4000

B. Comparative Algorithms

To evaluate the performance of the proposed DQN-RAO algorithm, we compare it against two common MEC baselines. The random offloading algorithm (ROA) dictates that each WD selects an edge server uniformly at random, repeating the process until a server below the maximum capacity ($M_{\max}$) is found. In contrast, the greedy offloading algorithm (GOA) mandates that each WD chooses the edge server that provides the highest instantaneous channel gain (or uplink data rate, R_{mn}), entirely disregarding the server's current computational load or available capacity.

C. Simulation Results and Evaluations

1) Training Loss: Figure 3 illustrates the trends of the DQN training loss over 500 episodes for the specified batch sizes.

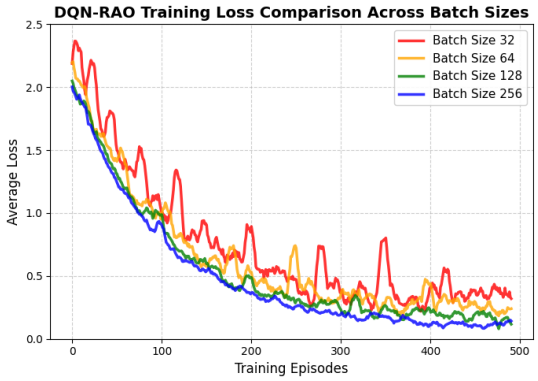

Fig. 3. Training loss of DQN with different batch sizes

The loss curves for smaller batch sizes (32 and 64) are more volatile (more jagged) due to the higher variance in the gradient estimation based on fewer samples per update. The loss curves for larger batch sizes (128 and 256) are smoother, indicating a more stable and accurate gradient estimation.

2) Averaged delay vs server CPU frequency: In the scenario of simulation, we consider the homogeneous servers. We change the maximum CPU frequency of servers in range [5e9,10e9]. The simulation results as shown in Fig. 4 confirms the substantial superiority of the DQN-RAO algorithm, which consistently achieves the minimum average task delay across the full edge server frequency range (5 GHz to 9 GHz). At the

lowest capacity (5 GHz), DQN-RAO yields a delay (≈ 0.60 s) that is over 14% lower than GOA (≈ 0.70 s) and over 29% lower than ROA (≈ 0.85 s). While all algorithms benefit from increased CPU frequency, the performance margin of DQN-RAO is maintained. This persistent gap validates that the DQN-RAO agent successfully learns an intelligent, coordinated policy, effectively navigating the trade-off between transmission quality and computational load balancing for significant efficiency improvements over myopic and randomized strategies in the MEC environment.

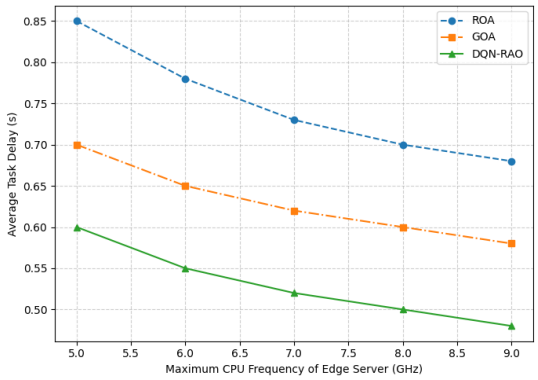

Fig. 4. The Averaged Delay of Algorithms

3) Averaged Delay vs. Task Complexity: In this scenario, where heterogeneous servers are considered, the task complexity is varied in the range $[500, 1000]$ CPU cycles per bit. Figure 5 illustrates the averaged task delay as task complexity increases. As expected, increasing task complexity increases

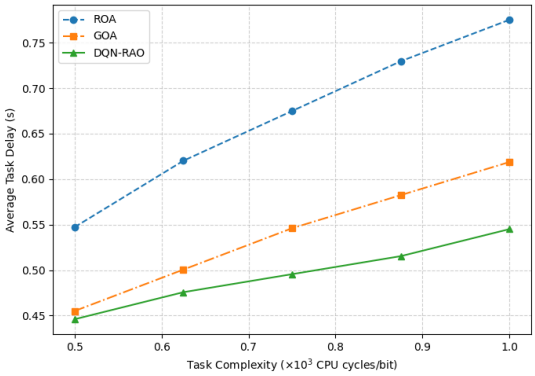

Fig. 5. The Averaged Delay of Algorithms

the average delay for all algorithms. The proposed DQN-RAO consistently yields the lowest overall delay, rising from ≈ 0.44 s to ≈ 0.54 s. ROA demonstrates the highest delay (≈ 0.55 s to ≈ 0.78 s) and is the most sensitive to complexity changes, showing the steepest delay increase (≈ 0.23 s). Both GOA and DQN-RAO are less sensitive due to their intelligent channel selection capabilities. The substantial performance gap maintained by DQN-RAO across the entire complexity range underscores its superior efficacy in achieving optimal load distribution and resource allocation for demanding applications.

V. Conclusion

This paper proposed the DQN-RAO algorithm to minimize the average task delay in multi-user multi-server MEC networks. The complex joint offloading and resource allocation problem was effectively handled using a two-phase decomposition: a DQN learned the optimal discrete offloading policy, followed by convex optimization for efficient CPU frequency allocation. Simulation results demonstrated that DQN-RAO consistently achieved the lowest average task delay, significantly surpassing both GOA and ROA algorithms across varying maximum server CPU frequencies and task complexity levels.

Acknowledgments

This work was supported in part by the Ministry of Science and ICT (MSIT), Korea, under the Innovative Human Resource Development for Local Intellectualization program, supervised by the Institute for Information and Communications Technology Planning and Evaluation (IITP) (IITP-2025-RS-2020-II201612 (25%), IITP-2025-RS-2024-00438430 (25%)) and Korea Research Fellowship Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (2018R1A6A1A03024003 (25%), RS-2023-00249687 (25%)).

References

[1] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 3, pp. 1628–1656.

[2] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 4, pp. 2322–2358.

[3] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, October 2016.

[4] K. Wang, S. F. Akhtar, and F. A. Al-Zahrani, "An efficient algorithm for resource allocation in mobile edge computing based on convex optimization and karush–kuhn–tucker method," *Complexity*, vol. 2023, pp. 1–15.

[5] L. Huang, X. Feng, L. Zhang, L. Qian, and Y. Wu, "Multi-server multi-user multi-task computation offloading for mobile edge computing networks," *Sensors*, vol. 19, no. 6, p. 1446.

[6] H. Tran-Dang, S. Bhardwaj, T. Rahim, A. Musaddiq, and D.-S. Kim, "Reinforcement learning based resource management for fog computing environment: Literature review, challenges, and open issues," *Journal of Communications and Networks*, pp. 1–16.

[7] Y. Liu, H. Yu, S. Xie, and Y. Zhang, "Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 11, pp. 11 158–11 168.

[8] Y. Qin, J. Chen, L. Jin, R. Yao, and Z. Gong, "Task offloading optimization in mobile edge computing based on a deep reinforcement learning algorithm using density clustering and ensemble learning," *Scientific Reports*, vol. 15, no. 1.

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

[10] T. T. Vu, N. V. Huynh, D. T. Hoang, D. N. Nguyen, and E. Dutkiewicz, "Offloading energy efficiency with delay constraint for cooperative mobile edge computing networks," in *2018 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6.