A Deep Reinforcement Learning-based Offloading Scheme for Multi-Access Edge Computing-supported eXtended Reality Systems

Bao Trinh* and Gabriel-Miro Muntean*

***Abstract*—In recent years, eXtended Reality (XR) applications have been employed increasingly in various scenarios in tourism, health care, education, manufacturing, etc. Such applications are now accessible via mobile devices, wearables devices, tablets, etc. However, mobile devices normally suffer from constraints in terms of battery capacity and processing power, limiting the range of applications supported or lowering user quality of experience when using them. One effective way to address these issues is to offload the computation to the cloud servers. The inherent limitation of the cloud computing approach is the long propagation distance to the end user from the processing server, that may result in long latency which is not tolerable by many mobile XR applications. To overcome such limitations, Multi-access Edge Computing (MEC) is proposed to bring the mobile computing, network control and storage services to the network edges (for example at base stations, access points, etc) so that the computation-intensive and latency-sensitive applications can be deployed at the resource limited mobile devices. This paper proposes a Deep Reinforcement Learning-based offloading scheme for XR applications (DRLXR). The problem is formulated as a utility function optimization equation that takes into account both energy consumption and execution delay at devices and the Markov Decision Process (MDP) framework is employed as a decision maker. Next the Deep Reinforcement Learning (DRL) technique is employed to train and derive the close-to-optimal offloading decision for mobile XR devices. The proposed DRLXR scheme is then validated in a simulation environment and compared against other novel offloading schemes. The simulation results show how our proposed scheme outperforms the other counterparts in terms of total execution latency and energy consumption.**

***Keywords*— eXtended Reality, Offloading, Multi-access Edge Computing, Deep Reinforcement Learning, Energy efficiency, Quality of Service**

I. Introduction

The eXtended Reality (XR) applications benefit from the latest developments in 5G and beyond network communications. XR can be defined as the combination of virtual 3D objects with real world content [1] consumed via smart devices such as handheld smart phones or head mounted glasses [1,2]. Depending on the balance between the amount of virtual content and reality, XR is denoted as Augmented Reality (AR), Mixed Reality (MR), or Virtual Reality (VR). However, regardless of the labeling, there is an exponential increase in XR applications in various scenarios, including in health care [2], tourism [3], education, and manufacturing.

Figure 1 illustrates a generic XR system, with the following essential components:

- **Input sensors** that acquire information via various type of built-in or companion sensors, such as: gyroscope, location, cameras, etc.

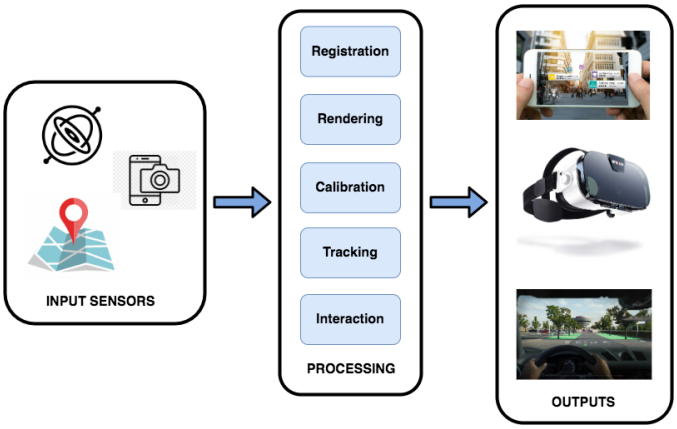

Figure 1: Components of an XR system [4]

- **Processing modules** are responsible for processing the collected data, which is performed locally or via offloading to a cloud server, fog server or edge server, depending on the required computational complexity and available processing power.
- **Outputs** refer to post-processing actions that involve the XR content display, including streaming of high definition video content [5], activating actuators and interaction with external devices. This stage uses head-mounted displays (HMD) [6], handheld displays [7] and/or devices such as haptic gloves, olfaction dispensers [8], etc.

Despite the latest fast pace of hardware design and development, the mobile devices used for XR applications are still limited in terms of resources in comparison to desktops or servers. The cost of high mobility and reduced size is paid in terms of battery capacity and processing power. On the other hand, due to the complex algorithms used mostly in relation to video content processing, XR applications require high computational resources. An effective way to cope with the challenge of supporting immersive XR applications run on resource-limited mobile devices is to offload the computation via the network to resource-rich devices, such as cloud or edge servers.

Cloud computing has been a successful new computing paradigm. Its intrinsic idea is the centralization of computing, storage and network management in the cloud, providing support via data centers, backbone networks and cellular core networks [9], [10]. In order to execute computation in the cloud, the mobile devices and servers are required to operate offloading frameworks, such as MAUI [11], or ThinkAir [12]. However, recently, the function of cloud computing is being increasingly moved towards the network edges, closer to user devices [13]. By harvesting the idle computation power and storage space distributed at network edges, sufficient support is made available to XR applications to perform computation-intensive and latency-critical tasks at user mobile devices. This principle is behind the Multi-Access Edge Computing (MEC) [14] paradigm, in which mobile devices can communicate and get support from MEC servers via multiple wireless communications technologies such as LTE, 5G, WiFi or a combination of them [15]. The general architecture of a MEC system is illustrated in Figure 2.

In a MEC-enhanced cloud computing context, the challenge remains to decide which XR processing-related tasks are to be offloaded and where, in order to best balance XR application requirements on one hand and make efficient use of device, MEC and cloud

*Bao Trinh Nguyen and Gabriel-Miro Muntean are with the Insight SFI Research Centre for Data Analytics and Performance Engineering Lab, School of Electronic Engineering, Dublin City University (DCU), Dublin, Ireland. emails: bao.trinh@insight-centre.org, gabriel.muntean@dcu.ie

[1]Google Glasses, https://www.google.com/glass/start/

[2]Microsoft Hololens, https://www.microsoft.com/en-us/hololens

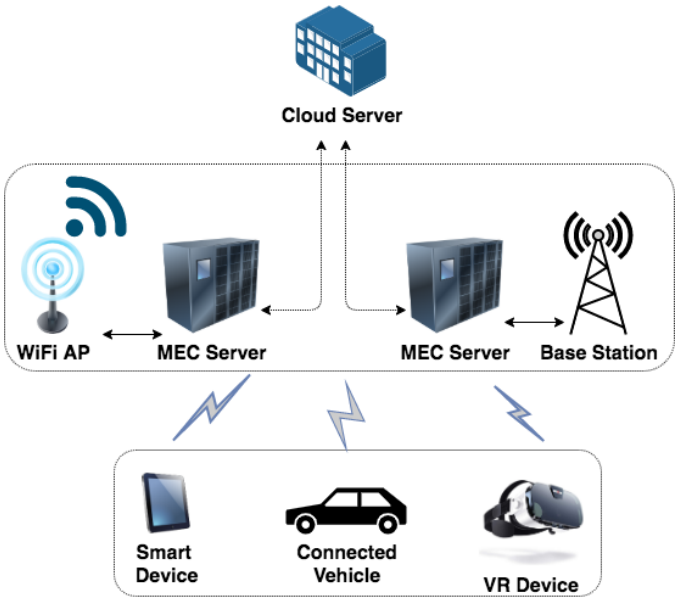

Figure 2: General architecture of a MEC-enhanced cloud computing system

computational, storage and network resources, on the other hand. This is not trivial and diverse solutions were proposed using heuristic or complex optimisation approaches [16].

This paper proposes **a Deep Reinforcement Learning-based offloading scheme for XR applications (DRLXR)** that distributes the computation between device, MEC and cloud in order to best balance the XR application performance and energy efficiency in given networked system resource constraints. The contributions of this paper are as follows:

- A three-layer architecture for XR systems is proposed, and the energy-efficient computation offloading issue to minimize the overall power consumption while satisfying the stringent delay constraints of XR applications is focused on.
- The problem is formulated by using the Markov Decision Process (MDP) framework and the close-to-optimal offloading decision making is derived via a Deep Reinforcement Learning (DRL) technique. The XR applications are decomposed into small tasks and are represented using Graph Theory.
- Finally, the proposed DRLXR solution is evaluated using Network Simulator NS-3 and Open Gym AI library and is benchmarked against other novel offloading schemes.

The rest of this paper is organized as follows: Section II surveys some novel offloading schemes found in the research literature. The technical background of the Deep Reinforcement Learning (DRL) is discussed in Section III. Section IV provides details about our proposed solution, including system architecture, problem formulation and the DRL-based offloading algorithm. We evaluate the proposed scheme in a simulation environment and discuss the results in Section V. Finally, the paper is concluded in Section VI.

II. Related Works

This sections discusses some state-of-the-art offloading schemes proposed in the research literature. According to their type of offloading, four main groups of such schemes are considered: *i)* binary offloading, *ii)* partial offloading, *iii)* stochastic model-based and *iv)* deep learning-based offloading schemes.

A. Binary offloading

Kumar *et. al.,* [17] provided guidelines for making offloading decisions with the aim to minimize both computation latency and energy consumption for mobile devices in a traditional cloud computing fashion. The key factors that are considered for offloading include: CPU speed at mobile devices and cloud servers, data size, and fixed rate of wireless communication links. However, the assumptions made in this paper are not realistic. The channel gain of wireless communication is time-varying. Besides, the CPU power consumption increases in proportional to CPU cycle frequency. So, adaptive offloading schemes are necessary to overcome such limitations.

The authors of [18] and [19] employed an optimization framework to formulate the offloading decision with the aim to minimize energy consumption. In [18], the researchers considered multimedia applications, which require the task to be completed within the deadline with a given probability τ. The offloading decisions are made following which computation modes (either local computing or offloading) incur less energy consumption. Internet of Things (IoT) systems where sensor nodes are powered using wireless power transfer (WPT) technology are considered in [19]. Alongside reducing the energy consumption, the optimization proposed in [19] also aims to maximize the computation rate of all network nodes.

In reality, mobile applications normally consist of multiple procedures/functions/components, like the components of the XR system illustrated in Figure 1. In this case, offloading the whole program or completely performing local execution as suggested by binary offloading is not suitable.

B. Partial offloading

Partial offloading of tasks refers to the decomposition of one application into two parts: one offloaded to edge servers and the other one executed locally at the mobile device. Kao *et al.* in [20] modeled the dependency between different procedures/components of an application by using a Directed Acyclic Graph (DAG). Next, the balance between energy consumption and delay is formulated via an optimization equation. Saleem *et. al.* [21] studied the problem of minimizing latency by considering the local energy constraint, while taking into account the limited energy availability at the user. This has a high impact on the data segmentation decision. Despite the manifold benefits, such partial offloading schemes are not examined under time-varying radio communications channels, such as poor channel conditions and scarce bandwidth may affect the offloading latency. In such case, multiuser cooperative edge computing can be considered as a promising solution, where proximal devices can collaborate with each other to scale up the services. An approach that combines MEC and Device-To-Device (D2D) communications is proposed in [22]. Based on monitoring the interference on the radio communications link, a device can decide to offload task execution to the edge server, to another nearby device, or execute it locally. [23] proposed a joint solution based on Mixed-Integer Nonlinear Programming (MINLP) that considered multi-task partial computation offloading and network flow scheduling problem in multi-hop network environments. The output of the proposed optimization problem is a partial offloading ratio.

C. Stochastic Task model-based Offloading

[24], [25], [26], and [27] are among solutions that consider stochastic task models that are characterized by random task arrivals. In [24], the problem of minimizing long-term execution cost was solved via jointly optimizing computation latency and energy consumption. The proposed scheme employed a semi-MDP framework to control local CPU frequency, modulation scheme and data rates. Zhang *et. al.* [25] proposed an optimization based offloading scheme for unmanned aerial vehicle (UAV) systems that aim to minimize the energy consumption subject to the constraints on the number of offloading computational tasks. These tasks were assumed to arrive in stochastic manner and be independent and identically distributed (iid). In [26], the problem of stochastic computation offloading is formulated by using the MDP framework and solved via using Q-learning algorithm. A joint solution that combines channel allocation and resource management for making offloading decision (JCRM) with the aim to maximize network utility was proposed in [27]. JCRM then leverages the Lyapunov optimization technique to make optimal offloading decisions.

D. Reinforcement Learning-based Offloading

Since there is limited training data and novel applications appear continually, supervised learning becomes difficult for feature learning. Although unsupervised learning is promising to exploit the features of network traffic, it is challenging to achieve real-time processing [28]. On the other hand, reinforcement learning paradigm can be used without having access to a pre-existing data set for training. Training can be achieved via direct interaction between learning agent and surrounding environment.

[29], [30], and [31] proposed to make use of reinforcement learning and/or combine it with deep learning in order to propose diverse offloading schemes for MEC-enhanced Internet of Vehicle (IoV) systems. In [29], Li *et. al.,* proposed an online reinforcement learning method from the feedback and traffic patterns to balance traffic loads. In order to fulfil high-efficient traffic management, a joint communication, caching and computing problem was investigated in [30]. [31] proposed an offloading scheme that addressed the trade-off between energy consumption and delay for IoV system. The RL based solution were then employed to derive offloading strategy for IoV nodes.

The authors of [32] considered IoT nodes that are powered following energy harvesting. The proposed scheme allowed IoT devices to select the edge server and offloading rate based on current battery level and previously monitored radio transmission rate. DRL was employed to improve the offloading performance in a highly complex state space.

Wang *et. al.* [33] transformed the original joint computation offloading and content caching issue into a convex problem then solved it in a distributed and efficient way. Hao *et. al.* [34] considered the offloading problem that takes into account both constraint of computing and storage capacity of mobile devices when optimizing the long term latency. The proposed scheme was formulated by using DRL and the solution proposed showed noticeable results in terms of convergence time and latency reduction.

Wang *et. al.* [35] proposed a Meta Reinforcement Learning-based scheme (MRLCO) to provide optimal offloading decision for User Equipment (UE). Mobile applications are modeled as Directed Acyclic Graphs (DAG). The author employs Meta Reinforcement Learning (MRL) in order to find the close-to-optimal offloading decisions for UEs with the aim to reduce latency. UE applications are defragmented into multiple sub-tasks. Each sub-task is then decided to be processed locally or offloaded to a virtual machine at MEC server. MRLCO outperforms the other baseline algorithms in terms of average latency. The main disadvantage of MRLCO is the lack of UE mobility and energy consumption consideration.

Despite pursuing different avenues, most of the existing works did not consider a holistic approach that takes into account the complexity of the latest applications, such as the XR ones. These applications comprise of many small tasks and their performance is influenced jointly by network conditions and energy consumption. This gap is bridged in this article.

III. Technical Background

This section briefly discusses the background related to Markov Decision Process (MDP) and Deep Reinforcement Learning (DLR), techniques used in the proposed solution.

A. Deep Reinforcement Learning

DRL is a research area of machine learning that combines Deep Neural Network and Reinforcement Learning (RL). Deep learning enables RL to scale problems that were previously intractable, i.e., the environment with a high dimensional state and large action spaces. Some successful applications of DRL include video games, robotics, etc.

In general, DRL can be formulated as an Markov Decision Process (MDP) framework using a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where:

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}^{a}_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}^{a}_{s} = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$
- γ is a discount factor $\gamma \in [0, 1]$

MDP uses a definition of total expected return, *return* G_t, or the total discounted reward from time-step t as follows:

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{1}$$

A policy π in an MDP is a distribution over actions given states:

$$\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]. \tag{2}$$

The goal of MDP is to derive an optimal *policy* $\pi^*(a|s) = \mathbb{P}[A_t = a|S_t = s]$, which is a distribution of actions in corresponding states, so as to maximize the total discounted cumulative reward.

In general, there are two main approaches to solving RL problems: value function based and policy search based methods.

1) **Value functions methods** are based on estimating the value (or expected return) of being in a given state. The state-value function $v_\pi(s)$ is the expected return when starting from state s and following policy π:

$$\begin{aligned} v_\pi(s) = &\ \mathbf{E}_\pi[G_t | S_t = s] \\ = &\ \mathbf{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]. \end{aligned} \tag{3}$$

The optimal policy, denoted as π^*, has a corresponding state value function $v_*(s)$ that is defined as:

$$v_*(s) = \max_\pi v_\pi(s). \tag{4}$$

If we know the value of $v_*(s)$, the optimal policy can be derived by choosing among all available actions in state s_t and picking the action a that maximizes $E_{s_{t+1} \sim \mathcal{P}(s_{t+1}|s_t, a)}$. In a RL environment, as the state transition probability matrix $\mathcal{P}$ is not available, another function, state-action value function $q_\pi(s, a)$ is constructed as follows:

$$\begin{aligned} q_\pi(s, a) = &\ \mathbf{E}_\pi[G_t | S_t = s, A_t = a] \\ = &\ \mathbf{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s]. \end{aligned} \tag{5}$$

The best policy, given $q_\pi(s, a)$ can be found by choosing a greedily in every state: $argmax_a q_\pi(s, a)$. Under this policy, the value $v_\pi(s)$ can be derived by maximizing $q_\pi(s, a)$: $v_\pi(s) = \max_a q_\pi(s, a)$

2) **Policy search methods** do not maintain a value function model, but directly search for an optimal policy π^*. In general, a parameterized policy π_θ is chosen, where parameters θ are updated to maximize the expected return $E[R|\theta]$ using either gradient-based or gradient-free optimization [36]. Gradient-free methods find the best policy via using heuristic search across a predefined class of models. For gradient-based learning, the gradient can be estimated [37].

In order to combine the advantages of value function and policy search methods, a hybrid solution that employs both value functions and policy search, named Actor-Critic [38], was introduced. **Actor-Critic method** combines a value function with an explicit representation of the policy, resulting in actor-critic methods, as shown in Figure 3. The **actor** (policy) learns by using feedback from the **critic** (value function). Actor-Critic methods use the value function as the baseline for policy gradients, so that the only fundamental difference between the actor-critic method and other baseline methods is that the actor-critic method utilizes a learnt value function. Some advantages of Actor-Critic methods [38] include: *i)* they require minimum computation in selecting actions in comparison to the other two methods; *ii)* they can learn an explicitly stochastic policy or optimal probabilities of selecting various actions. Due to these advantages, the Actor-Critic method is employed as a decision maker for XR device task offloading. This is discussed in details in the next section.

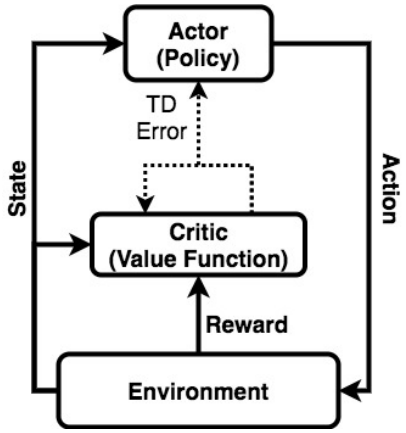

Figure 3: The concept of Actor-Critic [38]. The actor (policy) chooses an action following the received state from environment. At the same time, the critic (value function) receives the state and reward resulting from the last interaction. The critic uses TD error calculated to update itself and the actor.

Table I: Abbreviations

Parameter	Meaning
CPU	Central Processing Unit
CSI	Channel State Indicator
DAG	Directed Acyclc Graph
DRL	Deep Reinforcement Learning
LSTM	Long Short Term Memorty
MEO	Multi-Access Edge Orchestration
OSS	Operations Support System
RSSI	Received Signal Strength Indicator
VM	Virtual Machine

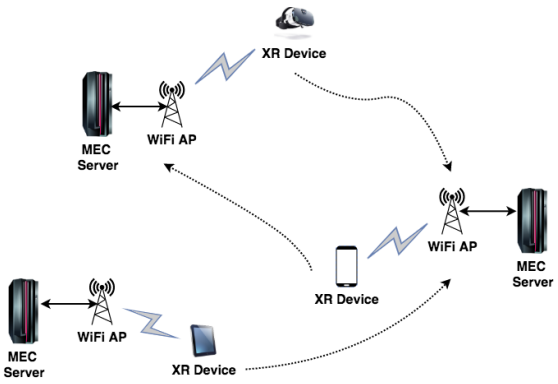

Figure 4: Testing topology

IV. Problem Formulation

This section discusses the proposed offloading scheme. First, the system architecture is described and then the details of problem formulation based on MDP are provided. Finally, the DLR-based offloading scheme is introduced in details. We include all the abbreviations used in this paper in Table I.

In order to evaluate and compare our proposed schemes against another algorithms, we use the following metrics:

- Average energy consumption (in Joules) over all devices
- Average total completion time of tasks

A. System Architecture

The general architecture of the MEC-enhanced network system is considered to consist of three levels: core network, edge network, and XR devices, as illustrated in Figure 5.

The Operations Support System (OSS) and Multi-Access Edge Orchestration (MEO) are located at the top core network level. The OSS block is responsible for receiving requests from customers, and determines requests granting, sending the requests to MEO. MEO maintains an overall view of the MEC-based system, knowing the available resources, services and deployed MEC hosts, and it also monitors the topology. MEO also selects the best hosts where to deploy an application, considering available resources, services availability and constraints such as latency.

At the Edge Network level, the major components are MEC Server, and MEC Platform. The latter is responsible for managing the life cycle of both applications and MEC platforms, informing the MEO if any relevant event happens. MEC platform manager allows for platform configuration and applications life cycle procedures.

Finally, at the bottom are XR devices that are running high computation-intensive applications, such as: deep learning-based object detection, 360° video streaming, etc. and need to offload some tasks to MEC servers.

Next, the block diagram for MEC server and XR devices, as illustrated in figure 6 is discussed.

- *At mobile XR device:* The *Application Monitor* block is responsible for monitoring all applications running in parallel at the device. A *Energy Consumption monitor* block notifies the remaining battery level and depletion rate. The *Channel State Information (CSI)* module keeps tracking the signal level, in terms of Received Signal Strength Indicator (RSSI). All these three blocks provide information to the *Local Trainer* for collecting data and *Deep RL based Decision Maker* for calculating the offloading decision. The computation is either fed into *Offloading Scheduler* module and then offloaded via *Radio Transmission Unit* to MEC server or executed locally at *Local Executor* block.
- *At MEC server:* The *Data Aggregation* part collects all the requests from all devices from *Radio Transmission Unit* in the vicinity then feeds them into *Traffic Management* block. The *Traffic Management* block manages all the Virtual Machine (VM) and assigned resources for corresponding mobile device's requests. All requests are then processed and the responds are sent back to XR devices via *Remote execution service* block. MEC sever also has connections to Remote Cloud servers, but in the scope of this paper, we ignore the effect of such communications.

B. Definitions and Assumptions for the Optimization Model

1) Multitasking Application Modelling: In this paper, we assume that an XR device is executing a resource-hungry multitasking XR application by offloading some sub-tasks to the MEC server. Such offloading decisions aim to minimize the device's energy consumption, whereas the predefined stringent requirements of completion time of the application are met.

A multitasking application can be decomposed into a set of fine granularity atomic non-preemtive tasks. We use a Directed Acyclic Graph (DAG) to formulate the dependencies between these tasks. Denote $G = (V, E)$ as the construction of multitasking, where V is the tasks and E refers to the dependencies. The total number of tasks of the application is $N = |V|$.

Depending on how developers model the applications [39] [40], there are, in general, three types of multitasking DAG: *i)* Sequential, *ii)* Parallel, and *iii)* General dependencies. Due to their simplicity, the Sequential and Parallel models cannot reflect the complexity of dependencies between sub-tasks of an XR application. Therefore, in this paper, we consider a general dependencies model for XR applications, as illustrated in Figure 7. Each node from 1 to $N = |V|$ represents a computation task of the application that can be executed locally or offloaded to the MEC server. Normally, for an XR initiated application, the first and last steps (i.e. 1 and N), which receive I/O data and display the final results on the device screen, respectively, must be executed at the XR device. XR devices decide for the tasks

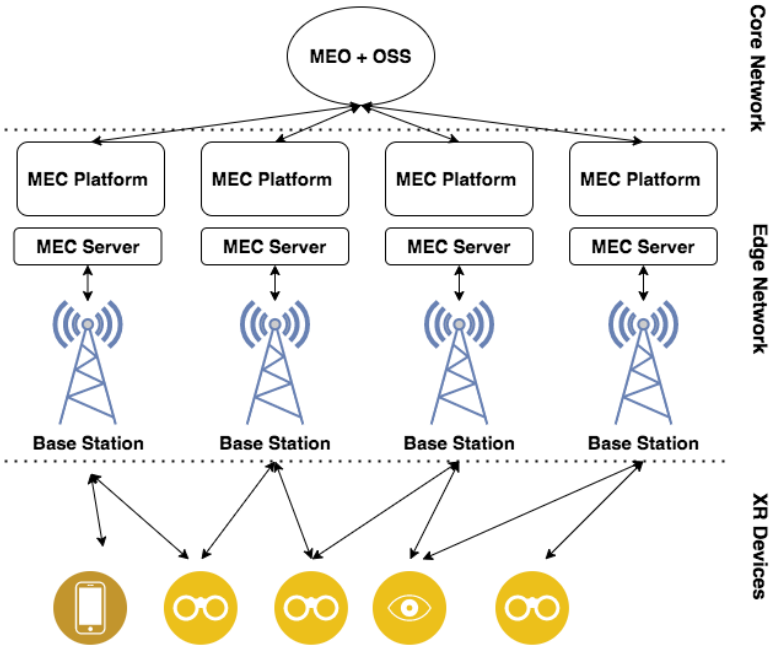

Figure 5: System Architecture of a MEC-based Network System

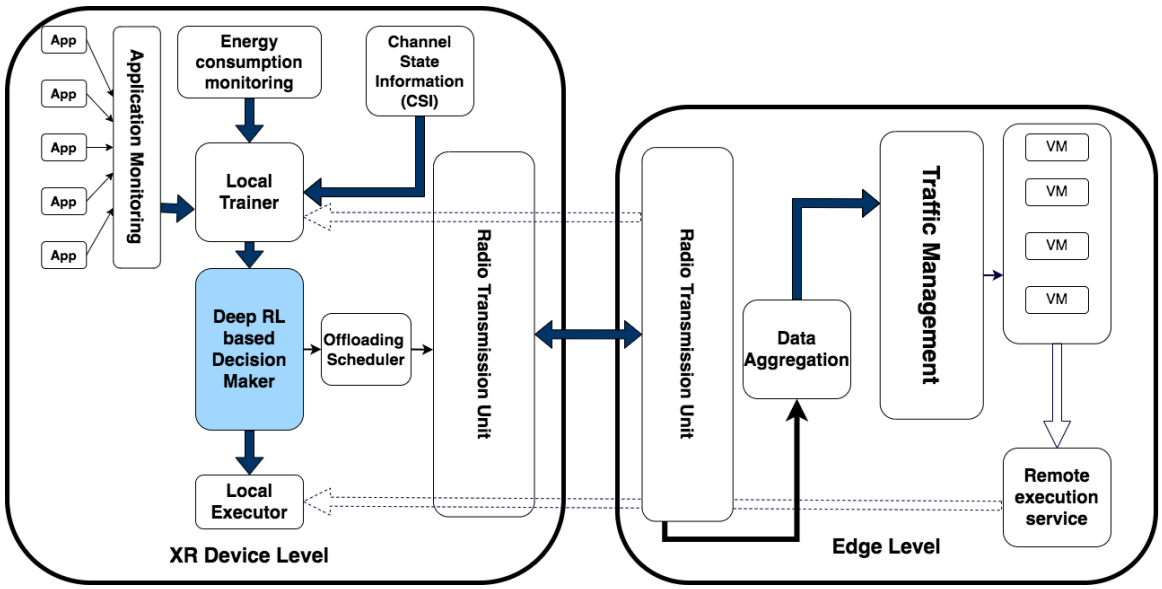

Figure 6: Block diagram of the proposed solution

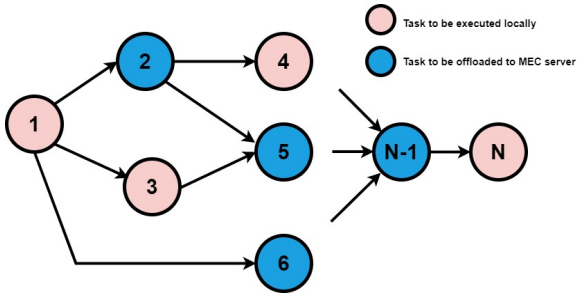

Figure 7: Example of a general dependencies model for XR application computation tasks

associated with the remaining nodes (i.e. from 2 to $N-1$) if they will be offloaded or executed locally. The tasks that are being offloaded to MEC server is highlighted in blue whereas the pink ones refers to the tasks that are executed locally at XR devices.

2) Energy Consumption Model: In general, the energy consumption of mobile device can be decomposed into four parts:

- The energy consumption by the local CPU due to local processing, denoted as $\epsilon_{processing}$.
- The energy consumed by wireless network interface when uploading to remote servers code source and data of offloaded tasks, denoted as ϵ_{up}.
- The energy consumed by wireless network interface when downloading task execution results from MEC servers, denoted as ϵ_{down}
- The energy consumed by wireless network interface when it is in idle mode. This mode is enabled when the mobile device is waiting for the execution of offloaded tasks, denoted as ϵ_{idle}.

Using the model from [40], [41] and following the previous considerations, the energy consumption ϵ for task t can be derived as follows:

$$\epsilon^t = \epsilon^t_{up} + \epsilon^t_{down} + \epsilon^t_{processing} + \epsilon^t_{idle}. \quad (6)$$

In case task t is executed locally, we have $\epsilon_{up} = \epsilon_{down} = 0$.

By summing up, the total energy consumption E of the application with n tasks is:

$$E(t) = \sum_{t=1}^{n} \epsilon^t. \quad (7)$$

3) Completion Time: When the computation is executed locally, it will utilize the computing resources of the mobile device, including CPU, memory, storage, battery capacity, etc. Denote CPU cycle frequency as f_m, task input-data size as L (bit), computation workload/intensity X (in CPU cycles per bit), the execution latency for local processing for task t is:

$$\tau^t_{local} = \frac{LX}{f_m}. \quad (8)$$

For the task that is offloaded to the MEC server, the time spent on transferring data is calculated as follows:

$$\tau^t_{off} = \tau_{up} + \tau_{down} + \tau_{queue} + \tau_{process}. \quad (9)$$

The completion of an application is obtained when the final task $n = |V|$ is executed. We use T to refer to the processing duration of all application tasks, plus transmission time to/from the MEC server.

$$T(t) = \sum_{t=1}^{n} \big((1 - x^t)\tau^t_{local} + x^t \tau^t_{off}\big), \quad (10)$$

where x^t denotes the offloading decision at time t. $x^t = 1$ refers to the offloading of the task at time t to a MEC server, and $x^t = 0$ indicates local task execution at the level of the XR device.

In order to meet the strict deadline τ_{max}, we have the condition: $T(t) < \tau_{max}$.

The utility function that takes into account the energy consumption and completion time is derived as follows:

$$U = -(\sigma \tilde{E}(t) + (1 - \sigma)\tilde{T}(t)), \quad (11)$$

where $\tilde{E}(t)$ and $\tilde{T}(t)$ are energy consumption and completion time values, after normalization.

C. DRL-based Offloading Algorithm Design

This section presents the algorithm of the DRL-based offloading scheme for XR devices. First, the problem formulation is described. It employs the Markov Decision Process (MDP) framework, as follows.

1) **STATE SPACE**
The state space of the agent (located at XR devices) includes all possible observations. Each observation is specified by a tuple $\langle \mathcal{P}, \mathcal{E}, \mathcal{C} \rangle$, where:
   - $\mathcal{P} = 0, 1, \ldots N$ denotes the set of Application sub-tasks that are specified as single-chain applications with N being the number of tasks.
   - $\mathcal{E}$ denotes the remaining energy of the XR device (expressed as percentage %)
   - $\mathcal{C}$ refers to the Channel State Information (CSI), monitored in the current state.
2) **ACTION SPACE**
The Action space incorporates $|A|$ available actions that the agent can perform in a given state. We define the action space with two values: $A = 0, 1, 2$ where: 0 and 1 denote local computing and offloading to MEC server, respectively, and 2 indicates that the device is in idle or waiting states.
3) **REWARD FUNCTION**
The reward signal is calculated by using eq. (11) to calculate the feedback of the chosen action for a specific state.

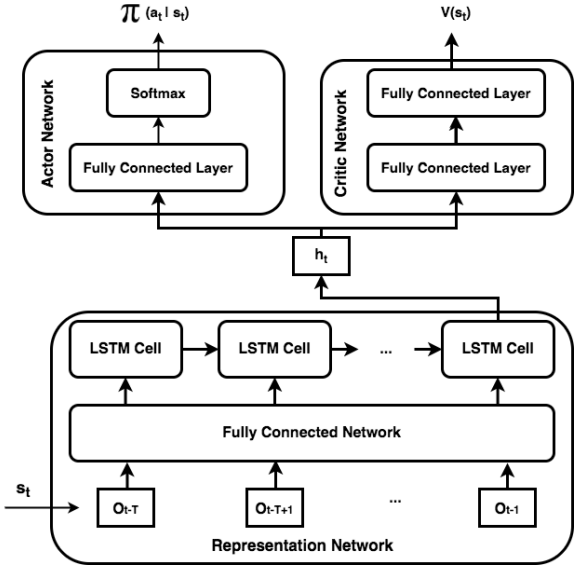

Figure 8: The LSTM based representation network

Figure 8 illustrates the Long Short Term Memory (LSTM) Actor Critic (AC) based architecture for solving the MDP. LSTM is a powerful artificial neural network architecture that is widely used in prediction and classification, such as in time series data [42]. In this paper, LSTM is used to learn the temporal regularity of states in terms of RSSI, energy consumption and application sub-task status due to device mobility. Details of the LSTM AC-based architecture are described next.

- *Representation network* incorporates a fully connected (FC) layer and an LSTM layer. This network is responsible for detecting the temporal correlation of states. The FC layer takes the buffer $\mathbf{B}$ as input and then feeds the extracted feature tensor to the LSTM layer. The output of the LSTM layer is the variation regularity of of states from the last T observation vectors in the buffer. After T updates, last LSTM cell outputs a completed representation of the environment h_t that is then used as input for both Actor and Critic networks.
- *Actor network* comprises one FC layer that takes the output from the *representation network* and generates actions for the current states that is specified by a Softmax function. The output of Softmax function is a probability of different available actions $\pi(a_t|s_t)$. Then, the taken action is sampled following $\pi(a_t|s_t)$.
- *Critic network* estimates value of current state and incorporates two FC layers. The first FC layer takes the h_t from representation network and extract value-related features. Then, the second FC layer output the estimated state value $V(s_t)$

Algorithm 1 presents the DRLXR scheme in details. θ and $\mathbf{w}$ are Actor and Critic network parameters. We use a buffer $\mathbf{B}$ with length T to concatenate a series of states to feed into the LSTM layer. We initialize the buffer via running a loop with T iterations to take a series of states into $\mathbf{B}$. From the beginning of each loop, all states in the buffer $\mathbf{B}$ are concatenated and fed into the representation network. The output h_t is then considered as input of both Critic and Actor networks. The action a_t is taken via sampling from the output of the Actor network and the next state s_t is then appended into the buffer $\mathbf{B}$. The output of the Critic network is the estimated value of $V(s_t)$. Next, the agent continues to concatenate the data from buffer $\mathbf{B}$ to form another input s_{t+1}. The value $V(s_{t+1})$ is then estimated from the output of Critic network. We calculate the Temporal Difference (TD) error δ by using equation $\delta = r_t + \gamma V(s_{t+1}) - V(s_t)$. If α_A and α_C are the learning rates of Actor and Critic networks, respectively, the values of θ and $\mathbf{w}$ are updated according to eq. (12) and eq. (13).

The parameters θ for the Actor network and $\mathbf{w}$ for the Critic network are updated based on the following equations:

$$\theta \leftarrow \theta + \alpha_A \delta \nabla ln \pi(a_t|s_t, \theta). \quad (12)$$

Algorithm 1 Deep Reinforcement Learning based Offloading

procedure $DRLXR$
Initialize Actor network parameters θ, Critic network parameters $\mathbf{w}$
Initialize an empty replay buffer $\mathbf{B}$ of length T
Output Offload decision $0, 1, 2$
for $i = 1$ to T do **do**
Randomly choose an action $a_i \in A$ and perform a_i
The agent takes the next state O_i
Append O_i to buffer $\mathbf{B}$
while TRUE **do**
Concatenate states in the buffer $\mathbf{B}$ to form $s_t = \{O_{t-T}, \ldots, O_{t-1}\}$
Feed O_t to the representation network and take the output h_t
Feed h_t to Critic network and calculate $V(s_t)$
Feed h_t to Actor network and take $\pi(a_t|s_t)$ and perform a_t
The agent receives the reward r_t and gets the new observation O_t
Append O_t to the buffer $\mathbf{B}$
Concatenate observations in the buffer $\mathbf{B}$ to form $s_{t+1} = \{O_{t-T+1}, \ldots, O_t\}$
Feed O_{t+1} to the representation network and take the output h_{t+1}
Feed h_{t+1} to Critic network and calculate $V(s_{t+1})$
Calculate Temporal Difference (TD) error $\delta = r_t + \gamma V(s_{t+1}) - V(s_t)$
Update θ of the Actor network following eq. (12)
Update $\mathbf{w}$ of the Critic network following eq. (13)

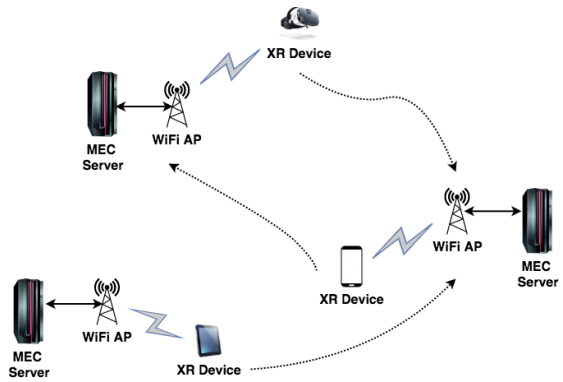

Figure 9: Testing topology

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha_C \delta \nabla \hat{v}(s_t, \mathbf{w}). \tag{13}$$

V. Performance Evaluation

This section discusses the validation of our proposed scheme in a simulation environment under different test scenarios.

A. Experimental Setup

We build our testing environment in Network Simulator NS-3 [43]. Then, we implement the Actor-Critic model on TensorFlow 2.4[3] and train the agent on OpenGym AI [44] framework. The computer for testing is installed with Ubuntu Linux 18.0 LTS and has 32 GB memory and an Intel Core i7 6^{th} gen processor. In this testing, there is no need for using a GPU for training. Figure 9 illustrates the network topology employed for testing. We assume that a number of mobile XR devices are moving around in an area at walking speed under the coverage of some MEC servers.

[3]https://blog.tensorflow.org/2020/12/whats-new-in-tensorflow-24.html

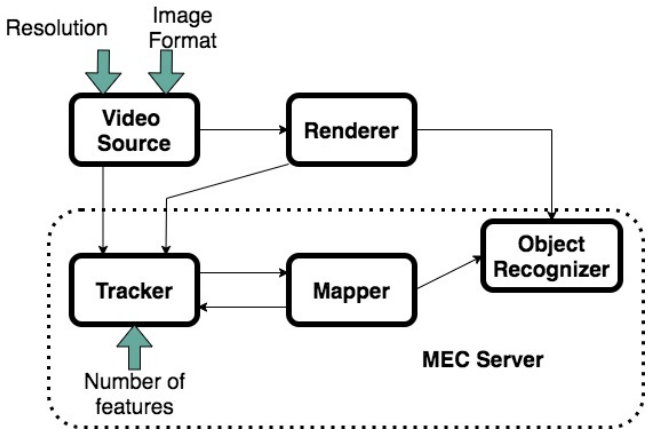

Figure 10: Example of main computation components in the XR application [45]

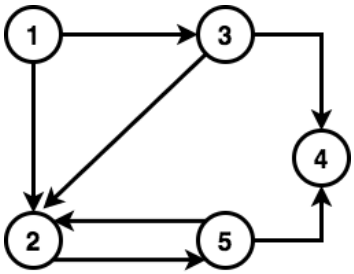

Figure 11: General dependency of a XR application

Figure 10 [45] illustrates the computation components of an XR application. The functionality of the major components is briefly introduced next.

- *Video Source* fetches video frames from the camera hardware.
- *Renderer* renders an overlay on the screen.
- *Tracker* component processes the camera frames and estimates the camera position with respect to the world based on a number of visual feature points. The more feature points we use, the more stable the tracking. Increased feature points also makes tracking the camera more robust during sudden movements.
- *Mapper* creates a model of the world by identifying new feature points and estimating their position, which can then be used for tracking.
- *ObjectRecognizer* tries to recognize known objects in the world and notifies the *Renderer* of their 3D position when found.

Depending on the latency requirement and current energy consumption situation, XR device can decide one component is either executed locally or offloaded to MEC server. For example, *Tracker*, *Mapper*, *ObjectRecognizer* components can be offloaded to MEC server whereas *Video Source* and *Renderer* computation are executed locally as illustrated in Figure 10. Based on the relation between components, we built the dependency model based on DAG, as illustrated in Figure 11. We assume that multiple applications are running in parallel in an XR device.

In order to evaluate and compare our proposed schemes to other algorithms, we use the following metrics:

- Average energy consumption (in Joules) across all devices
- Average total completion time of tasks

We compare our proposed solution DRLXR with the following baseline algorithms:

Table II: Simulation Setup Details

Parameter	Value
Simulation Length	100000 seconds
No. of nodes	5, 10, 15, 20, 30, 40, 50
No. of MEC servers	0, 2, 4, 6, 8, 10, 15
Cell layout	Single cell; Radius - 50 meters
WiFi Mode	IEEE 802.11ac 2.4/5.0 GHz
Antenna Model	Isotropic Antenna Model
WiFi 802.11ac	2.0 Mbps
Walking speed	5km/h

- No-Offloading scheme *(NO)* [46]: All tasks are handled locally at devices and all data is received from the network.
- Greedy policy (*Greedy*): Each task is greedily assigned to the XR device or a MEC server based on its estimated completion time.
- Q-Learning method *(Q-Learning)* [47]: That is a traditional temporal difference algorithm, which always pursues the largest reward in the next time step. In addition, Q-Learning always records rewards in each iteration. When system state or action spaces are large, this solution tends to use large memory.
- Dynamic RL Scheduling *(DRLS)* [48]: A reinforcement learning-based offloading scheme that combines both D2D and MEC systems.

B. Results Discussion

In all cases, the energy consumption and total completion time of the *Non-Offloading (NO)* scheme are unchanged due to the local execution. We consider this case as the baseline for the other schemes to compare against.

Figure 12 illustrates the average energy consumption with different offloading data sizes. We observe that the energy consumption of XR devices is proportional to the increase in the offloaded data size due to the energy usage for transmitting and receiving data over the radio link. When the offloading data size is small (less than 40MB), the average energy consumption of all cases is similar (experiences slight differences only). At the breaking point of 80MB, the *Greedy* method results are increasing sharply. Although the other schemes perform more stable, *DRLXR* has better results with lower energy consumption of about 150 $\times 10^6$ Joules in comparison to 160 $\times 10^6$ Joules and 177 $\times 10^6$ Joules of *DRLS* and *Q-Learning* methods, respectively.

Figure 13 and Figure 14 illustrate the average energy consumption and average total completion time with different numbers of MEC servers, respectively. It can be observed that the energy consumption decreases with the increase in the number of MEC servers used. Initially when there is no MEC server, all schemes consume about 150 ($\times 10^6$ Joules) energy and require 129 s completion time, respectively. The breaking point appears when number of MEC servers is equal to 8 and all schemes except *NO* show stability. *Greedy*, *Q-Learning*, and *DRLS* methods' average energy consumption are around 77 $\times 10^6$ Joules, 75 $\times 10^6$ Joules and 63 $\times 10^6$ Joules, respectively, whereas the result of *DRLXR* is around 60 $\times 10^6$ Joules. A similar situation also occurs at about 8 MEC servers and above related to the results of the average total completion time. Starting from 130 s, the average total completion time of all schemes decreases and is kept stable at 70 s, 68 s, 62 s and 60 s for *Greedy*, *Q-learning*, *DRLS* and *DRLXR*, respectively. The following are the reasons that explain the benefits of using *DRLXR* in comparison with the alternative solutions. In *DRLS*, XR devices can offload the computation to other peers via D2D communications, that lead to higher total energy consumption. On the other hand, *Q-learning* does not specify an exploration mechanism, but a greedy manner and requires all actions be tried infinitely in all states. Such a mechanism has lower accuracy when making offloading decisions. Unlike them, *DRLXR* employs the Actor-Critic method that specify a full exploration mechanism by the action probabilities of the Actor. In addition, *DRLXR* is trained from historical data that lead to higher accuracy of offloading decisions.

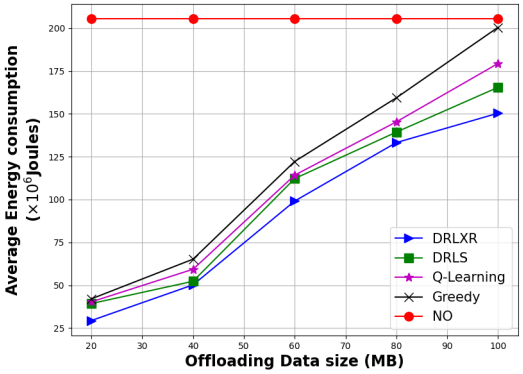

Figure 12: Average energy consumption with various offloading data sizes

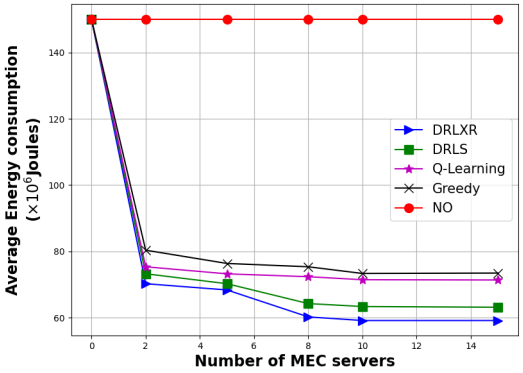

Figure 13: Average energy consumption with various number of MEC servers

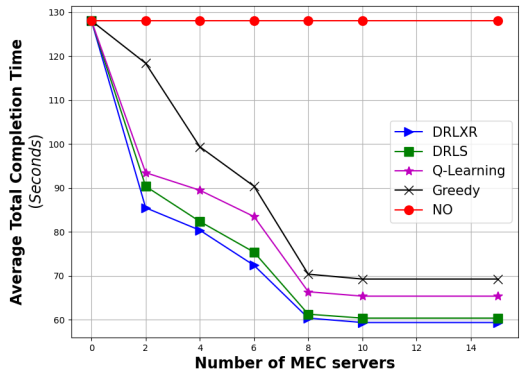

Figure 14: Average total completion time with various number of MEC servers

Finally, the total completion time with different numbers of XR devices is shown in Figure 15. The number of mobile devices at each MEC server is randomly generated by a uniform distribution, and the average total completion time is calculated as a performance indicator. *Greedy* and *Q-learning* methods' results are similar to those of the *NO* scheme for 50 XR devices, with a time completion of around 127 s. *DRLS* completion time increases at lower speed due to the probability of data exchange with other D2D peers. However, due to the limitation in computation, other XR devices that receive the offloaded computation from their peers cannot process the large amounts of data (due to the characteristics of XR applications) in

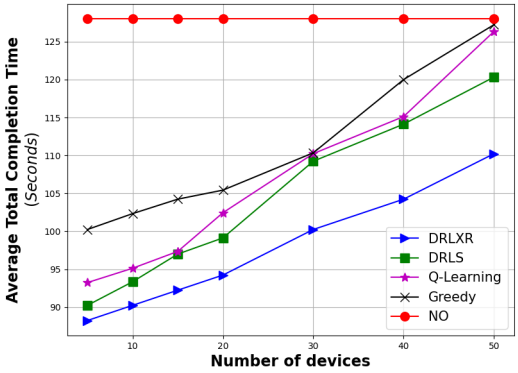

Figure 15: Average total completion time with various number of XR devices

a timely manner. On the contrary, XR devices in *DRLXR* make offloading decisions with higher accuracy than *Q-learning* and all high intensive computation tasks are guaranteed to be processed at MEC servers and the stringent latency requirements are met. As a consequence, *DRLXR* time completion increases at a lower pace and outperforms other counterparts.

VI. Conclusions and Future Works

This paper proposed and designed the Deep Reinforcement Learning-based Offloading scheme for XR devices (DRLXR) in the context of a MEC-enabled network environment. A hierarchical network architecture with three levels is considered. The task offloading problem at the XR device is formulated using DRL. Based on the data monitored at the XR devices, including radio signal quality, energy consumption and status of running application, the devices employ an Actor-Critic method for training and decision making on task offloading. The proposed DRLXR scheme is evaluated in a simulation environment and compared against other offloading methods. The simulation results show how DRLXR outperforms the other solutions in terms of average energy consumption and total completion time.

Future works will focus on a joint solution that combines the proposed offloading scheme and resource management at MEC server under heterogeneous QoS requirements.

Acknowledgement

This publication has received research funding support from the Science Foundation Ireland (SFI) under Grant Number 12/RC/2289_P2 for the Insight SFI Research Centre for Data Analytics, co-funded by the European Regional Development Fund.

References

[1] SangSu Choi, Kiwook Jung, and Sang Do Noh. Virtual reality applications in manufacturing industries: Past research, present findings, and future directions. *Concurrent Engineering*, 23(1):40–63, 2015.

[2] Ravi Pratap Singh, Mohd Javaid, Ravinder Kataria, Mohit Tyagi, Abid Haleem, and Rajiv Suman. Significant applications of virtual reality for covid-19 pandemic. *Diabetes & Metabolic Syndrome: Clinical Research & Reviews*, 14(4):661–664, 2020.

[3] Andrei OJ Kwok and Sharon GM Koh. Covid-19 and extended reality (xr). *Current Issues in Tourism*, pages 1–6, 2020.

[4] Dimitris Chatzopoulos, Carlos Bermejo, Zhanpeng Huang, and Pan Hui. Mobile augmented reality survey: From where we are to where we go. *Ieee Access*, 5:6917–6950, 2017.

[5] Abid Yaqoob, Ting Bi, and Gabriel-Miro Muntean. A survey on adaptive 360° video streaming: Solutions, challenges and opportunities. *IEEE Communications Surveys & Tutorials*, 22(4):2801–2838, 2020.

[6] Fabio Silva, Mohammed Amine Togou, and Gabriel-Miro Muntean. An innovative algorithm for improved quality multipath delivery of virtual reality content. In *2020 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, pages 1–6. IEEE, 2020.

[7] Daniel Wagner and Dieter Schmalstieg. Handheld augmented reality displays. In *IEEE Virtual Reality Conference (VR 2006)*, pages 321–321. IEEE, 2006.

[8] John Patrick Sexton, Anderson Augusto Simiscuka, Kevin Mcguinness, and Gabriel-Miro Muntean. Automatic cnn-based enhancement of 360° video experience with multisensorial effects. *IEEE Access*, 9:133156–133169, 2021.

[9] Armando Fox, Rean Griffith, Anthony Joseph, Randy Katz, Andrew Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. Above the clouds: A berkeley view of cloud computing. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, 28(13):2009, 2009.

[10] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18, 2010.

[11] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62, 2010.

[12] Sokol Kosta, Andrius Aucinas, Pan Hui, Richard Mortier, and Xinwen Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *2012 Proceedings IEEE Infocom*, pages 945–953. IEEE, 2012.

[13] Mung Chiang and Tao Zhang. Fog and iot: An overview of research opportunities. *IEEE Internet of Things Journal*, 3(6):854–864, 2016.

[14] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. Mobile edge computing—a key technology towards 5g. *ETSI white paper*, 11(11):1–16, 2015.

[15] Bao Trinh, Liam Murphy, and Gabriel-Miro Muntean. An energy-efficient congestion control scheme for mptcp in wireless multimedia sensor networks. In *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pages 1–7. IEEE, 2019.

[16] Pavel Mach and Zdenek Becvar. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys & Tutorials*, 19(3):1628–1656, 2017.

[17] Karthik Kumar, Jibang Liu, Yung-Hsiang Lu, and Bharat Bhargava. A survey of computation offloading for mobile systems. *Mobile networks and Applications*, 18(1):129–140, 2013.

[18] Weiwen Zhang, Yonggang Wen, Kyle Guan, Dan Kilper, Haiyun Luo, and Dapeng Oliver Wu. Energy-optimal mobile cloud computing under stochastic wireless channel. *IEEE Transactions on Wireless Communications*, 12(9):4569–4581, 2013.

[19] Suzhi Bi and Ying Jun Zhang. Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading. *IEEE Transactions on Wireless Communications*, 17(6):4177–4190, 2018.

[20] Yi-Hsuan Kao, Bhaskar Krishnamachari, Moo-Ryong Ra, and Fan Bai. Hermes: Latency optimal task assignment for resource-constrained mobile computing. *IEEE Transactions on Mobile Computing*, 16(11):3056–3069, 2017.

[21] Umber Saleem, Yu Liu, Sobia Jangsher, and Yong Li. Performance guaranteed partial offloading for mobile edge computing. In *IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2018.

[22] Umber Saleem, Yu Liu, Sobia Jangsher, Xiaoming Tao, and Yong Li. Latency minimization for d2d-enabled partial computation offloading in mobile edge computing. *IEEE Transactions on Vehicular Technology*, 69(4):4472–4486, 2020.

[23] Yuvraj Sahni, Jiannong Cao, Lei Yang, and Yusheng Ji. Multi-hop multi-task partial computation offloading in collaborative edge computing. *IEEE Transactions on Parallel and Distributed Systems*, 32(5):1133–1145, 2020.

[24] Sung-Tae Hong and Hyoil Kim. Qoe-aware computation offloading scheduling to capture energy-latency tradeoff in mobile clouds. In *2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pages 1–9. IEEE, 2016.

[25] Jiao Zhang, Li Zhou, Qi Tang, Edith C-H Ngai, Xiping Hu, Haitao Zhao, and Jibo Wei. Stochastic computation offloading and trajectory scheduling for uav-assisted mobile edge computing. *IEEE Internet of Things Journal*, 6(2):3688–3699, 2018.

[26] Xiao Zheng, Mingchu Li, Muhammad Tahir, Yuanfang Chen, and Muhammad Alam. Stochastic computation offloading and scheduling based on mobile edge computing. *IEEE Access*, 7:72247–72256, 2019.

[27] Ju Ren, Kadir Md Mahfujul, Feng Lyu, Sheng Yue, and Yaoxue Zhang. Joint channel allocation and resource management for stochastic computation offloading in mec. *IEEE Transactions on Vehicular Technology*, 69(8):8900–8913, 2020.

[28] Bomin Mao, Zubair Md Fadlullah, Fengxiao Tang, Nei Kato, Osamu Akashi, Takeru Inoue, and Kimihiro Mizutani. Routing or computing? the paradigm shift towards intelligent computer network packet transmission based on deep learning. *IEEE Transactions on Computers*, 66(11):1946–1960, 2017.

[29] Zhong Li, Cheng Wang, and Chang-Jun Jiang. User association for load balancing in vehicular networks: An online reinforcement learning approach. *IEEE Transactions on Intelligent Transportation Systems*, 18(8):2217–2228, 2017.

[30] Rose Qingyang Hu et al. Mobility-aware edge caching and computing in vehicle networks: A deep reinforcement learning. *IEEE Transactions on Vehicular Technology*, 67(11):10190–10203, 2018.

[31] Zhaolong Ning, Peiran Dong, Xiaojie Wang, Lei Guo, Joel JPC Rodrigues, Xiangjie Kong, Jun Huang, and Ricky YK Kwok. Deep reinforcement learning for intelligent internet of vehicles: An energy-efficient computational offloading scheme. *IEEE Transactions on Cognitive Communications and Networking*, 5(4):1060–1072, 2019.

[32] Minghui Min, Liang Xiao, Ye Chen, Peng Cheng, Di Wu, and Weihua Zhuang. Learning-based computation offloading for iot devices with energy harvesting. *IEEE Transactions on Vehicular Technology*, 68(2):1930–1941, 2019.

[33] Chenmeng Wang, Chengchao Liang, F Richard Yu, Qianbin Chen, and Lun Tang. Computation offloading and resource allocation in wireless cellular networks with mobile edge computing. *IEEE Transactions on Wireless Communications*, 16(8):4924–4938, 2017.

[34] Hao Hao, Changqiao Xu, Lujie Zhong, and Gabriel-Miro Muntean. A multi-update deep reinforcement learning algorithm for edge computing service offloading. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 3256–3264, 2020.

[35] Jin Wang, Jia Hu, Geyong Min, Albert Y Zomaya, and Nektarios Georgalas. Fast adaptive task offloading in edge computing based on meta reinforcement learning. *IEEE Transactions on Parallel and Distributed Systems*, 32(1):242–253, 2020.

[36] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and trends in Robotics*, 2(1-2):388–403, 2013.

[37] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[38] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[39] Songtao Guo, Bin Xiao, Yuanyuan Yang, and Yang Yang. Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9. IEEE, 2016.

[40] Houssemeddine Mazouzi, Nadjib Achir, and Khaled Boussetta. Dm2-ecop: An efficient computation offloading policy for multi-user multi-cloudlet mobile edge computing environment. *ACM Transactions on Internet Technology (TOIT)*, 19(2):1–24, 2019.

[41] Tuyen X Tran and Dario Pompili. Joint task offloading and resource allocation for multi-server mobile-edge computing networks. *IEEE Transactions on Vehicular Technology*, 68(1):856–868, 2018.

[42] Yuxiu Hua, Zhifeng Zhao, Rongpeng Li, Xianfu Chen, Zhiming Liu, and Honggang Zhang. Deep learning with long short-term memory for time series prediction. *IEEE Communications Magazine*, 57(6):114–119, 2019.

[43] Gustavo Carneiro. Ns-3: Network simulator 3. In *UTM Lab Meeting April*, volume 20, pages 4–5, 2010.

[44] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[45] Tim Verbelen, Pieter Simoens, Filip De Turck, and Bart Dhoedt. Leveraging cloudlets for immersive collaborative applications. *IEEE Pervasive Computing*, 12(4):30–38, 2013.

[46] Yejun He, Man Chen, Baohong Ge, and Mohsen Guizani. On wifi offloading in heterogeneous networks: Various incentives and trade-off strategies. *IEEE Communications Surveys & Tutorials*, 18(4):2345–2385, 2016.

[47] Zihan Gao, Wanming Hao, Zhuo Han, and Shouyi Yang. Q-learning-based task offloading and resources optimization for a collaborative computing system. *IEEE Access*, 8:149011–149024, 2020.

[48] Yixuan Wang, Kun Wang, Huawei Huang, Toshiaki Miyazaki, and Song Guo. Traffic and computation co-offloading with reinforcement learning in fog computing for industrial applications. *IEEE Transactions on Industrial Informatics*, 15(2):976–986, 2018.

Bao-Nguyen Trinh received the B.Eng. degree in telecommunications from Hanoi University of Science and Technology, Hanoi, Vietnam, in 2009 and the M.Eng. degree in Telecommunications from Dublin City University, Dublin, Ireland in 2014. He obtained the Ph.D. degree with the Performance Engineering Laboratory, School of Computer Science, University College Dublin, Ireland in 2020. He is currently working as a Postdoctoral Researcher at the Insight SFI Research Centre for Data Analytics, Dublin City University, Ireland. His research interests include Edge Computing, 5G, Deep Reinforcement Learning and eXtended Reality.

Gabriel-Miro Muntean (S'02–M'04-SM'17) received the B.Eng. and M.Sc. degrees in computer science engineering from the Politehnica University of Timisoara Romania, in 1996 and 1997, respectively, and the Ph.D. degree in electronic engineering from Dublin City University (DCU) Ireland, in 2003. He is currently a Professor with the DCU School of Electronic Engineering, co-Director of the DCU Performance Engineering Laboratory, and a Consultant Professor with the Beijing University of Posts and Telecommunications, China. He has authored or co-authored over 450 papers in prestigious international journals and conferences, has authored 4 books and 26 book chapters, and has edited 9 other books. His current research interests include quality-oriented and performance-related issues of adaptive multimedia delivery, performance of wired and wireless communications, energy-aware networking, and technology-enhanced learning. Prof. Muntean is a senior member of IEEE and IEEE Broadcast Technology Society. He is an Associate Editor of the IEEE TRANSACTIONS ON BROADCASTING, Multimedia Communications Area Editor of the IEEE COMMUNICATION SURVEYS AND TUTORIALS, and a reviewer for other international journals, conferences, and funding agencies. He coordinated the EU Horizon 2020 project NEWTON and leads the DCU team in the EU project TRACTION.