National College of Ireland

Hybrid Reinforcement Learning based code offloading in MEC

MSc Research Project
Cloud Computing

Abhinash Pati
Student ID: 18189440

School of Computing
National College of Ireland

Supervisor: Manuel Tova-Izquierdo

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Abhinash Pati
Student ID:	18189440
Programme:	Cloud Computing
Year:	2020
Module:	MSc Research Project
Supervisor:	Manuel Tova-Izquierdo
Submission Due Date:	17/08/2020
Project Title:	Hybrid Reinforcement Learning based code offloading in MEC
Word Count:	7115
Page Count:	20

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	16th August 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Hybrid Reinforcement Learning based code offloading in MEC

Abhinash Pati
18189440

Abstract

In recent times, the Mobile Edge Computing paradigm has become a popular paradigm in latency reduction for resource-constrained devices by utilizing the concept of code offloading to another execution platform. With this, the IoT devices have got a boot in data processing capabilities. This field is being widely researched; however, many issues persist. Although many method have been developed to get real-time results from such paradigms, it still not perfect. A variety of unpredictable constraints such as device roaming and unreliable network conditions hinders the optimal operation of the code offloading, resulting in delays.
This paper proposes an online algorithm using deep reinforcement learning and sampling and classification approach to find the most suitable execution node in an execution platform. The primary purpose of the algorithm is to obtain the best fit node where if the task is offloaded, the total cost of execution (TCE) will be minimum. The algorithm considers the Femto cloud-based mobile cluster as a potential platform for task offloading. It implements a Deep Q-Network which learns from the error generated during the decision-making process.
The results, which are based on publicly available datasets used to simulate the proposed system, show that the proposed algorithm performs better than the baseline algorithms without any learning component.

1 Introduction

Today's mobile devices are more powerful and sophisticated than the computers that NASA used back in 1969 Kent and Williams (2018). The latest generation of mobile devices has gone beyond being just a communication device. In this era of digitization, mobile devices and IoT devices have matured a lot and have reached a new height in integrating sophisticated technologies Torres Neto et al. (2019).
However, these devices still can be compared with traditional data centers that can process billions of terabytes of data in seconds. By shifting complex and computationally intensive tasks to these data centers, both IoT devices and mobile devices have solved many problems requiring high computation capabilities. However, for real-time based application, these solutions have not helped much because of latency issues.
Latency is the delay occurred due to the transmission of the data and code to another platform for execution. Real-time applications such as Vehicular Adhoc Network (VANETs) require the results to be acquired as quickly as possible and have a relatively low threshold for delay Vaquero et al. (2019). Mobile-Edge-Cloud paradigms have recently emerged as

a potential candidate platform for latency intolerant applications. They have significantly reduced these devices' energy utilization but still have setbacks in specific areas where the parameters responsible for reducing the latency are inconsistent. For example, device mobility is a hot topic among the researchers where they are trying to find the best way to reduce computation latency due to unpredictable mobility patterns of the users Cicconetti et al. (2019).

When a mobile device moves from the access range of one edge device to another edge device, latency can occur because of task migration. The latency that occurred due to this transmission depends on the transmission policy introduced in the application. There are always some risks involved when migrating an active task from one node to another. One can also program the application to forward the execution request to the original node via the new node, but the end SLO needs to be fulfilled Meurisch et al. (2017).

Many researchers have actively proposed many solutions to curb the latency, causing factors to make the execution of real-time application smooth. This paper proposes a hybrid algorithm composed of deep Q-Network and sample and classification approach for selecting the best node out of a cluster of nodes to schedule the code offloading to achieve a minimum cost of execution.

Code-offloading involves two major steps

1. **Partition the code into order to offload**": The code can be a class, a method, or merely a few lines within a method. This can be achieved by running the application through a code profiler. The more loose couple the code is, the more it can be sliced for code offloading and vice verse.

2. **Scheduling the task execution on a particular node**: This step involves selecting, scheduling, and queuing of the tasks in the node. The system has to identify a node in which the execution can occur most efficiently. Various factors that lead to latency are weighted in this step to analyze and select the best fit node that will generate a reward to the system of having the minimum cost of execution.

1.1 Research Question

Can a hybrid algorithm composed of deep Q-Network with sample and classification achieve a low latency cost while code-offloading in the MEC paradigm?

1.2 Justification

The proposed solution tries to solve 2 using a hybrid AI-based algorithm. The algorithm uses deep Q-Network based reinforcement learning to gradually determine the best node for scheduling the task that would have a minimum cost of execution.

A real-life use case scenario mentioned in Habak et al. (2015) as an example is taken up, and the entire solution is based on it.

Today, everyone has atleast one mobile device with them. Although every user has a different usage pattern, many user's devices are under-utilized most of the time. The utilization rate also differs at different times of the day. For example, during the night

when the user is sleeping, the device may be ideal for a vast amount of time, making these mobile devices a potential platform for code-offloading. The work proposed is limited to selecting the best node where the execution (latency) will be minimum. I am assuming that the code that needs to be offloaded, has already been identified from the application as mentioned in 1

This paper is further divided into five sections. Section 2 shows the background work done so far. Section 3 gives the general idea about the system and its component. The system architecture and system flow are described in section 4. Section 5 explains the proposed algorithm, along with its internal working. The proposed system is evaluated, and the corresponding results are compared in section 6, and finally, the paper is concluded in section 7.

2 Related Work

Latency during code-offloading has always been a widely researched area for empowering resource-constrained devices. In the paper Aazam et al. (2018), the author has examined numerous recommended mechanisms for Code-offloading. The author starts by illustrating different criteria used to decide whether to offload to a particular node or process it locally. Following this, the paper presents various platforms and models used with its advantages and disadvantages, showcasing its core purpose and flexibility. It also analyses various techniques, algorithms, and AI-based approaches proposed by various authors to solve specific issues during code offloading. The proposed paper has considered a few of these criteria and models for evaluating the decision for offloading while taking the MEC model and an ad-hoc cluster of mobile devices known as FemtoCloud. The different approaches analyzed in Aazam et al. (2018) have been taken as a baseline for creating the foundation of the problem discussed in the proposal which has been further discussed in section 4.

2.1 Code Scheduling

As discussed in section 1, once the parts of the application that demand execution elsewhere are identified, a scheduling policy needs to be placed that will meet the defined SLO. The task to be executed can be interpreted as a workflow or direct acyclic graph (DAG), consisting of interlinked or individual tasks. The workflow's computational requirement is based on the task size and the deadline detailed in the SLA and may vary as each task may not demand higher computation power. If such tasks are assigned to any random node, it may create the dilemma of resource over-utilization or resource under-utilization. The primary purpose of defining a scheduling policy is to make sure the resource consumption are optimal, and the latency cost is kept to a minimum ADHIKARI et al. (2019).
The authors in ADHIKARI et al. (2019) have studied numerous work related to scheduling policies for workload in multiple platforms, including MEC. The concept of workflow, along with its types, has been demonstrated and compared by the author. The strengths and weaknesses of popular algorithms associated with task scheduling have been illustrated along with various criteria for defining the scheduling policy. The detailed evaluation presented as well as the insights stated in the paper has contributed to

the development of the proposed online algorithm.
Cicconetti et al. (2019) has offered a serverless architecture skeleton to reduce the latency during offloading in an edge-cloud paradigm by choosing a node with the least executing time for the task by calculating the processing time using SRPT. The calculation of the overall processing time involves parameters such as the network delays and computational delay while determining the best node.
While determining the network delay is as simple as subtracting the total time taken with the computational time, determining the computation delay is not so simple, especially in a cluster consisting of heterogeneous nodes. The author proposes a mechanism to determine the computational delays from the history consisting of metadata about the tasks such as task size, system load during the execution. The equation stated in section 4 are derived using the equation proposed in this paper. The author has also proposed a reset mechanism for resetting the stored meta-data for resolving scenarios in which a new node while being evaluated, was estimated to be overloaded or down because of a temporary outrage and in such case, the system may never consider this node as reliable. The paper further presents an innovative skeleton to evaluate algorithms and protocols in MEC paradigms, and the results show that it works better than the other mentioned frameworks in the paper.
The framework uses a mix of mathematical, cloud, and testbed models to measure the latency produced by the supplied algorithm and compares its performance with two other baseline algorithms. Their evaluation's outcomes confirmed an improvement of satisfaction of users by one fifth when compared to the static allocation of the node and achieving similar performance to other approaches mentioned in their research.
However, the paper ignores the cold start cost of the serverless function. Also, the assumption taken that the nodes fail rarely is not ideal, especially in case mobility is of the devices is involved. While leaving one node's range, the device may enter a zone where no node is present and local processing is the only option in such scenarios.

2.2 Code Queuing

Code-offloading generally happens in an ever-changing setting with parameters such as network connection, the utilization rate of nodes, changing constantly, and therefore needs a practical policy to do it efficiently. Developing an AI-based approach can help accomplish an effective decision-building policy that favors the most reliable and cost-effective node for offloading by examining dynamic parameters' patterns. A scheduling policy's main objective is to meet the stated SLO, while resource utilization is kept optimal. If this can be converted into a series of automated steps, it would significantly raise its reliability and usability. Applying an AI-based approach can help gradually improve the applied policy and finally reach a stage where it is near perfect. When scheduled for offloading, any task can be analyzed beforehand based on its parameters, and a policy can be applied to determine the most suitable node for execution. A model can then be gradually trained for finding the best node when a task with similar parameters are scheduled in the future. The proposed work is based on such an approach where DQN based reinforcement learning is applied for determining and training the policy to decide the best node for offloading, further discussed in section 4.
In Xu, Li, Huang, Xue, Peng, Qi and Dou (2019), the author explains EACO, which is an Energy-aware computational offloading method. The recommended work attempts to reduce energy consumption and latency by practicing a non-dominated sorting genetic

algorithm II (NSGA-II) to optimize the offloading sequence. The author illustrates an application use-case of offloading a mobile device workflow via an Access Point (APs). When the system receives an offloading request, it tries to find the nearest AP connected to a data center to deliver the execution request. Once this is done, the problem is reduced to finding the shortest route between the connecting APs. NSGA-II helps in determining the shortest path and hence reduces the offloading time. However, this approach may encounter some hardship when multiple routes or more than one AP connected to a data center. To resolve this, simple additive weighting and multiple criteria decision making is applied. This approach is quite suitable for the proposed work as the FemoCloud based mobile cluster may have more than one suitable node with near similar parameters. Applying an MCDM or SAW approach to various weight parameters will help in determining the most appropriate node.

Applying MCDM can help train the Q-Network for making a quick decision on selecting the node in case ambiguous nodes are present in the cluster. However, only relying on MCDM or SAW may not necessarily produce the most dependable results and can even lead to exploitation vs. exploration problems.

When a roaming mobile device moves from the range of one node to another, it may end up connecting to a node that has never been evaluated, and the system is not confident about the reliability of the node. The authors in Meurisch et al. (2017) have taken up this issue. A state of exploitation is seen when a system over chooses a known node for all offloading tasks rather than exploring other nodes in fear of not finding the best node and hitting a penalty. Offloading any critical computation to these nodes is very risky as the node is not known in advance and may lead to system failure, especially in an application that demands a low latency offloading environment. To solve such exceptional cases, the author proposes splitting the task into numerous micro-tasks and executing them on the unknown nodes to test its reliability and computational capabilities and gradually build trust. The results show that by following such a strategy, the system achieved 85 % correct predictions about an unknown node's reliability and computational capability.

The exploration vs. exploitation problem has also been taken up in Pinto et al. (2018). In this paper, the author has taken this problem to develop a decision-making policy with a minimum cost of execution. The paper mentions a situation in which when a system encounters an unknown node, its decision making efficiency is affected as it falls into a dilemma of whether to explore the new node or exploit the known nodes only. A serverless solution is presented by the author to strengthen the decision making policy. The system follows a sequence of steps to decide the offloading node. The tasks are split into light, heavy, super heavy, and obese categories and different parameters of the nodes are weight to generate a rank to decide the best suitable node for execution. In case the system does not have any information about any node in the cluster, it selects a node based on a greedy approach, primarily focusing on the average time taken for task completion. An Upper Confidence Bounds (UCB) based approach with Hoeffding's Inequality is presented in the paper, to generate the selection policy to balance between exploration and exploitation problem. For generating selection policy for known devices, a Bayesian UCB based approach is used.

Confirming the reliability of unknown nodes is quite relevant to the proposed solution as it involves FemtoCloud based mobile clusters, which can have a large number of heterogeneous devices that can join and leave at any time. The solution proposed by Meurisch et al. (2017) and Pinto et al. (2018) are well thought out and solve the problem; however, they still might not be able to solve all aspects of the issue. In the case of Meurisch et al.

(2017), splitting of tasks into numerous micro-tasks can be quite tedious and depend on the level of cohesion of the code. In the case of Pinto et al. (2018), the node history is not maintained, and if a node is marked as unreliable, the system may never explore it again.
The authors of Cicconetti et al. (2019) have proposed a reset mechanism to resolve such scenarios. The mechanism resets the meta-data stored after a certain pre-defined threshold of time. The same mechanism can also be applied while probing an unknown node. While probing, store the data about the node and periodically delete it and again probe it to understand the node's current reliability level and change its ranking accordingly.
Consider a real-life example of hiring an employee. The interview process can be termed as the probing process to understand the capabilities of the candidate. After getting a rough idea about their capabilities, few non-critical tasks can be assigned to them for trust-building and, therefore, gradually increase the candidate's ranking. For tasks that do not demand more computation power or are not urgent, they can be scheduled on unknown nodes to understand the node's capabilities.
In Shekhar et al. (2020), a middleware for selecting a node of offloading is proposed. The decision is made based on factors that direly impact latency, such as network reliability. The selection problem is divided into two sub-problems: to offload or process locally and where to offload? It selects the node for offloading by calculating the total cost of execution, including pre-processing cost, local execution cost, offload cost, and network cost. The decision to offload or process locally is made via a greedy heuristic-based algorithm. The decision process focuses mainly on the execution cost as well as the energy consumption locally. Executing a task in the device may result in a battery drain, but if the total cost of offload exceeds the defined deadline, the offload's purpose is defeated, and therefore the system makes a trade-off to meet the deadline. With each successful iteration, the middleware refines its prediction model to predict the variance in parameters and its corresponding latencies, making the decision process nearly perfect.
Lyapunov drift-plus-penalty optimization, Markov decision process (MDP), SAW (simple additive weighting) and MCDM (multiple criteria decision making) are widely used for creating an AI-based selection solution Li et al. (2019), Alam et al. (2019) ,Xu, Liu, Luo, Peng, Zhang, Meng and Qi (2019). However, these are not perfect and have some drawbacks. For example, if we use the Markov Decision Process-based algorithm to find the optimal node, we need to first discrete the system, but that may not be feasible if the solution space becomes too large.
Liu et al. proposal in Liu et al. (2019) can be asserted as the most similar to what I have proposed. The author proposes a deep Neural Network (DNN) based optimization solution that achieves minimum latency and low battery consumption levels. Code offloading is described as a discrete-time Markov decision process(discrete-time MDP) problem, and implementing an AI-based approach can help find the best solution. For this, the author has applied an actor-critic reinforcement model that uses policies to advance its actor to get the maximum reward. The model consists of actors: action to perform, a states: offloading or not, and a reward: minimum cost of execution. The benchmarking is done in two scenarios: offloading to the cloud and offloading to the edge node. The results show that power consumption is inversely proportional to task size decrease, but the overall performance is better than DQN.
Nevertheless, the mobility factor of a mobile device and its computational capabilities is ignored. In my proposal, I tried to integrate the mobile's computation environment

as a potential platform for the task while taking a scenario that involves the mobility of multiple mobile devices that can arrive and leave the range of the edge devices at any point in time.

3 Methodology

This section presents the associated hypotheses 3.2 related to the proposed solution along with an application use case 3.1

3.1 Application use-case

As mentioned in section 1, the classroom use-case discussed in Habak et al. (2015) is used for explaining the validity of the proposed solution. Many mobile devices are available from different users in public places such as train stains or airports or a cafe shop. These devices do not necessarily use 100 % of their computation power and can be considered a potential execution platform. At NCI, we use RFID readers to mark attendance. Consider a classroom with a camera-based IoT device responsible for taking attendance from the video feed. The camera module records the video of the class periodically, meaning at any time it may or may not be recording for attendance. The system is responsible for marking the students' attendance with a deadline of fifteen minutes before the end of the class.
Applying a face-matching algorithm to several video-frames may require a higher computation power. Moreover, face matching should be done without breaching the deadline; otherwise, the attendance marked may not be valid. The system has to decide whether to process the data locally or offload it to some execution platform. The system needs to choose the execution platform, which can provide the minimum latency results without breaching the deadline. The execution platform consists of FemtoCloud based mobile device cluster, edge node, where the system is deployed, and a cloud data-center.

3.2 Assumptions

The following section contains pre-requisites and assumptions made for the proposed system.

3.2.1 Pre-requisites

- An independent attendance management system.
- FemtoCloud based mobile cluster, as explained in Habak et al. (2015).
- A local network for connecting the cluster of devices with the system, preferably WiFi, similar to NCI's "Eduroam".

3.2.2 Assumptions

- The client-side is an IoT device with a camera having little computational capabilities such as in a Raspberry pie or Pandaboard connected with a 1080p camera.

- The video frame quality is clear and high.
- For each period, an offloading task can be submitted with a fixed deadline.
- The input data is assumed to be more extensive compared to the output result, and therefore, the download latency is taken as 0.
- Each student within the campus have atleast one active mobile device, connected and registered with the system via the client-side application.
- Any device can join or leave the cluster anytime.
- The network quality of the cluster devices is not constant and varies with the network signal strength.
- Every connected node has some load at any point in time.
- Latency delay for cloud is far worse than the edge, while the latency delay for FemtoCloud is acceptable.
- Device mobility is within the same complex and, therefore, in a small geographical area, which is negligible. For simplicity, I am ignoring any altitude-related attribute when calculating the distance between a mobile device and the edge node.

4 Design Specification

4.1 Architecture

The system architecture is shown in Fig. 1. The system comprises three components: camera-based IoT device for recording and sending video frames, control Manager (CRM) for generating a node selection policy, and node cluster for offloading tasks.
Consider a set of edge nodes represented by $E = e_1, e_2, e_3...$ and $M = m_1, m_2, m_3...$ number of mobile devices present in the FemtoCloud. Consider a time period containing $T = t_1, t_2, t_3...$ time slots. As stated in 3.2, the mobile devices are not stationary and move from the range of one edge device to another at different time slots. The client can generate an offload request at any period, which will have a deadline before which it needs to be completed. If this is not meet, the task is a timeout and the request is dropped from the current execution platform and directly sent to the cloud for the completion.
The main focus of the proposal is the smart selection of the execution platform and therefore for simplicity, I am assuming that all the execution platform can communicate and collaborate fluently and if an application workflow is migrated from one execution platform to another (such as from FemtoCloud to Cloud Data Center), the chances of failure due to migration is ignored.
Fig 2 represents the application scenario taken up in this proposal. The scenario is composed of several mobile devices that are either stationary at a location or either moving along a path. The scenario also consists of various edge nodes interconnected via LAN for data transmission and communication. The scenario is divided into few time slots, and each time slot has a different number of available mobile devices in the FemtoCloud, and they may not be connected with the edge node in their vicinity. The edge nodes are stationary, but the mobile devices can be both stationary and moving in

the vicinity of the edge nodes.
The Total cost of latency in our scenario is composed of six components: pre-processing time including the video frame upload to the CRM, time is taken by the CRM to select the best execution platform, the actual execution time of the task and the communication cost between the CRM-Platform-Client. The total cost of execution TCE is the summation of the time taken from the task initiation to completion.

$$TCE = L_{pp} + L_{ns} + L_{te} + L_n \tag{1}$$

L_{pp} is the local processing latency of the client device, including the upload of the video frame to the CRM. L_{ns} is the latency due to the analysis and calculation of the workflow for determining the best fit node. L_{te} is the time taken for execution of the workflow in the selected node, and L_n is the network delay.
Each task needs to be completed within a deadline. therefore,

$$T_{deadline} \geq \max[L_{pp} + L_{ns} + L_{te} + L_n] \tag{2}$$

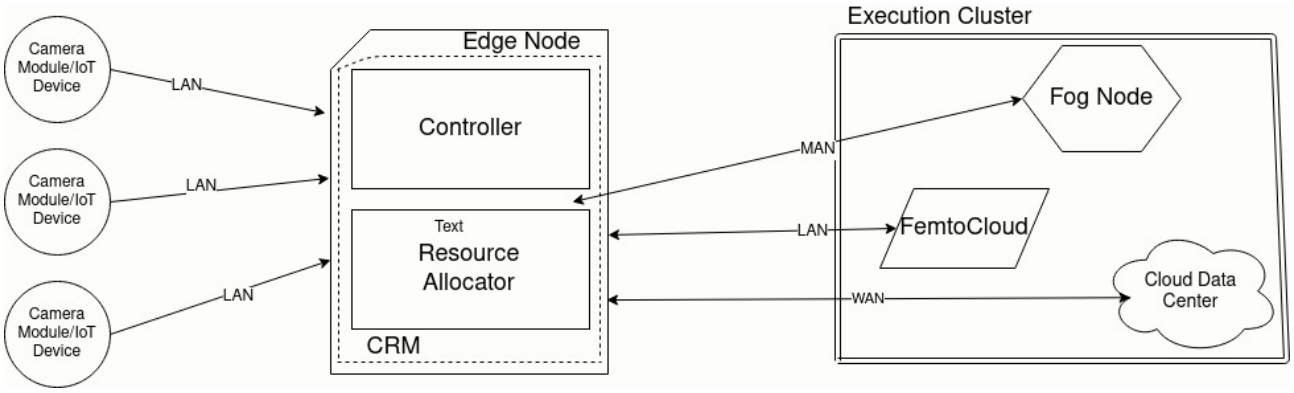

Figure 1: Proposed System Architecture

If we assume that the client IoT device and the node in which CRM resides in the same complex, the variance of L_{pp} will depend on the distance and the uplink network speed between the device and the node. Liu et al. (2019) is using signal-to-noise ratio and bandwidth to calculate the uplink speed. If we ignore the client and the node's altitude distance, we can fairly say that $L_{pp} \approxeq 0$ or $L_{pp} \approxeq \infty$ if the node or the network is down. Similarly, L_{ns} depends on the distance between CRM and the selected execution node. Whereas, L_{te} depends on the utilization rate of the node and the size of the task.

4.2 System Flow

The client device extracts the required data from the video recorded and uploads it to CRM for processing. The CRM receives the request along with the data and the deadline. It then calculates the total execution time based on different parameters to generate a selection policy to select the best fit node with minimum latency. The CRM also trains its Q-Network to predict the latency changes due to the variations in the parameters and gradually polishes the selection policy, further explained in section 4.2.
Since any mobile device can join and leave the FemtoCloud cluster at any time without a need of any administrator, the node availability is highly unpredictable. In case a node

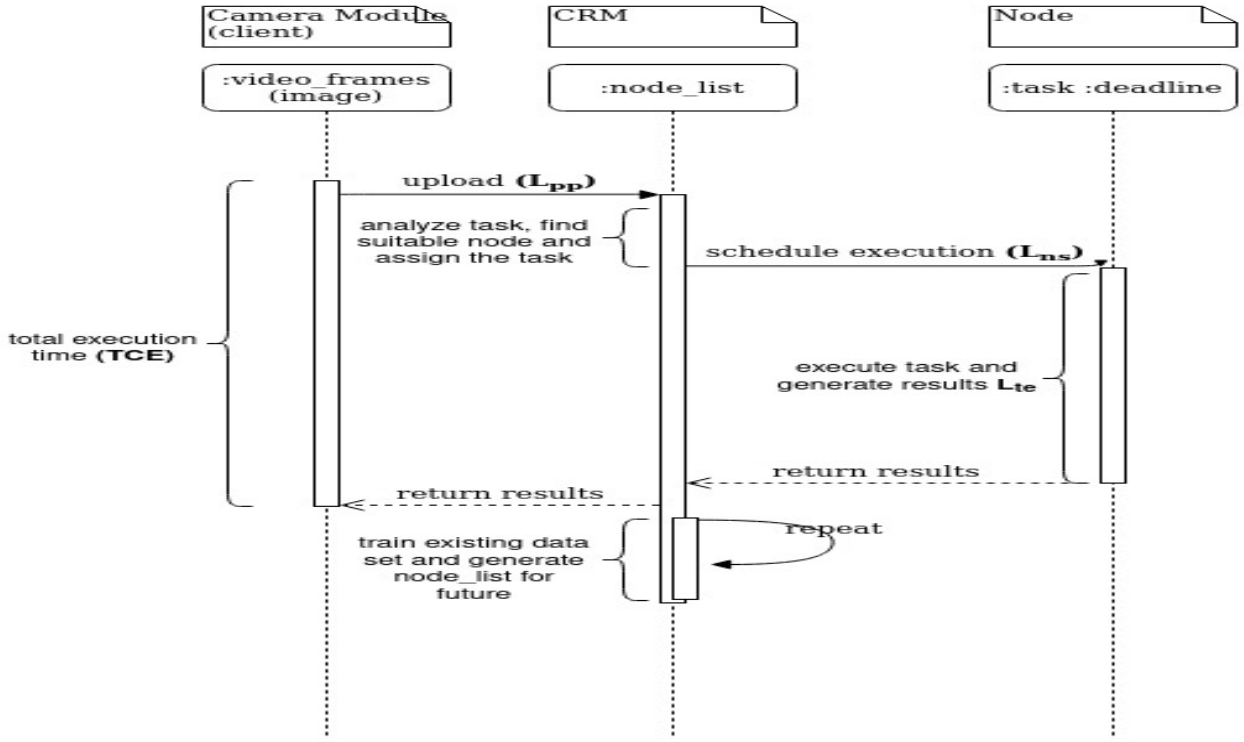

Figure 2: System Sequence Diagram

is currently executing an offloaded task drops from the cluster, the system will hit a time penalty and have to restart the task with higher priority to make up for the loss of time. A heartbeat mechanism with a fifteen minutes time delay may help in avoiding such scenarios. This heartbeat can also be used to update the node ranking by checking parameters such as the battery, utilization rate, and network strength.

If, at any point, a new device joins the Femto cluster, which has not been evaluated yet, the ranking of the device will be based on the model number of the device. The system ranks the device based on Linpack value. "Linpack is the most popular benchmark for ranking of supercomputers and high performance systems by performance. General idea of Linpack benchmark is to measure the number of floating point operations per second (flops) used to solve the system of linear equations. Top500 list of the most powerful supercomputers in the world is based on the Linpack results" **?**. I am using the Linpack ranking dataset for the mobile device available in **?**, which consists of around 1000 mobile devices with attributes such as CPU type, CPU Cores, Clock Speed, OS, and ram. It contains the data for both android as well as iOS devices, and the ranking of Linpack is based on basic vector and matrix operations.

A notification is sent by the node to the CRM when it finishes the task execution. The CRM retrieves the output in a deadline first approach. While receiving the output, the system fires its AI module to build a training model from the memory to train the Q-Network. Both network and execution delay are calculated based on the equations derived from Cicconetti et al. (2019) and Shekhar et al. (2020) along with weighted parameters such as Linpack ranking.

Since mobile devices have a limited battery capacity, if too much battery is consumed by the task that has been executed on it, it may affect the battery life, and the user may or may not offer its device of executing a task in the future. For preventing this, I am

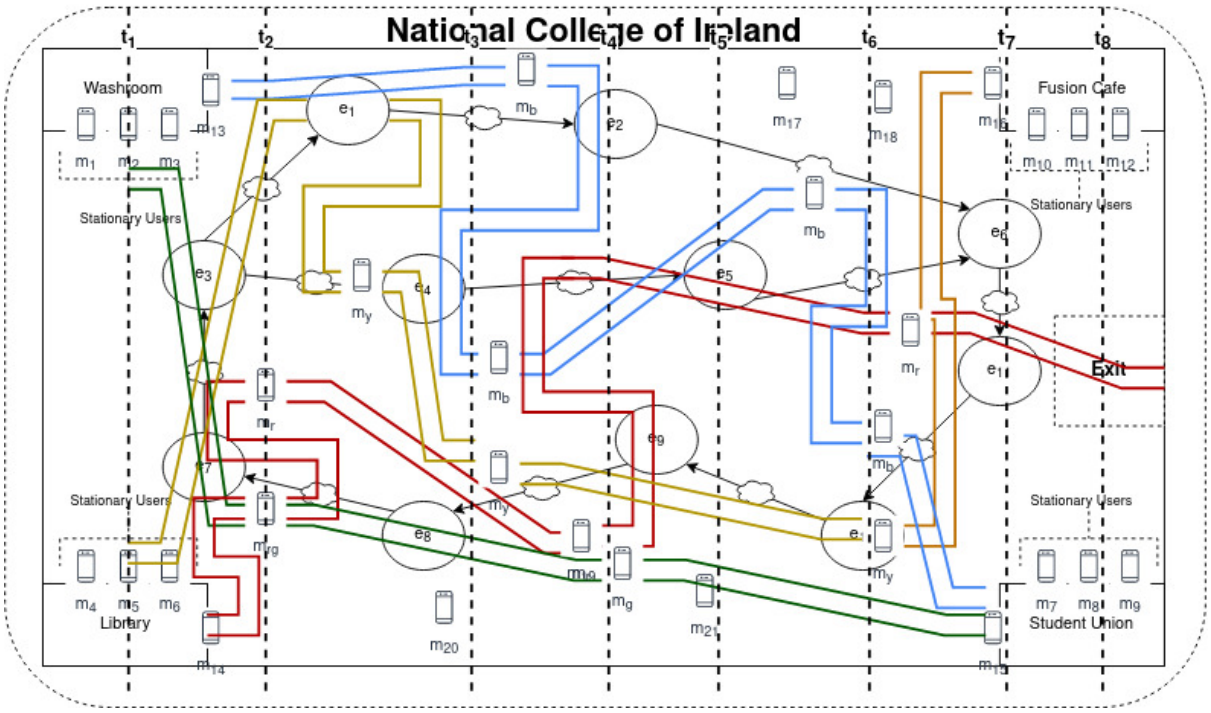

Figure 3: Application use case

placing a threshold for battery consumption $BT_{threshold}$. W.r.t, we can say that whatever the task is, if the battery consumption has exceeded the threshold defined, the task will be dropped from the device immediately. Task will also be dropped if the execution time exceed the defined deadline i.e., $T_{deadline} > \max TCE$. Lets denote the dropped task as $D_n(t) = 0, 1$, where t is the time slot at which the task is dropped from the mobile device n. The dropping of the task will certainly involve some cost D_{cost}, therefore the total cost can be further described as:

$$TCE(t) = \max[L_{pp} + L_{ns}] + [L_{te} + D_n(t) * D_{cost}] + L_n] \tag{3}$$

With this, we can decide on selecting the execution platform where TCE is the minimum.

$$Node_{selection} = \min_{\forall m \subset M(t), \forall e \subset E(t)} \lim_{T \to T_{deadline}} \sum_{t=0}^{T-1} TCE(t) \tag{4}$$

The node selection can be made using various optimal routes finding algorithms such as meta-heuristic or genetic or MDP. In Aazam et al. (2018), the author has clearly defined all the possible scheduling algorithms used by other researchers to find the optimal node for scheduling the task. However, most of these traditional optimization algorithms have some demerits, as described in section 2. ADHIKARI et al. (2019) also mentions that some authors have used hybrid algorithms composed of two or more types of an algorithm to balance out the disadvantages. In this paper, I intend to take the same approach and solve the optimal node problem with an algorithm composed of Deep Q Network as well as SAC based algorithm.

Deep Q Network is a type of reinforcement learning in which an agent or an actor learn an action or behavior that would give it the most reward. Moreover, Q-network is used to examine the Q-function using neural networks. Using Q-Network, one can reduce the

variation among the optimal values and predicted values Liu et al. (2019). I choose to use DQN because of two reasons: firstly, both the optimal as well as predicted values in Q-Network are based on the same network, and therefore it is not easy to make the predictive values as similar as the optimal value which makes the actor learn policies that may or may not lead to the desired result. Secondly, DQN is more abstract, and because it can use data stored in the buffer for leaning, the trainable weights of the network (represented by θ) become independent, and any update to the predicted network does not impact the optimal network Liu et al. (2019).
However, this model of actor and action may have a performance impact. To solve this part, this paper has applied an approach that also involves SAC based optimization.

5 Implementation

The sequence diagram of the system is shown in Fig. 3. After the client uploads the video frame, the CRM performs analysis and estimates the execution time and the priority based on its deadline. The selection of the node for offloading is then followed.

The process flow of the proposed Deep Q-Network algorithm is shown in the state

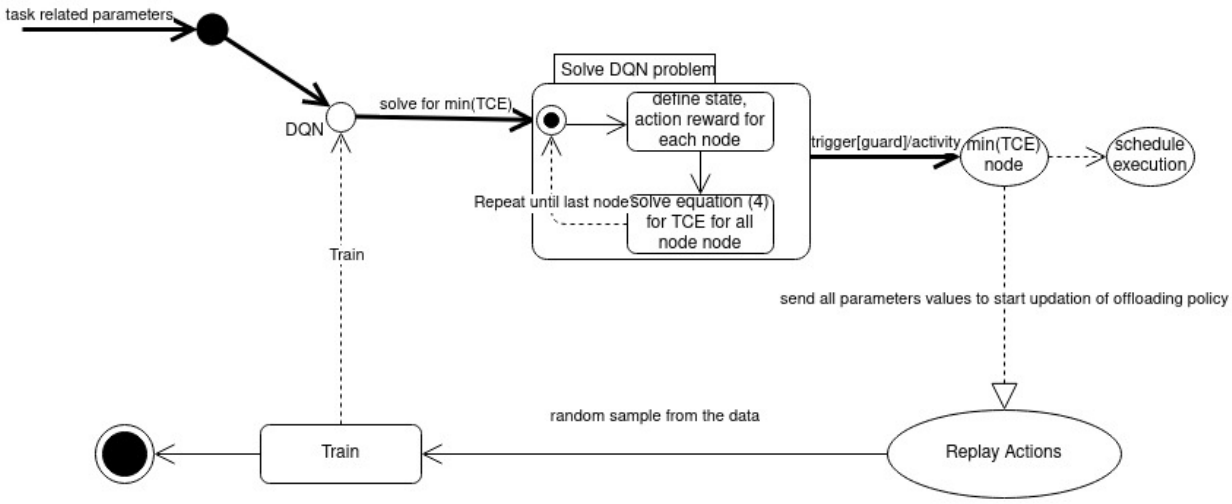

Figure 4: System State Diagram

diagram in Fig 4. The entire semantics of the proposed DQN algorithm is divided into two sections, solving the $Node_{selection}$ problem to find the node with min TCE along with scheduling the offload (action) and updating this decision mechanism to yield the best results (reward). The action can be achieved using DQN, and the desired rewards can be tailored to the target values by training a set of a random batch of samples from the buffer memory to train the DQN model.
Let us assume that a mobile device M_i is connected with the edge node E_i at a time slot t_i. At that time slot, the CRM may or may not have an offloading task for the devices. This problem of task availability at a certain time slot can be stated as a probability function of Bernoulli distribution $P_{t_i} = [0, 1]$ where $P_{t_i} = 1$ represents that there is a task available for offloading. Let us say the video size that needs to be processed and split into a set of images be D_{client}, and the data size received for the offloading be D_off. With this, let us

also assume that the processing power or the CPU cycles required for the execution of the data is C_{client} and C_{off} for edge and mobile devices respectively with F_{client} and F_{off} as the frequency of the CPU at the respective platforms. Each edge node can have a limited number of tasks running at any point in time, each mobile device can have only one offloading task running, and the cloud data center can have any offloading task running. This assumption is made purely based on the resource capacity of these platforms. Lets denote $N = n_1, n_2, n_3...$ as the node where the offloading is to be scheduled if at time slot t, $P_{t_i} = 1$. N can be a local device, Femto cloud-connected mobile device, and a cloud data center. When computing the best fit node for execution, if the task at hand has a very short deadline $T_{dealine} \geq \max TEC$ or the current capacity of the local device where CRM is deployed is enough to execute the task without breaching the deadline then its better to execute the task locally. In such scenario, $TCE \simeq 0$

5.1 Proposed algorithm

As mention in the section 4, I intend to solve the problem of selection of node for code offloading using deep Q-Network as well as Lyapunov optimization. Any DQN algorithm is composed of three things: state, action, and a reward.

1. State: The capacity or the capabilities of the execution platform.
2. Action: Decision to offload to a particular platform, either Femto cloud, edge, or the cloud.
3. Reward: Minimum latency during code offloading.

This can be represented as a function of Q Network. The system's state can be divided into two parts: the maximum capacity as the device or platform and the available capacity at time slot t. The action is the probability function stated P_{t_i} in section5. The reward is executing the task with a minimum cost of execution, i.e., executing the task with minimum latency. Let us denote the total cost of execution in the local environment and offloaded environment as TCE_{local} and TCE_{off}, respectively. The final aim of the algorithm is to maximize the reward of R.

$$TCE_{DQN} = \sum_{N=n_1}^{n_n} \alpha_N * TCE_{offN} + (1 - \alpha_N) * TCE_{localN} \tag{5}$$

This equation 5 is derived from the DQN Q Function, as stated in Liu et al. (2019), where α is the decision to act or not. From this, we can calculate the reward of R.

$$R = \Upsilon * R_{TCE} \qquad \text{where,} \tag{6}$$

$$R_{TCE} = 1 - \frac{TCE_{off}}{TCE_{local}} \tag{7}$$

Solving the best node problem in order to determine $Node_{selection}$ is a bit difficult task as it is a non-convex optimization problem ADHIKARI et al. (2019), **?**. This is why I choose to use a Sample and Classification (SAC) approach to solve this problem. In my algorithm, I assume that the error produced is independent of the target value in each iteration of the learning phase. This approach is also applied in **?**.
The The proposed algorithm first initializes the DQN network with random values for θ,

representing the different parameters to be used in DQN for building offloading policy. The parameters are set randomly for every time slot $t \subset T$. The general idea is to generate a set of casual offloading actions. These generated actions will then be tested against the offloading policy to determine whether the corresponding action yields the most reward or not.
The DQN takes this set of actions, generates the reward based on the action parameters, creates a new pair of state and action, and stores it in the buffer memory. Before doing so, it has to quantize the actions into K-binary action meaning, creating a set of state-action pairs that need to be tested again, a hypothesis, out of which the best-fit action is chosen by solving equation 4. However, this best fit node is not necessarily the best node because we have taken random variables in the parameters. The algorithm applies a Sample and classification approach to check if its truly the best node or not asserting a hypothesis. Sampling is done from the action previously generated by the DQN.
Here, sampling is based on a rule that once a task hits its deadline $T_{deadline}$, there involves a penalty, and the task is to be immediately shifted to either local or cloud data center. It is best to avoid penalties as it decreases the reward of the action. Before proceeding to solve the best node, any mobile device where a task is already running $P_{t_i} = 1$ is removed from the set. Although a mobile device which has a task running may have enough capacity to accommodate another task since the environment of the Femto cloud is too volatile, i.e., any device can join and leave at any time, there is a high risk of hitting a penalty if the selected node leaves the premises. To avoid having multiple penalties, each mobile device is only chosen iff it is not running any task currently as well as the battery threshold of the mobile device has not been exceeded.
To verify if the generated solution is the best node or not, again a random action is generated with a greedy approach and then compared with the previously generated action. Simultaneously, when the previously generated action is not the best fit node, the entire process is repeated until we find the best fit node. Once we find the best fit node from the set of a generated node (action) by DQN, we proceed to train the DQN to select similar values for the node for generating future actions. To do so, a random batch is taken from the buffer memory to train the DQN. Based on the training, a new offloading policy is generated, and in the next iteration of the time slot, this newly updated policy is used to generate better action, therefore, improving the decision making process. The best fit node returned as the output.
To solve equation 4, the system compares the parameters of various weights during the start of the decision making process. The weights are task size, Linpack ranking of mobile devices connected, the available capacity of the edge devices, the distance between the device and CRM, and battery level of mobile devices. Based on these, the system calculates the most suitable node for executing the task.

Algorithm 1 Proposed Algorithms using DQN and SAC approach

Input: $task\_info$
Output: $best\_fit\_node$
Data: $selected\_node\_history$
Initialize variables
$SelectedNode = null$
for *each t in TimeSlot T* **do**
  Initialize DQN with random values θ and empty memory.
  Generate a greedy offloading action by selecting random node.
  Quantize the generated action into K-binary
  Calculate 5 by solving 4
  S = Generate best fit action
  for *each candidate node S* **do**
    if *At time slot t* $\forall$ *T,* $P_{t_i} = 1$ **then**
      if *S.Contains(Type."Mobile")* **then**
        remove all nodes where battery threshold is $BT_{threshold} \geq 15\%$
          remove all nodes from S where $P_{t_i} = 1$ `/* If any task is already running remove it from the list of potential candidate */`
    select random nodes from candidate Node S from all possible nodes available where $\forall m \subset M$, $\forall e \subset E$
      set $N_(S_t) = 1$ `/* Assign the randomly selected node */`
  N(t) $\rightarrow \min TCE_{DQN}$ `/* Calculate TCE for the node by solving 5 */`
  if *node is Type."Mobile"* **then**
    CurrentBatteryOfDevice = S.Node.Battery
  let *rand* = uniformly sampled value from S **for** *each node in rand* **do**
    apply greedy policy for selected node and store results
    S_{new} = Generate best fit action using values from *rand*
  if S_{new} *is not better than* $\boldsymbol{S}$ **then**
    repeat the process untill best fit node found
  else
    $S_{new}(P_{ti}) = 1$
    update memory related variables
    CurrentBatteryOfDevice = CurrentBatteryOfDevice - S.Node.Battery
    update battery threshold related variables
    $SelectedNode = S_{new}$ Uniformly sample a batch of data set from the memory
    Train the DQN
return $SelectedNode$

s

6 Evaluation

6.1 System Simulation

In my simulation environment, I have used three datasets for simulating the proposed algorithm. I have used edge sites location in Melbourne data taken from **?**. The dataset is generated from the Australian Communications and Media Authority for the radio

base station dataset and user location details. Each edge site has a limited coverage area around which users are connected to it. **?** is the dataset that contains the ranking of different mobile devices with their configuration and OS details. The system configuration that are used are presented in Table 1. The values used are all random and doesn't have any correlating with each other. This is done in order to simplify the evaluation otherwise, we need to go further and define each parameter based on various other factors. The overall cost for task execution TCE is the main criteria taken for evaluation.

Table 1: Configuration Parameters

Parameters	**Assigned values**
Time Slot Length	3 ms
Number of Time Slots	50
Task Dropped Penalty	2 ms
Average Task Deadline	2 ms
Probability of a task available for offloading	0.6
Unit CPU Cycle	737.5
Local Input Size	100 bits
Local CPU frequency	1.5 GHz
Max battery consumption allowed	15%
Edge Input Size	3000 bits
Max Assigned Tasks in edge	4
Edge CPU freq	32 GHz
Bandwidth	1.5 GHz
Min Distance	150
Max distance	400
Number of Mobile devices	10
Capacity of Memory Structure	1024
Update interval for adaptive K	32
Size of train data	80%
Size of Test data	20%
Learning rate	0.01
Batch size	100

6.2 Benchmarking

In paper Aazam et al. (2018), Alam et al. (2019), Cicconetti et al. (2019), the author have evaluated their proposed system with general baseline algorithm as a Greedy algorithm. The authors have tested their proposal against the results obtained without the system's presence using a baseline algorithm. I also took the same approach as I could not find any direct code related to the implementation of the proposed solution in various papers reviewed in section 2.
The proposed algorithm is evaluated against three algorithms.

1. Randomly selected: Each time a task is available for offload at any time, out of all the nodes available in the cluster, a random node is selected at once, and the task is scheduled, and no consideration to any parameters is given.

2. Nearest node selected: Each time a task is available for offloading at any time, out of all the nodes available, the node which is the nearest to CRM (around 150 to 200m) is selected, and the task is scheduled there for execution.

3. Local node only: Each time a task is available for offloading, the execution takes place locally on the edge device where the CRM is deployed.

6.3 Results

As shown in Fig 5, the experiment done shows that the proposed algorithm has a better performance when compared with at least two baseline algorithms define in 6.2. It shows tremendous improvement over randomly selected nodes for offloading. The graph indicates that my algorithm performed at approx. 81% better than randomly selection algorithm, 27% improvement over the nearest selection algorithm.
However, compared to local execution, the proposal algorithm lags around 38%. This can be since there is no cost of analysis of the task in local processing and no cost of network delay. The only cost that involves in executing a task in the local device is the cost of execution of the task, i.e., L_{te} and therefore, in this case, $TCE =\approx 0$.

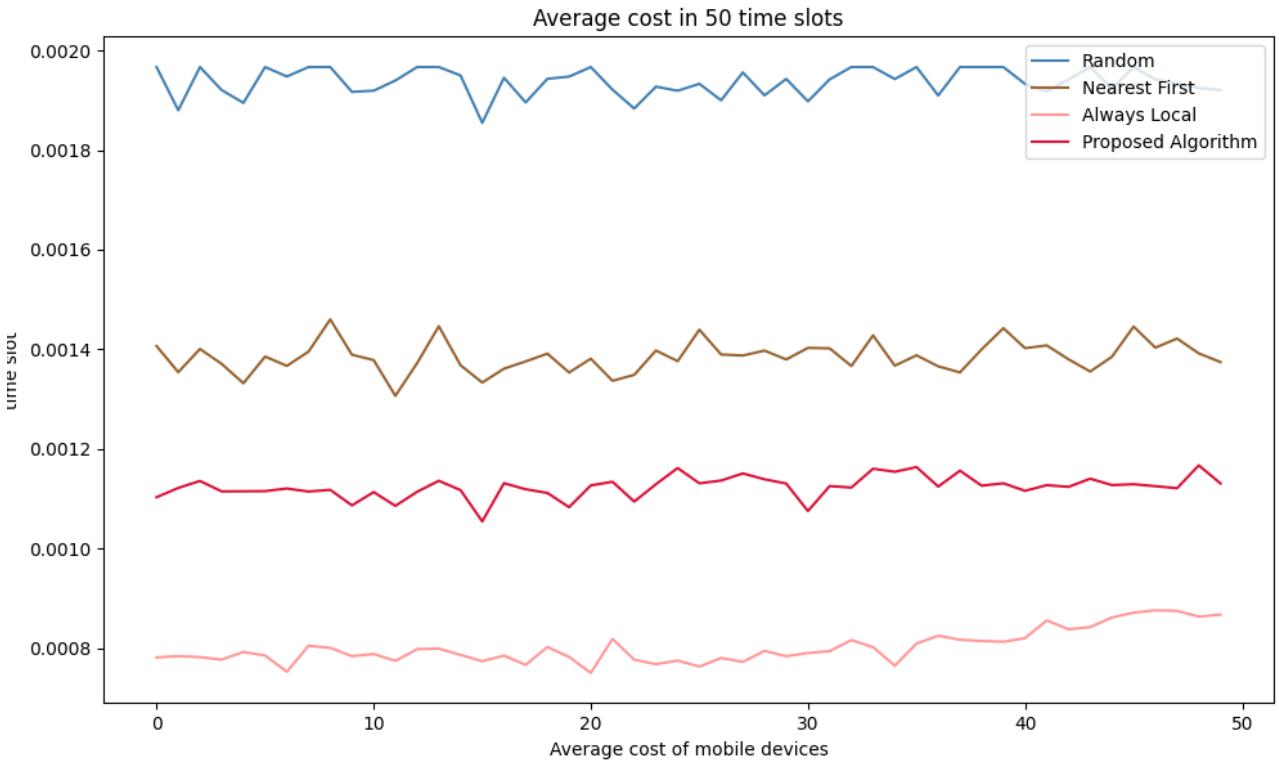

Figure 5: Proposed System Architecture

7 Conclusion and Future Work

This paper has proposed a reinforcement learning-based algorithm that determines the best fit node among several nodes involving edge device and Femto clouds based mobile

devices for scheduling code offloading. The proposed work can efficiently choose an execution platform by taking various parameters available then by weighing each parameter and applying a hybrid approach involving DQN and SAC to solve the problem with the shortest total cost of execution. The learning component used in the algorithm makes use of Q-Network to learn the past offloading action taken up by the system and gradually improves it. The results are based on dataset taken from public sources and show that the proposed algorithm performs better than a typical baseline algorithm; however, it still lacks behind when the processing is done locally, which is natural.
However, the proposed algorithm does not consider parallel execution. If the task execution is made so that it can be spliced and executed among several nodes parallel, it may further reduce the latency. Another aspect in which this proposal lacks is the ability to do a cross-platform execution.

References

Aazam, M., Zeadally, S. and Harras, K. A. (2018). Offloading in fog computing for iot: Review, enabling technologies, and research opportunities, *Future Generation Computer Systems* **87**: 278 – 289. Impact Factor 4.639.
URL: *http://www.sciencedirect.com/science/article/pii/S0167739X18301973*

ADHIKARI, M., AMGOTH, T. and SRIRAMA, S. N. (2019). A survey on scheduling strategies for workflows in cloud environment and emerging trends., *ACM Computing Surveys* **52**(4): 1 – 36.
URL: *http://search.ebscohost.com/login.aspx?direct=trueAuthType=ip,cookie,shibdb=bthAN=13860 livescope=sitecustid=ncirlib*

Alam, M. G. R., Hassan, M. M., Uddin, M. Z., Almogren, A. and Fortino, G. (2019). Autonomic computation offloading in mobile edge for iot applications, *Future Generation Computer Systems* **90**: 149 – 157. Impact Factor 4.639.
URL: *http://www.sciencedirect.com/science/article/pii/S0167739X18303996*

Baresi, L. and Filgueira Mendonça, D. (2019). Towards a serverless platform for edge computing, *2019 IEEE International Conference on Fog Computing (ICFC)*, pp. 1–10. Core Ranking C.

Cicconetti, C., Conti, M. and Passarella, A. (2019). Architecture and performance evaluation of distributed computation offloading in edge computing, *Simulation Modelling Practice and Theory* pp. 1–21. Impact Factor 2.092.
URL: *http://www.sciencedirect.com/science/article/pii/S1569190X19301406*

Habak, K., Ammar, M., Harras, K. A. and Zegura, E. (2015). Femto clouds: Leveraging mobile devices to provide cloud service at the edge., *2015 IEEE 8th International Conference on Cloud Computing* p. 9.

Hai Duc Nguyen, A., Chaojie Zhang, A., Zhujun Xiao, A. and Andrew A. Chien, A. (2019). Real-time serverless : Enabling application performance guarantees., *Serverless Computing* p. 1.
URL: *http://search.ebscohost.com/login.aspx?direct=trueAuthType=ip,cookie,shibdb=edscmaAN=ea livescope=sitecustid=ncirlib*

Hall, A. and Ramachandran, U. (2019). An execution model for serverless functions at the edge, *Proceedings of the International Conference on Internet of Things Design and Implementation*, IoTDI '19, Association for Computing Machinery, New York, NY, USA, p. 225–236. Core Ranking B.
URL: *https://doi.org/10.1145/3302505.3310084*

Kent, A. and Williams, J. G. (2018). *Computers in Spaceflight:The NASA Experience*, 2 edn, NASA. Chapter 2-5, https://www.hq.nasa.gov/office/pao/History/computers/Ch2-5.html.

Li, L., Guo, M., Ma, L., Mao, H. and Guan, Q. (2019). Online workload allocation via fog-fog-cloud cooperation to reduce iot task service delay., *Sensors* **19**(18): 3830. Impact Factor 2.475.
URL: *http://search.ebscohost.com/login.aspx?direct=trueAuthType=ip,cookie,shibdb=a9hAN=1390*
livescope=sitecustid=ncirlib

Liu, Y., Cui, Q., Zhang, J., Chen, Y. and Hou, Y. (2019). An actor-critic deep reinforcement learning based computation offloading for three-tier mobile computing networks, *2019 11th International Conference on Wireless Communications and Signal Processing (WCSP)*, pp. 1–6.

Meurisch, C., Gedeon, J., Nguyen, T. A. B., Kaup, F. and Muhlhauser, M. (2017). Decision support for computational offloading by probing unknown services, *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–9. Core Ranking A.

Nobre, J. C., de Souza, A. M., Rosário, D., Both, C., Villas, L. A., Cerqueira, E., Braun, T. and Gerla, M. (2019). Vehicular software-defined networking and fog computing: Integration and design principles, *Ad Hoc Networks* **82**: 172 – 181. Impact Factor 3.151.
URL: *http://www.sciencedirect.com/science/article/pii/S1570870518305080*

Pinto, D., Dias, J. P. and Sereno Ferreira, H. (2018). Dynamic allocation of serverless functions in iot environments, *2018 IEEE 16th International Conference on Embedded and Ubiquitous Computing (EUC)*, pp. 1–8. Core Ranking C.

Shekhar, S., Chhokra, A., Sun, H., Gokhale, A., Dubey, A., Koutsoukos, X. and Karsai, G. (2020). Urmila: Dynamically trading-off fog and edge resources for performance and mobility-aware iot services, *Journal of Systems Architecture* **107**: 101710.
URL: *http://www.sciencedirect.com/science/article/pii/S1383762120300047*

Torres Neto, J. R., Rocha Filho, G. P., Mano, L. Y., Villas, L. A. and Ueyama, J. (2019). Exploiting offloading in iot-based microfog: Experiments with face recognition and fall detection., *Geofluids* pp. 1 – 13.
URL: *http://search.ebscohost.com/login.aspx?direct=trueAuthType=ip,cookie,shibdb=a9hAN=1362*
livescope=sitecustid=ncirlib

Vaquero, L. M., Cuadrado, F., Elkhatib, Y., Bernal-Bernabe, J., Srirama, S. N. and Zhani, M. F. (2019). Research challenges in nextgen service orchestration, *Future Generation Computer Systems* **90**: 20 – 38. Impact Factor 4.639.
URL: *http://www.sciencedirect.com/science/article/pii/S0167739X18303157*

Xu, X., Li, Y., Huang, T., Xue, Y., Peng, K., Qi, L. and Dou, W. (2019). An energy-aware computation offloading method for smart edge computing in wireless metropolitan area networks., *Journal of Network and Computer Applications* **133**: 75 – 85. Impact Factor 3.991.
URL: *http://search.ebscohost.com/login.aspx?direct=trueAuthType=ip,cookie,shibdb=edselpAN=S1*
livescope=sitecustid=ncirlib

Xu, X., Liu, Q., Luo, Y., Peng, K., Zhang, X., Meng, S. and Qi, L. (2019). A computation offloading method over big data for iot-enabled cloud-edge computing, *Future Generation Computer Systems* **95**: 522 – 533. Impact Factor 4.639.
URL: *http://www.sciencedirect.com/science/article/pii/S0167739X18319770*