Cooperative Task Offloading with Multi-agent Deep Reinforcement Learning

Xinyao Qiu *xyqiu@umich.edu*
Yuqi Mai *yqmai@umich.edu*

1. Introduction

The latest advances in the Internet of Things (IoT) have facilitated the proliferation of a large number of mobile applications that rely primarily on edge devices (EDs), such as laptops, tablets, and smartphones, to collect data from IoT sensors to serve end users as shown in Figure 1.

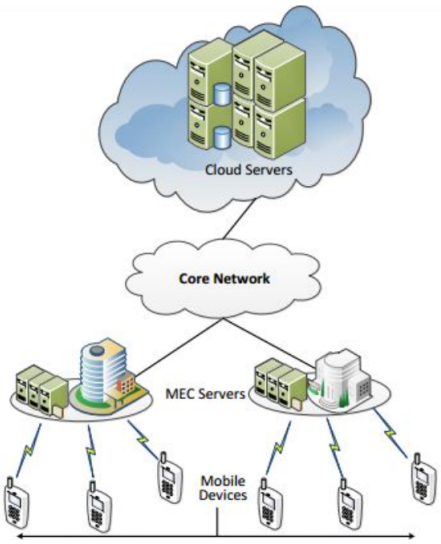

Figure 1. Edge Computing

To meet the growing computing needs of users, mobile edge computing (MEC) is a promising technology that can improve the computing experience of electronic devices by offloading computation-based tasks to MEC servers located near the cloud servers. MEC task offloading has become a viable solution to meet various EDs' computing needs and improve the quality of end-user experience.

However, it is not easy to design an efficient task-offloading strategy for the whole MEC system. Specifically, each ED always tries to maximize its own utility of the whole resources, which tends to lead to network traffic congestion and thus degrade the end-user experience. Additionally, because of the lack of knowledge about a practical multi-user MEC system's present state, it is hard to figure out an optimal offloading solution for each ED.

Recently, many edge task offloading schemes have been proposed [1], but most of these efforts consider single-agent unloading scenarios using traditional convex optimization tools. Deep reinforcement learning (DRL) techniques, such as deep Q-learning (DQN), have emerged as a promising alternative by modeling offloading problems as Markov decision processes (MDP) using deep neural networks (DNN) for function approximation. However, these efforts only use a single agent to handle the entire uninstall process and do not work well in a large-scale distributed MEC environment. An interesting alternative is to use a multi-agent DRL (MA-DRL) to support smart task offloading in a MEC network [2].

In this project, we simplify the MEC problem as video task (VT) processing. When we compress a video, two options can be chosen, which are processed locally or offloaded video tasks to the MEC server for cloud computing. We try to gain a deeper insight into Reinforcement Learning (RL), which is not covered in class due to timing issues, through this project. To be more specific, we apply three different Deep Reinforcement Learning methods which are all based on Actor-Critic structure, namely Multi-Agent Advantage Actor-Critic (MAA2C), Multi-Agent Proximal Policy Optimization (MAPPO), and Multi-Agent Deep Deterministic Policy Gradient (DDPG) on a basic MEC problem. We compare the reward function for different environment parameters as well as their final results.

2. Method

2.1. Deep Reinforcement Learning Algorithms

Deep reinforcement learning is a subfield of machine learning that combines reinforcement and deep learning. RL considers the problem of computational agents learning decisions by trial and error. DRL integrates deep learning into solutions, allowing agents to make decisions from unstructured input data without having to manually design state spaces. DRL algorithms can take very large inputs (for example, each pixel rendered to the screen in a video game) and decide what to do to optimize the goal such as maximizing the game score.

2.1.1. Actor Critic. Actor-Critic [3] is a classical RL algorithm, which contains two parts, the Actor, which is responsible to create actions for the agent, and the Critic, which is used to evaluate the Actor's performance as well as guide the Actor to the next stage's action as shown in Fig 2.

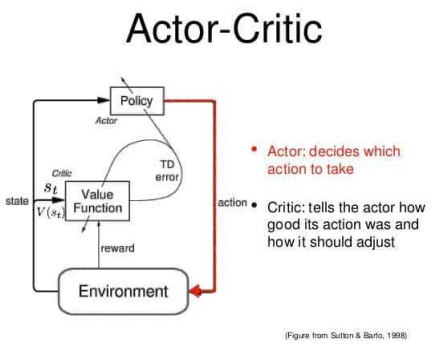

Figure 2. Actor Critic

2.1.2. Advantage Actor Critic (A2C). Advantage Actor-Critic [4] has a very similar structure to the Actor-Critic algorithm which uses the Advantage Function to replace the reward function in the critic network of AC. The general form for the Q-value in such an algorithm is expressed as Eq. 1

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s) \tag{1}$$

The Advantage function intuitively indicates whether a state is better or worse than the model's expectation. If an action is better than the average expectation, it is more likely that the actor will be encouraged to take that action. And on the contrary, if an action is worse than average, the actor will be encouraged to take the opposite of the action.

2.1.3. Proximal Policy Optimization (PPO). Proximal Policy Optimization [5] is another algorithm that bases on AC's structure. The advance in PPO is that it tackles the problem with continuous action space compared to AC, which is only capable of discrete action space. In PPO, the policy distribution is assumed to follow a Gaussian Distribution and thus describes the probability of a continuous action space. Besides, PPO also uses important sampling to change an On-policy training process to an Off-policy training, which reduces the risk of high variance.

2.1.4. Deep Deterministic Policy Gradient (DDPG). The deep deterministic policy gradient [6] also has a very similar structure to Actor-Critic (AC) model, which contains two deep neural networks, an actor network, and a critic network. The actor network takes in the current state and outputs selected actions while the critic network takes in the state and the action and evaluates the Q-value. Unlike PPO, which outputs a distribution of action probability, the output of DDPG's algorithm is a simple action.

In this project, We use modified versions of the above algorithms in our project, MAA2C, MAPPO, and MADDPG, which stands for Multi-Agents A2C, PPO, and DDPG. The major difference between the MA version and the original one is in MA's, the critic neural network takes in multiple state information sensed by different agents to evaluate.

2.2. Evaluation Metric

We use two baseline methods to evaluate different RL algorithms. One baseline is Non-Adaptive Compression (NAC), where all video tasks are compressed to a prefix compression rate instead of a range of satisfaction. Another baseline is All-Edges (ALLES), which means all the video tasks are offloaded to the MEC server and will not consider computing locally. The details of these baselines will be discussed in the following section.

3. Model

Here we put forward the network model and computing of our video task processing.

Before processing the video task, we need to choose whether to process it locally (local mode) or to offload it to the MEC server (MEC mode), which also requires our selection among different channels.

So we give a task offloading policy $\boldsymbol{X} = \{x_n^k | n \in \boldsymbol{N},\ k \in \boldsymbol{K}\}$ to describe the mode & channel selection of each VT_n, where

$$x_n^k = \begin{cases} 1 & \text{, MEC mode \& using channel } k \\ 0 & \text{, otherwise.} \end{cases} \tag{2}$$

Here, $\boldsymbol{N} = \{1, ...N\}$ denotes the set of video tasks, and $\boldsymbol{K} = \{1, ...K\}$ denotes the K selectable channels. The binary variable x_n^k ($n \in \boldsymbol{N}$) indicates the transmission state of VT_n. $x_n^k = 1$ means that the task is offloaded to the MEC server via channel k and $x_n^k = 0$ otherwise.

Further, we denote x_n as the indicator of whether VT_n is offloaded to the MEC server, which is also a binary variable:

$$x_n = \sum_{k \in \boldsymbol{K}} x_n^k \tag{3}$$

Moreover, we introduce the transmit power policy $\boldsymbol{P} = \{p_n^k | 0 \le p_n^k \le p_{max},\ n \in \boldsymbol{N},\ k \in \boldsymbol{K}\}$, where p_n^k denotes the transmit power when offloading the task VT_n to the MEC server via the channel k, subject to a maximum budget p_{max}.

Under the selection of task offloading policy, there are two possible computing modes: local mode and MEC mode.

3.1. Local Mode

We define $\beta_n \in [0,1]$ as the compression rate of VT_n. A higher compression rate leads to better transformation in the quality of a video task. Also, we denote f_n^l as the local computational resource allocated to VT_n, which is confined within the resource allocation limit $[f_{min}, f_{max}]$. Thus, a local computational resource allocation policy can be introduced as $\boldsymbol{F} = \{f_n^l | f_{min} \leq f_n^l \leq f_{max}\}$.Then, the local computing latency for processing the n-th video task VT_n is expressed as

$$T_n^{loc} = \frac{\beta_n c_n}{f_n^l}, \tag{4}$$

where c_n denotes the required CPU cycles for processing VT_n [2].

Then we introduce an energy consumption coefficient κ. For VT_n with a size s_n, the energy consumption for processing the video task locally is [7]

$$E_n^{loc} = \kappa s_n \beta_n (f_n^l)^2. \tag{5}$$

For our video processing quality, we define

$$\phi_n^{loc} = \nu^{loc} log_2 \left(1 + \frac{\beta_n}{\theta^{loc}}\right), \tag{6}$$

where ν is the quality coefficient and θ is the normalization coefficient [8].

3.2. MEC Mode

Under this mode, the video tasks are offloaded to the MEC server for cloud computing. For task transmitting, the latency and energy consumption cost by VT_n and the video processing quality are respectively expressed as

$$T_n^{off} = \frac{\beta_n s_n}{r_n} + \frac{\beta_n c_n}{f^e}, \tag{7}$$

$$E_n^{off} = \sum_{k \in \boldsymbol{K}} p_n^k T_n^{off}, \tag{8}$$

$$\phi_n^{off} = \nu^{off} log_2 \left(1 + \frac{\beta_n}{\theta^{off}}\right). \tag{9}$$

Particularly, r_n denotes the data transmission rate and has an expression of

$$r_n = W log_2 \left(1 + \frac{p_n^k h_n^k}{\sigma^2 + \sum_{i \in \boldsymbol{N}, i \neq n} x_i^k p_i^k h_i^k}\right), \tag{10}$$

where W is a constant coefficient which is the bandwidth of the channels, σ^2 is the background noise variance, p_i^k is the transmit power of VT_i via the channel k, and h_i^k is the uplink channel gain between VT_i and channel k [2].

4. Problem Formulation

4.1. System Utility Formulation

4.1.1. Processing Utility. For video task processing, the total processing latency, energy consumption, and processing quality equal to

$$T_n = (1 - x_n) T_n^{loc} + x_n T_n^{off}, \tag{11}$$

$$E_n = (1 - x_n) E_n^{loc} + x_n E_n^{off}, \tag{12}$$

$$\phi_n = (1 - x_n) \phi_n^{loc} + x_n \phi_n^{off} \tag{13}$$

Considering the utility of a processed video task is collectively determined by its energy consumption and its quality, the utility function for processing VT_n is specified as

$$R_n = \lambda_1 \frac{\phi_n^{loc} - \phi_n}{\phi_n^{loc}} + \lambda_2 \frac{E_n^{loc} - E_n}{E_n^{loc}}, \tag{14}$$

where λ_1 and λ_2 are weight coefficient and $\lambda_1 + \lambda_2 = 1$.

4.1.2. System Utility Formulation. In this project, our goal is to solve the optimization problem that can be formulated as follows [2]

$$\underset{\boldsymbol{X}, \boldsymbol{P}, \boldsymbol{F}}{\text{maximize}} \quad \frac{1}{N} \sum_{n \in \boldsymbol{N}} R_n \tag{15a}$$

$$\text{subject to} \quad x_n^k \in \{0,1\}, \quad x_n \leq 1, \tag{15b}$$

$$0 \leq p_n^k \leq p_{max}, \tag{15c}$$

$$T_n \leq \tau_n, \quad \phi_n \geq \epsilon_n, \tag{15d}$$

$$f_{min} \leq f_n^l \leq f_{max}. \tag{15e}$$

where each constraint is explained as

- Equation 15b implies that video tasks are either processed locally or uploaded to the MEC server. One channel can be occupied by at most one task, and one task can only occupy one channel.
- Equation 15c indicates that the transmission power should lie between 0 and the maximum value p_{max}.
- Equation 15d implies that the processing time of each video task cannot exceed its processing deadline, and the transmitted quality should be greater than the lowest quality.
- Equation 15e tells that the local computational resource allocated to each video task should lie in the preset minimum and maximum values.

4.2. Cooperative Learning Formulation

To solve the system utility optimization problem 15a, we convert the optimization problem to a reward maximization problem and apply our proposed MA-DRL algorithm.

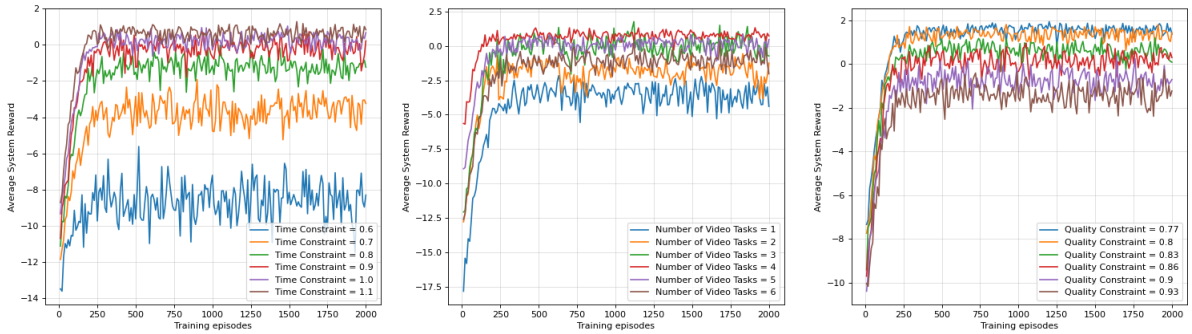

Figure 3. Comparison of average system rewards with different parameters.

4.2.1. State. The environment state of the cooperative network is featured by four components: task state $S_{task}(t)$, mode & channel selection state $S_{channel}(t)$, power transmission state $S_{pow}(t)$, and local resource allocation state $S_{res}(t)$. Thus, the system state can be defined as a matrix:

$$\mathcal{S}(t) = \{S_{task}(t), S_{channel}(t), S_{pow}(t), S_{res}(t)\}, \quad (16)$$

where each component is explained as follows.

- **Task state:** $S_{task}(t) = [s_n(t), c_n(t)]$. $s_n(t)$ represents the task size of VT_n, and $c_n(t)$ represents the required CPU cycle to compute VT_n.
- **Mode & channel selection state:** $S_{channel}(t)$ is specified as

$$S_{channel}(t) = \boldsymbol{X}(t) = \begin{pmatrix} x_1^1(t) & \cdots & x_1^K(t) \\ \vdots & \ddots & \vdots \\ x_n^1(t) & \cdots & x_n^K(t) \end{pmatrix}, \quad (17)$$

  where $x_n^k(t) = 1$ when the channel k is used by VT_n at time t, and $x_n^k(t) = 0$ otherwise.
- **Power transmission state:** $S_{pow}(t)$ is specified as

$$S_{pow}(t) = \boldsymbol{P}(t) = \begin{pmatrix} p_1^1(t) & \cdots & p_1^K(t) \\ \vdots & \ddots & \vdots \\ p_n^1(t) & \cdots & p_n^K(t) \end{pmatrix}, \quad (18)$$

  where $p_n^k(t)$ represents the transmission power allocated to the k-th channel that lies in $[0, p_{max}]$.
- **Local resource allocation state:** $S_{res}(t) = f_n^l(t)$, where $f_n^l(t)$ is the local computational resource allocated to VT_n.

4.2.2. Action. After observing the system states, each VT needs to take actions including task offloading decision, channel selection, local computational resource allocation, and compression rate selection. The action space can be expressed as:

$$\mathcal{A}(t) = \{x_n^k(t), k(t), f_n^l(t), \beta_n^k(t)\}, \quad (19)$$

The corresponding explanation of each component is as follows:

- **Task offloading decision** $x_n^k(t)$**:**

$$x_n^k(t) = \begin{cases} 1 & \text{, MEC mode \& using channel } k \\ 0 & \text{, otherwise.} \end{cases} \quad (20)$$

  where the value of $x_n^k(t)$ is based on the current task state $S_{task}(t)$.
- **Channel selection** $k(t)$**:** $k(t) = [1, 2, ..., K]$. Each VT selects one available channel to offload the task to the MEC server, which is based on the current channel state $S_{channel}(t)$.
- **local computational resource** $f_n^l(t)$**:** Based on the current state $S_{res}(t)$ and $S_{task}(t)$, each VT allocates the computational resource to execute the task locally.
- **compression rate selection** $\beta_n^k(t)$**:** $\beta_n^k(t) \in [0, 1]$. Each VT selects its compression rate between 0 and 1.

4.2.3. System Reward Function. Since the design of our cooperative learning formulation is based on the optimization problem 15a, our system reward function should be equal to the system utility function (Equation 14). For different video tasks, we just add their reward function together to get the total reward function. Thus, we can formulate the system reward function as

$$r(\mathcal{S}(t), \mathcal{A}(t)) = \frac{1}{N} \sum_{n \in \boldsymbol{N}} r(\mathcal{S}_n(t), \mathcal{A}_n(t)) = \frac{1}{N} \sum_{n \in \boldsymbol{N}} R_n(t). \quad (21)$$

5. Simulation and Analysis

In this section, we do experiments to simulate our model. Our code[1] is based on PyTorch and Numpy.

5.1. Simulation Settings

We use similar simulation conditions and datasets from article [2]. Here we change the edge devices number

1. https://github.com/XinyaoQiu/eecs498project

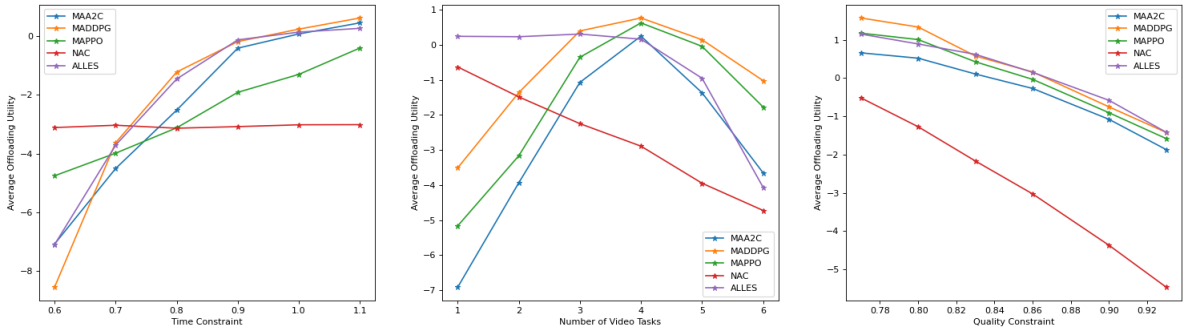

Figure 4. Comparison of average system rewards with different algorithms.

Parameter	Value
Number of VTs N	[1-6]
Number of channels K	4
Time Constraint τ_n	[0.6-1.1] s
Quality Constraint ϵ_n	[0.77-0.93]
Local Quality Coefficient μ^{loc}	0.125
MEC Quality Coefficient μ^{off}	0.13
Local Normalization Coefficient θ^{loc}	1/1600
MEC Normalization Coefficient θ^{off}	1/1700
Bandwidth W	20 MHz
Power Maximum p_{max}	24 dBm
Background Noise Variance σ^2	100 dBm
Uplink Channel Gain h_n^k	8 dB
Local Computing Resource f_n^l	[0.4-2.3] GHz
MEC Computing Resource f^e	5 GHz
Task Size s_n	50 MB
CPU Cycles c_n	2 Gcycles
The VT's energy coefficient κ	$5 * 10^{-27}$
Weight Coefficient λ_1, λ_2	0.6, 0.4

TABLE 1. SIMULATION PARAMETERS.

to a smaller value and dismiss its block-chain part. We use 1-6 mobile phones as VTs and some channels connecting to the MEC server with a fixed number of 4. Other parameters are listed in Table 1.

5.2. Analysis

First, we change three conditions and are to see the convergence process. The three changed conditions are the number of VTs N, time constraint τ_n, and quality constraint ϵ_n.

Figure 3 shows the performance of average system rewards with different N, τ_n, and ϵ_n. Based on our simulation results, we can see the average system rewards would increase as τ_n increase and would decrease as ϵ_n increase because when τ_n is larger or ϵ_n is smaller, the penalty caused by longer processing time or weaker quality would be smaller, so the reward would increase. The reward increases then decreases as the number of agents N increases because when N is small, the MEC system can support sufficient computing resources to handle all offloaded VTs. However, after exceeding some critical point (e.g., $N = 4$), the system utility decreases because the larger the number of offloaded VTs, the higher the competition for resource usage. We see the configured number of channels $K = 4$ is relatively small with respect to the increase in the N, thus the channel bandwidth allocated for each VT decreases when there are more VTs in the system. This increases the offloading latency and thus, decreases the overall offloading reward.

Figure 4 shows the comparison of average converged rewards in different algorithms such as MADDPG, MAA2C, and MAPPO, where NAC and ALLES are baselines. Here to simulate NAC, we make all compression rate β_n^k equal to 0.2, and to simulate ALLES, we make all offloading decision policy x_n^k equal to 1. For different algorithms, we can see MADDPG is a little better than MAA2C and MAPPO if we choose the best choice of parameters. This is because, in the MADDPG, each agent can obtain the full knowledge of other agents' information such as the information on offloading decisions via collaborative interactions. This allows each agent to determine the optimal offloading strategy to achieve the converged point of Nash equilibrium.

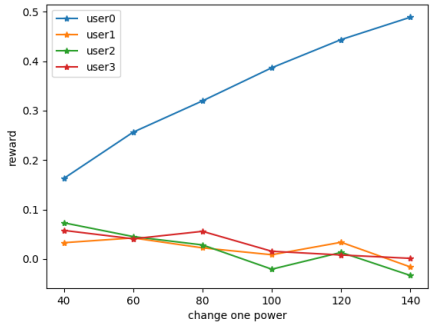

Figure 5. Average agents' rewards when changing one agent's power maximum.

Figure 5, which is the relationship of a single agent's reward with the changed power maximum, shows the influence of agent on agent when we only change one agent's power maximum p_{max}. We can see if we change the user$_0$'s power maximum, its reward increases with other agents' rewards decreasing. This is because it

tends to offload the video task to the MEC server if it has a larger transmitting power. If so, its reward would increase. However, it can cause other agents to have fewer offloading channels, then they need to take more resources to compute locally, which would make their rewards decrease a little.

6. Conclusion and Future Works

In this project, we have learned a lot about edge computing and deep reinforcement learning algorithms. First, we proposed a mobile edge computing system that is widely used today and introduced its computing and offloading process. Then, we give the details of the deep reinforcement learning algorithms and show how it works. Moreover, we create a model that transforms the MEC system into some variables. Given the formulations of reward, we set up the state and action, which is a method to solve optimization problems in deep reinforcement. Also, we simulate the model using PyTorch and Numpy. In the end, we get the result in graphs and analyze the tendency of the change.

Our proposed approach has the potential for future mobile networks, where EDs are able to build distributed intelligent solutions via our MADDPG model for enabling intelligent computation, communications, and network control. In future work, it is of interest to consider fair resource allocation strategies for simultaneously supporting edge computation. The trade-off between quality and energy consumption should also be studied to strike a beneficial balance between these two important design factors before integrating into MEC. Moreover, resource trading solutions should be developed to enable energy purchases for resource-constrained edge nodes in MEC systems.

References

[1] X. Xiong, K. Zheng, L. Lei, and L. Hou, "Resource allocation based on deep reinforcement learning in iot edge computing," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1133–1146, 2020.

[2] D. Nguyen, M. Ding, P. Pathirana, A. Seneviratne, J. Li, and V. Poor, "Cooperative task offloading and block mining in blockchain-based edge computing with multi-agent deep reinforcement learning," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2021.

[3] A. Barto, R. Sutton, and C. Anderson, "Neuron like elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, & Cybernetics*, pp. 1–1, 1983.

[4] Openai, "Openai baselines: Acktr a2c," *https://openai.com/blog/baselines-acktr-a2c/*.

[5] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[6] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2016.

[7] B. Yang, X. Cao, J. Bassey, X. Li, and L. Qian, "Computation offloading in multi-access edge computing: A multi-task learning approach," *IEEE Trans. Mob Comupt.*, pp. 1–1, 2021, doi:10.1109/TMC.2020.2990630.

[8] Z. Shou, X. Lin, Y. Kalantidis, L. Sevilla-Lara, M. Rohrbach, S.-F. Chang, and Z. Yan, "Dmc-net: Generating discriminative motion cues for fast compressed video action recognition," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 1268–1277.