User Allocation in Mobile Edge Computing: A Deep Reinforcement Learning Approach

Subrat Prasad Panda   Ansuman Banerjee
Indian Statistical Institute
Email: {subratprasad.mail@gmail.com, ansuman@isical.ac.in}

Arani Bhattacharya
Indraprastha Institute of Information Technology Delhi
Email: arani@iiitd.ac.in

Abstract—**In recent times, the need for low latency has made it necessary to deploy application services physically and logically close to the users rather than using the cloud for hosting services. This paradigm of computing, known as edge or fog computing, is becoming increasingly popular. An edge user allocation policy determines how to allocate service requests from mobile users to MEC servers. Current state-of-the-art techniques assume that the total resource utilization on an edge server is equal to the sum of the individual resource utilizations of services provisioned from the edge server. However, the relationship between resources utilized on an edge server with the number of service requests served from there is usually highly non-linear, hence, mathematically modelling the resource utilization is challenging. This is especially true in case of an environment with CPU-GPU co-execution, as commonly observed in modern edge computing. In this work, we provide an on-device Deep Reinforcement Learning (DRL) framework to predict the resource utilization of incoming service requests from users, thereby estimating the number of users an edge server can accommodate for a given latency threshold. We further propose an algorithm to obtain the user allocation policy. We compare the performance of the proposed DRL framework with traditional allocation approaches and show that the DRL framework outperforms deterministic approaches by at least 10% in terms of the number of users allocated.**

Index Terms—**Mobile Edge Computing, Edge User Allocation, Service Latency, Deep Reinforcement Learning**

I. Introduction

Mobile Edge Computing (MEC) [1] is a promising new paradigm in which computing devices or edge servers provide compute-intensive low-latency services by being installed much closer to the user, in cellular towers, mini data centres or even in homes of mobile users. Service requests from mobile users are offloaded to nearby MEC servers, as an alternative to executing them on the resource-deprived mobile devices or sending them to a distant cloud server. The increasing demand for compute-intensive applications like real-time vehicle identification, object detection and route prediction is gradually leading to widespread adoption of MEC, and installation of MEC servers alongside mobile base stations [2].

A major challenge in MEC is to determine the user-server binding policy for routing of service requests. This problem, called Edge User Allocation (EUA), aims to ensure that users are allocated resources on edge servers while satisfying constraints on coverage, resource availability, latency requirement and so on. The EUA problem becomes challenging since it needs to satisfy such a diverse variety of constraints. A number of algorithms have been proposed to solve the EUA problem [3]–[5]. Allocation polices in recent literature often provide an optimization solution based on metrics like latency, count of allocated users, energy etc. However, all these works assume *linear dependence* of resource utilization of services on the number of services provisioned on an edge server. Works that consider the non-linear relationship, such as [6], do not consider CPU-GPU co-execution. In contrast, we show via experiments on both CPUs and GPUs that this relation is usually highly non-linear, i.e. the resource utilization by services does not scale linearly if the number of requests grows, as shown in [7]–[11] using the Google cluster trace dataset [12]. The non-linearity arises due to the diverse effects of various internal system attributes such as software / hardware architecture, operating system policies, number of cores, varying nature of service workloads in CPU / GPU, service invocation patterns etc.

In conventional allocation approaches, the total resource utilization at each server is assumed to be the cumulative sum of the resource utilization footprint of each service request [3], [4], [13]. In the real world, the amount of resources that may be utilized by any service during execution is highly dynamic, which is often difficult to model mathematically. Generally, the resource utilization values for MEC services in prior research articles are taken as the mean or maximum based on the records of service execution. However, these approaches lead to resource under-use or overflow on the edge servers due to sub-optimal allocation. In our work, we learn the resource utilization of services using machine learning methods instead of just assuming average or maximum values from past execution records. While utilizing supervised models is one possible way, they require an extensive amount of data for training, and cannot adapt if the recorded data at some instance of time changes in future. Therefore, we utilize a Deep Reinforcement Learning (DRL) [14] based approach that incrementally learns the appropriate resource allocation based on experience and interaction with the MEC system, instead of just the training dataset obtained a priori [15]. This makes the DRL based framework trainable on the device itself. This is quite advantageous since the agent can continually adapt to variations in the MEC environment and thereby create policies which best approximate the resource utilisation.

In our proposed work, the DRL agent learns the system environment of edge servers i.e. it learns the number of users

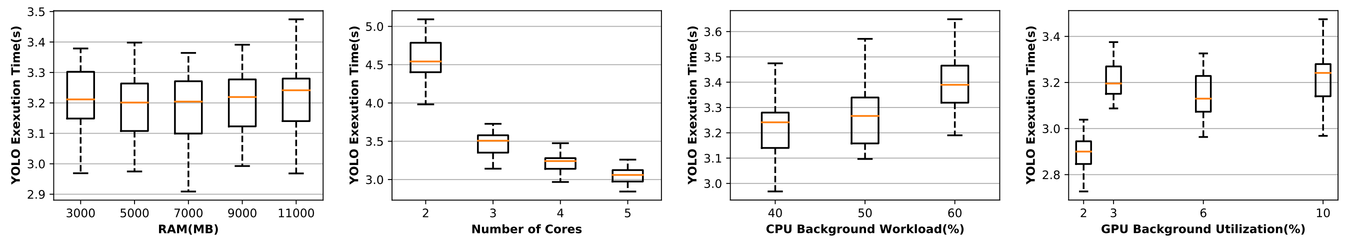


Fig. 1: Non-Linear relationship between different resource attributes and YOLO execution time (a) varying RAM (b) varying number of cores (c) varying CPU workload (d) varying GPU utilization in a **CPU-GPU co-execution** environment.

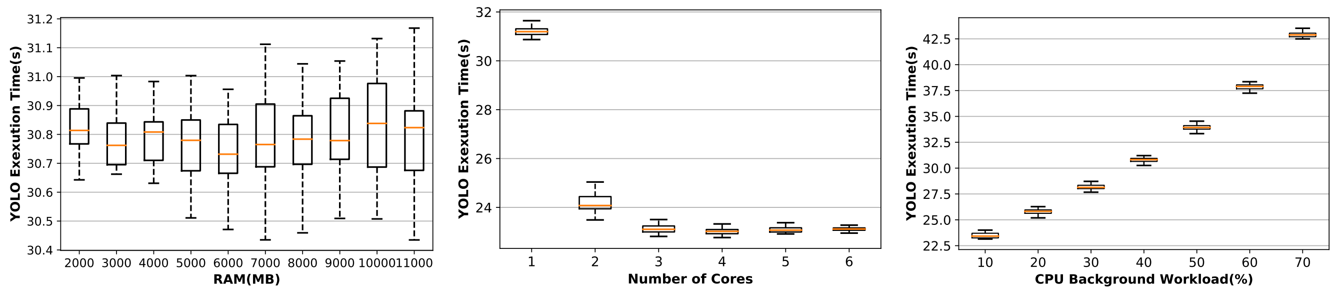


Fig. 2: Non-Linear relationship between different resource attributes and YOLO execution time using only CPU (a) varying RAM (b) varying number of cores (c) varying CPU workload in a **CPU-only** environment.

which can be served at a particular edge server under the constraint of a defined service latency threshold. Rather than formulating a complex mathematical model of the system, the DRL agents inherently learn non-linear dependencies directly from the edge server by observing system parameters over a while. The DRL agents do not need the dataset to perform the training task, hence can be trained online directly on the edge. We propose to use the DRL for on-device training to learn the edge system dynamics, thereby, reducing the overhead of the requirement of the training dataset. We propose an algorithm to obtain the user allocation policy from the trained DRL agent.

We have evaluated our approach using a real world dataset [4], which consists of locations of users and edge servers at the Central Business District (CBD), City of Melbourne, Australia. We have compared the number of users allocated with our proposed method with two deterministic baseline approaches: (a) the deterministic Integer Linear Programming (ILP) approach inspired from works in [3] [4] and (b) a Greedy solution to the ILP based on user allocation to nearest neighbourhood. Since the proposed method is based on DRL, unlike other Machine Learning (ML) approaches, the requirement of large-scale dataset is not necessary. Our experiments outperform the traditional linear approach by approximately 10% with 500 users and 50 edge servers in the MEC environment.

The rest of this paper is organized as follows. We illustrate a motivating scenario for this work in section II-B. In Section III, we provide a short description of DRL and formulate our DRL agent to solve the EUA problem, while in Section IV, we present a deterministic approach used as our baseline. Subsequently, in Section V we discuss experiments and results obtained using our proposed approach. We then discuss related work in Section VI. Finally, we discuss some limitations and open issues in Section VII and then conclude in Section VIII.

II. Motivation

In this section, we first show through experiments how the standard assumption of linear dependence of resource utilization breaks in practice. We then use a motivating example to demonstrate how these assumptions lead to inefficient allocations of users to edge devices.

A. *Observations to Verify Assumptions*

We first observe the service execution times of a widely used object detection application YOLO [16] in Figures 1 and 2 both using and without using a GPU respectively, to process an image. We run the experiments on a machine with Intel(R) Xeon(R) CPU E5-1650 v4 processor, 64GB RAM and Quadro P4000 8GB GPU (further details in Section V). By default, we retain the amount of RAM at 11000 MB, number of cores equal to 4, CPU background workload at 40% and GPU background workload at 10%. We then vary only a single parameter for each experiment, where the parameters are (a) amount of available RAM, (b) number of available cores, (c) CPU background workload, and (d) GPU background workload. We repeat the experiments a total of 20 times and show the execution times in box plots.

Figure 1 shows the execution time when we vary the amount of RAM available (Figure 1(a)), the number of available CPU cores (Figure 1(b)), the background workload on each CPU (Figure 1(c)) and the background workload on GPU (Figure 1(d)). Note that the standard YOLO implementation uses both CPU and GPU. We find in Figure 1(a) that changing the amount of available RAM had relatively minor effect on the execution time. Increasing the number of cores in Figure 1(b), however, had a major impact, with a reduction in the execution time. However, the reduction is highest when we increase the number of cores from 2 to 3 (equal to 22.22%), whereas it is relatively small when we increase it from 4 to 5 (equal to 8.5%). Thus, this relationship is non-linear. We again change the CPU utilization adding background jobs to the CPU, with the CPU utilization varying from 40% to 60%. As with the number of cores, we find that if the CPU background workload is low, increasing it increases the execution time of YOLO by a modest amount. Finally, we note that increasing the background workload on the GPU in Figure 1(d) increases the execution time initially, but beyond a point, there is no general trend. Figure 2 shows the execution time of YOLO while varying the same parameters but on a system where only CPU is present. We note that due to the absence of GPUs, the execution time is on the higher side. However, the trends in execution time visible in Figure 1 are all identical. Thus, Figures 1-2 show that the non-linear relationship between execution times and CPU/GPU parameters depends on a number of hidden parameters related to CPU/GPU availability and is relatively difficult to model.

Based on the findings above, we note that modeling service execution times is challenging due to the following factors:

- **Non-linear relationship between available processor resources and execution time:** The relationship between the execution time and the CPU/GPU parameter is non-linear. For example, in Figure 1(b), we find that reducing the number of available cores increases the execution time by a much higher factor when the number of available cores is small. Similarly, increasing the background workload of CPU and GPU slows down execution much more if the workload is already high. Moreover, there are significant variations in the execution time, making them difficult to model directly. The optimization models in prior research do not accommodate these factors.
- **Variation Across Time:** Figures 1 and 2 show that there is a substantial difference in execution times *even with identical configurations on the same machines.* For example, in Figure 1(a), the execution times vary from 2.82s to 3.47s. This occurs because execution of services depends on multiple hidden parameters, such as service invocation patterns, temperature, etc. Since most modeling techniques utilize deterministic values of execution times, it can lead to inefficient edge user allocations.
- **Variation Across Services:** Users are likely to invoke different services and each service invoked has its own pattern of execution time. This complicates the task of modeling the service execution times. For example, Table 1 shows that the execution time of Yolo on e_1 increases approximately linearly on increasing the number of users, but that of MobileNet on e_1 is non-linear.

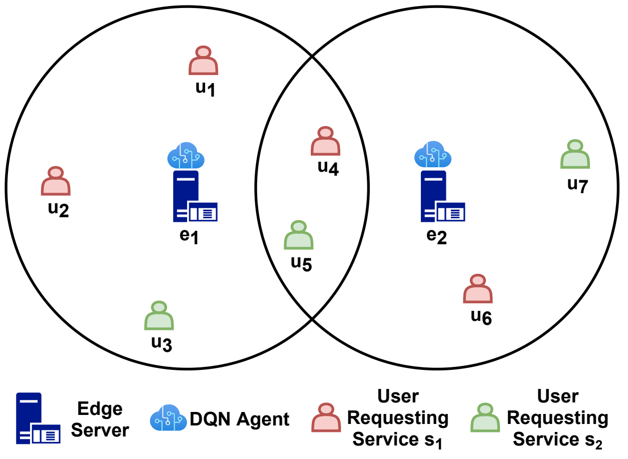


Fig. 3: Representative MEC Server Allocation Scenario

B. A Motivating Example

Consider a simple scenario with seven mobile users $u_1, u_2 \dots u_7$ and two edge servers e_1 and e_2 as illustrated in Figure 3. Each user is requesting for one of the two services s_1 and s_2 available on the edge servers. The users u_1, u_2, u_4 and u_6 are requesting for the service s_1, whereas, the remaining users are requesting for service s_2. For this example, the service s_1 corresponds to the YOLO [16] application and service s_2 corresponds to the MobileNetV2 [17] application. Each edge server is denoted by a resource vector which represents the resources available and the status of the edge server for service execution. We denote the resource vector on each edge server as a 4-tuple *(Available RAM, Number of Cores, CPU Background Workload%, GPU Utilized%)*. The GPU Utilized(%) denotes the current GPU memory utilization (in percentage) by the background processes. In this example, the resource vectors for edge servers e_1 and e_2 are taken as $(15000, 8, 60\%, 10\%)$ and $(6000, 4, 40\%, 6\%)$ respectively. We now explain how a conventional deterministic approach leads to inefficient user-to-server allocation.

Allocation with Deterministic Approaches: For allocation of users to the edge servers with constraints on latency threshold, a deterministic value for service execution needs to be determined from historical execution footprints using approaches like averaging, or computation of median, maximum or regression. If we use a simple linear approach to find the service execution time, for edge server e_1, we find that the execution time for a single user request for service s_1 is 3.12s. Linearly interpolating this value for 4 users gives us an execution time of $12.48s$. However, in real-world execution, the execution time for 4 users is $3.46s$ as shown in Table I. Let us assume that we are given a latency threshold of $6.5s$, i.e. users should be allocated in such a way that their execution finishes in $6.5s$. A deterministic approach considering only the execution time of a single request for services will produce an allocation of u_1, u_2 and u_3 to e_1. Note that only two users can be served for service s_1 as each will take execution time of $3.12s$ producing a total of $6.24s$. Similarly, only one user can

Users	YOLO on e1	YOLO on e2	MobileNet on e1	MobileNet on e2
1	3.12	3.26	6.32	6.03
2	3.23	3.29	6.40	6.12
3	3.35	3.37	6.42	6.18
4	3.46	3.50	6.54	6.26

TABLE I: Service execution times (in seconds) of YOLO and MobileNetV2 on edge servers e_1 and e_2.

be served for service s_2 as it takes $6.32s$ of execution time for a single service. Thus, the total number of users we are able to allocate using the deterministic approach is equal to 3.

Potential of Data-driven Allocation Approach: As illustrated in Table I, the execution time of four users running YOLO is below the latency threshold of $6.5s$. Thus, it was actually possible to allocate the users u_1, u_2, u_4 on e_1, as it only takes $3.35s$. Furthermore, it is also possible to accomodate u_5 and u_7 on e_2 as two users only take a total of $6.12s$. Thus, we are able to allocate a total of 5 users (i.e., 2 more users than the deterministic approach) using the data-driven approach with more accurate modeling of resource utilization.

III. Allocation with Reinforcement Learning

The MEC environment comprises of edge servers denoted as $E = \{e_1, e_2, \dots e_j\}$, where each edge server e_j has a coverage radius of r_j. The mobile users located within the coverage radius of an edge server can request for services hosted on that server. A set of users $U = \{u_1, u_2, \dots u_i\}$ may request for services from the set $S = \{s_1, s_2, \dots s_k\}$ hosted on an edge server. The resources available on each edge server is denoted by the resource vector *(RAM, Cores, CPU Background Workload%, GPU Utilized%)*. Since users are mobile and service requests are dynamic, the allocation algorithm discussed later in this section, is executed to obtain an allocation policy (which decides the user-server binding) whenever: (a) new users join the coverage area of an edge server, (b) users move away from the coverage area of an edge server, (c) user service requests change, or (d) edge servers or mobile users go offline.

The goal of the allocation policy in this work is to serve as many possible service requests as possible while strictly honouring a service execution latency threshold Γ. The knowledge of service execution time is needed to make such decisions of whether to assign a user's request to an edge server or not. For deterministic allocation approaches, the service execution time for services can be obtained from historical data by statistical methods, and then the determined value is used to obtain an allocation policy. However, due to the dynamic nature of execution time, the allocation policy can over or under allocate users to edge servers during real execution. In this work, we propose an RL based learning framework to obtain user-server binding decisions honouring the threshold Γ by learning the service execution patterns from experience directly on the edge server.

The agent in the RL framework learns the environment to choose better action choices by exploring the environment

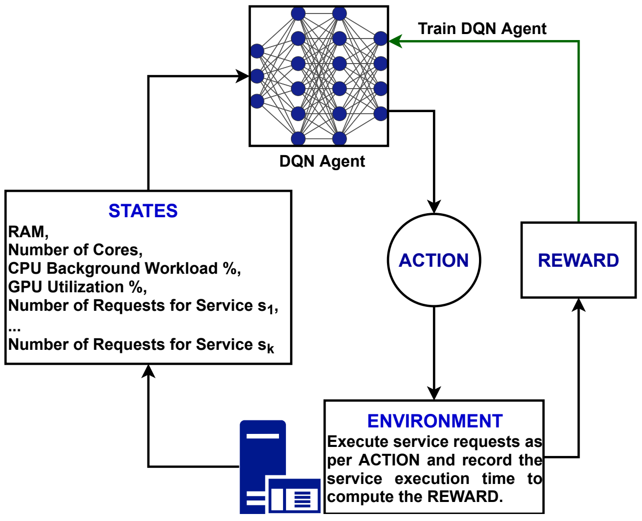

Train DQN Agent
DQN Agent
STATES
RAM,
Number of Cores,
CPU Background Workload %,
GPU Utilization %,
Number of Requests for Service s_1,
...
Number of Requests for Service s_k
ACTION
REWARD
ENVIRONMENT
Execute service requests as per ACTION and record the service execution time to compute the REWARD.

Fig. 4: Illustration of Reinforcement Learning Framework

and receiving feedback from the action. The primary advantage of RL is that it can learn the underlying environment without requiring massive amount of labelled data. In this RL framework, the agent continuously interacts with the edge servers to take actions i.e. executes several service requests and obtains the corresponding rewards according to the execution footprint. As shown in Figure 4, the state is denoted by the resource vector of the edge server along with the number of service requests. The action represents the set of service requests waiting to be executed on the edge server. For our RL problem, the agent is modeled as a Markov Decision Process (MDP) [18] represented as a tuple (Σ, A, R), as illustrated in Figure 5. The notations used throughout the paper are shown in Table II. The entries of the MDP are as follows:

- Σ is a finite set of states represented with six attributes as (R: RAM(MB), C:Number of Cores, CW: CPU Background Workload%, GU: GPU Utilization%, N_{s1}: Number of service requests for Service s_1, ..., N_{sk}: Number of service requests for Service s_k). The values for RAM, Number of Cores, CPU Background Workload percentage and GPU Utilization percentage are taken from the resource vector of the edge server. Moreover, additional attributes are added to include the number of service requests for each service $s \in S$. The number of service requests represents the number of users requesting to get served for a particular service hosted on the edge server. For example, the state (5000, 4, 40, 10, 100, 300) represents an edge server that hosts two services s_1 and s_2 with the currently available resource of $5000MB$, 4 CPU cores, CPU background workload at 40% and GPU utilization of 10%, and 100 users requesting for service s_1 and 300 users requesting for service s_2.
- A is the set of actions represented by the number of user service requests executed on an edge server. An action $a_n \in A$ is represented by a tuple $(p_{n1}, p_{n2} \dots p_{nk})$, where k is the total number of services hosted on the edge server

and p_{nk} represents the number of service requests for service s_k to be executed on an edge server. For example, for an edge server hosting two services s_1 and s_2, the action $(50, 100) \in A$ represents 50 service requests for service s_1 and 100 service requests for service s_2 to be executed. Since the action space has size $O(|U| \times |S|)$, we reduce the cardinality of the action space using quantization of size λ. For example, the quantization size $\lambda = 10$ produces a new action space where the new action tuple $(2, 2)$ represents all the actions in the range $(11 - 20, 11 - 20)$ in the old action space. We discuss the impact of λ on performance in the experiments.

- $R(\sigma, a_n)$ is the immediate reward received after the agent takes a particular action $a_n \in A$ at state $\sigma \in \Sigma$. The reward is computed from the total latency L_j^{tot} resulting due to an action a_n at an edge server $e_j \in E$. Given a state $\sigma = (R, C, CW, GU, N_{s1}, \ldots, N_{sk}) \in \Sigma$ and an action $a_n = (p_{n1}, \ldots, p_{nk}) \in A$ on the edge server e_j hosting k services with hard service latency threshold of Γ, the reward is the sum of the services accommodated, multiplied by a damping factor η. Note that if the services chosen cannot be accommodated, then we have a reward of zero. Formally,

$$R_{lin}(\sigma, a_n) = \begin{cases} \eta \sum\limits_{i=1}^{k} p_{nk} & \text{if latency } L < \Gamma \\ 0 & \text{Otherwise} \end{cases} \quad (1)$$

The agent in the MDP learns the optimal action i.e. the maximum number of service requests that can be deployed on the edge server so that the total latency L_j^{tot} does not exceed Γ where $L_j^{tot} = L_j^{ser} + L_j^{nwk}$, the sum of service execution time L_j^{ser} and network latency L_j^{nwk} for edge server e_j. The network latency L_j^{nwk} for an edge server e_j is the maximum possible latency incurred due to network communication delays. The reward returned is higher for actions with a higher number of service requests executed under the latency threshold of Γ, however, the reward is low whenever the service latency Γ is not honoured for certain actions. The agent learns the optimal action by exploring and exploiting the environment [18]. The environment for our problem is the real system which provides the real latencies (L_j^{tot}) and state of the system. The latency generated by a particular action $a_n \in A$ is derived directly from the system by executing a number of services on the edge server e_j due to the action a_n. The latency L_j^{tot} is used to return the reward $R(\sigma, a_n)$ for an action a_n at state σ on an edge server e_j.

The RL agent is trained using the Deep-Q learning [14] paradigm. In Deep-Q learning, the states of the RL agent are input to a neural network and Q-values of each action are the output of the neural network. The Q-values at time step t for state s_t and action a_t are calculated as per the equation given in Equation 2 [18], where α is the learning rate and β is the discount factor.

Notations	Descriptions
U	The set of users $\{u_1, u_2 \ldots u_i\}$
E	The set of edge servers $\{e_1, e_2 \ldots e_j\}$
S	The set of services $\{s_1, s_2 \ldots s_k\}$
Σ	State of the MDP (RAM, Cores, CPU Background Workload(%), GPU Utilization(%), Number of requests for service s_1, ... Number of requests for service s_k)
A	Action space of the MDP
$R(\sigma, a_n)$	Reward due the action $a_n \in A$ at state $\sigma \in \Sigma$
λ	Quantization size for action space reduction
L_j^{ser}	Service execution latency for the edge server e_j
L_j^{nwk}	Network latency for the edge server e_j
L_j^{tot}	$L_j^{ser} + L_j^{nwk}$ for an edge server e_j
Γ	The latency threshold for MEC environment
γ_{kj}	Latency of single request for service s_k at edge server e_j
η	Damping factor in reward function
α	Learning rate in Q-value update
β	Reward discount factor in Q-value update
U_j^{opt}	Predicted optimal number of user service request tuples on the edge server e_j
r_j	The coverage radius of edge server e_j
d_{ij}	The distance between user u_i and server e_j
$\Omega(u)$	Returns the index of service requested by user u

TABLE II: List of Notations

$$\begin{aligned} Q(s_t, a_t) &\leftarrow Q(s_t, a_t) + \\ &\alpha\{R(s_t, a_t) + \beta \max_{a_{t+1} \in A}[Q(s_{t+1}, a_{t+1})] - Q(s_t, a_t)\} \end{aligned} \quad (2)$$

The optimal policy is to select the action with the maximum Q-Value. The agent can be trained on an edge server during service installations to predict the number of users that could get deployed there. This reduces the effort for offline training in contrast to simple supervised learning approaches.

Algorithm 1 presents our heuristic to obtain the user-server allocation policy. The proposed algorithm performs a load balancing of service requests while computing the allocation policy. The optimal number of user service requests U_j^{opt} that can be allocated to a particular edge server e_j predicted from the trained Deep Q agent is a tuple of size equal to the number of services hosted on the edge server i.e. $U_j^{opt} = (u_1^{opt} \ldots u_k^{opt})$. For example, the tuple of predicted users $U_1^{opt} = (100, 200)$ with an edge server e_j hosting two services s_1 and s_2 denotes that the optimal number of service requests that can be accommodated on e_j honouring the given latency threshold is 100 for service s_1 and 200 for service s_2.

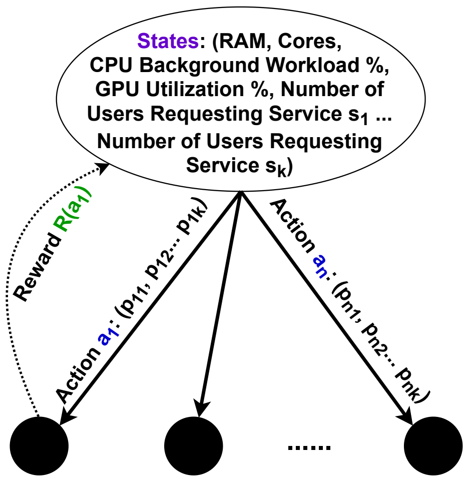

States: (RAM, Cores, CPU Background Workload %, GPU Utilization %, Number of Users Requesting Service s1 ... Number of Users Requesting Service sk)
Reward R(a1)
Action a1: (p11, p12... p1k)
Action an: (pn1, pn2... pnk)
......

Fig. 5: Our RL model for a single episode.

Algorithm 1: Algorithm for User Allocation with RL

```
Input  : $U \leftarrow$ Users, $E \leftarrow$ Servers, DQN Agent
Output: Returns User-Server Allocation List $Alloc[]$
1  foreach $e \in E$ do
2      $S \leftarrow$ State of the edge server $e$
3      $U_e^{opt}[k] \leftarrow$ Tuple of predicted number of user
         service requests given the state $\sigma \in \Sigma$ for an edge
         server $e$ using the trained DQN RL Agent
4  end
5  foreach $u \in U$ do
6      $e_{list} \leftarrow$ List of servers covering user $u$
7      $e_{selected} \leftarrow$ The server with maximum $U^{opt}[\Omega(u)]$
         in the list $e_{list}$
8      $Alloc[] \leftarrow$ Append $(u, e_{selected})$ which assigns user
         $u$ to the server $e_{selected}$
9      Decrement $U^{opt}_{e_{selected}}[\Omega(u)]$ by 1
10 end
```

IV. Deterministic Approach Used as Baseline

In this section, we present a baseline ILP model that is based on a conventional allocation policy that works with deterministic service execution times. Given the initial resource state vector on an edge server e_j, the execution time for a single request for service s_k (denoted by γ_{kj}) on that particular edge server is typically determined by averaging over historical service execution data. This value of γ_{kj} is used in the Integer Linear Programming (ILP) model below to determine the number of users that can be allocated on an edge server. The ILP formulation generates the user-server binding policy with an objective of maximizing the number of users allocated to the edge servers. The allocation of user $u_i \in U$ to the edge server $e_j \in E$ is denoted by the binary variable x_{ij} and the distance between the corresponding user and edge server is denoted as d_{ij}. The function $\Omega(u)$ returns the index of the service from S requested by the user $u \in U$. For allocation, the distance between the user and edge server d_{ij} should not exceed the coverage radius of the edge server r_j which is represented as a constraint in Equation 4. The total latency caused due to users assigned to a particular edge server e_j should not exceed the latency threshold Γ, as in constraint Equation 5 for our ILP formulation. The constraint in Equation 6 ensures the allocation of a particular user to a maximum of only one edge server. The deterministic ILP allocation policy is developed along the approach followed in prior research [3] [4] etc. The solution returned by an ILP solver is used to compare against our proposed approach with the RL agent. It may be noted that in this ILP formulation, we consider only the latency constraint, while neglecting additional resource constraints of the edge devices. Inclusion of additional resource constraints would make the ILP model conservative to allocation. Furthermore, multiple works ([4], [5]) utilize ILP as a standard technique to further build their own approaches based on it. The latency threshold constraint used in modelling of ILP indirectly involves soft constraints on resources. Hence, the threshold on execution time keeps a check on resource overflow.

Objective:

$$Maximize : \sum_{i=1}^{|U|} \sum_{j=1}^{|E|} x_{ij} \tag{3}$$

where,

$$x_{ij} = \begin{cases} 1, & \text{If user } u_i \text{ is allocated to server } e_j \\ 0, & \text{Otherwise} \end{cases}$$

Subject To:

1) Coverage Constraint:

$$d_{ij} \leq r_j \tag{4}$$

2) Latency Threshold Constraint:

$$L_j^{nwk} + \sum_{i=1}^{|U|} (x_{ij} \times \gamma_{\Omega(i)j}) \leq \Gamma : \forall j \in \{1, \ldots |E|\} \tag{5}$$

3) User-Server Mapping:

$$\sum_{j=1}^{|E|} x_{ij} \leq 1 : \forall i \in \{1, \ldots |U|\} \tag{6}$$

4) Integer Constraint:

$$x_{ij} \in \{0, 1\} : \forall i \in \{1, ..|U|\}, \forall j \in \{1, ..|E|\} \tag{7}$$

Considering the hardness of the above, solving for the optimal allocation policy on a real system makes it harder for the baseline allocation scheme to be implemented in the real world for large workloads. A greedy approximation strategy is therefore proposed for solving the allocation problem. We use a simple greedy heuristic based on the nearest neighbourhood allocation principle for comparison with our RL based approach, i.e., allocate an user to the nearest edge server with available resources to accommodate the user's request under the given latency threshold. We present experimental results to compare the relative performances in the following section.

V. Experiments and Analysis of Results

All experiments are conducted on a machine with Intel(R) Xeon(R) CPU E5-1650 v4 processor, 64GB RAM and Quadro P4000 8GB GPU. The services YOLO [16] and MobileNetV2 [17] are used widely for object detection in images and videos. Both of these use both CPU and GPU for computation. Also, the applications are computationally heavy. So, the aforementioned services are used as representative services in our MEC environment to show the effectiveness of the learning approach to embracing more hidden parameters as GPU is also involved. All programs are written in Python, the software library Stable-Baseline3 [19] is used for training of RL agents and the Python Mixed-Integer-Programming library is used as the ILP solver. The results from the RL-based approach discussed in Section III are compared with the deterministic formulation shown in Section IV which is similar to the modelling approach of [3] [4]. The number of users allocated and execution time for running the algorithms are demonstrated for the following: (a) ILP in Section IV **[ILP]**, (b) Greedy algorithm in Section IV **[Greedy]** and (c) Proposed RL approach with reward in Equation 1 **[RL]**. Experimental results clearly show the effectiveness of our work.

A. Experiment Setup

We use the data-set for edge server locations as in [3], which includes data of base stations and users within the Melbourne Central Business District area. The coverage area of edge servers is set to 150 meters radius. The edge servers are assigned with the initial resource vector randomly as shown in Table III. The RL agent proposed in this work is trained using total latency which is the sum of network latency and service execution latency. We use network latencies from the real world PlanetLab and Seattle latency data-set [20]. Since the PlanetLab and Seattle latency data-set comprises latencies from across the world, which is not fully representative of latencies in an MEC environment, we cluster the data-set into 150 clusters considering devices which are in proximity of each other. A cluster is picked according to the distance of an user from an edge server and a representative network latency is assigned. The service execution latency is obtained by executing YOLO and MobileNetV2 varying RAM, Number of Cores, Workload(%), GPU Utilization (%) and Number of Service Requests. The trained model is then used for predicting the number of users that can be accommodated on the server given the state of the edge server. The quantization size of the action space is set to $\lambda = 100$. The damping factor in the reward function is set to $\eta = 0.01$ to avoid gradient overflow while training the DQN agent.

The DQN agent in the RL framework uses a neural network of 2 layers having 64 nodes at each layer with layer normalisation for our proposed model. The learning rate α is set to 0.0001 and discount factor β in Q-value calculation is set to 1. The exploration fraction for the agent is set to 0.4 for our experiment. The DQN agent is trained for $5,00,000$ time steps with each step corresponding to one episode during exploration, which takes around 0.6 milliseconds for each time step during training of the agent. Overall it takes around 50 minutes to train for $5,00,000$ training steps. The γ_j for each server in the deterministic ILP is obtained from the server by first executing YOLO and MobileNetV2, then computing the average time taken for execution given the initial resource state vector of the server. Our experiment has by default a total of 500 users, the number of edge servers as 20-80, and latency threshold Γ of 50ms. We also have a set of experiments to study the influence of varying each of these individual parameters. We repeat each experiment 50 times and then record the average allocation results for the sake of comparison. For each experiment, we show the (i) average reward using our technique, (ii) training loss using our technique, (iii) number of users allocated using our technique and the baselines, and (iv) the execution time (in log scale) of running our technique and the baselines.

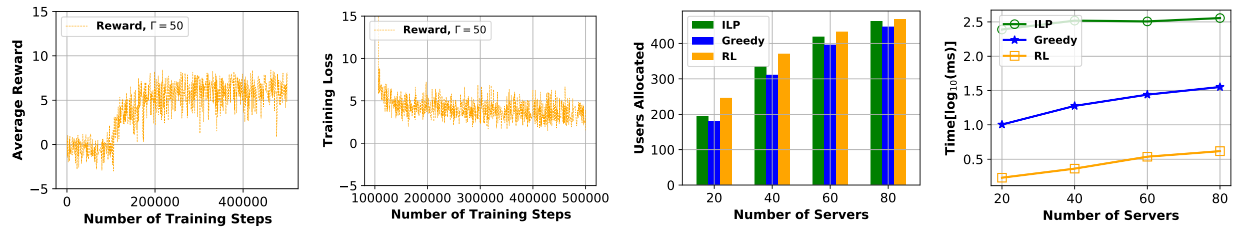
Parameter	Assumed Value
Initial RAM	3000 - 11000MB
Number of CPU cores	2 - 5
Initial CPU Workload	40 - 60%
Initial GPU Utilization	1 - 10%
Quantization Size λ	100
Damping Factor in reward η	0.01
Learning rate in neural network α	0.0001
Discount factor β	1
Number of time steps (episode)	5,00,000
Number of Users	100-500
Number of Servers	20-80
Latency threshold	20-50s

TABLE III: Values of different parameters used for evaluation.

B. Experimental Results

Default Configuration: Figure 6 shows the performance for the default configurations. We first note that the average reward converges after 200,000 rounds. This corresponds to around $1200s$ of training time. After this, further training does not increase the reward significantly. Thus, the training loss also does not reduce from this point (Figure 6(b)).

We then record the number of users that can be allocated by varying the number of servers between 20 to 80. We find that the number of users allocated is higher using RL as compared to ILP and the Greedy technique by up to 16% and 18% respectively. This improvement is highest when the number of edge servers is equal to 40. On increasing the number of edge servers further, RL still performs better than ILP, but by a more modest amount. This is because when the number of servers is sufficiently high, even a simple allocation algorithm leads to allocation of most users. Thus, RL is most effective when the number of servers present is limited.



(a) Average Reward (b) Training Loss (c) Number of Allocated Users (d) Execution Time

Fig. 6: Comparison of different performance parameters under the default configurations: RL versus baseline

We also compare the performance in terms of execution time of each of these techniques. Once again, we find that RL has the least execution time, with allocation for even 80 servers taking only around $0.1s$, in contrast to the Greedy heuristic and ILP which take $0.32s$ and $1.2s$ respectively. Thus, RL performs better in terms of both number of users allocated as well as execution time under these configurations.

Performance under Varying Threshold Latency and Varying Number of Users: We now consider the performance of RL and baseline techniques when we vary the threshold latency with number of users varying between 100-500 and the number of servers fixed at 50. Figure 7 shows the performance in terms of the number of allocated users for each of the techniques. We find that the number of users allocated falls the least with a decrease in the threshold latency. This is different from the ILP and greedy techniques, where we find that there is a more significant decrease in performance with a reduction in the threshold latency. For example, when the number of users present is equal to 400, the ILP can allocate 280 and 310 for latency thresholds of 30s and 50s respectively. On the other hand, RL allocates around 320 users in both the cases. This shows the importance of RL especially in cases where the latency constraint is tighter. The execution time is lowest for RL, thereby outperforming the ILP and greedy techniques.

Performance under Varying Threshold Latency and Varying Number of Servers: We now consider the performance of RL and baseline techniques when we vary the threshold latency with number of servers varying between 20-80 and the number of users fixed at 500. Figure 8 shows the performance in terms of number of allocated users for each of the techniques. We find that the difference in number of users allocated by RL compared to the other approaches to be significant when the latency threshold is lower. The deterministic approaches struggle to allocated users with strict latency constraints. The number of servers in the MEC environment also affects the performance of the algorithms since more servers in the MEC environment makes the situation easier for the deterministic algorithms to allocate users. In such cases, the ILP and the greedy techniques attain results comparable to our proposed RL agent. As expected, the execution time is less for RL than those approaches.

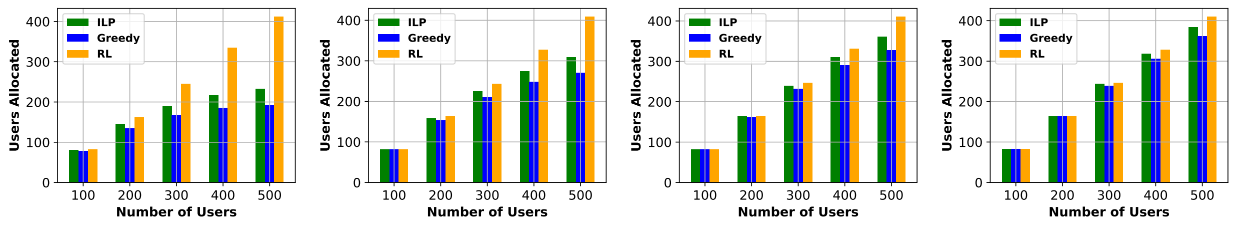
Impact of Training Time: We consider the number of allocations generated by an under-trained RL agent with $30,000$ training steps and compare it against a properly trained RL agent with $1,50,000$ training steps. The quantization size of the action space is kept at $\lambda = 2$ for both the cases. The results are illustrated in Figure 9. Figure 9a shows the allocation with varying number of users with the number of servers fixed at 30, while, Figure 9b shows the result for varying number of servers with number of users fixed at 500 in the MEC environment with a latency threshold of $\Gamma = 10s$. The properly trained RL agent produces better results in comparison to the under-trained agent which is seen as inadequate to capture the non-linearity in execution time.

Impact of Quantization Parameter: We also study the impact of quantization size λ on the RL agent for generating allocation policies. Figure 10 shows the effect of quantization size of $\lambda = 5$ and $\lambda = 100$ with the quantization size of $\lambda = 2$. The model with $\lambda = 2$ produces better allocation results compared to a higher quantization size of $\lambda = 5$ and $\lambda = 100$. The higher value of quantization size reduces the action space significantly while sacrificing the accuracy for allocation, hence the allocation results with the RL agents of quantization size $\lambda = 5$ and $\lambda = 100$ do not perform as expected due the reduced accuracy of prediction. The better allocation result produced by $\lambda = 100$ as compared to $\lambda = 5$ does not always mean $\lambda = 100$ is better than $\lambda = 5$. Changing the quantization value affects the accuracy of allocation, so, the allocation result may over-shoot or get damped according to different MEC scenarios. The change in allocation result due to variation in λ is insignificant especially considering the significant reduction in action space due to use of quantization.

VI. Related Work

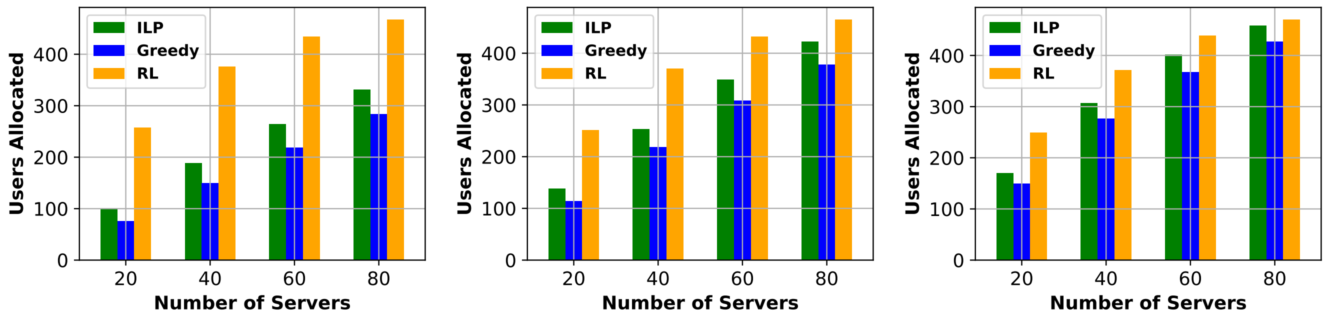
Prior research falls into three major categories – articles that have studied the EUA problem, studies that have modeled the performance of cloud / edge using ML techniques, and additional system prototypes that have used Deep Reinforcement Learning (DRL) to solve resource allocation problems.

EUA Problem: A number of works formulate EUA as an optimization problem, and use a variety of techniques such



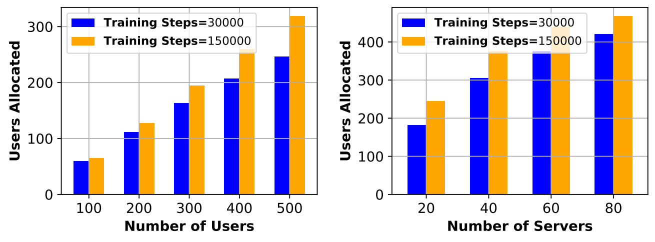
(a) Latency Threshold=20s (b) Latency Threshold=30s (c) Latency Threshold=40s (d) Latency Threshold=50s

Fig. 7: Comparison of the number of allocated users when the latency threshold is varied with varying number of users.



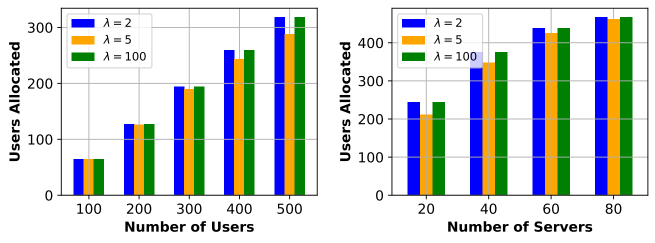
(a) Latency Threshold=20s (b) Latency Threshold=30s (c) Latency Threshold=40s

Fig. 8: Comparison of the number of allocated users when the latency threshold is varied with varying number of servers.



(a) Varying Users (b) Varying Servers

Fig. 9: Impact of under training on allocation



(a) Varying Users (b) Varying Servers

Fig. 10: Impact of Quantization parameter on allocation

as ILP, approximation algorithms or heuristics to solve them [3], [4], [21]–[27]. For example, [3] and [4] formulate the allocation problem as a version of the bin-packing problem, with the objective being to maximize the number of users allocated to the cloud and the QoE of users. Authors in [21] propose optimal and approximate mechanisms for allocating network resources in MEC. The work [22] formulates joint allocation and service placement as that of minimizing the number of users allocated to each cloud server. In [28] the authors consider minimizing each MEC server's energy consumption. In [23], the authors derive an approximation algorithm by incorporating rewards that are awarded when user requirements are honoured. In [24], the authors formulate a time-slotted model and develop a polynomial-time approximation by jointly optimizing service placement and request scheduling, i.e., which user requests are to be routed to deployed services. In [25], [26] and [27] the authors develop a mathematical model of an edge system and solve the optimization problem using reinforcement learning. In all of these works, the MEC environment is modelled by innately assuming that the resource utilization of a service on an edge server scales linearly with the number of services deployed. Although DRL is used for the EUA problem in [29], it still assumes a linear relationship between resource utilization and execution time. As shown in Section II, this is not true in practice. Our work, therefore, uses a more general technique that does not depend on the linear assumption. The work in [6] uses a game-theoretic approach to solve the user allocation problem from an application provider perspective. Though it introduces non-linearity in the mathematical formulation by considering weighted cumulative sum of resource utilization of the services on a given edge server, the resource utilization of a service on an edge server is still determined from past resource utilization footprints. This approach therefore, partially misses to model the dynamic setting. The work [30] investigates the EUA problem in a multi-cell multi-channel downlink power-domain Non-orthogonal Multiple Access (NOMA)-based MEC system with an objective of maximizing the benefit to mobile application vendors. The work [31] handles communication interference caused due to user allocation and proposes an interference aware allocation policy using a game-theoretic approach. The work [32] models user allocation considering user migration from the perspective of service providers as a stochastic optimization problem. This proposal puts forward an online Lyapunov optimization-based algorithm, and proves its performance bounds. The work [33] handles the trade-off between multi-tenancy and interference in the pursuit of a cost-effective service user allocation by proposing a game-theoretic approach, namely MI-SUAGame. Further, [30], [31], [32] and [33] propose weighted non-linear mathematical formulations for computing the total resource utilization and use optimization approaches to obtain an allocation scheme. The formulation explicitly models the MEC environment using a predefined mathematical formulation which ignores system attributes like service execution at GPUs on an edge server or dynamics in service resource utilization.

ML-based Performance Models: Some works utilize machine learning based performance models to predict the service parameters for different cloud architectures. For example, PARIS [34] and CherryPick [35] use random forests and Bayesian optimization respectively to identify the best VM for different workloads. SLAOrchestrator [36] uses a linear regression model to learn the performance of workloads. AutoPilot [37] uses a multi-armed bandit technique to identify an action to scale up or down execution on cloud systems. The work in [38] uses a deep neural network to learn the system dynamics of LTE Network devices to allocate users to different base stations. Our work utilizes DRL to allocate users, as we find that the linear models do not work well in practice, while also observing that deep neural networks cannot be run on the edge due to performance reasons.

Deep Reinforcement Learning Based Prototypes: A number of systems utilize DRL to optimize their performance, albeit not necessarily for allocation of users to cloud or edge [39]. For example, Pensieve [40] uses DRL to allocate bitrates to video streaming clients. DRL is used by [41] to allocate channel bands for transmission to IoT devices using DRL and [42] to allocate power to different antennas. The work [43] uses DRL to perform accurate indoor localization of users using Bluetooth Low Energy (BLE) signals. Finally, in the context of MEC, DRL has been used for caching data close to the users [44] and even computation offloading [29], [45]. In particular, [29] uses DRL to solve the optimization formulation instead of conventional optimization methods, while assuming a linear relationship in mathematically modelling the MEC system. Also, the work in [46] uses Sequence-to-Sequence (S2S) neural network models with DRL training to solve the problem of task offloading in MEC. These works are based on the observation that simplistic models often fail to accurately take into account the relationship between resource availability and performance in actual systems. We leverage the same observation in the context of the EUA problem. To the best of our knowledge, this is the first work that learns the relationship between resource utilization and edge server system performance using DRL to predict the number of users that can be allocated to a particular edge server.

VII. Limitations and Open Issues

While our RL approach promises to better allocate users to edge servers, there are a number of open issues and threats to validity. We now discuss some of them.

- **Presence of Heterogeneous Edge:** Our trace-driven simulation is based on the assumption that the edge servers are homogeneous in nature, i.e. the hardware configuration of each edge server is assumed to be identical. The recent rise of heterogenous edge [47] can create situations where the edge servers can have different configurations. While we believe that our approach should work well under such settings, we have not considered them in our evaluation.
- **Unpredictable Traffic Patterns:** The rise of the Internet of Things (IoT) is gradually making traffic patterns unpredictable due to the increase in the number of control signals [48]. While we have shown that our model works well under current traffic patterns, how our technique would react to a case with such unpredictable traffic patterns is currently left for future work.
- **Training on User Device:** One drawback of our approach is that the RL is trained on the user device. While we show that the training on user device is lightweight enough to be feasible, there might be concerns raised by users about their battery consumption.

VIII. Conclusion and Future Work

In this paper, we propose the RL approach for online training and edge user allocation in the context of mobile edge computing. Our approach eliminates the need for modeling exact execution times a priori, whereby we show through experiments that many of the standard assumptions related to resource utilization do not hold in practice. The proposed RL framework automatically infers the resource utilization relation by executing services on the edge server and allocates users to edge servers while honoring the defined latency threshold. We carry out our experiments using a real-world dataset and service execution data. Our experiments show that the RL based approach outperforms deterministic approaches that carry out resource allocation based on historical execution footprints. As future work, we plan to extend our work to implement this framework on an actual edge testbed.

References

[1] F. Bonomi, R. A. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing, MCC@SIGCOMM 2012, Helsinki, Finland, August 17, 2012* (M. Gerla and D. Huang, eds.), pp. 13–16, ACM, 2012.

[2] "IoT Edge | Microsoft Azure."

[3] P. Lai, Q. He, M. Abdelrazek, F. Chen, J. Hosking, J. Grundy, and Y. Yang, "Optimal edge user allocation in edge computing with variable sized vector bin packing," in *ICSOC*, pp. 230–245, Springer, 2018.

[4] P. Lai, Q. He, G. Cui, X. Xia, M. Abdelrazek, F. Chen, J. G. Hosking, J. C. Grundy, and Y. Yang, "Edge user allocation with dynamic quality of service," in *Service-Oriented Computing - 17th International Conference, ICSOC 2019, Toulouse, France, October 28-31, 2019, Proceedings* (S. Yangui, I. B. Rodriguez, K. Drira, and Z. Tari, eds.), vol. 11895 of *Lecture Notes in Computer Science*, pp. 86–101, Springer, 2019.

[5] K. Poularakis, J. Llorca, A. M. Tulino, I. J. Taylor, and L. Tassiulas, "Joint service placement and request routing in multi-cell mobile edge computing networks," in *2019 IEEE Conference on Computer Communications, INFOCOM 2019, Paris, France, April 29 - May 2, 2019*, pp. 10–18, IEEE, 2019.

[6] Q. He, G. Cui, X. Zhang, F. Chen, S. Deng, H. Jin, Y. Li, and Y. Yang, "A game-theoretical approach for user allocation in edge computing environment," *IEEE Trans. Parallel Distributed Syst.*, vol. 31, no. 3, pp. 515–529, 2020.

[7] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, (New York, NY, USA), Association for Computing Machinery, 2012.

[8] S. Gupta and D. A. Dinesh, "Online adaptation models for resource usage prediction in cloud network," in *Twenty-third National Conference on Communications, NCC 2017, Chennai, India, March 2-4, 2017*, pp. 1–6, IEEE, 2017.

[9] B. R. Ray, M. U. Chowdhury, and U. Atif, "Is high performance computing (HPC) ready to handle big data?," in *Future Network Systems and Security - Third International Conference, FNSS 2017, Gainesville, FL, USA, August 31 - September 2, 2017, Proceedings* (R. Doss, S. Piramuthu, and W. Zhou, eds.), vol. 759 of *Communications in Computer and Information Science*, pp. 97–112, Springer, 2017.

[10] P. Minet, E. Renault, I. Khoufi, and S. Boumerdassi, "Analyzing traces from a google data center," in *14th International Wireless Communications & Mobile Computing Conference, IWCMC 2018, Limassol, Cyprus, June 25-29, 2018*, pp. 1167–1172, IEEE, 2018.

[11] M. Tirmazi, A. Barker, N. Deng, M. E. Haque, Z. G. Qin, S. Hand, M. Harchol-Balter, and J. Wilkes, "Borg: the Next Generation," in *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys'20)*, (Heraklion, Greece), ACM, 2020.

[12] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: format + schema," technical report, Google Inc., Mountain View, CA, USA, Nov. 2011. Revised 2014-11-17 for version 2.1. Posted at https://github.com/google/cluster-data.

[13] Q. Peng, Y. Xia, F. Zeng, J. Lee, C. Wu, X. Luo, W. Zheng, H. Liu, Y. Qin, and P. Chen, "Mobility-aware and migration-enabled online edge user allocation in mobile edge computing," in *2019 IEEE International Conference on Web Services, ICWS 2019, Milan, Italy, July 8-13, 2019* (E. Bertino, C. K. Chang, P. Chen, E. Damiani, M. Goul, and K. Oyama, eds.), pp. 91–98, IEEE, 2019.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.

[16] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 779–788, 2016.

[17] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pp. 4510–4520, IEEE Computer Society, 2018.

[18] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[19] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann, "Stable baselines3." https://github.com/DLR-RM/stable-baselines3, 2019.

[20] R. Zhu, B. Liu, D. Niu, Z. Li, and H. V. Zhao, "Network latency estimation for personal devices: A matrix completion approach," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 724–737, 2017.

[21] C. You, K. Huang, H. Chae, and B. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Trans. Wirel. Commun.*, vol. 16, no. 3, pp. 1397–1411, 2017.

[22] K. Poularakis, J. Llorca, A. M. Tulino, I. J. Taylor, and L. Tassiulas, "Service placement and request routing in MEC networks with storage, computation, and communication constraints," *IEEE/ACM Trans. Netw.*, vol. 28, no. 3, pp. 1047–1060, 2020.

[23] S. Pasteris, S. Wang, M. Herbster, and T. He, "Service placement with provable guarantees in heterogeneous edge computing systems," in *2019 IEEE Conference on Computer Communications, INFOCOM 2019, Paris, France, April 29 - May 2, 2019*, pp. 514–522, IEEE, 2019.

[24] V. Farhadi, F. Mehmeti, T. He, T. L. Porta, H. Khamfroush, S. Wang, and K. S. Chan, "Service placement and request scheduling for data-intensive applications in edge clouds," in *2019 IEEE Conference on Computer Communications, INFOCOM 2019, Paris, France, April 29 - May 2, 2019*, pp. 1279–1287, IEEE, 2019.

[25] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. Shen, "Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach," *IEEE Trans. Mob. Comput.*, vol. 20, no. 3, pp. 939–951, 2021.

[26] T. Alfakih, M. M. Hassan, A. Gumaei, C. Savaglio, and G. Fortino, "Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on SARSA," *IEEE Access*, vol. 8, pp. 54074–54084, 2020.

[27] J. Wang, L. Zhao, J. Liu, and N. Kato, "Smart resource allocation for mobile edge computing: A deep reinforcement learning approach," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2019.

[28] F. Wang, J. Xu, X. Wang, and S. Cui, "Joint offloading and computing optimization in wireless powered mobile-edge computing systems," *IEEE Trans. Wirel. Commun.*, vol. 17, no. 3, pp. 1784–1797, 2018.

[29] J. Li, H. Gao, T. Lv, and Y. Lu, "Deep reinforcement learning based computation offloading and resource allocation for MEC," in *2018 IEEE Wireless Communications and Networking Conference, WCNC 2018, Barcelona, Spain, April 15-18, 2018*, pp. 1–6, IEEE, 2018.

[30] P. Lai, Q. He, G. Cui, F. Chen, J. Grundy, M. Abdelrazek, J. G. Hosking, and Y. Yang, "Cost-effective user allocation in 5g noma-based mobile edge computing systems," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2021.

[31] G. Cui, Q. He, F. Chen, Y. Zhang, H. Jin, and Y. Yang, "Interference-aware game-theoretic device allocation for mobile edge computing," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2021.

[32] P. Lai, Q. He, X. Xia, F. Chen, M. Abdelrazek, J. Grundy, J. G. Hosking, and Y. Yang, "Dynamic user allocation in stochastic mobile edge computing systems," *IEEE Transactions on Services Computing*, pp. 1–1, 2021.

[33] G. Cui, Q. He, F. Chen, H. Jin, and Y. Yang, "Trading off between multi-tenancy and interference: A service user allocation game," *IEEE Transactions on Services Computing*, pp. 1–1, 2020.

[34] N. J. Yadwadkar, B. Hariharan, J. E. Gonzalez, B. Smith, and R. H. Katz, "Selecting the best vm across multiple public clouds: A data-driven performance modeling approach," in *Proceedings of the 2017 Symposium on Cloud Computing*, SoCC '17, (New York, NY, USA), p. 452–465, Association for Computing Machinery, 2017.

[35] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang, "Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics," in *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017* (A. Akella and J. Howell, eds.), pp. 469–482, USENIX Association, 2017.

[36] J. Ortiz, B. Lee, M. Balazinska, J. Gehrke, and J. L. Hellerstein, "Slaorchestrator: Reducing the cost of performance slas for cloud data analytics," in *2018 USENIX Annual Technical Conference, USENIX ATC 2018, Boston, MA, USA, July 11-13, 2018* (H. S. Gunawi and B. Reed, eds.), pp. 547–560, USENIX Association, 2018.

[37] K. Rzadca, P. Findeisen, J. Swiderski, P. Zych, P. Broniek, J. Kusmierek, P. Nowak, B. Strack, P. Witusowski, S. Hand, and J. Wilkes, "Autopilot: Workload autoscaling at google," in *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, (New York, NY, USA), Association for Computing Machinery, 2020.

[38] A. K. Albanna and H. Yousefi'zadeh, "Congestion minimization of LTE networks: A deep learning approach," *IEEE/ACM Trans. Netw.*, vol. 28, no. 1, pp. 347–359, 2020.

[39] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Commun. Surv. Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

[40] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2017, Los Angeles, CA, USA, August 21-25, 2017*, pp. 197–210, ACM, 2017.

[41] S. Wang, H. Liu, P. H. Gomes, and B. Krishnamachari, "Deep reinforcement learning for dynamic multichannel access in wireless networks," *IEEE Transactions on Cognitive Communications and Networking*, vol. 4, no. 2, pp. 257–265, 2018.

[42] A. Zappone, M. Debbah, and Z. Altman, "Online energy-efficient power control in wireless networks by deep neural networks," in *19th IEEE International Workshop on Signal Processing Advances in Wireless Communications, SPAWC 2018, Kalamata, Greece, June 25-28, 2018*, pp. 1–5, IEEE, 2018.

[43] M. Mohammadi, A. I. Al-Fuqaha, M. Guizani, and J. Oh, "Semisupervised deep reinforcement learning in support of iot and smart city services," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 624–635, 2018.

[44] H. Zhu, Y. Cao, X. Wei, W. Wang, T. Jiang, and S. Jin, "Caching transient data for internet of things: A deep reinforcement learning approach," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2074–2083, 2019.

[45] J. Chen, S. Chen, Q. Wang, B. Cao, G. Feng, and J. Hu, "iraf: A deep reinforcement learning approach for collaborative mobile edge computing iot networks," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 7011–7024, 2019.

[46] J. Wang, J. Hu, G. Min, W. Zhan, Q. Ni, and N. Georgalas, "Computation offloading in multi-access edge computing using a deep sequential model based on reinforcement learning," *IEEE Communications Magazine*, vol. 57, no. 5, pp. 64–69, 2019.

[47] W. Zhang, S. Li, L. Liu, Z. Jia, Y. Zhang, and D. Raychaudhuri, "Heteroedge: Orchestration of real-time vision applications on heterogeneous edge clouds," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 1270–1278, IEEE, 2019.

[48] V. Nagendra, A. Bhattacharya, A. Gandhi, and S. R. Das, "Mmlite: A scalable and resource efficient control plane for next generation cellular packet core," in *Proceedings of the 2019 ACM Symposium on SDN Research*, pp. 69–83, 2019.