TPTO: A Transformer-PPO based Task Offloading Solution for Multi-Access Edge Computing

Niloofar Gholipour[1], Marcos D. Assuncao[1], Pranav Agarwal[1], Julien Gascon-Samson[1], and Rajkumar Buyya[2]

[1] École de Technologie Supérieure, University of Quebec, Canada
{niloofar.gholipour,pranav.agarwal}.1@ens.etsmtl.ca
{marcos.dias-de-assuncao,julien.gascon-samson}@etsmtl.ca
[2] University of Melbourne, Australia
rbuyya@unimelb.edu.au

Abstract. Emerging applications in healthcare, autonomous vehicles, and wearable assistance require interactive and low-latency data analysis services. Unfortunately, cloud-centric architectures cannot fulfill the low-latency demands of these applications, as user devices are often distant from cloud data centers. Multi-Access Edge Computing (MEC) aims to reduce the latency by enabling processing tasks to be offloaded to resources located at the network's edge. However, determining which tasks must be offloaded to edge servers to reduce the latency of application requests is not trivial, especially if the tasks present dependencies. This paper proposes a Deep Reinforcement Learning (DRL) approach called TPTO, which leverages Transformer Networks and Proximal Policy Optimization (PPO) to offload dependent tasks of IoT applications in MEC. We consider users with various preferences, where devices can offload computation to a MEC server via wireless channels. Performance evaluation results demonstrated that under fat application graphs, TPTO is more effective than state-of-the-art methods, such as HEFT, Greedy, and MRLCO, by reducing latency by 3.44%, 30.61%, and 19.17%, respectively. In addition, TPTO presents a training time approximately 2.5 times faster than an existing DRL approach.

Keywords: edge computing · reinforcement learning · transformers

1 Introduction

Multi-Access Edge Computing (MEC), by complementing the cloud, can enable an increasing range of IoT applications that produce vast amounts of time-sensitive data requiring prompt analysis, such as in autonomous driving, healthcare, online video processing, and wearable assistance [4, 28]. In autonomous driving, for instance, latency is a critical factor in ensuring the safety of passengers and pedestrians. A minor delay in processing sensor data or making control decisions can not only degrade the users' quality of experience but also result in accidents or compromised safety. MEC provides computing services (e.g., base stations, access points, and edge routers) that are closer to end-users, contributing to lower the latency of application requests, their energy consumption, and the amount of data transferred to the cloud for processing [12].

Reducing the latency of IoT application requests requires offloading data processing tasks to MEC servers, an activity that often poses significant challenges. Offloading tasks can free constrained resources of user devices, but on the other hand, transferring data between the user devices and remote MEC servers can impact the latency [23]. Moreover, according to research conducted by Alibaba, around 75% of real-world applications have interdependent tasks, commonly structured as a Directed Acyclic Graph (DAG), where the vertices represent data sources, data sinks, end-users, and operators, and the edges represent data streaming from one operator to another [8, 20]. Trying to devise efficient offloading decisions for these applications can often result in NP-hard problems, which require sophisticated algorithms to address them effectively.

Several heuristics, meta-heuristics, and model-based approaches exist for offloading decisions in MEC, most of which are unsuitable to stochastic environments where resource availability is continuously evolving [7, 25]. MEC is also stochastic when considering the number of applications, the number of tasks in an application, their arrival rate, their dependencies, and their resource requirements [3]. DRL with policy optimization is a promising approach to address these challenges and design agents interacting with the environment to learn an optimal policy, enhanced over time through trial and error [2]. DRL agents can learn a stochastic policy without having preliminary information about the environment, making them suitable for stochastic and complex systems like MEC [6, 7, 9, 25, 29].

We formulate the task offloading decision as a binary optimization problem and propose a solution, Transformer-PPO based Task-Offloading (TPTO), which utilizes a combination of Markov Decision Process (MDP), Reinforcement Learning (RL), and transformers [24]. While RL provides a learning mechanism to optimize offloading decisions over time, the Transformer model enhances the solution's performance by enabling it to learn from previous tasks and apply the knowledge to future offloading decisions. TPTO trains transformers for various MEC tasks and quickly adapts to new ones with less training time and shorter latency. Our proposed approach features Bidirectional Encoder Representations from Transformers (BERT) architecture incorporating multi-head attention, layer normalization, and feed-forward fully connected layers. The predictions made by the transformer, provided to a Softmax function, act as the actions that guide the training process in collaboration with the PPO algorithm. This results in a more efficient and effective solution. To our knowledge, it is the first work to apply BERT for offloading decisions in MEC environments. To validate our approach, we carry out simulations using synthetic DAGs that reflect real-world applications and network topologies with multiple wireless transmission rates. Our experimental results demonstrate our approach's effectiveness in optimizing the offloading problem.

The main contributions of this paper are: A novel latency-aware task offloading approach, TPTO, that leverages the transformer model and that quickly adapts to stochastic MEC environments; and a new policy that jointly uses transformers and PPO to determine the best action for task offloading – i.e., offloaded to the MEC server or processed locally to minimize end-to-end latency.

The paper is structured as follows: Section 2 describes the problem and presents a formulation. Section 3 presents TPTO, whereas Section 4 analyzes and compares its efficiency against state-of-the-art techniques. Section 5 reviews related work, and Section 6 concludes the paper and discusses future work.

2 Problem Description and Formulation

A real-time object detection system presents a typical example of an application that requires computation offloading to MEC. In this scenario, a user device often captures a video stream from a camera and aims to detect and recognize objects from the video feed in real time. This scenario reflects applications in facial recognition [25], pest bird detection systems [14], and the detection of traffic signs [17]. The user device can carry out data pre-processing and execute a lightweight object detection model locally, identifying some features, but the type of computations it can perform will largely depend on the system status, the resources available, and their constraints, or offload them to a MEC server.

An application A is a DAG $G = (V, E)$ where each vertex $v_1 \in V$ represents a task and each directed edge $e(v_i, v_j) \in E$ is a dependence constraint in which task v_i must complete before task v_j starts. *Entry tasks* are tasks without parent tasks, whereas *exit tasks* or *sinks* are tasks without children. The computation of task v_i corresponds to the number of CPU cycles needed for its execution, given by c_i. Moreover, we define as data_i^{up} and data_i^{do} the amount of data required to upload and download, respectively, task v_i to/from a MEC server.

The computing capacity of a resource m_j (user device or MEC server), denoted as cs_j, reflects its clock speed times the number of cores available in the system. Similar to previous work [21,25], a user device is associated with a dedicated Virtual Machine (VM) or container providing the computing and network resources that an application requires. The VMs share the computing resources equally, such that the capacity of a VM on a MEC server m_j is $cs_{vm} = cs_j/k$, where k is the number of users in m_j. This approach ensures that each VM receives a fair and proportional share of the computing capacity, enabling efficient utilization of the resources in the MEC environment.

The user device can execute a task locally or offload its computation to a MEC server via *wireless channels*. A wireless channel's uplink and downlink transmission rates are r_{up} and r_{do}. Three steps are required to offload a task v_i to a MEC server m_j: first, the user device sends the task to the MEC server via a wireless channel. Second, the MEC server executes the task. Finally, the MEC server sends the execution results back to the user's device. The overall task latency depends on the task requirements and system status. Hence, the time in uploading task v_i to MEC server m_j (t_i^{up}) is the time to execute the task on the MEC server (t_i^{ex}) and the time to download the data back to the user device (t_i^{do}), and can be computed as:

$$t_i^{up} = data_i^{up}/r_{up}, \qquad t_i^{ex} = c_i/cs_{vm}, \qquad t_i^{do} = data_i^{do}/r_{do} \tag{1}$$

When offloaded to a MEC server, the overall end-to-end latency of task v_i represents the sum of the above times in (1). On the other hand, if a user device executes a task v_i locally, hence using resource m_k (the user device), its latency consists only of the task execution time (*i.e.* $t_i^{ex} = c_i/cs_k$). In addition, for a task v_i scheduled for execution, we establish four task finish times, namely FT_i^{ud}, FT_i^{up}, FT_i^{mec}, and FT_i^{do}, to denote the task finish time on the user device, on the upload link, on the MEC server and the download link. If the task v_i runs locally on the user device, then $FT_i^{up} = FT_i^{mec} = FT_i^{do} = 0$. Otherwise, $FT_i^{ud} = 0$ if v_i is offloaded to a MEC server.

Before scheduling a task v_i, all preceding tasks (*i.e.*, its parent tasks) must already have been scheduled. In this way, we denote RT_i^{ud}, RT_i^{up}, RT_i^{mec}, and RT_i^{do} as the ready time, the earliest time that task v_i can be executed on a resource (user device, upload link, MEC server, download link) so that the precedence constraints are maintained. Therefore, for task v_i, scheduled on the user device, we can calculate its ready time as:

$$RT_i^{ud} = \max_{j \in parent(v_i)} max\left\{FT_j^{ud}, FT_j^{do}\right\} \tag{2}$$

where $\boldsymbol{parent}(v_i)$ is the set of parent tasks immediately before task v_i. RT_i^{ud} is the earliest time at which all the tasks preceding v_i will have completed and produced the results that v_i requires. When a task v_j preceding v_i is scheduled locally, then $max\{FT_j^{ud}, FT_j^{do}\} = FT_j^{ud}$; otherwise, when offloaded to the MEC server, $max\{FT_j^{ud}, FT_j^{do}\} = FT_j^{do}$. Task v_i can only start executing once v_j has freed the wireless download channel.

On the other hand, if that task v_i is to be offloaded to the MEC server, then its ready time on the upload channel (RT_i^{up}) is given by:

$$RT_i^{up} = \max_{j \in parent(v_i)} max\left\{FT_j^{ud}, FT_j^{up}\right\} \tag{3}$$

where RT_i^{up} is the earliest time when v_i can use the upload channel while meeting precedence constraints. When a task v_j preceding v_i is scheduled locally, then $max\{FT_j^{ud}, FT_j^{up}\} = FT_j^{ud}$; otherwise, when offloaded to the MEC server, then $max\{FT_j^{ud}, FT_j^{up}\} = FT_j^{up}$. Task v_i can only start execution once v_j has freed the wireless download channel.

The ready time of a task v_i on a MEC server is:

$$RT_i^{mec} = max\left\{FT_i^{up}, \max_{j \in parent(v_i)} FT_j^{mec}\right\} \tag{4}$$

where RT_i^{mec} is the earliest time v_i can execute on the MEC server while respecting precedence constraints. If a task v_j preceding v_i is scheduled locally, then $FT_j^{mec} = 0$. Hence, $\max_{j \in parent(v_i)} FT_j^{mec}$ is the earliest time when all offloaded tasks preceding v_i have finished execution.

The earliest time for sending the results of task v_i back to the user device is:

$$RT_i^{do} = FT_i^{mec} \tag{5}$$

The offloading goal is to compute an offloading plan $O_n = (o_1, o_2, \ldots, o_n)$ that minimizes the latency of an application DAG $G(V, E)$, where $n = |V|$ and o_i denotes the offloading decision for task v_i. Before offloading, tasks are sorted by priority, as discussed later, so that O_{n-1}, for example, represents the partial offloading plan comprising all tasks from v_1 to v_{n-1}.

The optimization goal is, hence, to minimize the overall *Application Latency*:

$$AL_{O_n} = max \left[max_{v_e \in \mathcal{E}} (FT_e^{ud}, FT_e^{do}) \right] \tag{6}$$

where $\mathcal{E}$ is the set of exit tasks (*i.e.* tasks with no children). The equation considers the maximum task latency within a DAG to compute the overall application latency. This maximum time represents the duration of the critical path of the DAG, which is the longest path from a start task to any of the exit tasks.

Table 1. Notation used in this paper.

Notation	Description
$G(V, E)$	Application DAG where V is the set of tasks and E the task precedence constraints
$v_i \in V$	Computing task v_i
$e(v_j, v_i) \in E$	Precedence constraint, task v_j must execute before v_i can start
$data_i^{up}$, $data_i^{do}$	Number of bytes to upload/download to/from a MEC server when offloading task v_i
r_{up}, r_{do}	Transmission rates of wireless uplink and downlink channels
cs_k, cs_{vm}	Computing capacity of resource m_k, and of a VM
t_i^{up}, t_i^{ex}, t_i^{do}	Time required for uploading, executing and downloading task v_i to MEC server m_k
FT_i^{ud}, FT_i^{up}, FT_i^{mec}, FT_i^{do}	Finish time of task v_i on user device, uplink channel, MEC server, and downlink channel
RT_i^{ud}, RT_i^{up}, RT_i^{mec}, RT_i^{do}	Earliest time when task v_i can use the user device, uplink channel, MEC server, and downlink channel)

3 Transformer-Based Task Offloading Solution

This section presents our transformer-PPO based task offloading solution.

3.1 TPTO: Transformer-PPO based Task Offloading

In standard RL settings, an agent interacts with an environment, trying to learn a policy to take actions that maximize the accumulated reward. An MDP, commonly used to represent RL problems [22], consists of a tuple (S, A, P, R, γ), where S represents the set of possible states; A represents the action space; $P(s'|s,a)$ denotes the probability of transitioning to state s' when taking action a under the current state s; $R(s,a,s')$ represents the immediate reward received when transitioning from s to s' by taking action a; γ is a discount factor. The goal is to find a policy $\pi(s)$ that maximizes the expected cumulative reward over time. A policy network $\pi(a|s,\theta)$ takes the state s as input and outputs a probability distribution over the actions a, where θ represents the neural network parameters. Training the policy network involves finding the optimal parameters θ^* that maximize the expected cumulative reward, a process typically performed using policy gradient algorithms that seek to maximize the expected return. TPTO optimizes the policy network parameters using PPO [19]. During training, PPO uses a batch of sampled trajectories to update the network weights. The following describes the main elements of our MDP:

State S: A state comprises the task profile (CPU cycle requirements and data sizes), the DAG topologies, the wireless transmission rates, and the status of MEC resources. The status of a MEC resource depends on the offloading decisions for tasks preceding v_i. Hence, we can express the state combining the encoded DAG and the partial offloading plan as:

$$S = \{s_i | s_i = (G(V,E), O_i)\} \tag{7}$$

where $i \in [1, |V|]$, $G(V,E)$ represents the sequence of embedding tasks and O_i is the partial offloading plan of task v_i. We use the approach outlined in [25] to convert a DAG into a sequence of embedding tasks. First, we assign a priority to each task that reflects the rank in the DAG and sort them in ascending order. By representing each task as an embedding, we can capture the relationships between tasks, including parent-child dependencies, used to optimize task scheduling. To create a task embedding, we use three vectors. The first vector embeds the current task index and the normalized task profile. The second vector contains the indices of the immediate parent tasks, and the third vector contains the indices of the immediate child tasks. If the number of parent or child tasks is less than the length of the task vector, we pad the vector with -1. The size of the parent/child task index vectors is limited to the length of the task vector.

Action A: As the scheduling for each task is a binary choice, executing the task either on the user device or on a MEC server, the action space is $A := 0, 1$, where 0 represents execution on the user device and, 1 represents offloading.

Reward function R: The objective is to minimize the total application latency, defined in Equation 6. Hence, the reward function estimates the negative increase in latency resulting from an offloading decision for a particular task: $\varDelta AL_{O_i} = AL_{O_i} - AL_{O_{i-1}}$, where AL_{O_i} represents the total latency when taking a given action for task v_i and $AL_{O_{i-1}}$ represents the total latency the partial offloading plan for the previous task.

Assume that $\pi(a_i|G(V,E), O_{i-1})$ represents the likelihood of the offloading plan O_{i-1} given the graph $G(V,E)$, we can compute $\pi(O_n|G(V,E))$ by using the chain rule of probability on each $\pi(a_i|O_{i-1}, G(V,E))$ as follows:

$$\pi(O_n|G(V,E)) = \prod_{i=1}^{n} \pi(a_i|O_{i-1}, G(V,E)) \tag{8}$$

We use transformers to implement the policy. Transformers use an encoder-decoder architecture to overcome various challenges of Recurrent Neural Networks (RNNs). Generally, the encoder comprises embedding, multi-head attention, residual connection and normalization, feed-forward networks, and softmax. They primarily differ from prior architectures by including a self-attention mechanism to extract data dependency [24]. In TPTO, a transformer takes as input the task embeddings of the sequence $(v_1, v_2, ..., v_n)$ of a DAG and generates a new representation of the input sequence processed through a stack of Transformer layers. Based on the Transformer's output, the Actor generates corresponding offloading decisions for each task $(o_1, o_2, ..., o_n)$. The critic computes the value function for each task. Separate, fully connected layers generate these outputs.

3.2 Implementing TPTO

As Figure 1 outlines, TPTO employs the transformer model and PPO to update the policy network. First, the transformer receives an observation of the environment and produces two results: the policy logits and the value function. The policy logits are passed through a softmax function to obtain a proper probability distribution of the available actions. Next, the actor network takes the transformer's output and produces the final policy, which provides a probability distribution for the available actions. Finally, the critic network takes the transformer's output and generates the estimated value of the current state. The advantage function captures the difference between the actual and estimated return and the estimated value of the current state.

We use PPO as the policy optimization method. For a given learning task $\mathcal{T}$, PPO creates trajectories using a sample policy $\pi_{\theta_{sam}}$ and updates the target policy π_θ over multiple epochs, where θ and θ_{sam} are the parameters of the target and sample policies, respectively. At the initial epoch, $\theta = \theta_{sam}$. Then the probability ratio $r_t(\theta)$ at a time step t is:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{sam}}(a_t|s_t)} \tag{9}$$

where $s_t = G(V,E), O_t$. To update the actor's policy, PPO uses a clipped surrogate objective to avoid extensive policy updates:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right] \tag{10}$$

where $\hat{A}_t$ is the advantage function at time step t, and $\hat{\mathbb{E}}$ is the average expectation over a set of samples in an algorithm that alternates between sampling and optimization [19]. As the policy and value functions share most of their parameters, facilitating mutual training, we also employ the entropy coefficient to

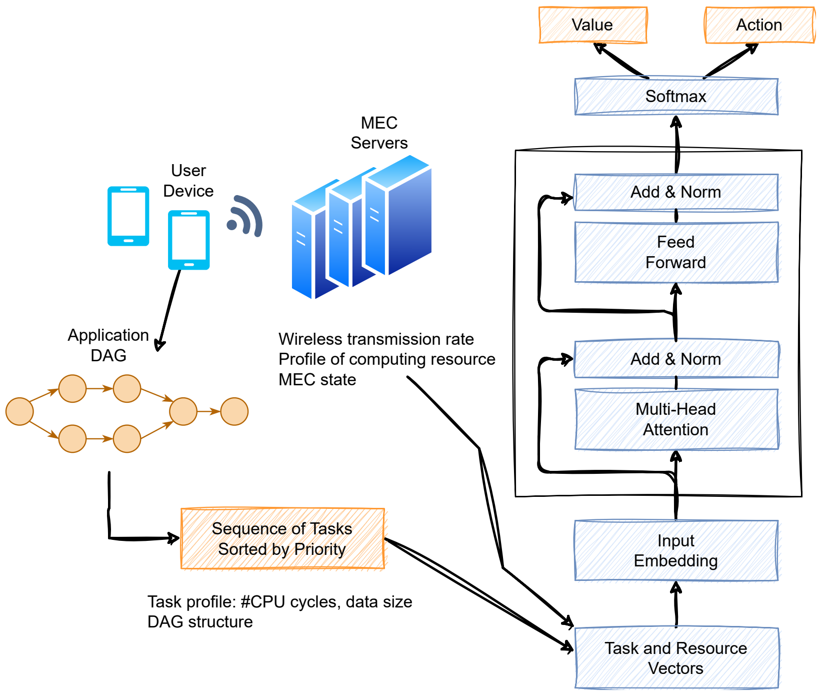


Fig. 1. Overview of TPTO.

compute the entropy bonus, added to the policy loss, to encourage exploration in the policy space. The combined objective is, therefore:

$$L^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t \left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t) \right] \tag{11}$$

where c_1 and c_2 are coefficients, $S[\pi_\theta](s_t)$ represents the entropy bonus, and $L_t^{VF}(\theta)$ is the squared-error loss: $(V_\theta(s_t) - V_t^{targ})^2$, where V is a state-value function.

The advantage function at time step t, denoted by $\hat{A}_t$, is calculated using General Advantage Estimator (GAE) [18]. GAE is a specific type of advantage function estimated as follows:

$$\hat{A}_t = \sum_{l=0}^{n-t+1} (\gamma\lambda)^k \left[r_t + \gamma V(s_{t+k+1}) - V(s_{t+k}) \right] \tag{12}$$

where λ is in the interval $(0, 1)$ and determines the equation's balance between bias and variance. We can then use gradient ascent to maximize $L^{CLIP+VF+S}(\theta)$.

Algorithm 1 Transformer-PPO based task offloading

```
Require: Task distribution $r(\mathcal{T})$, learning rate $\alpha$
Ensure: Updated policy parameters $\theta$
 1: Randomly initialize the parameters of the policy, $\theta$;
 2: for iterations $k \in \{1, 2, \ldots, K\}$ do
 3:     Sample $n$ learning tasks $\{\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_n\}$ from $r(\mathcal{T})$;
 4:     for each task $\mathcal{T}_i$ do
 5:         Initialize $\theta_{\text{sam}} \leftarrow \theta$
 6:         Sample trajectory set $S = (\tau_0, \tau_1, \ldots, \tau_n)$ from $\mathcal{T}_i$ using policy $\pi(\theta_{\text{sam}})$;
 7:         Calculate the advantage estimates $\hat{A}_1, \hat{A}_2, .., \hat{A}_T$;
 8:         Compute the policy gradient:
 9:         $L^{\text{TPTO}}_{\tau_{\text{sam}}}(\theta_{\text{sam}}) = \nabla_{\theta_{\text{sam}}} L^{\text{CLIP+VF+S}}(\theta_{\text{sam}})$
10:     end for
11:     Update the policy network parameters $\theta$ using Adagrad optimizer with gradi-
        ents computed by the TPTO loss function with trajectory set $S$ for $m$ steps:
12:     $\theta \leftarrow \theta + \alpha L^{\text{TPTO}}_{\tau_{\text{sam}}}(\theta_{\text{sam}})$
13: end for
```

Algorithm 1 outlines how TPTO performs the offloading decision and generates trajectories. First, the algorithm samples an n sized batch of learning tasks τ and performs the training loop for each sampled learning task. Following the completion of the training loop, the algorithm then updates the policy parameters θ using gradient ascent $\theta \leftarrow \theta + \beta L^{TPTO}$ using Adam optimizer [11], where β is the learning rate of training loop.

4 Performance Evaluation

This section first outlines the experimental setup and the baseline algorithms. Then it presents performance evaluation results.

4.1 Experimental Setup

We use simulation to assess TPTO's performance as it provides a controllable and repeatable environment. The simulation environment is similar to that described by Wang *et al.* [25]. We consider a cellular network whose data transmission rate varies based on the user devices' position. Also, a user device's CPU clock speed is $1GHz$, denoted by f_1. In contrast, each virtual machine in a MEC host has four cores, each core running at $2.5GHz$, represented by f_s. Consequently, offloaded tasks can simultaneously use all cores, resulting in a combined CPU clock speed of $10GHz$ for each VM.

We consider latency under multiple scenarios to evaluate TPTO's efficiency comprehensively in dynamic environments. We use a synthetic DAG generator tool[3] to generate heterogeneous DAGs representing various real-world applications with distinctive structures and task profiles. The generator receives four

[3] `https://github.com/frs69wq/daggen`

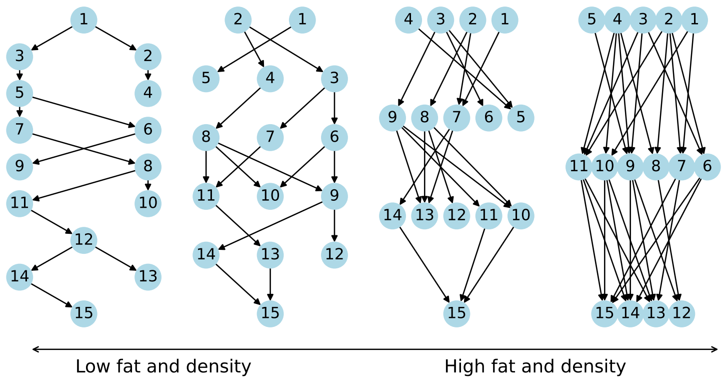


Fig. 2. Examples of produced DAGs.

parameters: n, fat, $density$, and ccr. The n represents the number of tasks; fat determines the DAG's width and height; $density$ sets the number of edges between two levels of the DAG; and computation to communication ratio, ccr, specifies the ratio between tasks' communication and computation cost.

To model the mobile network users' diverse preferences, we generated 25 DAG datasets, each consisting of 100 DAGs with various fat and densities, key parameters impacting the DAG topology. Each DAG has 20 tasks, and we pick the fat and density values randomly from $\{0.4, 0.5, 0.6, 0.7, 0.8\}$. These DAGs emulate a variety of user preferences under different transmission data speeds. We randomly select 22 DAG sets as training datasets and the remaining three as unseen testing datasets with different DAG topologies. Figure 2 illustrates DAGs generated by the synthetic DAG generator, varying fat and density values.

Table 2. TPTO's hyperparameters.

Hyperparameter	Value
Number of Layers	3
Num Attention Head	8
Dimension of Key Vector	1024
Dimension of Value Vector	1024
Dimension of FF network	512
Hidden Size	512
Dropout Rate	0.4
Policy Learning Rate	0.1
Valuefunc Learning Rate	0.01
Batch Size	100
Clip ratio	0.2
Activation Function	Relu
Optimization Method	Adagrad
Discount Factor	0.99
Entropy coefficient	0.5

TPTO is implemented using Tensorflow, with 3 layers of transformer encoders having 128 hidden units per layer and layer normalization included. Table 2 summarizes the hyperparameters for training TPTO. To ensure the robustness of the TPTO policy, we trained it using a range of transmission rates between 4Mbps to 22Mbps, with a step size of 3Mbps. To evaluate its performance on previously unseen transmission rates and topologies, we tested the trained policy on data rates of 5.5Mbps, 8.5Mbps, and 11.5Mbps, not seen during training, following a similar methodology as in [25] with sampling 20 trajectories for a DAG on the dataset. In addition, as we aim to assess how TPTO performs in different dynamic scenarios, the task data size varies from $5KB$ to $50KB$, while the CPU cycle requirements range from 10^7 to 10^8 cycles per task, as reported in [5]. Furthermore, the length of the parent/child task indices vector is 12. By testing TPTO's performance on these diverse sets of DAGs, we aim to gain insights into its ability to effectively provision network resources and meet the varying needs of mobile users.

4.2 Baseline Algorithms

We assess TPTO's performance against three state-of-the-art algorithms:

MRLCO: this algorithm, proposed by Wang *et al.* [25], integrates meta reinforcement learning and a Seq2Seq neural network. The approach focuses on modeling task offloading using meta-reinforcement learning and an offloading policy based on a custom Seq2Seq neural network.

HEFT based: this algorithm, based on the work by Lin *et al.* [13], involves prioritizing tasks using the HEFT method and scheduling each task according to its earliest estimated finish time.

Greedy: a greedy approach considers the estimated finish time of each task to decide whether to assign a task to the user device or a MEC server.

4.3 Result Analysis

Figures 3(a) and 3(b) depict the average latency of simulation results during training for TPTO and MRLCO. The results demonstrate that TPTO converges faster than MRLCO while being more stable and general, mainly due to TPTO's ability to effectively capture the diverse preferences of mobile users through its training on a wide range of network topologies and transmission rates. Figure 3(c) and 3(d) show the performance of HEFT and Greedy algorithms.

Table 3 summarizes the average latency of TPTO and the baseline algorithms. TPTO outperforms heuristic and meta-learning algorithms for the various wireless transmission rates. Overall, the Greedy algorithm has the highest latency, while TPTO achieves lower latency under various network conditions, indicating its effectiveness in provisioning network resources to meet the needs of mobile users. Moreover, distinct topologies reflect the diverse preferences of user requests. Increasing the transmission rate can further reduce latency as offloaded tasks traverse the wireless channels faster. Overall, the results show that TPTO is a promising solution for optimizing network performance and enhancing user experience in mobile networks. The results show that the TPTO achieves a training time 2.5 faster than MRLCO. The transformer architecture of TPTO is mainly responsible for this training time difference. Transformers are known for their parallelizability and efficient utilization of self-attention mechanisms, which can exploit the parallel processing capabilities of modern hardware architectures,

Table 3. Average latency (ms) on multiple testing datasets.

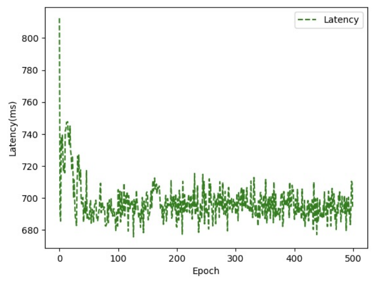
Topology Sets	**Algorithm**	**Wireless Transmission Rate**	
		$r_{up} = r_{do} = 8.5$**Mbps**	$r_{up} = r_{do} = 11.5$**Mbps**
1	**HEFT**	1064	835
	Greedy	1033	837
	MRLCO	846	760
	TPTO	741	581
2	**HEFT**	1157	849
	Greedy	1462	952
	MRLCO	989	869
	TPTO	1022	811
3	**HEFT**	1521	943
	Greedy	1009	822
	MRLCO	894	810
	TPTO	900	719

resulting in a faster training process. These results underscore the potential benefits of employing Transformer-based models for optimizing offloading decisions in the MEC environment.

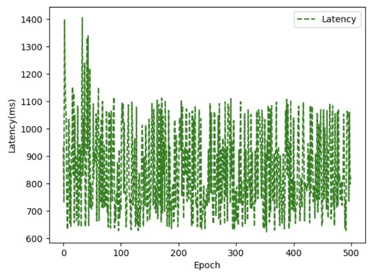
5 Related Work

This section reviews selected related work on task offloading in MEC.

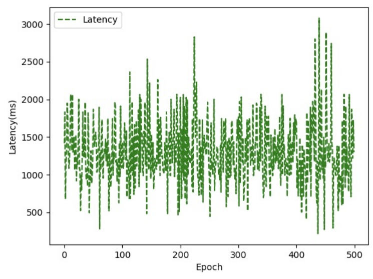
Machine-Learning Offloading Approaches: Qu *et al.* present a framework for IoT devices to offload computing tasks to Edge servers [16]. The work uses deep meta-reinforcement learning to minimize energy consumption, task computation, and transmission delays by dividing applications into sequential workflows. The proposed framework, called Deep Meta Reinforcement learning based Offloading (DMRO), includes an inner and outer loop. The former relies on Q-learning, whereas the latter employs a meta-algorithm to learn the initial parameters and adapt to changing environments, quickly converging to optimal offloading solutions. The work of Huang *et al.* [10] proposes MELO, a Meta-Learning-based computation Offloading algorithm for dynamic computation tasks in MEC. The system consists of one edge server and N wireless devices, each with a prioritized task to execute. They applied binary offloading in which the tasks run locally on a device or the edge server. The approach focuses on minimizing latency, communication, and computation delay. Yang *et al.* [27] tackle joint offloading optimization and bandwidth allocation, modeled as a mixed-integer programming (MIP) problem. The work proposes the Deep Supervised Learning-based computational Offloading (DSLO) algorithm that considers task delay and energy consumption. Furthermore, the authors enhance the convergence speed of the algorithm by incorporating Batch Normalization (BN) into two classical neural network architectures, CNN and DNN.



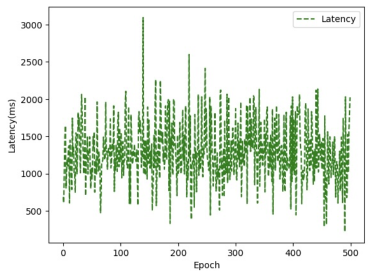
(a) TPTO



(b) MRLCO



(c) HEFT



(d) Greedy

Fig. 3. Influence of wireless transmission rate and network topology on latency.

Optimization-based Offloading Techniques: The work of Nguyen *et al.* [15] introduces a collaborative scheme for Unmanned Aerial Vehicless (UAVs) to share workloads. The authors consider the task topology, which involves breaking down a task into multiple sub-tasks with dependencies and the power consumption constraints of the UAVs in MEC. The authors use the discrete whale optimization algorithm and the SCS solver in the CVXPY library to solve the optimization problem, modeled as a mixed-integer, non-linear, and non-convex problem. Abbas *et al.* [1] present classical approaches for optimal task offloading in MEC environments. They use well-known meta-heuristics such as the ant colony optimization algorithm, whale optimization algorithm, and Grey wolf optimization algorithm, adapting these algorithms to their problem. The goal is to minimize the energy consumption of user devices and IoT and minimize response time for task computation at MEC servers. A search-based meta-heuristic model, introduced by Xu *et al.* [26], also handles task offloading and time allocation in MEC. Considering computation rate and task execution latency, they formulated the problem as a Mixed Integer Programming (MIP) and divided it into sub-problems: offloading decision and resource allocation. They proposed an "order-preserving policy generation method", which works well in large networks. They also utilized a one-dimensional bisection search over the variable associated with allocation time constraints.

6 Conclusions and Future Work

This work proposed a distributed DRL-based approach called TPTO for optimizing offloading decisions in MEC. By leveraging transformers, TPTO seeks to minimize the latency associated with offloading tasks for DAG-structured user applications. We first introduced a latency model that optimizes the task execution time, communication, and offloading in a MEC environment. This model serves as the basis for the decision-making process in TPTO. Then, experimental results demonstrated the effectiveness of TPTO under various network conditions and topologies. TPTO presents superior performance compared to three baseline algorithms: MRLCO, HEFT, and Greedy. In addition, TPTO consistently achieved the lowest latency, showcasing its ability to make efficient offloading decisions. These findings highlight the potential of utilizing transformer-based DRL approaches, particularly TPTO, in real-world MEC.

Future work will evaluate the scalability of TPTO to handle large-scale MEC environments with many user devices and more complex task dependencies. We will also consider multiple optimization criteria, including energy consumption, execution cost, and latency, to enhance the decision-making process.

References

1. Abbas, A., Raza, A., Aadil, F., Maqsood, M.: Meta-heuristic-based offloading task optimization in mobile edge computing. Int. Journal of Distributed Sensor Networks **17**(6) (2021)
2. Arulkumaran, K., Deisenroth, M.P., Brundage, M., Bharath, A.A.: A brief survey of deep reinforcement learning. arXiv preprint arXiv:1708.05866 (2017)
3. Cao, B., Zhang, L., Li, Y., Feng, D., Cao, W.: Intelligent offloading in multi-access edge computing: A state-of-the-art review and framework. IEEE Communications Magazine **57**(3), 56–62 (2019)
4. Chen, M., Wang, T., Zhang, S., Liu, A.: Deep reinforcement learning for computation offloading in mobile edge computing environment. Computer Communications **175**, 1–12 (2021)
5. Dinh, T.Q., Tang, J., La, Q.D., Quek, T.Q.: Offloading in mobile edge computing: Task allocation and computational frequency scaling. IEEE Transactions on Communications **65**(8), 3571–3584 (2017)
6. Faraji-Mehmandar, M., Jabbehdari, S., Javadi, H.H.S.: A self-learning approach for proactive resource and service provisioning in fog environment. The Journal of Supercomputing pp. 1–30 (2022)
7. Goudarzi, M., Palaniswami, M.S., Buyya, R.: A distributed deep reinforcement learning technique for application placement in edge and fog computing environments. IEEE Transactions on Mobile Computing (2021)
8. Goudarzi, M., Wu, H., Palaniswami, M., Buyya, R.: An application placement technique for concurrent iot applications in edge and fog computing environments. IEEE Transactions on Mobile Computing **20**(4), 1298–1311 (2020)
9. Hashem, W., Attia, R., Nashaat, H., Rizk, R.: Advanced deep reinforcement learning protocol to improve task offloading for edge and cloud computing. In: Int. Conf. on Advanced Machine Learning Technologies and Applications. pp. 615–628 (2022)

10. Huang, L., Zhang, L., Yang, S., Qian, L.P., Wu, Y.: Meta-learning based dynamic computation task offloading for mobile edge computing networks. IEEE Communications Letters **25**(5), 1568–1572 (2020)
11. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
12. Kirkpatrick, K.: Software-defined networking. CACM **56**(9), 16–19 (2013)
13. Lin, X., Wang, Y., Xie, Q., Pedram, M.: Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment. IEEE Transactions on Services Computing **8**(2), 175–186 (2014)
14. Mahmud, R., Toosi, A.N.: Con-pi: A distributed container-based edge and fog computing framework. IEEE Internet of Things Journal **9**(6), 4125–4138 (2022)
15. Nguyen, L.X., Tun, Y.K., Dang, T.N., Park, Y.M., Han, Z., Hong, C.S.: Dependency tasks offloading and communication resource allocation in collaborative uavs networks: A meta-heuristic approach. IEEE Internet of Things Journal (2023)
16. Qu, G., Wu, H., Li, R., Jiao, P.: Dmro: A deep meta reinforcement learning-based task offloading framework for edge-cloud computing. IEEE Transactions on Network and Service Management **18**(3), 3448–3459 (2021)
17. Saadna, Y., Behloul, A.: An overview of traffic sign detection and classification methods. Int. Journal of Multimedia Information Retrieval **6**, 193–210 (2017)
18. Schulman, J., Moritz, P., Levine, S., Jordan, M., Abbeel, P.: High-dimensional continuous control using generalized advantage estimation. arXiv preprint arXiv:1506.02438 (2015)
19. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
20. Souza, F.R.d., Silva Veith, A.D., Dias de Assunção, M., Caron, E.: Scalable joint optimization of placement and parallelism of data stream processing applications on cloud-edge infrastructure. In: ICSOC. pp. 149–164 (2020)
21. Sun, X., Ansari, N.: Adaptive avatar handoff in the cloudlet network. IEEE Transactions on Cloud Computing **7**(3), 664–676 (2019)
22. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Intr. MIT press (2018)
23. Tong, Z., Deng, X., Ye, F., Basodi, S., Xiao, X., Pan, Y.: Adaptive computation offloading and resource allocation strategy in a mobile edge computing environment. Information Sciences **537**, 116–131 (2020)
24. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)
25. Wang, J., Hu, J., Min, G., Zomaya, A.Y., Georgalas, N.: Fast adaptive task offloading in edge computing based on meta reinforcement learning. IEEE Transactions on Parallel and Distributed Systems **32**(1), 242–253 (2020)
26. Xu, Y., Wang, Y., Yang, J.: Meta-heuristic search based model for task offloading and time allocation in mobile edge computing. In: Proc. the 6th International Conference on Computing and Artificial Intelligence. pp. 117–121 (2020)
27. Yang, S., Lee, G., Huang, L.: Deep learning-based dynamic computation task offloading for mobile edge computing networks. Sensors **22**(11), 4088 (2022)
28. Yousefpour, A., Fung, C., Nguyen, T., Kadiyala, K., Jalali, F., Niakanlahiji, A., Kong, J., Jue, J.P.: All one needs to know about fog computing and related edge computing paradigms: A complete survey. Journal of Systems Architecture **98**, 289–330 (2019)
29. Zheng, T., Wan, J., Zhang, J., Jiang, C.: Deep reinforcement learning-based workload scheduling for edge computing. Journal of Cloud Computing **11**(1), 1–13 (2022)