Actor Critic-based Multi Objective Reinforcement Learning for Multi Access Edge Computing

Vishal Khot[1], Vallisha M[2], Sharan S Pai[3], Chandra Shekar R K[4], Kayarvizhy N[5]
Department of Computer Science and Engineering, BMS College of Engineering, Bangalore, India[1, 2, 3, 5]
Department of CSIS, BITS Pilani, Goa, India[4]

***Abstract*—In recent times, large applications that need near real-time processing are increasingly being used on devices with limited resources. Multi access edge computing is a computing paradigm that provides a solution to this problem by placing servers as close to resource constrained devices as possible. However, the edge device must consider multiple conflicting objectives, viz., energy consumption, latency, task drop rate and quality of experience. Many previous approaches optimize on only one objective or a fixed linear combination of multiple objectives. These approaches don't ensure best performance for applications that run on edge servers, as there is no guarantee that the solution obtained by these approaches lies on the pareto-front. In this work, Multi Objective Reinforcement Learning with Actor-Critic model is proposed to optimize the drop rate, latency and energy consumption parameters during offloading decision. The model is compared with MORL-Tabular, MORL-Deep Q Network and MORL-Double Deep Q Network models. The proposed model outperforms all the other models in terms of drop rate and latency.**

***Keywords*—Edge computing; reinforcement learning; multi objective optimization; neural networks; deep learning**

I. Introduction

In the modern day, mobile devices handle more computationally demanding activities, including data processing, artificial intelligence, and virtual reality. Despite advancements in mobile technology, these devices lack sufficient computational capacity to complete all of their duties locally with low latency and reasonable energy consumption. Mobile apps for online gaming, signal or image processing (such as facial recognition), augmented reality, and real-time translation services are some examples of computational domains whose use has grown drastically that places a substantial computing demand on mobile devices (MDs) which have a limited amount of resources.

Mobile edge computing (MEC), also known as fog computing and multi-access edge computing, is a technology that enables effective job processing. [1] It is a new computing paradigm in which computing, network, storage, capabilities are migrated to edge nodes, which is closer to end-users to meet real-time needs of fast changing IT industries. The demand for on-demand computation close to mobile devices is only expected to grow. Additionally, as 5G networks become more and more popular, the three main services - massive machine communication, enhanced mobile broadband and ultra-reliable low-latency communication pose network, computing, storage, and application core capabilities. As a result, their applications can be run on the edge server, enabling faster network service response and satisfying the real-time processing, intelligent application, security, and other requirements. Despite edge computing's enormous potential, there are many obstacles. Mobile real-time apps are extremely sensitive to latency and power usage. However, the prolonged time of execution of these applications can result in excessive energy consumption owing to the randomness and volatility of mobile edge networks.

Single objective reinforcement learning algorithms perform considerably well in environments where there is only one objective to optimize, which is often not the case in real world scenarios. The offloading requirement in the case of multi-access edge computing needs to satisfy many conflicting requirements like latency, energy, drop rate, QoS, and cost, among others. Optimization of just one objective can provide neither a guarantee of pareto optimality nor a control over the order of preference of the multiple objectives to suit the specific use-case, albeit at the cost of pareto optimality. Multi objective reinforcement learning (MORL) approaches can be leveraged to overcome the above shortcomings whilst maintaining adaptability to work in dynamic environments.

The research contribution of this work is the application of the actor critic method in multi objective reinforcement learning algorithms for the task offloading problem as opposed to previous literature that have used the actor critic method for single objective reinforcement learning.

II. Related Work

The decision to offload a task or not is a complex one, with multiple factors about the problem itself and the solution to be considered. Firstly, tasks can be considered to either be dependent or independent of one another. Literatures choosing to work on dependent tasks usually consider a directed acyclic graph to represent task dependencies. Secondly, the decision to offload or not can be made either centrally or by each mobile device, in a decentralized fashion. We have considered the papers that have used reinforcement learning to make the decision to offload or not. The network architecture of mobile devices and servers must be considered. The parameters for making the decision to offload or not are the task size, algorithmic complexity, the time by which the task needs to be completed, task interdependencies, and bandwidth. Authors choose a subset of these parameters for their system. The RL system can either be based on table or on function approximation. The reward for the RL agent can be based on latency, energy, cost, drop rate, QoS considerations.

Tang et al. [2] propose a cost optimized reinforcement learning algorithm, where every mobile device makes an independent decision to offload or not, while also being aware of edge load dynamics. T Alfaikh et al. [3] propose using SARSA for making the decision to offload or not to the closest server or adjacent server or to compute it locally. J Wang et al. [4] propose using meta RL for faster adaptability and use a sequence2sequence network for making the decision to offload or not. J Wang et al. [5] combine their approach with a specific off-policy policy gradient algorithm with a clipped surrogate objective. Liang Huang et al. [6] propose using deep q learning while optimizing on energy with constraints on bandwidth. Peizhi Yan et al. [7] propose using deep q learning with both node and edge level offloading. Xiaowei Liu et al. [8] propose using a parameterized, indexed-value function for value estimation for achieving faster convergence.

Zhenjiang Zhang et al. have proposed a multi-agent load balancing distribution deep reinforcement learning algorithm [9]. It minimizes the latency, load factor and the algorithm complexity as compared to a centralized algorithm. It uses a genetic algorithm to identify the decision to offload or not while still meeting the QoS requirements of all the tasks. Yu Dai et al. propose a federated reinforcement learning algorithm to address the issue of weak generalization of the model and privacy leakage caused by sharing user sensitive information to the central server [10]. Attention is used to aggregate the parameter weights resulting in the reduction of the processing time of the task. Yuanchao Xu et al. [11] explore decentralized multi-agent reinforcement learning algorithms to solve the problem of task offloading accounting for reward uncertainty. They try different approaches like Multi-Agent Deep-Deterministic Policy-Gradient (MADDPG), Robust MADDPG and Decentralized Partially-Observable Markov Decision Process (Dec-POMDP). Baris Yamansavascilar et al. [12] propose a task orchestrator based on deep reinforcement learning which learns to satisfy different task requirements without any human interaction. The problem is modeled as a Markov decision process and the Double Deep Q-Network algorithm is used to minimize the task drop rate. Xiangjun Zhang et al.[13] propose a task offloading algorithm for Reconfigurable Intelligent Surface (RIS) empowered Mobile Edge Computing networks. The problem is formulated as a Markov Decision Process (MDP) where latency, energy consumption and operating costs are minimized. DDPG is used to jointly optimize the phase shift and amplitude of RIS, task allocation strategy and offloading decision. Tu et al. [14] design a predictive offloading algorithm that makes use of deep RL and long short-term memory (LSTM) networks. The MEC server's load is monitored and the next task is predicted using the LSTM network. The amount of latency, energy used, and work abandonment is decreased. Liang Huang et al [15] propose an online, deep RL based algorithm to adapt task offloading and resource allocation decisions to the time varying wireless channel conditions. They aim to minimize the latency. Mingjie Feng et al. [16] explore offloading of tasks between MEC servers and cloud servers using DRL. They aim to minimize average latency and achieve optimization in task partitioning ratio and cloud selection.

Yi Ouyang et al. propose a task offloading algorithm for a vehicle edge computing environment based on Dueling-DQN [17]. The proposed approach considers the mobility of the vehicles and the changing network conditions make a better decision to offload or not. The results of the experiment demonstrate the effectiveness and superiority of the proposed algorithm with respect to existing approaches. Fuhong Song et al. have proposed an approach which uses multi-objective reinforcement learning for optimizing the UAV's trajectory and offloading decisions in real-time [18]. The algorithm considers multiple objectives, including minimizing energy consumption, maximizing data processing efficiency while maintaining a stable network connection. Xianfu Chen et al. propose a new approach based on deep reinforcement learning which optimizes performance of offloading algorithms in virtual edge computing systems [19]. Unlike other approaches which use traditional heuristics or machine learning algorithms, the proposed approach uses a deep-neural network to learn the optimal offloading policy in a data-driven manner. Junyao Yang et al. [20] propose an inverse order based optimization technique for resource allocation and task offloading in multi-access edge computing systems. It makes the offloading decision and resource allocation and optimizes them jointly using an inverse order optimization strategy unlike other approaches which consider them separately. Ting Wang et al. [21] propose an approach based on deep reinforcement learning (DRL) for improving the performance of task offloading in the Internet of Vehicles (IoV) scenario. The proposed approach uses a deep neural network to learn the optimal offloading policy based on IoV's dynamic and uncertain environment. Hongxia Zhang et al. [22] propose an ultra-low latency multi-task offloading approach for making task offloading decisions in mobile edge computing that considers multiple tasks with different requirements, such as computation, communication, and storage. The proposed approach uses a multi-agent reinforcement learning algorithm to make the decision to offload or not while satisfying the different task requirements. X Zhang et al. [23] propose an approach based on deep reinforcement learning for energy-efficient task offloading in secondary mobile edge systems. The proposed approach uses a deep-neural network to learn the optimal offloading policy in accordance with the dynamic and uncertain environment of the edge system. Mushu Li et al. [24] propose creating a cooperative edge computing framework to lower latency and increase dependability for vehicular networks. In order to determine the best option that reduces the cost of the service, a DDPG algorithm is used.

Juan Chen et al. [25] propose a multi-agent DRL solution to minimize total cost in terms of energy requirement of IOT device and long term renting cloud costs. They have explored centralized training and decentralized execution, hence, each IOT device will be a decision making agent. Xing Chen et al. [26] explore a federated DDPG solution for combined optimization of energy consumption and reduction in latency. The federated learning procedure ensures privacy of user data because only parameters of locally trained models are sent to central servers. Hao Meng et al. [27] constructs a DRL model with a new reward function design for optimizing the battery power of mobile devices. The reward function simulates the tradeoff between latency and battery consumption. Kun Wang

et al. [28] formulated a Double DQN model to apply the task offloading concept to the Internet of Vehicles. This proposed algorithm solves real-time changes in the network due to user movement. Fuhong Song et al. propose a MORL algorithm to make the decisions with respect to task offloading when subject to multiple dependent tasks to optimize on three objectives, energy consumption, latency and user costs, simultaneously and using independent rewards for each [29]. They use tournament selection schemes to maintain previously learnt policies. Yu Chen et al. [30] investigated multi-user edge video analytics task offloading problems. The authors design two algorithms, one based on Game Theory and another based on the Actor-Critic method. The proposed A2C is observed to be more flexible, users can adjust accuracy decisions and achieve the converged reward.

III. MEC Architecture

The considered architecture shown in Fig. 1 is similar to [2] where there is a set of resource constrained mobile devices M = {1, 2, 3, …. , m} and a single edge server in the MEC environment. We measure time using discrete timesteps where each timestep is equal to 100 milliseconds. All mobile devices are polled at the start of each timestep to check if any tasks have been generated. The following section explains the device and server models.

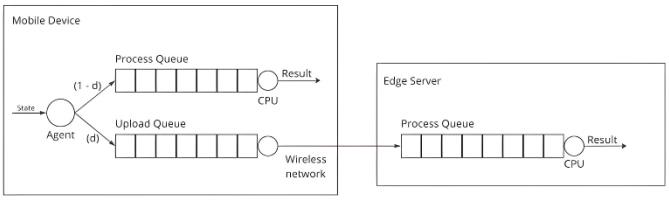

Fig. 1. MEC architecture

A. Device Model

The test environment consists of multiple mobile devices which generate non-divisible tasks with no interdependency which can either be computed locally on the device or on an edge server. Tasks can only be generated at the start of each timestep. The decision d as to whether a task has to be computed locally on the resource constrained mobile device or offloaded to an edge server is taken by a reinforcement learning (RL) agent. If the task has to be computed locally, it is first pushed into the local process queue of the device. The process queue follows a first-in first-out (FIFO) principle and once the task is computed, its result is returned. If the task is supposed to be offloaded, it is pushed into the upload queue of the device and subsequently uploaded to the edge server. Similar to the process queue, the upload queue follows a FIFO principle. It is assumed that once a task is computed or uploaded, the computation or upload process of the next task in the respective queue will be started only in the next timestep.

1) Task model: It is assumed that new tasks are only generated at the start of each timestep. Tasks are generated in all mobile devices with a probability of 0.3. For each task the parameters considered are, the task size (in bits), task timeout (in timesteps), algorithmic complexity and start time of the task. The task start time is the timestep at which the task was generated. In case a task is computed locally, the total execution time is equal to the sum of the duration of time spent in the process queue of the device and the actual computation time. Otherwise, if it is offloaded, the total execution is equal to the sum of the duration of time spent in the upload queue of the device, the time necessary to upload the task to the edge server, the duration of time spent in the process queue of the edge server and the time necessary to execute the task on the edge server. Each task has an associated timeout and if the task is not executed within the timeout, it is considered to be dropped.

2) Offloading decision: The Agent A makes the decision to offload or not d, depending on the state which contains the task timeout, task size, the time required to upload the task, time required to execute the task on the server and the local execution time of the task. d is a binary variable d {0, 1} where d is 1 if the task is to be offloaded or 0 if it is to be processed locally. As seen in Fig. 1, if d is 1 it is pushed into the upload queue of the device, else it is pushed into the local process queue of the device.

B. Edge Server Model

A single edge server is considered in our MEC environment. Tasks which are uploaded to the edge server are pushed into the process queue of the server only at the start of a time step. Before a task is pushed into the process queue of the server, we check if the task is going to be dropped, i.e, it is checked if the task has no possibility of getting executed within its timeout and if so, the task is considered to be dropped and the task will not be pushed into the process queue of the server. The tasks in the queue are computed in a First-In First-Out order. Instead of having one process queue for each mobile device, we maintain a single process queue for all devices. This has the same effect as having a process queue for each device and a First-In First-Out scheduler to pick tasks from all the queues.

C. Reinforcement Learning Environment

We consider a single edge server connected to multiple resource constrained mobile devices via a wireless network.

1) State space: State space is essentially the input to the RL agent to make the task offloading decision. It is a vector of task size, algorithmic complexity and the time before which the task must be processed.

2) Action space: Action space is the set of values that can be returned by the agent. The agent makes a binary decision (0 or 1) i.e. to offload or not.

IV. Multi Objective Reinforcement Learning

Multi objective reinforcement learning is a type of reinforcement learning that aims to optimize multiple potentially conflicting objectives simultaneously to achieve ideal performance in real world scenarios. The architecture explained in the previous section is used to compare four multi-objective reinforcement learning algorithms in a simulated environment containing ten resource constrained mobile devices connected to a single edge server.

A. Actor Critic Method

The actor-critic method is a popular temporal difference (TD) learning approach that consists of two main components, the actor and the critic. The actor suggests actions that can be taken based on a particular state and the critic evaluates the actions taken after being suggested by the actor. Four neural networks are considered namely, actor, critic, actor-target and critic-target. The actor network outputs the expected Q value for each action and the action which has the highest expected Q value is chosen as the next action to be taken. The critic network is used to evaluate how good the action suggested by the actor network is. In addition to the state, the critic network also takes the action taken and the reward obtained as the input. Then, the loss is computed for both the actor and the critic networks and update them.

The actor-target and critic-target networks are used to stabilize the training of the actor and critic networks. At the end of each timestep the network parameters are partially copied from the main networks to the target networks via soft updates. Experience replay is used every 25 timesteps to effectively train the model. This algorithm outperforms all the previous algorithms with respect to drop rate and latency.

Algorithm 1: Actor-Critic method for task offloading

Params: Learning rate $\alpha \in [0, 1)$, small epsilon ε, $0 < \varepsilon < 1$, convergence parameter γ close to 1
Initialize networks Q_a, $Q_a`$, $Q_c$ and $Q_c`$
foreach *timestep* with *task* **do**
  select action a from the set of possible actions A
  using policy derived from $Q_a`$(ε-greedy);
  if *a* is *offload*:
    add *task* to *upload*_queue at mobile device
  elseif *a* is local_*computation*:
    add task to *process*_queue of the respective mobile device
  observe reward R, next state *s`*;
  store the experience (*s, a, R, s`*) in *replay-buffer*
  $Q^*(s, a) = R + \gamma \max Q_c`(s, a)$
  compute mean squared error between $Q_c(s, a)$ and $Q^*(s, a)$, update critic network
  update actor network using loss$\leftarrow \frac{1}{N} \Sigma_i Q_c (s_i , \max Q_a(s_i, a))$
  update target networks
    $\theta_a` \leftarrow \tau \theta_a + (1 - \tau) \theta_a`$
    $\theta_c` \leftarrow \tau \theta_c + (1 - \tau) \theta_c`$
  $s \leftarrow s`$
  use experience replay to train the actor and critic networks every 25 timesteps
end foreach

B. Objectives

The multi objective reinforcement learning model aims to optimize three objectives namely, the task drop rate, latency and the energy consumed by the device.

Task drop rate: The task drop rate is the number of tasks dropped until the current timestep divided by the total number of tasks that were generated until the current timestep.

Latency: Latency is the duration of time between task generation and task completion.

latency = d * (upload_latency + process_latency_server) + (1 - d) * (process_latency_device)

where,

d is the decision to offload or not which takes a binary value (0 or 1)

upload_latency is the total timesteps required to upload the task

process_latency_server is the total timesteps which the server takes to execute the task taking into account the server load (time spent by the task in the process queue of the server)

process_latency_device is the total timesteps taken by the mobile device to execute the task considering the time spent by the task in the process queue of the device.

Energy: It is the energy utilized by the mobile device from the moment the task is generated on it, to the moment the task is completely processed. Naturally, offloaded tasks use a lot less energy than the tasks that are processed locally.

energy = d * (latency * idle_energy_rate) + (1 - d) * (latency * active_energy_rate)

where,

idle_energy_rate is the energy utilized by the device in its idle state

active_energy_rate is the energy utilized by the device during execution

Single objective optimization usually comes at the expense of other objectives which might make the solution not feasible for a lot of real-world applications where we often need to jointly optimize multiple objectives. Multi objective reinforcement algorithms are handy in such scenarios.

C. Reward Engineering

The reinforcement learning agent is given some reward as a result of every action it takes. It gets a positive reward if it takes an action that contributes towards optimization of the objective, otherwise it is punished with a negative reward.

When it comes to multi-objective reinforcement learning where the model optimizes the task drop rate, energy and latency, three rewards need to be considered.

1) A reward of +1 is given to the agent if the task gets executed successfully within its timeout, else it is given a reward of -1. A task is considered to be dropped if it cannot be executed within the timeout specified for that particular task.

2) With regard to energy, we consider a threshold of 0.5 mJ if the task is offloaded and 160 mJ if the task is computed locally. This difference is due to the fact that when a task is offloaded, the mobile device consumes energy only to offload the task and not execute it. The agent gets a reward of +1 if the energy consumed for the given task is below the threshold, otherwise it gets -1 as the reward.

3) Similarly, if the latency for the given task is below 2000 ms, the agent gets a reward of +1, else -1.

The establishment of these thresholds was the outcome of experimentation. It was observed that the average amount of energy consumed per task and the average latency over a period of time for the given task parameters were 0.5 mJ / 160 mJ and 2000 ms respectively in our simulation environment. The three rewards are combined using a set of weights that add up to 1 to obtain a single total reward that is utilized to train the multi-objective reinforcement learning agent. The weights can be changed accordingly if a particular objective is required to have a higher priority than other objectives.

V. Experiments

The experiment was run with different models with tasks generated at the mobile devices with a probability of 0.3 at each timestep. The model under training makes the decision as to either offload the task or to process it locally. The task is appended to the respective mobile devices' upload queue if it is to be offloaded. If the task is to be computed locally, the task is appended to the local computation queue of the mobile device. The experiment is run with ten mobile devices and one edge server for 10,000 tasks. The latency, energy and drop rate is monitored and recorded during the experiment.

VI. Experimental Results

The MEC environment is simulated with ten resource constrained mobile devices and one edge server. We implemented and compared four multi objective optimization algorithms in the same environment with uniform random policy, which is a standard benchmark for any RL model. All the multi objective reinforcement learning models aim to optimize the task drop rate, energy and latency.

We compare the performance of the MORL Actor-Critic model with three other MORL models.

1) MORL-Tabular: We apply the Tabular Q learning algorithm to make the task offloading decision in our MEC system. We consider the four parameters task size, algorithmic complexity, timeout in timesteps and store the action-value pair in a table. In each iteration, the Q-learning model refers to the previous action-value pairs from the table and based on the bellman equation for each update the new action-value pair. Tabular Q Learning can be used for solving problems where the number of states is not large as the storage is limited. Once the problem size increases, we cannot scale Tabular Q Learning, especially in environments where the state space has continuous values.

2) MORL-DQN: We implemented the Deep Q Learning algorithm where, instead of a table we make use of a neural network to map a particular state to an action value, i.e the neural network takes the current state as the input and we get the expected Q value as the output. The action which has the highest Q value stands as the most optimal choice for the present state. We consider the task size, task timeout, time taken to upload the task, time taken to compute the task on the server and time taken to complete the task on the mobile device to make the decision as to whether the particular task at hand should be offloaded to the MEC server.

3) MORL-DDQN: Although Deep Q Learning algorithms perform well, there exists a maximization bias. If at any point in time the Q value is overestimated, the error gets compounded overtime and leads to suboptimal policies and poor exploration. To overcome this issue, in Double Deep Q Learning, two neural networks are used instead of one. We assume one network as the target Q network which is updated less frequently and is used to calculate the target Q values for the Q value update of the other neural network called the online network. The online network is used to determine the best known action in a particular state. The weights of the online network are copied to the target network intermittently.

B. Cumulative Reward

In MORL, a positive reward is given to the model for not just executing the task successfully but also if the task is executed within a limited latency and limited battery utilization of the edge device. A negative reward is given if the task is dropped or if the task is not executed within the thresholds of energy or latency. Fig. 2 shows the plot of cumulative reward vs. timesteps. Although, an analysis is necessary, it is unreasonable to infer the comparison of agent performance from this plot alone.

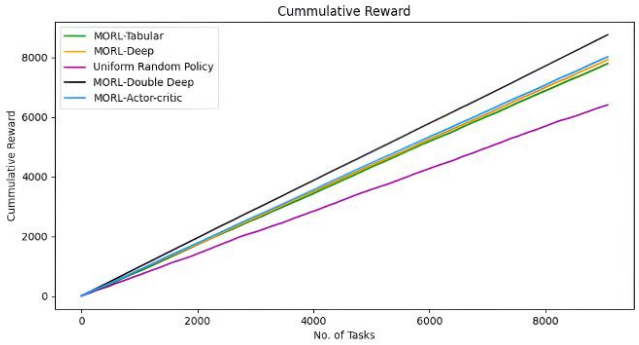

Fig. 2. Cumulative reward of different models.

C. Drop Rate

The crucial factor to adjudicate the performance is the total tasks dropped as a result of the decision made by the models. Each task has a timeout within which the task must be executed, otherwise, it is considered to be dropped. At each timestep we calculate the number of dropped tasks. This factor can be used to consider the task offloading problem as a minimization problem. Fig. 3 shows that uniform random policy drops a significant number of tasks. Deep MORL models, on the other hand, have the running task drop rate close to zero. The MORL actor critic model not only decreases the drop rate but also improves the stability of its output compared to both DQN models.

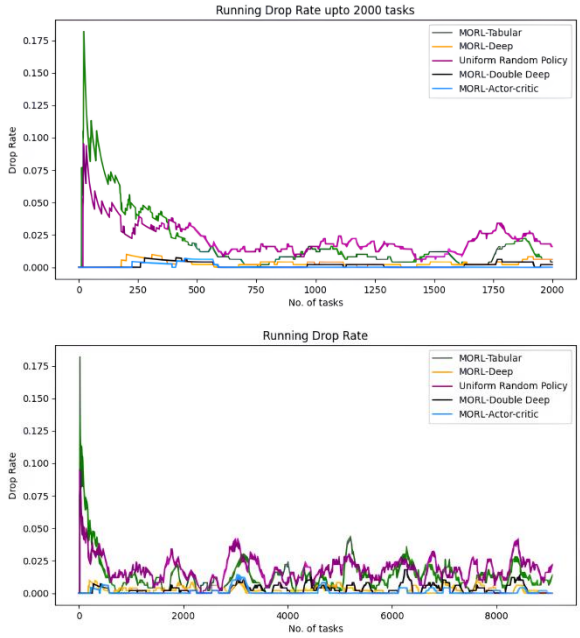

Fig. 3. Running drop rate of tasks.

D. Latency

The second most important factor to adjudicate the performance of our models is how quickly we are able to execute all the tasks. The agent gets a positive reward if a task is executed within the latency threshold or else it gets a negative reward. From Fig. 4, it can be seen that the MORL-DQN model performs considerably well compared to the tabular MORL model and uniform random policy. The MORL actor critic model estimates and optimizes the latency significantly compared to the other approaches.

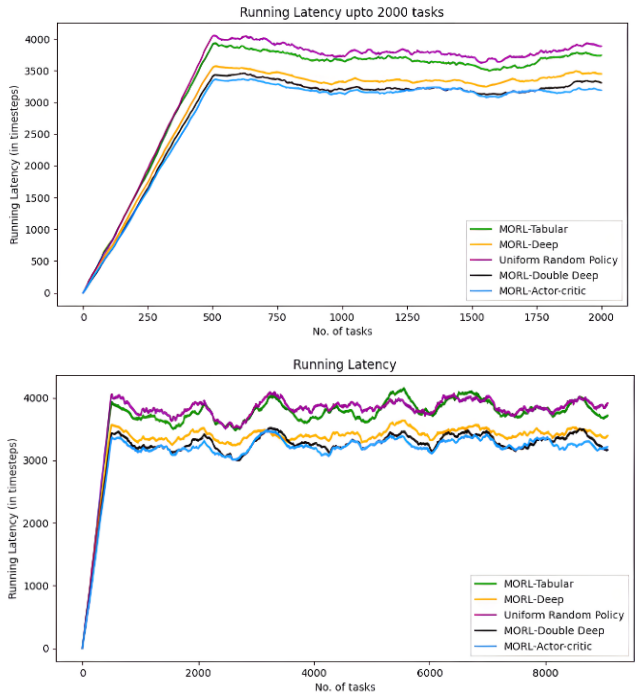

Fig. 4. Running latency of execution of tasks.

E. Energy

Another important factor to consider is the battery consumption to process the tasks at the edge device. This factor allows the model to take into consideration that we intend to minimize the battery utilization of the edge device. Hence the decision made by MORL is not just influenced by latency or drop rate but also by battery consumption. We have set a threshold for each device and if a task is going to get executed locally, we want those tasks to be executed within this threshold. In such cases, we provide the agent with a positive reward and if any task's execution crosses the threshold, we provide the agent with a negative reward.

From Fig. 5, it is observed that the Deep MORL model does considerably well compared to all the other models. The tabular MORL model performs quite well but doesn't provide the same stability that the deep MORL model gives. It is true that MORL actor-critic leads to a greater amount of energy consumption on average than other approaches, but it is due to the fact that more tasks are executed on the mobile device as compared to other algorithms to achieve better drop rate and latency. It shows a greater degree of stability compared and better overall performance on all objectives.

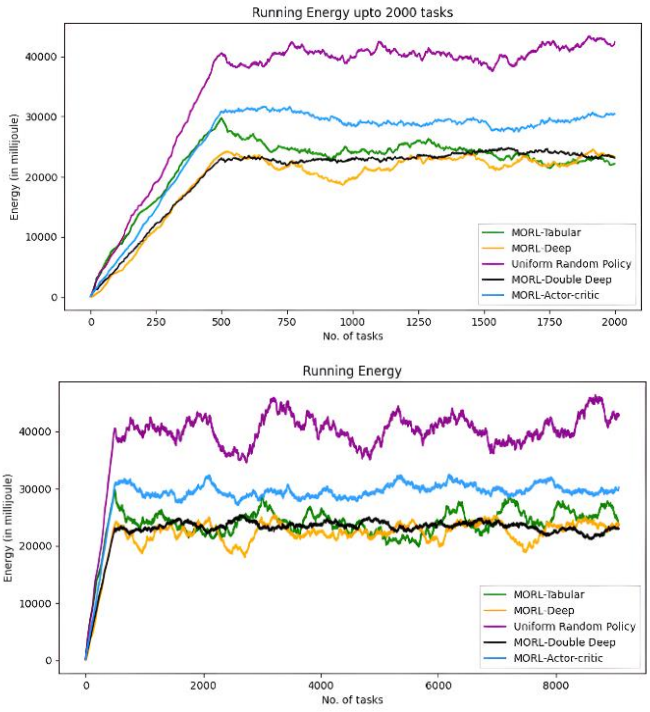

Fig. 5. Running energy consumed by the edge device.

F. Comparison

Table I provides a comparison of the results obtained when all algorithms were applied to 3000 episodes using 10 edge devices connected to a single MEC server. All edge devices generated a total of 9073 tasks.

G. Benefits of Multi-objective Optimization over Single-objective Optimization

Single objective approaches seem to outperform the multi objective ones on the objective which they have been trained to

optimize. However, a closer look into the performance of all objectives will reveal the obvious superiority of multi-objective reinforcement learning approaches. To better understand the aforementioned claim, we compare a SORL model [31] which optimizes energy consumption and the MORL actor-critic model.

TABLE I. RESULT EVALUATION TABLE

	Objectives			Decisions	
Model	Net Drop Rate	Mean Latency (in timesteps)	Mean Energy consumerd (in mJ)	Offload	Local
MORL-Tabular	0.0406	22.963	147.5110	5891	3182
MORL-DQN	0.0083	20.690	**136.1579**	6304	2769
MORL-DDQN	0.0081	19.853	140.6437	5187	3886
MORL-Actor Critic	**0.0034**	**19.62**	179.6439	4861	4212
Uniform Random Policy	0.0590	23.521	248.1667	4515	4558

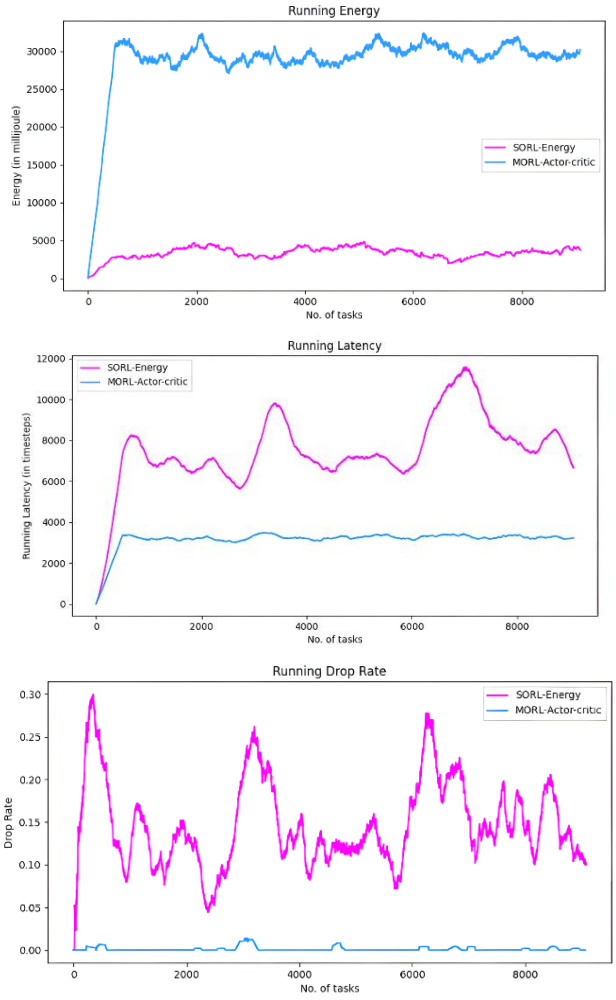

Fig. 6. Comparison of objectives' performance using single-objective and multi-objective optimization techniques.

Fig. 6 shows that the SORL model performs exceptionally well in terms of the objective that it optimizes, i.e, energy consumed. It offloads most of the tasks to the edge server to cut down on the energy consumption. However, this comes at a cost of drop rate and latency. The drop rate and latency achieved by the SORL model is unacceptable in real world applications, whereas the MORL actor-critic model optimizes all three objectives reasonably well which makes it more practical.

We conclude with the observation that the agent trained using the MORL actor critic method exhibited the best overall performance.

VII. CONCLUSION

Multi-access Edge Computing (MEC) tries to improve user experience and reduce energy consumption. It is a popular and emerging paradigm that takes the cloud closer to resource constrained mobile devices. A general scenario depicting the interaction between a few mobile devices and a MEC Server is simulated in which four multi-objective reinforcement learning algorithms are compared. From the results observed, we can conclude that multi-objective reinforcement learning based actor critic model outperforms other models in terms of both latency as well as task drop rate.

REFERENCES

[1] N. Hassan, K. -L. A. Yau and C. Wu, "Edge Computing in 5G: A Review," in IEEE Access, vol. 7, pp. 127276-127289, 2019, doi: 10.1109/ACCESS.2019.2938534.

[2] M. Tang and V. W. S. Wong, "Deep Reinforcement Learning for Task Offloading in Mobile Edge Computing Systems," in IEEE Transactions on Mobile Computing, vol. 21, no. 6, pp. 1985-1997, 1 June 2022, doi: 10.1109/TMC.2020.3036871.

[3] 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," in IEEE Access, vol. 8, pp. 54074-54084, 2020, doi: 10.1109/ACCESS.2020.2981434.

[4] J. Wang, J. Hu, G. Min, A. Y. Zomaya and N. Georgalas, "Fast Adaptive Task Offloading in Edge Computing Based on Meta Reinforcement Learning," in IEEE Transactions on Parallel and Distributed Systems, vol. 32, no. 1, pp. 242-253, 1 Jan. 2021, doi: 10.1109/TPDS.2020.3014896.

[5] J. Wang, J. Hu, G. Min, W. Zhan, A. Y. Zomaya and N. Georgalas, "Dependent Task Offloading for Edge Computing based on Deep Reinforcement Learning," in IEEE Transactions on Computers, vol. 71, no. 10, pp. 2449-2461, 1 Oct. 2022, doi: 10.1109/TC.2021.3131040.

[6] Huang, L., Feng, X., Qian, L., Wu, Y. (2018). Deep Reinforcement Learning-Based Task Offloading and Resource Allocation for Mobile Edge Computing. In: Meng, L., Zhang, Y. (eds) Machine Learning and Intelligent Communications. MLICOM 2018. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 251. Springer, Cham. https://doi.org/10.1007/978-3-030-00557-3_4

[7] P. Yan and S. Choudhury, "Optimizing Mobile Edge Computing Multi-Level Task Offloading via Deep Reinforcement Learning," ICC 2020 - 2020 IEEE International Conference on Communications (ICC), Dublin, Ireland, 2020, pp. 1-7, doi: 10.1109/ICC40277.2020.9149024.

[8] X. Liu, S. Jiang, and Y. Wu, "A Novel Deep Reinforcement Learning Approach for Task Offloading in MEC Systems," Applied Sciences, vol. 12, no. 21, p. 11260, Nov. 2022, doi: 10.3390/app122111260.

[9] Zhang, Z., Li, C., Peng, S. et al. A new task offloading algorithm in edge computing. J Wireless Com Network 2021, 17 (2021). https://doi.org/10.1186/s13638-021-01895-6

[10] Dai, Yu, et al. "Offloading in Mobile Edge Computing Based on Federated Reinforcement Learning." Wireless Communications and Mobile Computing 2022 (2022): 1-10.

[11] Xu, Yuanchao, Amal Feriani, and Ekram Hossain. "Decentralized multi-agent reinforcement learning for task offloading under uncertainty." arXiv preprint arXiv:2107.08114 (2021).

[12] Yamansavascilar, Baris, et al. "Deepedge: A deep reinforcement learning based task orchestrator for edge computing." IEEE Transactions on Network Science and Engineering 10.1 (2022): 538-552.

[13] Zhang, Xiangjun, et al. "An efficient computation offloading and resource allocation algorithm in RIS empowered MEC." Computer Communications 197 (2023): 113-123.

[14] Tu, Youpeng, et al. "Task offloading based on LSTM prediction and deep reinforcement learning for efficient edge computing in IoT." Future Internet 14.2 (2022): 30.

[15] Huang, Liang, Suzhi Bi, and Ying-Jun Angela Zhang. "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks." IEEE Transactions on Mobile Computing 19.11 (2019): 2581-2593.

[16] Feng, Mingjie, et al. "Task assignment in mobile edge computing networks: a deep reinforcement learning approach." Sensors and Systems for Space Applications XIV. Vol. 11755. SPIE, 2021.

[17] Ouyang, Yi. "Task offloading algorithm of vehicle edge computing environment based on Dueling-DQN." Journal of Physics: Conference Series. Vol. 1873. No. 1. IOP Publishing, 2021.

[18] Song, Fuhong, et al. "Evolutionary multi-objective reinforcement learning based trajectory control and task offloading in UAV-assisted mobile edge computing." IEEE Transactions on Mobile Computing (2022).

[19] Chen, Xianfu, et al. "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning." IEEE Internet of Things Journal 6.3 (2018): 4005-4018.

[20] Yang, Junyao, Yan Wang, and Zijian Li. "Inverse order based optimization method for task offloading and resource allocation in mobile edge computing." Applied Soft Computing 116 (2022): 108361.

[21] Wang, Ting, Xiong Luo, and Wenbing Zhao. "Improving the performance of tasks offloading for internet of vehicles via deep reinforcement learning methods." IET communications 16.10 (2022): 1230-1240.

[22] Zhang, Hongxia, et al. "Ultra-low latency multi-task offloading in mobile edge computing." IEEE Access 9 (2021): 32569-32581.

[23] Zhang, Xiaojie, Amitangshu Pal, and Saptarshi Debroy. "Deep reinforcement learning based energy-efficient task offloading for secondary mobile edge systems." 2020 IEEE 45th LCN Symposium on Emerging Topics in Networking (LCN Symposium). IEEE, 2020.

[24] Li, Mushu, et al. "Deep reinforcement learning for collaborative edge computing in vehicular networks." IEEE Transactions on Cognitive Communications and Networking 6.4 (2020): 1122-1135.

[25] Chen, Juan, et al. "Task offloading in hybrid-decision-based multi-cloud computing network: a cooperative multi-agent deep reinforcement learning." Journal of Cloud Computing 11.1 (2022): 1-17.

[26] Chen, Xing, and Guizhong Liu. "Federated deep reinforcement learning-based task offloading and resource allocation for smart cities in a mobile edge network." Sensors 22.13 (2022): 4738.

[27] Meng, Hao, Daichong Chao, and Qianying Guo. "Deep reinforcement learning based task offloading algorithm for mobile-edge computing systems." Proceedings of the 2019 4th International Conference on Mathematics and Artificial Intelligence. 2019.

[28] Wang, Kun, et al. "Task offloading strategy based on reinforcement learning computing in edge computing architecture of internet of vehicles." IEEE Access 8 (2020): 173779-173789.

[29] Song, Fuhong, et al. "Offloading dependent tasks in multi-access edge computing: A multi-objective reinforcement learning approach." Future Generation Computer Systems 128 (2022): 333-348.

[30] Chen, Yu, et al. "Multi-user edge-assisted video analytics task offloading game based on deep reinforcement learning." 2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS). IEEE, 2020.

[31] V. M, V. Khot, S. S. Pai, V. R. Rao and K. N, "Deep Reinforcement Learning for Task Offloading in a Multi-Access Edge Computing Environment," 2023 International Conference on Network, Multimedia and Information Technology (NMITCON), Bengaluru, India, 2023, pp. 1-6, doi: 10.1109/NMITCON58196.2023.10275998.