Multi-Objective Robust Workflow Offloading in Edge-to-Cloud Continuum

Hongyun Liu* Ruyue Xin* Peng Chen‡ Zhiming Zhao*†
*Multiscale Networked Systems (MNS), University of Amsterdam, Amsterdam, the Netherlands
†LifeWatch ERIC Virtual Lab and Innovation Center, Amsterdam, the Nederlands
‡School of Computer and Software Engineering, Xihua University, Chengdu, China
Email: {h.liu|r.xin|z.zhao}@uva.nl, chenpeng@mail.xhu.edu.cn

***Abstract*—Workflow offloading in the edge-to-cloud continuum copes with an extended calculation network among edge devices and cloud platforms. With the growing significance of edge and cloud technologies, workflow offloading among these environments has been investigated in recent years. However, the dynamics of offloading optimization objectives, i.e., latency, resource utilization rate, and energy consumption among the edge and cloud sides, have hardly been researched. Consequently, the Quality of Service(QoS) and offloading performance also experience uncertain deviation. In this work, we propose a multi-objective robust offloading algorithm to address this issue, dealing with dynamics and multi-objective optimization. The workflow request model in this work is modeled as Directed Acyclic Graph(DAG). An LSTM-based sequence-to-sequence neural network learns the offloading policy. We then conduct comprehensive implementations to validate the robustness of our algorithm. As a result, our algorithm achieves better offloading performance regarding each objective and faster adaptation to newly changed environments than fine-tuned typical single-objective RL-based offloading methods.**

***Index Terms*—offloading, Meta-Learning, Reinforcement Learning, robustness, multi-objective learning, LSTM**

I. Introduction

The highly developing Edge-to-cloud continuum, e.g. Multi-access edge computing (MEC) [1] offers more intelligent and diverse application use cases in our life: smart cities, smart homes, E-Healthcare, smart transportation, smart factories. The amount of end-users and cloud platforms has grown enormously within the past decade. The corresponding applications and the calculation are also getting more and more complex. This growth brings massive network traffic and calculation workload among end-users and cloud platforms. In many scenarios, the end devices do not have enough calculation capability to execute the applications efficiently. To this end, workflow offloading is a typical approach to solving this issue by better-utilizing computation resources on the user side, edge, and cloud. Within the MEC environment empowered by developments of hardware and software, the offloading problem matters more than ever in many manners: energy consumption, resource utilization rate, latency and QoS [2].

With the large scale of the cloud platform, more objectives are taken into account to optimize offloading: resource utilization rate and energy consumption. Each of them affects the offloading performance. To this end, offloading turns into a multi-objective optimization problem. On the edge side, the execution takes more time with lower execution capability, while execution on the cloud side has higher calculation capability but takes more time to deliver tasks and receive results. Optimizing those factors simultaneously also turns the offloading problem into an NP-hard problem.

To cope with the offloading problem, lots of attention has been paid to Machine Learning-based approaches [3] [4]. Among them, Reinforcement-Learning(RL)-based approaches [5] [6] [4] also has been investigated a lot because it does not ask for data labeling. Many works have been done to investigate the offloading problem also in the context of heterogeneous environment [7] [8]. When the latency-critical tasks together with dependencies form the workflow, then the workflow becomes latency-critical as well [9] [10]. Consequently, offloading policies also need to take the latency required by those workflows into account [11] [12].

However, not many works addressed the robustness of the offloading policies from a multi-objective perspective. As a result, the offloading performance deviation leads to uncertain latency, resource utilization rate, and energy consumption, consequently influencing the QoS even violation of the Service Legal Agreement(SLA). In this work, we propose a Meta-Reinforcement-Learning-based multi-objective robust offloading algorithm(MRL-MO-RO) to address this issue. The main contributions of this paper include:

1) Multi-objective offloading optimization: we propose a multi-objective learning framework for the offloading optimization.
2) LSTM-based Sequence-to-Sequence neural network for workflow offloading prediction: we propose an optimized neural network, LSTM-based Sequence-to-Sequence neural network, to make the offloading prediction.
3) Robustness improvement of offloading: we integrate Meta-Learning with our multi-objective Reinforcement Learning to improve the robustness of offloading performance.

In the remainder of this paper, we will first review the existing RL-based solutions for workflow offloading optimization in Section II. Then, in Section III, we offer the background of our methodology. In Section IV, we present the detailed methodology and algorithm of the proposed MRL-MO-RO

approach. Next, we evaluate the approach in Section V. Section VI presents further discussions on the experimental results and potential future work. Finally, Section VII summarizes the whole paper.

II. Related Work

During the past decades, offloading has been extensively studied [13]. Different offloading solutions have been developed: using hierarchical method [14], or collaborative optimization method [15], energy-efficient method [16]. Those classical approaches often rely on explicit models of resources or workflows to design the offloading policies and strategies for a specific system. Machine Learning-based approaches have hence been investigated [3] [4]. Among them, RL offers an interactive approach without data labelling [5] [6] [17] [4]. However, the performance still highly depends on the design and configuration of the learning pipeline. When applying a mature model to a new environment, we can observe some performance deviation among those methods. This issue is also called the robustness of the offloading performance. These are some work addressed this issue: adaptive methods [18], connection stability [19], robust network contention [20]. However, compared with throughput or energy consumption, the issue of offloading robustness from a multi-objective perspective has gained much less attention during past years. In the next section, we will formulate our approach step by step.

III. Background

A. Multi-Objective Learning(MOL)

A typical MOL [21] setup is given a collection of input points and sets of targets for various tasks per point. A common way to set up the inductive bias across tasks is to design a characterized hypothesis class that shares some parameters across tasks. Typically, these parameters are learned by solving an optimization problem that minimizes a weighted sum of the empirical risk for each task. However, the linear-combination formulation is only sensible when a parameter set is effective across all tasks. In other words, minimization of a weighted sum of empirical risk is only valid if tasks are not competing, which is rarely the case. MOL with conflicting objectives requires modeling the trade-off between tasks beyond what a linear combination achieves.

An alternative objective for MOL is finding solutions that are not dominated by others. Such solutions are said to be Pareto optimal. In this paper, we cast the objective of MOL in terms of finding Pareto optimal solutions.

B. Reinforcement Learning

Reinforcement Learning [22] considers learning from interactions with environment to maximize the accumulated reward. A learning task is formulated as an Markov Decision Process (MDP), which is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$. Here, $\mathcal{S}$ is the state space, $\mathcal{A}$ denotes the action space, $\mathcal{R}$ is a reward function, $\mathcal{P}$ is the state-transition probabilities matrix, $\gamma \in [0, 1]$ is the discount factor. A policy $\pi(a|s)$, where $a \in \mathcal{A}$ and $s \in \mathcal{S}$, is a mapping from state s to the probability of selecting action a. We define the trajectories sampled from the environment according to the policy π as $\tau_\pi = (s_0, a_0, r_0, s_1, a_1, r_1, ...)$, where $a_t \sim \pi(\cdot|s_t)$ and r_t is a reward at time step t.

The state value function of a state s t under a characterized policy $\pi(a|s : \theta)$, denoted as $v_\pi(s_t)$, is the expected return when starting in s_t and following $\pi(a|s; \theta)$ thereafter. Here, θ is the vector of policy parameters and $v_\pi(s_t)$ can be calculated by:

$$v_\pi(s_t) = \mathbb{E}_{\tau \sim P_{\mathcal{T}(\tau|\theta)}}[\sum_{k=t} \gamma^{k-t} r_k] \tag{1}$$

where $P_{\mathcal{T}(\tau|\theta)}$ is the probability distribution of sampled trajectories based on $\pi(a|s; \theta)$. The objective in general is to search for an optimal policy formulated as $\pi(a|s; \theta^*)$ to maximize the expected total rewards: $J = \sum_{s \in P} v_\pi(s_k)$.

C. Meta Learning

Meta-Learning [23] aims to learn a more general model instead of learning a specific one. The robustness of Meta-Learning comes at the randomization of the training environment, which is thus the dynamics. The model trained by meta-learning can effectively exploit and adapt to changes brought incurred by dynamics faster than re-training the model from scratch. In a typical Meta-Learning setting, the task distribution Λ provides the training set and adaption set (new tasks). The training process is to learn a policy model, denoted as π_θ, characterized by θ. π_θ optimizes the objective function while minimizing learning loss denoted as $\mathcal{L}_\mathcal{D}$.

In this paper, we introduce the gradient-based meta-learning into RL, which updates the learning parameter in two steps:

1) Inner layer update: Doing training on the sample drawn for training $\mathcal{D}_{tr}$ from task distribution $\mathcal{D}$ to calculate the updated θ' according to the following update function:

$$\theta' = \Phi_\beta(\mathcal{D}_{tr}, \theta) \tag{2}$$

2) Outer layer update: Using the updated θ' to apply testing procedure among tasks $\mathcal{D}_{te}$, which is from the same data set with $\mathcal{D}_{tr}$, to update the parameter of the model when achieving minimal of loss, we will demonstrate the definition of the loss function in next section.

$$\min_{\theta,\beta} \mathbb{E}_\mathcal{D}[\mathcal{L}(\mathcal{D}_{te}, \theta')] \tag{3}$$

For the inner layer update, we adopt the gradient descent method to update θ as follows:

$$\Phi_\beta(\mathcal{D}_{tr}, \theta) = \theta - \alpha \nabla \mathcal{L}(\mathcal{D}_{tr}, \theta) \tag{4}$$

Repeatedly, according to the convergence, after specific times mutation of environment [24], a general model is achieved. When a learned model encounters a new data set or a new environment, it just needs to adapt itself by few times learning new features. Moreover, the inner layer learns a specific offloading model for a specific data set from a cloud log period. As is known, all kinds of dynamics exist in resource

TABLE I: Notation Summary

Notation	Description
$\mathbb{E}$	Mean value
ta_i	task i
Da_i^s, Da_i^r	Size of data sending to or receiving from a task ta_i
UT, DT	Transmission rate of uplink and downlink
Cap_{Lo}, Cap_l ,Cap$_t$	Computation capacity of user equipment (UE), VM l and MEC host at time t
$Lat_i^{ul}, Lat_i^s, Lat_i^{dl}, Lat_i^{UE}$	Latency for task i on uplink channel, MEC host, downlink channel, and UE.
Enc$_i$	Energy consumption of task i
RU$_t$	Resource utilization rate at time t
EC$_t$	Energy consumption on MEC host at time t
$\mathcal{T}_i^U, \mathcal{T}_i^s, \mathcal{T}_i^D, \mathcal{T}_i^{UE}$	Finish time for task t i on uplink channel, MEC host, downlink channel, and UE
$Av_i^U, Av_i^s, Av_i^D, Av_i^{UE}$	Resource available time for task ta_i on uplink channel, MEC host, downlink channel, and UE
$Pol_{1:n}$	Computational offloading plan for n tasks
$\mathcal{T}_i, \rho(\mathcal{T})$	A learning task and distribution of learning tasks
s_i, a_i, r_i	State, action, and reward of an MDP at time step i
$\pi(a\|s;\theta), v(s;\theta)$	Parametrized policy and value function for computation offloading.
τ_π	Trajectories sampled from the environment based on the policy π.
$\mathcal{F}_{en}, \mathcal{F}_{de}$	Functions of encoder and decoder
e_i, d_i	Output of encoder and decoder at time step i
c_i	Context vector at decoding step i
$\hat{A}_t$	Advantage function at time step t
$Up(\theta, \mathcal{T}_i)$	Update function (e.g., Adam) for the learning task

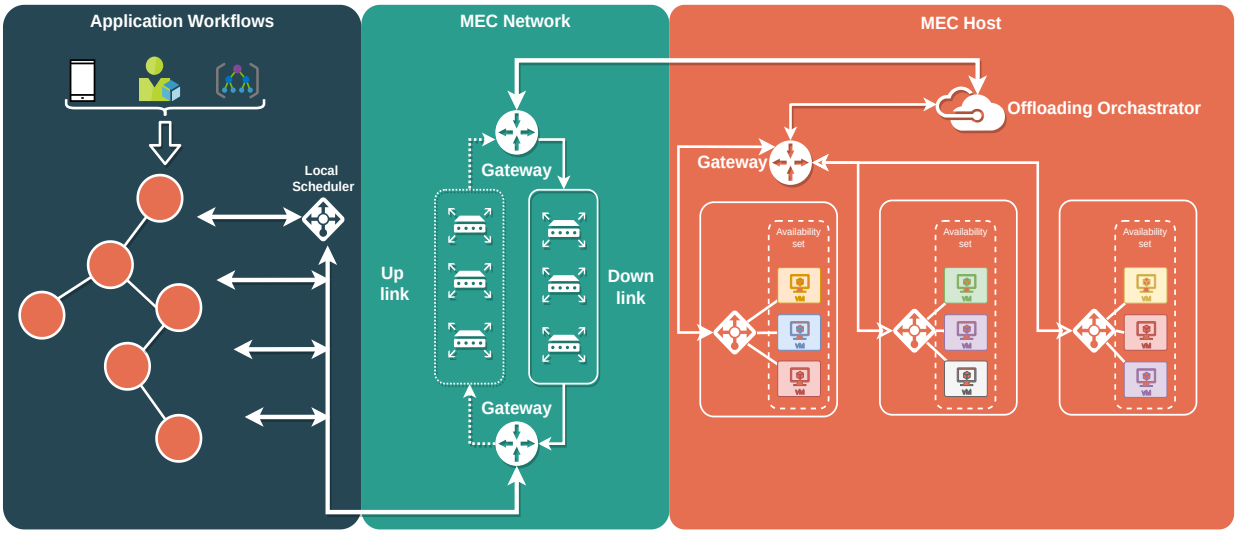

Fig. 1: Typical Offloading Process

availability and task demands. The data trajectories consequently change dynamically, where the outer layer is working on learning across different data trajectories to achieve more features improving robustness. As to the inner layer, the learning goal is to learn the offloading model, which acts as a scheduler in the system interacting with the task model and cloud platform models. Therefore, the inner layer's learning approach has decent interactive ability while learning for this role. Reinforcement Learning is ideal for this mission among different learning approaches as its structure fits the problem set up in this work. The following subsection will introduce the details of the RL approach's feasibility and its design.

IV. Methodology

A. Problem Formulation

As is shown in Figure 1, in a typical offloading process, the application requests together with dependencies form the workflow. The decision made by the local scheduler is whether to offload the tasks to the MEC host based on the user side local calculation capability and the resource availability of the MEC host. On the MEC host, the orchestrater allocates tasks to VMs. The DAG models of workflows are $\mathcal{D} = (TA, \vec{ED})$, where vector TA represents the tasks and vector $\vec{ED}$ represents the directed edge of each task dependencies among the tasks, respectively. $\vec{ed} = (ta_i, ta_j)$ denotes the dependency between task ta_i and task ta_j: ta_j is an immediate successor task of ta_i. A successor task has to wait for the ending of

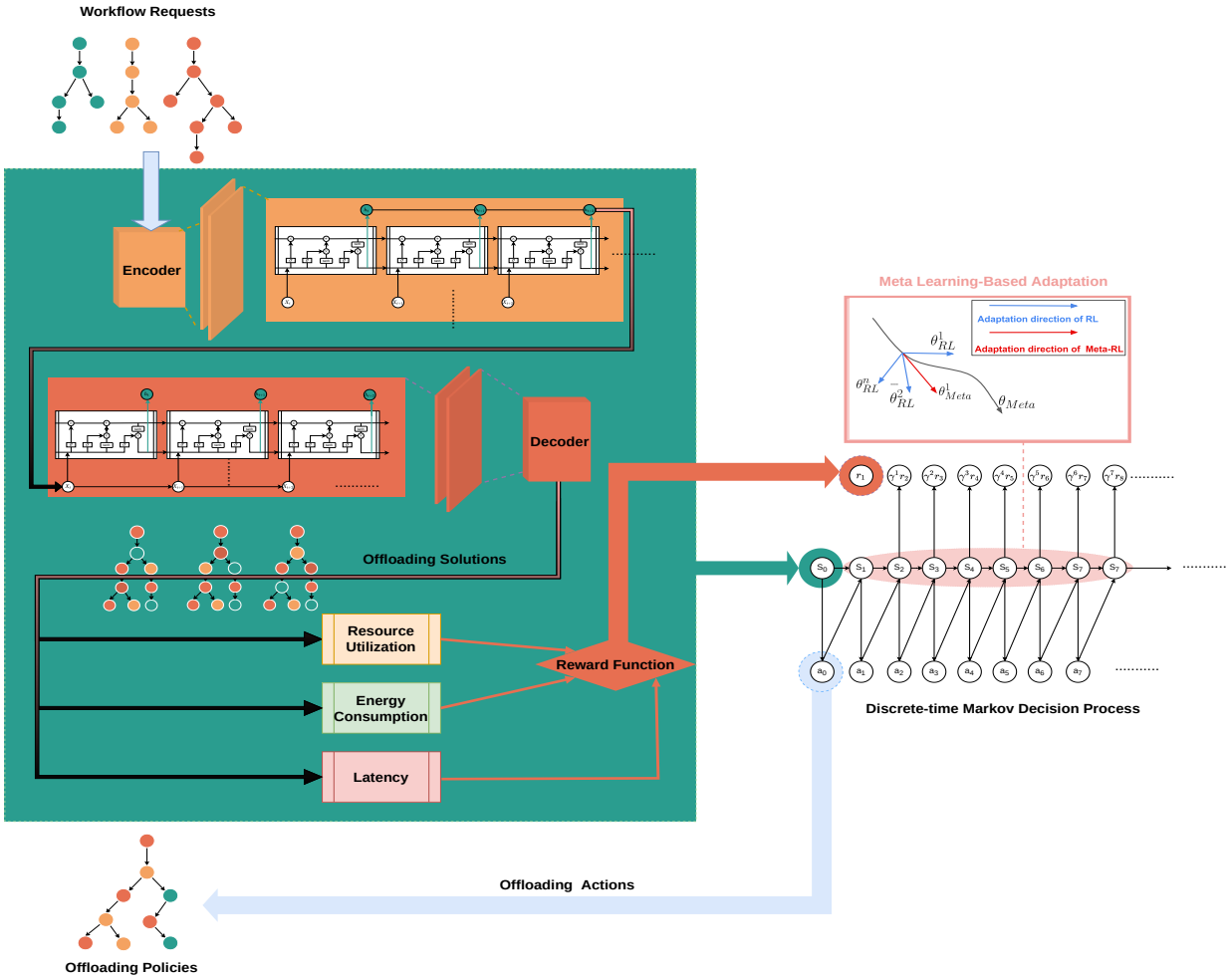

Fig. 2: Paradigm of MRL-MO-RO

its leading task's execution until the last one. Given ξ type of VMs with heterogeneous resources with different type of resources combination, the computation capacity of each is $Cap_l, l \in [1, 2, 3, ..., \xi]$. The task profile of ta_i includes the required resource requirement for running the task, Cp_i, data sizes of the task sent, Da_i^s, and the result data received, Da_i^r. The state information of the MEC host includes the transmission rate of the wireless uplink channel, UT, and the rate of the downlink channel, DT. Therefore, the latency for sending data, Lat_i^U, executing on the MEC host, Ex_i^s, and receiving the result, Lat_i^D, energy consumption Enc_i of task ta_i can be calculated as:

$$\begin{aligned} &Lat_i^U = Da_i^s/UT, Ex_i^s = Cp_i/Cap_l, \\ &Lat_i^D = Da_i^r/DT, Enc_i = m(Lat_i^U + Lat_i^D) + nEx_i^s \end{aligned} \tag{5}$$

where m, n are thresholds between latency and energy consumption. If task ta_i runs locally on the end-user side, there is only running latency on the end-user side, which can be obtained by $Lat_i^{Lo} = Cp_i/Cap_{Lo}$, where Cap_{Lo} denotes the computation capacity of the end-user. The end-to-end latency of a task offloading process includes local processing, uplink transmission, and remote processing latency and results receiving latency, as shown in Figure 1.The offloading policy is denoted as $Pol_{1:n} = a_1, a_2, ..., a_n$, where $|TA| = n$ and a_i represents the offloading decision of ta_i.

The finish time on the uplink channel, $\mathcal{T}_i^U$, are defined as:

$$\begin{aligned} &\mathcal{T}_i^U = max\{Av_i^U, \max_{j \in parent(t_i)}\{\mathcal{T}_j^{UE}, \mathcal{T}_j^D\}\} + Lat_i^U, \\ &Av_i^U = max\{Av_{i-1}^U, \mathcal{T}_{i-1}^U\} \end{aligned} \tag{6}$$

The finish time of ta_i on the MEC host, FT_i^s, resource utilization rate RU_t, energy consumption EC_t, and latency on the downllink channel, FT_i^D are defined as:

$$\begin{aligned} &\mathcal{T}_i^s = max\{Av_i^s, max\{\mathcal{T}_i^U, \max_{j \in parent(t_i)}\{\mathcal{T}_j^s\}\}\} + Lat_i^U, \\ &Av_i^s = max\{Av_{i-1}^s,\ \mathcal{T}_{i-1}^s\}, \\ &\mathcal{T}_i^D = max\{Av_i^D, \mathcal{T}_i^s\} + Lat_i^D, \\ &Av_i^D = max\{Av_{i-1}^D, \mathcal{T}_{i-1}^D\} \\ &RU_t = Cap_t/Cap_l \\ &EC_t = p(FT_i^s - T_i^U) + qRU_t \end{aligned} \tag{7}$$

where p, q are energy consumption threshold. The finishing time of ta_i on the end user side, FT_i^{UE} are defined as:

$$\begin{aligned} \mathcal{T}_i^{UE} &= max\{Av_i^{UE}, \max_{j \in parent(t_i)}\{\mathcal{T}_j^{UE}, \mathcal{T}_j^{D}\}\} + Lat_i^{UE}, \\ Av_i^{UE} &= max\{Av_{i-1}^{UE}, \mathcal{T}_{i-1}^{UE}\}. \end{aligned} \tag{8}$$

Overall, given a offloading plan $Pol_{1:n}$, the total latency of a DAG: $Lat^c_{A_{1:n}}$, is defined as:

$$Lat^c_{Pol_{1:n}} = max[\max_{t_k \in \mathcal{K}}\{\mathcal{T}_k^{UE}, \mathcal{T}_k^{D}\}] \tag{9}$$

where $\mathcal{K}$ is the set of exit tasks which have no successor tasks. In the next section, we present the details of out proposed algorithm dealing with this problem.

B. Multi-Objective Offloading

As is shown in Figure 2, the whole offloading process starts when the new workflows form. First, they will be embedded and input in *Encoder*, depicted on top of the green area in orange color. Then, the output states of Encoder are the input of Decoder, depicted in the middle of the green area in red color, to get the offloading policies prediction. Finally, based on those offloading policies prediction, the RL agent calculates the rewards of resource utilization, energy consumption, and latency, which are depicted at the bottom of the green area. After the learning process by the RL agent, as depicted in the middle green part, the meta learner adapts to the gradients updates learned by RL agents to pick the action of offloading policy for each state. The Meta-Learning process is depicted in the pink block at the right top of the figure, named "Meta Learning-Based Adaptation." Finally, the framework updates the overall offloading policy model and related parts: resource availability and energy consumption.

More specifically, we formulate the algorithm step by step in the following part; firstly we start with the MDP formulating of the RL part:

1) **State**: When offloading a task ta_i, we define the state as a combination of the encoded DAG and the partial offloading plan:

$$\mathcal{S} := \{s_i | s_i = (\mathcal{D} = (TA, \vec{ED}), Pol_{1:i})\}, i \in [1, |TA|], \tag{10}$$

where $\mathcal{D} = (TA, \vec{ED})$ is comprised of a sequence of task embedding, and $Pol_{1:i}$ is the offloading policy of the tasks scheduled before ta_i.

2) **Action**: The offloading for each task is a binary choice; thus, the action space is defined as: $\mathcal{A}$, which includes actions: execution locally, execution on different VMs with different resources.

3) **Reward**: The objective is to minimize loss given by Equation 15. In order to achieve this goal, we define the reward function as the estimated negative increment of the reward function after making an offloading decision for a task. Formally, when taking action for the task ta_i, the increment is defined as:

$$\Delta r_i^c = r_{A^c_{1:i}} - r_{A^c_{1:i-1}} \tag{11}$$

Based on the above MDP definition, we denote the policy when offloading ta_i as:

$$Pol(a_i | \mathcal{D} = (TA, \vec{ED}), A_{1:i-1}) \tag{12}$$

and the policies for workflow as

$$\begin{aligned} &Pol(A_{1:n} | \mathcal{D} = (TA, \vec{ED})) \\ &= \prod_{i=1}^{n} Pol(a_i | \mathcal{D} = (TA, \vec{ED}), A_{1:i-1}) \end{aligned} \tag{13}$$

Figure 2 shows our design of offloading training model. In the training paradigm, both encoder and decoder are built with recurrent neural networks(RNN) [25]. The input of the encoder is the sequence of task embedding, while the output of the decoder is the offloading decisions of each tasks. We include the attention mechanism [26] to allow the decoder to attend to different parts of the source sequence without information loss issue. We define the functions of the encoder and decoder as $\mathcal{F}_{en}$ and $\mathcal{F}_{de}$ respectively. At each step of encoding, the output of the encoder, e_i, is obtained by $e_i = \mathcal{F}_{en}(ta_i, e_{i-1})$. After encoding all the input task embedding, we have the output vector as $\vec{e} = [e_1, e_2, ..., e_n]$. At each decoding step, we define the output of the decoder, d_j, as $d_j = \mathcal{F}_{de}(d_{j-1}, a_{j-1}, c_j)$, where c_j is the context vector at decoding step j and is computed as a weighted sum of the encoder: $c_j = \sum_{i=0}^{n} \alpha_{ji} e_i$

The weight α_{ji} of each output of encoder, e_i is computed by

$$\alpha_{ji} = \frac{exp(f(d_{j-1}, e_i))}{\sum_{k=1}^{n} exp(f(d_{j-1}, e_i))}, \tag{14}$$

where the score function, $f(d_{j-1}, e_i)$, is used to measure how well the input at position i and the output at position j match. We use the sequence-to-sequence neural network [27] to approximate both policy $Pol(a_j|s_j)$ and value function $v_{Pol}(s_j)$ by passing the output of decoder to two separate fully connected layers. During training for the sequence-to-sequence neural network, the action a_j is generated through sampling from the policy $Pol(a_j|a_j)$. Once the training is finished, the action a_j is generated by $a_j = argmax_{a_j} Pol(a_j|s_j)$.

The overall objectives we investigate in this work include latency, resource utilization rate, and energy consumption, denoted respectively by OB_l, OB_{ur}, and OB_{ec}, respectively. Therefore, given N training data samples, the overall objective of the learning approach can be formulated as:

$$\begin{aligned} \underset{\boldsymbol{r}^{OB_l}, \boldsymbol{r}^{OB_{ur}}, \boldsymbol{r}^{OB_{ec}}}{\arg\max} r_i = \sum_{i=1}^{N} \Big[& r(Lat^c_{Pol_{1:n}}, Pol_{1:n}) \\ & + w_a r(RU_t, Pol_{1:n}) \\ & + w_b r(EC_t, Pol_{1:n}) \Big] \end{aligned} \tag{15}$$

where r_i represents the reward of i-th data sample: $r_i^{OB_l}$, $r_i^{OB_{ur}}$, and $r_i^{OB_{ec}}$, namely rewards of latency, resource utilization rate and energy consumption. w_a, w_b, are weights that

signifies the importance of different optimization objectives. The corresponding rewards are calculated as follows:

$$\begin{cases} r(Lat^c_{Pol_{1:n}}, Pol_{1:n}) = \max[0, (Lat^c_{Pol_{1:n}} - \Upsilon)]P_a, \\ r(RU_t, Pol_{1:n}) = \Big[\max[0, (RU_t - RU_m)] \\ \qquad + w_a Lat^c_{Pol_{1:n}}\Big]P_a, \\ r(EC_t, Pol_{1:n}) = \Big[\max[0, (EC_t - EC_m)] \\ \qquad + w_b Lat^c_{Pol_{1:n}}\Big]P_a, \end{cases} \tag{16}$$

where, P_a denotes a constant representing penalty, Υ is latency-critical threshold, RU_m, EC_m are the upper bounds of resource utilization rate and energy consumption expense.

C. Robust Offloading

After formulation of multi-objective offloading, we turn to Meta-Learning formulation [23], which consists of two loops for training: *inner loop* and *outer loop*. We define the objective function based on Proximal Policy Optimization (PPO) [28]:

$$J^C_{ta_i}(\theta_i) = \mathbb{E}_{\tau \in P_{ta_i}(\tau, \theta^o_i)}\left[\sum_{t=1}^{n} min\left(Pr_t, \hat{A}_t, slice^{1+\epsilon}_{1-\epsilon}(Pr_t)\hat{A}_t\right)\right] \tag{17}$$

where, $\pi_{\theta^o_i}$ is the sample policy, θ^o_i is the vector of parameters of the sample policy network, π_{θ_i} is the target policy, where θ_i equals to θ^o_i at the initial epoch. Pr_t is the probability ratio between the sample policy and target policy, which is defined as

$$Pr_t = \frac{\pi_{\theta_i}(a_t|\mathcal{D}(TA, \vec{ED}), A_{1:t})}{\pi_{\theta^o_i}(a_t|\mathcal{D}(TA, \vec{ED}), A_{1:t})} \tag{18}$$

The slice function $slice^{1+\epsilon}_{1-\epsilon}(Pr_t)$ aims to limit the value of Pr_t, in order to remove the incentive for moving Pr_t outside the interval $[1-\epsilon, 1+\epsilon]$. $\hat{A}_t$ is the advantage function at time step t. Specially, we use general advantage estimator (GAE) [29] as our advantage function, which is defined as:

$$\hat{A}_t = \sum_{k=0}^{n-t+1} (\gamma\lambda)^k (r_{l+k} + \gamma\nu_\pi(s_{t+k+1}) - \nu_\pi(s_{t+k})), \tag{19}$$

where$\lambda \in [0, 1]$ is used to control the trade-off between bias and variance. The value function loss is defined as:

$$J^{VE}_{\mathcal{T}_i}(\theta_i) = \mathbb{E}_{\tau \sim P_{ta\,i}}(\tau, \theta^o_i)\left[\sum_{t=1}^{n} (\nu_\pi(s_t) - \hat{\nu}_\pi(s_t))^2\right] \tag{20}$$

where $\hat{\nu}_\pi(s_t) = \sum_{k=0}^{n-t+1} \gamma^k r_{t+k}$.

Overall, we combine Equation 17 and Equation 20, defining the objective function for each *inner layer* task learning as:

$$J^{PPO}_{\mathcal{T}_i}(\theta_i) = J^C_{ta_i}(\theta_i) - c_1 J^{VE}_{\mathcal{T}_i}(\theta_i), \tag{21}$$

where c_1 is the coefficient of value function loss. The outer layer objective is expressed as:

$$J^{MLD}(\theta) = \mathbb{E}_{\mathcal{T}_i \sim \rho(\mathcal{T}), \tau \sim P_{\mathcal{T}_i}(\tau, \theta'_i)}[J^{PPO}_{\mathcal{T}_i}(\theta'_i)], \tag{22}$$

where $\theta'_i = Up_{\tau \sim P_{\mathcal{T}_i}(\tau, \theta_i)}(\theta_i, \mathcal{T}_i), \theta_i = \theta$. To simplify the calculation complexity, we use the fist-order approximation to replace the second-order derivatives which is defined as

$$Grad^{MLD} := \frac{1}{n}\sum_{i=1}^{n}\left[(\theta'_i - \theta)/\alpha/h\right], \tag{23}$$

where n is the number of sampled learning tasks in the *outer loop*, α is the learning rate of *inner loop* training, and h is the conducted gradient steps for the *inner loop* training.

D. MRL-MO-RO Algorithm

Algorithm 1 Meta-Reinforcement-Learning-based multi-objective robust offloading algorithm(MRL-MO-RO)

Require: Distribution over workflows: Λ, Learning rate: $\alpha, \beta \sim \mathbb{R}^+$
1: Initialize the policy π_θ and $\mathcal{D} \leftarrow \varnothing$
Require: Number of environments: N
2: **for** $i = 1, ...N$ **do**
3: | Use pre-adapted policies $\pi_{\theta'_H}$ to sample $\mathcal{D}' \sim \Lambda$
4: | Add samples: $\mathcal{D} \leftarrow \mathcal{D}'$
5: | **for** $j = 1, ..., H$ **do**
6: | | Set a sliding door with the size $h_i \in (0, H)$
7: | | Use policies π_θ to sample trajectories within first h_i samples $\tau_{h_i} \sim H$
8: | | Use τ_H to calculate adapted parameters:
9: | | $\theta'_{h_i} = \theta + \alpha \sum_{j=1}^{h_i} \nabla \log J^{PPO}(\theta)$
10: | | Use adapted policy $\pi_{\theta'_{h_i}}$ sample trajectories $\tau'_{h_i} \sim H$
11: | **end for**
12: | Use τ_H to calculate adapted parameters:
13: | $\theta'_H = \theta + \alpha \sum_{h_i} \nabla \log J^{PPO}(\theta'_{h_i})$
14: | Use adapted policy θ'_H sample trajectories $\tau'_H \sim \mathcal{D}$
15: **end for**
16: Calculate update:
17: $\theta \leftarrow \theta - \beta \nabla_\theta \frac{1}{H} \sum_{j=1}^{H} J^{MLD}_j(\theta'_H)$ using τ'_H
18: **return** θ

This section elaborates on the overall algorithm in detail, merging all components mentioned in previous sections. As is shown in algorithm 1, distribution over tasks, and learning rates of the outer and inner loop are required. The first step is the initialization of the algorithm: setting the initial parameters of the policy model and resetting the data set $\mathcal{D}$. We also set a sliding learning window with a size of h to improve the algorithm as a continuous learning process. From line 2 to line 4 is the sampling step: based on the number of environments, N data trajectories are sampled from the distribution Λ according to the current policy model and added to the data set $\mathcal{D}$. In the following inner layer learning loop from lines 5 to 9, the learning agent samples a smaller sliding learning window with a size of h to calculate updated θ'_h based on each loss function line 10. Unlike conventional RL or other learning methods, the overall policy model is not updated by any parallel inner layer learning agent. After achieving updated θ'_h, RL agent uses θ'_{h_i} model to sample new data samples τ'_{h_i} from $\mathcal{D}$. In line 12, the learning agent finishes a sliding window learning and then continues to the next sliding window. The learned τ'_{h_i} are used to calculate θ'_H and update τ'_H. From lines 16 to 17, the overall parameter update for offloading policy model is calculated based on the gradient of θ'_H learned by the RL agent from each sliding window within each environment. After each iteration, the sliding window continues to sample

TABLE II: Fine-tuned baseline approaches: we train DQN, Double-DQN, CEM based approaches as baselines of our proposed MRL-MO-RO.

Fine-tuned RL Approaches						
Parameters / Baseline Approaches	NN Layers	Replay Buffer Size	Optimizer	ρ	Learning Rate	Activation Function
DQN	4	–	$Adam$	0.95	1e-3	ReLU, Softmax
Double-DQN	3	500	$Adam$	0.95	1e-3	ReLU, Softmax
CEM	3	–	$Adam$	0.95	1e-3	ReLU, Softmax

TABLE III: MRL-MO-RO Hyperparameter set up

Hyperparameter	**Set up**	**Hyperparameter**	**Set up**
Encoder	LSTM, 2 Layers, norm: on	Outer Loop Learning Rate	5×10^{-4}
Number of Neurons	256	Activation Function	tanh
Decoder	LSTM, 2 layers, norm: on	Loss Coefficient	0.5
Inner Loop Learning Rate	5×10^{-4}	Slice Constant	0.2
Optimizer	Adam	Discount Factor	0.99
Gradient steps h	3	Adv Discount Factor	0.95

another data set then the whole algorithm stays a continuous learning process.

V. Evaluation

In this section we implement comprehensive evaluation to validate performance of MRL-MO-RO from following two perspectives:

1) *How is the overall offloading optimization performance of MRL-MO-RO compared with single-objective offloading optimization approaches?*
2) *How is the performance robustness of MRL-MO-RO against dynamics?*

A. Evaluation Measurements

To answer these two questions, we define two groups of measurements. The first group is correspondingly related to three objectives: latency, resource utilization rate, and energy consumption:

QoS Latency Critical Rate (QLCR) [5]: total percentage of executed tasks that meet latency required by QoS.

Resource Utilization Rate (RUR) [30]: average percentage of utilized resources among executed tasks.

Energy Consumption Expense (millions) (ECE) [30]: total electricity consumption bill of executed tasks that meet expected latency. ECE indicates the level of energy consumption for each method.

Robustness measurements [5] includes: Offloading Performance Deviation(OPD) and Adaptation Steps and Data Usage for Performance Recovery(ASDUPR). They are formulated as follows:

$$OPD = \frac{PER_{after} - PER_{before}}{PER_{before}} \tag{24}$$

where, PER_{after} denotes the instant average offloading loss after the influence of dynamic, PER_{before} indicates the previous converged average offloading loss. Besides the instant performance deviation, ASDUPR is proposed to describe adaptation, include time and data iteration needed for adaptation after performance deviation incurred by dynamics:

$$ASDUPR = OPD * ITER * t^o \tag{25}$$

where $ITER$ demonstrates the iteration time, t^o describes time spent for each iteration.

B. Set up

Platform settings: We implement the experiments on a hardware platform consists of 18 nodes, each node has: 4 x GTX 1080 Ti, 2 x Intel(R) Xeon(R) Gold 5118 CPU @ 2.30GHz (12 cores per cpu), 128 GB memory, 2 x 10 TB local HDD, 2 x 4 TB local SSD. The software environment includes: Anaconda, python-numpy, python-scipy, python-dev, python-pip, python-nose, g++ libopenblas-dev, git, Thensorflow, and python-matplotlib.

Simulation Environment: We consider a cellular network, where the data transmission rate varies with the UE position. The CPU clock speed of UE, f_{UE} is set to 1GHz. There are four cores in each VM of the MEC host with a CPU clock speed of 2.5 GHz per core. The CPU clock speed of a VM, f_{VM} is $4 \times 2.5 = 10$ GHz. We implement a synthetic DAG generator according to [31] based on four parameters: n, *fat*, *density*, and ccr, where n represents the task number, *fat* controls the width and height of the DAG, *density* decides the number of edges between two levels of the DAG, and ccr denotes the ratio between the communication and computation cost of tasks.

C. Results

To validate the offloading optimization performance of the MRL-MO-RO algorithm, as is shown in Table IV, we compare the offloading performance of our MRL-MO-RO algorithm with RL-based approaches on the DAG data. Each of the RL-based approaches has a single objective separately. We change the workload in the same manner for each method under the same resource availability setup to get the average optimization results of each objective to compare the

TABLE IV: Offloading Performance Comparison: we compare MRL-MO-RO with fine-tuned DQN(optimizes latency), Double-DQN(optimizes resource utilization rate) and CEM (optimizes energy consumption) to show that MRL-MO-RO achieves better latency-critical offloading performance

Indicators / Approaches / Workflow Topology	MRL-MO-RO			DQN			Double-DQN			CEM		
	QLCR	RUR	ECE	QLCR	RUR	ECE	QLCR	RUR	ECE	QLCR	RUR	ECE
Topology 1	**95.33%±0.26%**	**56.13%±2.24%**	**15.32±2.36**	89.56%±2.36%	-	-	-	69.34%±2.15%	-	-	-	17.38 ± 1.67
Topology 2	**93.33%±0.51%**	**63.35%±1.05%**	**16.75±0.66**	85.33%±2.05%	-	-	-	75.63%±3.12%	-	-	-	19.34 ± 0.88
n=20	**96.16%±0.33%**	**50.71%±2.65%**	**12.67±2.34**	90.78%±0.43%	-	-	-	67.37%±3.65%	-	-	-	15.38 ± 1.67
n=30	**92.34%±0.23%**	**57.66%±3.27%**	**15.02±1.32**	88.53%±2.54%	-	-	-	73.54%±1.36%	-	-	-	17.86 ± 0.65
UT=DT=8.5Mbps	**97.53%±0.42%**	**60.13%±2.58%**	**18.25±2.41**	90.47%±2.34%	-	-	-	68.59%±3.07%	-	-	-	21.57 ± 1.87
UT=DT=5.5Mbps	**95.67%±0.58%**	**72.63%±0.57%**	**20.15±1.56**	87.21%±1.06%	-	-	-	79.32%±1.68%	-	-	-	23.77 ± 1.16

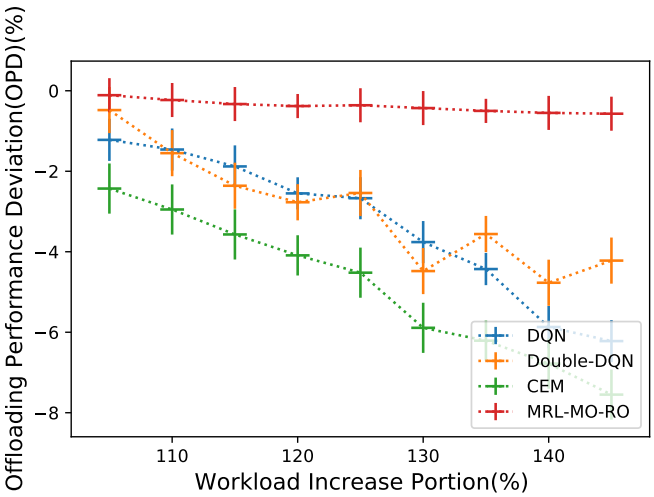

Fig. 3: Comparison of performance deviation between MRL-MO-RO and other 3 fine-tuned single-objective RL-based approaches: with the same amount of workload increase, MRL-MO-RO experiences lower performance deviation

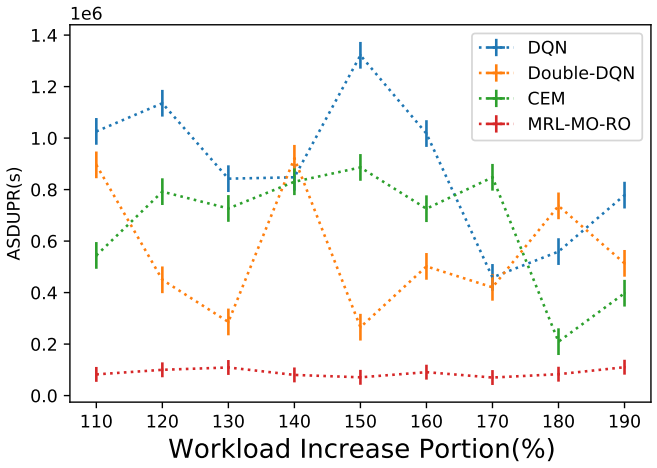

Fig. 4: Comparison of adaptation time spent on retraining after workload changes between MRL-MO-RO and other fine-tuned single-objective RL-based approaches: with the same amount of workload increase, MRL-MO-RO spend less time to recover offloading performance

proposed MRL-MO-RO offloading performance. As can be seen, all numbers in bold are the ones with the best performance. MRL-MO-RO stably offers offloading policies in each environment, outperforming the other three fine-tuned RL methods regarding each objective: expected latency-critical rate, resource utilization rate, and energy consumption. Overall more than $92.34\% \pm 0.23\%$ tasks offloaded meet the expected latency 720ms [18], less than $72.63\% \pm 0.57\%$ resources are utilized, energy consumption is less than 20.15 ± 1.56 (million). In contrast, other tasks offloaded by other RL methods have a $9.12\% - 14.67\%$ violation rate of latency requirement, using 10.35% more resources and 3.56 million more electricity. Moreover, for heavier tasks (topology 2, n=30, UT=DT=5.5Mbps), shown in Table IV, our method MRL-MO-RO still offers better offloading performance regarding multi-objective among different environments. Thus MRL-MO-RO outperforms the other three fine-tuned single-objective RL-based approaches in offlaoding performance.

To validate the robustness of the MRL-MO-RO algorithm's offloading performance, as shown in Figure 3: with the increasing workload, the performance deviation of MRL-MO-RO remains stable within 65%, for some range even under 23%. On the other hand, other RL approaches' performance decreases from more than 50% even beyond 320% with the increased workload. From this, the robustness of our MRL-MO-RO outperforms the conventional RL approach. Figure 4 shows the adaptation speed or performance recovery speed discounted by the performance deviation portion, which balances the adaptation speed and robustness performance. As is shown, we cloud see that the adaptation speed of MRL-MO-RO is more than four times faster than RL averagely after every time increase of workload, at some point, even more, proving its robustness to dynamics of the environment.

VI. Discussion

As shown in results Section V, our proposed approach MRL-MO-RO outperforms the fine-tuned conventional single-objective RL-based approaches at both multi-objective offloading optimization and the offloading robustness against dynamics among environments. Our approach's offloading deviation and adaptation speed change in a similar trend during increased workload. Firstly they both increase within the range of 10%-30% workload increase then decrease within the range of 30%-65% which increases again. Thus when given a 50% or minor workload increase, our approach could still keep

robustness without lower than 30% performance deviation. The robustness decreases when workload increases beyond 50% but still with lower than 50% performance deviation, much lower than fine-tuned single-objective RL-based methods (more than 320%). We are currently working on expanding the robustness range where our framework's framework could keep robustness with lower performance deviation to improve the overall offloading robustness further. However, there is still room for further improvement, particularly for reducing the deviation right after the dynamic workload changes. Furthermore, by investigating alternative offloading strategies like migrations of tasks or policies redundancy, we could improve offloading performance robustness.

VII. Conclusion

In this work, within the Edge-to-cloud continuum environment, MRL-MO-RO, a robust multi-objective workflow offloading algorithm, is presented to offer offloading optimization from objectives: latency, resources utilization rate, and energy consumption. We propose a Meta-Reinforcement sequence-to-sequence robust learning framework to quickly adapt an offloading policy model to a newly changed environment while optimizing multi-objective at the same time. Experimental results show that our approach can provide better offloading optimization performance regarding three objectives, outperforming fine-tuned single-objective RL-based methods. Furthermore, our MRL-MO-RO approach finishes adaptation in new environments using fewer training iterations, 320%-510% faster than the fine-tuned single-objective RL approach, achieving better robustness while offering better offloading performance.

Acknowledgment

This research is funded by the European Union's Horizon 2020 research and innovation program under grant agreements 825134 (ARTICONF project), 862409(BlueCloud project) and 824068 (ENVRI-FAIR project). The research is also supported by the Chinese Scholarship Council, and EU LifeWatch ERIC.

References

[1] D. Sabella, V. Sukhomlinov, L. Trang, S. Kekki, P. Paglierani, R. Rossbach, X. Li, Y. Fang, D. Druta, F. Giust, *et al.*, "Developing software for multi-access edge computing," *ETSI white paper*, vol. 20, pp. 1–38, 2019.

[2] Q.-V. Pham, F. Fang, V. N. Ha, M. J. Piran, M. Le, L. B. Le, W.-J. Hwang, and Z. Ding, "A survey of multi-access edge computing in 5g and beyond: Fundamentals, technology integration, and state-of-the-art," *IEEE Access*, vol. 8, pp. 116974–117017, 2020.

[3] S. Yu, X. Wang, and R. Langar, "Computation offloading for mobile edge computing: A deep learning approach," in *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pp. 1–6, IEEE, 2017.

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

[5] H. Liu, P. Chen, and Z. Zhao, "Towards a robust meta-reinforcement learning-based scheduling framework for time critical tasks in cloud environments," in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, pp. 637–647, IEEE, 2021.

[6] T. Q. Dinh, Q. D. La, T. Q. Quek, and H. Shin, "Learning for computation offloading in mobile edge computing," *IEEE Transactions on Communications*, vol. 66, no. 12, pp. 6353–6367, 2018.

[7] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang, "Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks," *IEEE access*, vol. 4, pp. 5896–5907, 2016.

[8] H. Zhou, Y. Hu, X. Ouyang, J. Su, S. Koulouzis, C. Laat, and Z. Zhao, "CloudsStorm: A framework for seamlessly programming and controlling virtual infrastructure functions during the DevOps lifecycle of cloud applications," *Software: Practice and Experience*, vol. 49, pp. 1421–1447, Oct. 2019.

[9] Z. Zhao, P. Grosso, J. van der Ham, R. Koning, and C. de Laat, "An agent based network resource planner for workflow applications," *Multiagent and Grid Systems*, vol. 7, pp. 187–202, Dec. 2011.

[10] Y. Hu, C. de Laat, and Z. Zhao, "Optimizing Service Placement for Microservice Architecture in Clouds," *Applied Sciences*, vol. 9, p. 4663, Nov. 2019.

[11] Y. Ye, R. Q. Hu, G. Lu, and L. Shi, "Enhance latency-constrained computation in mec networks using uplink noma," *IEEE Transactions on Communications*, vol. 68, no. 4, pp. 2409–2425, 2020.

[12] Y. Hu, H. Zhou, C. de Laat, and Z. Zhao, "ECSched: Efficient Container Scheduling on Heterogeneous Clusters," in *Euro-Par 2018: Parallel Processing* (M. Aldinucci, L. Padovani, and M. Torquati, eds.), vol. 11014, pp. 365–377, Cham: Springer International Publishing, 2018. Series Title: Lecture Notes in Computer Science.

[13] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[14] Z. Zhao, R. Zhao, J. Xia, X. Lei, D. Li, C. Yuen, and L. Fan, "A novel framework of three-hierarchical offloading optimization for mec in industrial iot networks," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 8, pp. 5424–5434, 2019.

[15] M. Huang, W. Liu, T. Wang, A. Liu, and S. Zhang, "A cloud–mec collaborative task offloading scheme with service orchestration," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 5792–5805, 2019.

[16] X. Yang, X. Yu, H. Huang, and H. Zhu, "Energy efficiency based joint computation offloading and resource allocation in multi-access mec systems," *IEEE Access*, vol. 7, pp. 117054–117062, 2019.

[17] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4005–4018, 2018.

[18] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, "Fast adaptive task offloading in edge computing based on meta reinforcement learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 242–253, 2020.

[19] M. Chen, S. Guo, K. Liu, X. Liao, and B. Xiao, "Robust computation offloading and resource scheduling in cloudlet-based mobile cloud computing," *IEEE Transactions on Mobile Computing*, vol. 20, no. 5, pp. 2025–2040, 2020.

[20] E. Hyytiä, T. Spyropoulos, and J. Ott, "Offload (only) the right jobs: Robust offloading using the markov decision processes," in *2015 IEEE 16th international symposium on a world of wireless, mobile and multimedia networks (WoWMoM)*, pp. 1–9, IEEE, 2015.

[21] O. Sener and V. Koltun, "Multi-task learning as multi-objective optimization," *Advances in neural information processing systems*, vol. 31, 2018.

[22] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[23] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International conference on machine learning*, pp. 1126–1135, PMLR, 2017.

[24] D. Li, Y. Yang, Y.-Z. Song, and T. M. Hospedales, "Learning to generalize: Meta-learning for domain generalization," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[25] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[26] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[27] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in neural information processing systems*, vol. 27, 2014.

[28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[29] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[30] Y. C. Lee and A. Y. Zomaya, "Energy efficient utilization of resources in cloud computing systems," *The Journal of Supercomputing*, vol. 60, no. 2, pp. 268–280, 2012.

[31] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 682–694, 2013.