2023 IEEE International Conference on Web Services (ICWS) | 979-8-3503-0485-5/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICWS60048.2023.00049

A Deep Reinforcement Learning-Based Pointer Scoring Network for Edge Task Scheduling

Yuqi Zhao*, Delun Jiang*, Bing Li*†, Lei Fu‡, Jian Wang*, and Duantengchuan Li*
*School of Computer Science, Wuhan University, Wuhan, China
†Hubei Luojia Laboratory, Wuhan, China
‡Anhui Thingo Intelligent Technology Co., Ltd., Hefei, China
Email:{yuqizhao, jiangdelun, bingli, 2019101130036, jianwang, dtclee1222}@whu.edu.cn

***Abstract*—In edge computing, service providers aim to provide users with low latency, high reliability, and more secure services by deploying services to edge servers close to users. However, scheduling service requests in edge computing scenarios is challenging because the edge service nodes typically have limited resources. This limitation makes it difficult to simultaneously optimize multiple objectives, including resource utilization, total running time, and average waiting time. Existing methods are insufficient to provide intelligent scheduling services that consider multiple optimization objectives, resulting in unsatisfactory scheduling results. To address this problem, we propose PSNet, a deep reinforcement learning-based Pointer Scoring Network, for edge task scheduling. PSNet integrates multiple pointer networks to optimize multiple objectives and obtain final scheduling results. Experiments carried out on three publicly available real-world datasets show that our method can effectively solve the multi-objective optimization problem on edge service requests, with an 11% improvement over several state-of-the-art methods.**

***Index Terms*—Service Requests Scheduling, Edge Computing, Pointer Scoring Network, Deep Reinforcement Learning**

I. Introduction

With the development of Internet of Things (IoT) technologies and intelligent devices, edge computing has gained significant attention. Unlike the classical cloud computing model, edge computing deploys services to edge servers close to users and offers more personalized and convenient services [1]. The limited resources of edge servers, however, make dynamically scheduling tasks on edge server nodes a significant challenge as the number of accesses to edge servers increases [2].

There are two main types of approaches to scheduling service requests: strategy-based approaches and machine-learning approaches. Some strategy-based approaches [3]–[9] focus on multi-objective optimization, while the majority of these methods primarily address resource utilization or running time, rather than considering multi-objective optimization issues comprehensively. Additionally, strategy-based algorithms often require extensive iteration time to obtain a superior solution, which is incompatible with the low latency requirements of service requests in edge contexts. On the other hand, certain researchers [10]–[13] utilize deep learning techniques to estimate task scheduling and predict future system load, demonstrating significant performance improvements over meta-heuristic algorithms. However, many deep learning-based approaches struggle to achieve optimal scheduling performance due to the absence of quality training tags. Some methods attempt multi-objective optimization without fully considering the relevance of objectives. Generally, the sub-objectives of a multi-objective optimization problem contradict each other, and improving one sub-objective may result in the degradation of performance in another or several other sub-objectives. Therefore, analyzing the correlation between the optimization objectives can help in addressing the challenges of multi-objective optimization problems.

Recently, traditional operating system scheduling techniques have been adapted for edge computing environments. For instance, deep reinforcement algorithms [14], [15] incorporate the status of edge servers when training models by dividing service request sequences into multiple time slices. However, these approaches often concentrate only on the state of the services and servers, neglecting the sequential nature of the scheduling problem. In recent years, several studies have considered the serialization issues associated with service requests, which can make it challenging for a server to schedule requests over time slices. A recent work attempts to address the issue of service request scheduling using deep reinforcement learning-enhanced pointer networks [16], [17]. The study considers factors such as resource utilization, total running time, and average waiting time as separate optimization objectives, and combines multiple optimization objectives through linear weighting. However, it is challenging to determine the optimal weights and requires a significant number of experiments. Additionally, finding parameter weights that can be effectively applied across various experimental scenarios is also challenging.

To address these challenges, we introduce PSNet, a novel approach that utilizes deep reinforcement learning techniques to integrate multiple pointer networks. By incorporating a sophisticated scoring mechanism, PSNet generates a comprehensive score that facilitates optimal scheduling outcomes. Our pointer network model consists of several networks, each of which is optimized for a specific indicator to accommodate the complexities of multi-objective optimization problems. The main contributions of this paper are outlined as follows:

- We conduct an experimental analysis to examine the correlation between multiple optimization objectives, which can help reduce the number of optimization objectives required.
- We propose a multiple pointer networks model specifically designed for multi-objective optimization problems, with each network optimized for a different metric.
- We perform experiments on three publicly available

979-8-3503-0485-5/23/$31.00 ©2023 IEEE
DOI 10.1109/ICWS60048.2023.00049

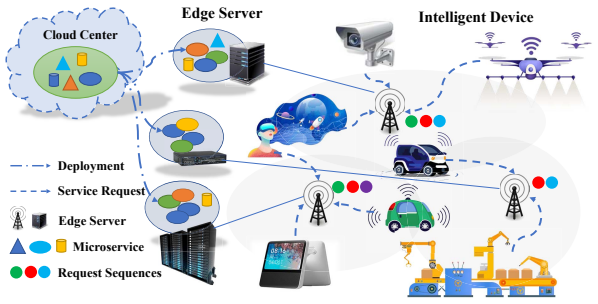


Fig. 1. A typical service requests schedule scenario in edge computing

datasets, and the results demonstrate that our proposed approach outperforms several state-of-the-art methods.

The rest of the paper is organized as follows. In Section II, the scenario and the problem are described. A detailed description of the proposed model is provided in Section III. Section IV reports the experimental results on real-world datasets. More information about the related work can be found in Section V. Section VI concludes this study and introduces the future work.

II. Problem Formulation

In this section, we first highlight the significance of the service request scheduling problem in edge computing. We then formalize the edge service request scheduling problem and propose an optimization objective for this problem, which serves as the basis for introducing our subsequent model.

A. Scenario Description

We utilize the scenario of autonomous driving as an example to illustrate the significance of service request scheduling in edge computing environments. In Fig. 1, we present a typical service request scheduling scenario in edge computing, where vehicles can establish communication with their surroundings and offer various value-added services. Caching content at the edge, particularly music, video, and web content, can enhance the performance of content distribution. The reduction in latency can make a considerable impact. Service content providers are in search of a content delivery network (CDN) with an extensive distribution footprint, ensuring network adaptability and customization to meet user traffic demand. With the growing popularity of autopilot, the number of vehicles connected to the network is increasing, resulting in a substantial surge in communication data. Traditional cloud computing technologies are unable to meet the criteria of low latency and high reliability set by the IoT.

B. Problem Definition

In this section, we define the key concepts and optimization objectives in the edge computing scenario.

1) Edge Server: Services in the edge environment are deployed in the edge servers.

$$E = \{e_1, e_2, \ldots, e_m\}. \tag{1}$$

Each edge server consists of a quadratic vector representing the edge server's CPU capacity consumption, I/O occupation, bandwidth, and memory usage. It is expressed as follows:

$$e_i = (c_i, o_i, b_i, m_i). \tag{2}$$

This gives a formal representation of the edge server, and the request for the edge service is presented next.

2) Service Request: We suppose that n service requests are awaiting for execution at a particular edge server at a given time. The service request set S is defined as follows:

$$S = \{s_1, s_2, \ldots, s_n\}. \tag{3}$$

We denote each request s_j by a vector with seven dimensions as:

$$s_j = (c_j, o_j, b_j, m_j, \tau_j, t_j, \pi_j), \tag{4}$$

where the first four-dimensional vectors (c_j, o_j, b_j, m_j) denote, respectively, the CPU capacity consumption, I/O occupation, bandwidth, and memory usage required to execute the request s_j. τ_j denotes the arrival time of the request s_j to the edge server. t_j indicates the amount of time necessary to execute s_j. Calculated in accordance with the coverage connection between service requests and edge servers, π_j represents the set of edge servers available to handle the request. π_j is represented as:

$$\pi_j = \left\{e_i \mid l_i \geq \|\alpha_j - \beta_j\|_2, e_i \in E\right\}, \tag{5}$$

where l_i denotes the edge server's coverage radius, α_j denotes the location of the edge server e_i, and β_j denotes the service request's coordinates.

3) Optimization objectives: The scheduling problem in mobile edge computing involves three optimization objectives: resource utilization, total running time, and average waiting time. By studying the task scheduling problem in a real-world setting, we can identify the following principles. The running time tends to change proportionally to resource utilization: as resource utilization increases, the running time tends to be shorter, and as resource utilization decreases, the running time tends to be longer. This can be compared to the traditional boxing problem, where the number of required boxes depends on the space utilization inside each box. Higher space utilization results in fewer boxes needed, while lower space utilization requires more boxes.

In our study, we have reduced the three optimization objectives to two: total running time and average waiting time. The specific experimental analysis of these objectives is presented in the experimental section.

The total running time is calculated by averaging the response times of all servers. The collection of edge servers E is known without taking the request timeout into account. The operating time cost set for edge servers corresponding

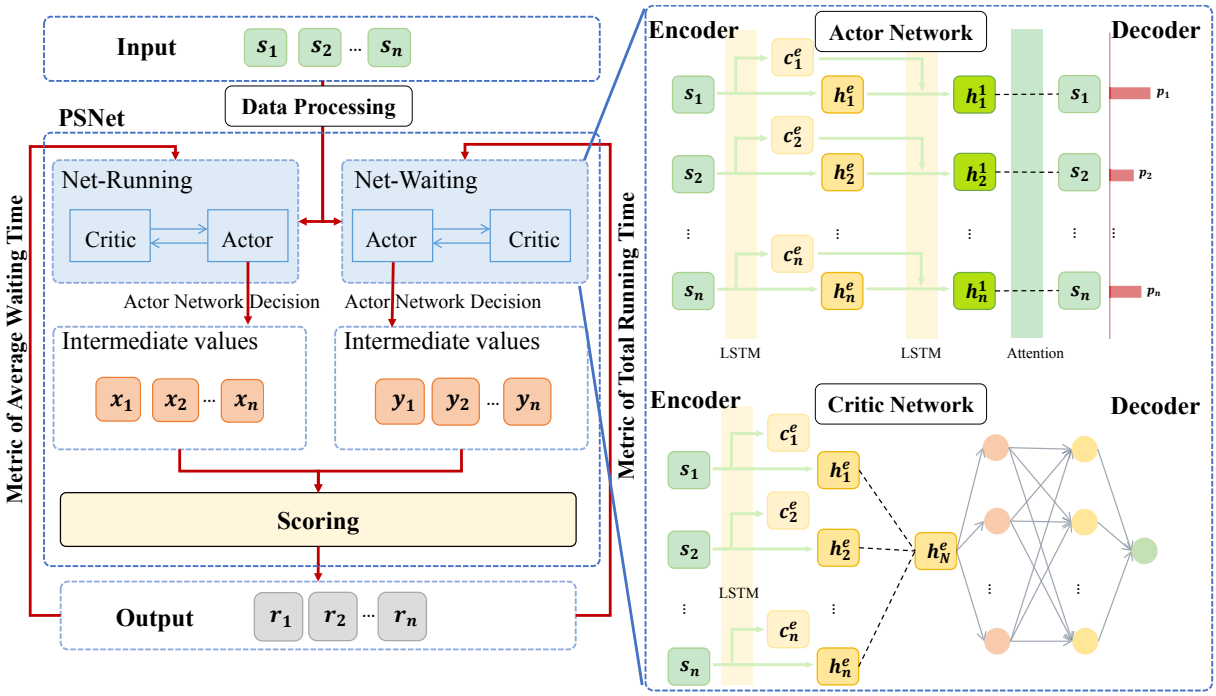


Fig. 2. Framework of PSNet

to (S, C^S) is represented by the returned result τ_{map}. Net-Running uses the total running time as its reward function, defined as follows.

$$reward_{Net-Running} = \frac{1}{m}\sum_{i=1}^{m}\tau_{map_i}, \tag{6}$$

where τ_{map_i} is the i-th edge server's current operating time. The objective of the optimization is to lower $reward_{Net-Running}$.

Net-Waiting uses the average waiting time as its reward function, defined as follows:

$$reward_{Net-Waiting} = \frac{1}{m}\sum_{j=1}^{m}(\frac{1}{n}\sum_{i=1}^{n}W_i), \tag{7}$$

where the waiting time W_i of a service request s_j represents the time between the receipt of the request and its fulfillment by the edge server.

III. The Proposed PSNet Method

The proposed PSNet comprises three components, as illustrated in Fig. 2. The first component is data processing, where the user requests and server information are taken as inputs. The output of this component is the order of request processing after intelligent scheduling. The second component involves the construction of pointer networks, namely Net-Running and Net-Waiting. These networks are designed to optimize the two objectives of total running time and average waiting time. The third component integrates the networks with different objectives using a scoring mechanism. This mechanism is trained through deep reinforcement learning to obtain the final output request sequence.

A. Data Processing

The edge server closest to the service request s_j in the set of S is prioritized. The model scheduling technique outputs the service request subscript sequence. The report shows the edge server set E execution sequence for microservice request set S. When the edge server has enough resources, it may perform many service requests in the parallel output:

$$C^S = \{ C_1 , C_2 , \ldots , C_n\} \tag{8}$$

In a multi-server scenario, the edge server independently executes the model scheduling strategy. The servers generate output denoted as C^S. The solution sequence C^S corresponds to the input request set S and has the same length as S. Similar to the traveling salesman problem, C^S represents the indices of the service requests in our context.

B. Pointer Network

In the previous section, we transformed the original three optimization objectives into two: total running time and average waiting time. However, using linear weighting to address the challenges of edge multi-objective optimization has its drawbacks. It is challenging to determine the optimal weights, which requires a significant number of experiments. To overcome these limitations, we propose a new network model called the Pointer Scoring Network. This model enhances the

single actor-critic network by introducing two groups: Net-Running and Net-Waiting. Each group consists of actor-critic networks that specifically focus on the optimization objectives of total running time and average waiting time, respectively.

Both the Net-Running and Net-Waiting network models follow the structure of the pointer network in RLPNet [16], with the only difference being the reward function utilized during deep reinforcement learning training. The pointer network is a neural network architecture that simplifies the attention mechanism by interpreting the attention weights a_j^i as pointers to elements in the input sequence, rather than as a weighted sum of the input sequence elements. In the pointer network, the attention is controlled by the following mechanism.

$$u_j^i = v^T \, tanh \, (\, W_1 \, e_j + W_2 d_i \,) \quad i, j \in (1, \ldots, n), \tag{9}$$

where parameters v^T , W_1 and W_2 in Equation (9) can be trained by the model.

$$a_j^i = softmax \, (u_i) \; i, \; j \in (1, \ldots, n), \tag{10}$$

where a_j^i is the weight the decoder placed on the $j{-}th$ element of the input sequence at the $i-th$ stage.

$$d_i' = \sum_{j=1}^{n} a_j^i e_j \; . \tag{11}$$

At the $i-th$ iteration of the decoder, the resulting d_i' is combined with the output d_i of the hidden layer.

$$p \, (\, C_i \, | C_1 \, , \ldots , \, C_{i-1}, \, Q \,) = softmax \, (u_i). \tag{12}$$

For each individual actor-critic network, such as Net-Running and Net-Waiting, the actor-network is trained using the policy gradient algorithm, while the critic network is trained using the stochastic gradient algorithm. The training results obtained from the critic network are then used as the baseline function during the training of the actor-network.

C. Scoring Mechanism

We design a scoring mechanism for PSNet to integrate the intermediate results output from the two networks, Net-Running and Net-Waiting, into a final result, as follows. The intermediate results of the existing Net-Running network $\{x_1, \; x_2, \; \ldots, x_n\}$, for the service request corresponding to x_1, its score is n. for the service, a request corresponding to x_2, its score is $n-1$. For each service request, the score can be calculated as follow: $\text{score}_1 = \{n, n-1, \cdots, 1\}$. The more preceding elements in the sequence, the higher the score; the more posterior elements, the lower the score. The same scoring strategy is applied to the intermediate results $\{y_1, \; y_2, \; \ldots, y_n\}$ of the Net-Waiting network to obtain the scoring set $score_2$. The different scores of the same service request s in $score_1$ and $score_2$ are summed to obtain the total score of the service request. The full ratings of all service requests are sorted from the highest to the lowest, and the final output sequence $\{r_1, \; r_2, \; \ldots, r_n\}$ is obtained, which is fed back to Net-Running and Net-Waiting for training.

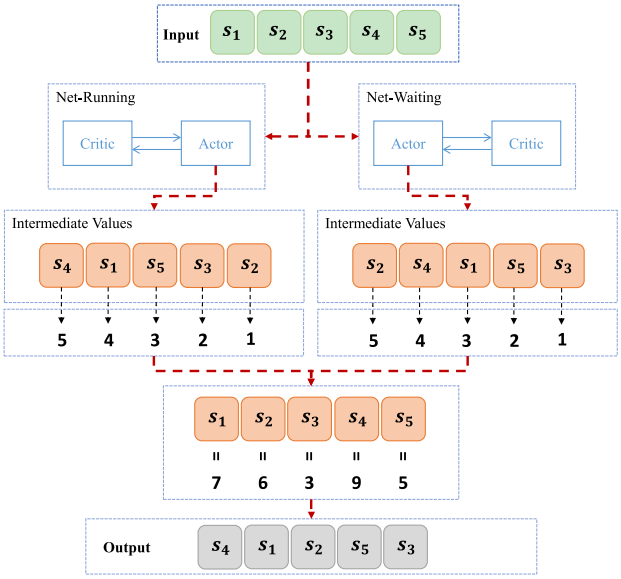


Fig. 3. Illustration of the scoring process in PSNet

Fig. 3 shows a specific scoring process with the input sequence length of five as an example. As we can see from the figure, when five requests enter, the corresponding intermediate results will first be calculated by Net-Running and Net-Waiting, respectively. This intermediate result will correspond to a specific score. Finally, all requests will be sorted and output based on the final score to get the order of service request execution after scheduling.

D. Model Training

The overall training process of PSNet is summarized as follows: the input service request sequence $\{s_1, \; s_2, \; \ldots, s_n\}$ is input to Net-Running and Net-Waiting, respectively, and the intermediate results $\{x_1, \; x_2, \; \ldots, x_n\}$ and $\{y_1, \; y_2, \; \ldots, y_n\}$, where each element represents a service request in the input sequence. After the scoring process, the two intermediate results are combined into the final result $\{r_1, \; r_2, \; \ldots, r_n\}$, where each element also represents a service request in the input sequence. The final results are fed to Net-Running and Net-Waiting for training, and the pseudo-code of the training process is shown in Algorithm 1.

For the Net-Running and Net-Waiting networks, both consist of two stages: Actor and Critic. The key distinction lies in the reward function used during their deep reinforcement learning training. Specifically, Net-Running employs the reward function of total running time as its optimization objective, while Net-Waiting utilizes the reward function of average waiting time as its optimization objective.

The Actor stage follows a specific process, which is defined as follows: The original input data undergoes an embedding operation and is then fed into the encoder. The encoder, essentially a recurrent neural network consisting of LSTM neurons, produces the hidden layer state h and intermediate

Algorithm 1: Training Process of PSNet

Input: Define Q as the training samples in each round;
N as the number of training rounds;
B as the size of the input batch in each round.
Output: the chosen edge server's id

begin
  Initialize Net-Running network with actor-network parameter $\theta_{running}$, Commenter network parameter $\theta_v\_running$, Net-Waiting network with actor-network parameter $\theta_{waiting}$, Commenter network parameter $\theta_v\_waiting$;
  for $step = 1$ *to* N **do**
    / Score Process */*
    for $i \in \{1, \ldots, B\ \}$ **do**
      ${C_i^Q}_r \leftarrow Actor_R(Q)$
      ${C_i^Q}_w \leftarrow Actor_W(Q)$
      ${C_i^Q}_s \leftarrow Score({C_i^Q}_R,\ {C_i^Q}_w)$
    / Net-Running Training */*
    for $i \in \{1, \ldots, B\}$ **do**
      $b_i\_r \leftarrow Critic\_R(Q)$
      $j_{\theta-r} \leftarrow \frac{1}{B}\sum_{i=1}^{B}(\mathscr{R}_{N-R}\left(C_i^Q{}_{scored}\middle|Q_i\right) -$
      $\mathrm{b}_{ir})\nabla_\theta \log_{p\theta}\left(C_i^Q{}_{scored}\middle|Q_i\right)$
      $l_{\theta_v-r} \leftarrow$
      $\frac{1}{B}\sum_{i=1}^{B}||b_{\theta_v}(Q_i) - \mathscr{R}_{N-R}\left(C_i^Q{}_{scored}\middle|Q_i\right)||_2^2$
      $\theta_r \leftarrow ADAM(\theta, j_{\theta-r})$
      $\theta_{vr} \leftarrow ADAM(\theta_v, \nabla_{\theta_v},\ l_{\theta_v-r})$
    / Net-Waiting Training */*
    for $i \in \{1, \ldots, B\}$ **do**
      $b_{iw} \leftarrow Critic_W(Q)$
      $j_{\theta-w} \leftarrow$
      $\frac{1}{B}\sum_{i=1}^{B}(\mathscr{R}_{N-W}\left(C_i^Q{}_{scored}\middle|Q_i\right) -$
      $\mathrm{b}_{iw})\nabla_\theta \log_{p\theta}\left(C_i^Q{}_{scored}\middle|Q_i\right)$
      $l_{\theta_v-w} \leftarrow$
      $\frac{1}{B}\sum_{i=1}^{B}||b_{\theta_v}(Q_i) - \mathscr{R}_{N-W}\left(C_i^Q{}_{scored}\middle|Q_i\right)||_2^2$
      $\theta_w(\theta, j_{\theta-w})$
      $\theta_{vw}(\theta_v, \nabla_{\theta_v},\ l_{\theta_v-w})$

vector c. The process iterates for n times, where n represents the number of hidden layers, and its value equals the length of the input task sequence. During each iteration, h and c serve as input to the decoder, which generates the hidden layer state and the intermediate vector for the current round of decoding. Similar to the encoder, the decoder is a recurrent neural network composed of LSTM neurons. The values of h and c are updated with the output of the current round of the decoder, serving as input for the subsequent round. The target tasks for the current round are selected through sampling or a greedy approach. After n rounds of iterations, an output task sequence consisting of n target tasks is obtained.

The Critic stage shares similarities with the Actor stage in the first half. After performing the embedding operation on the original input data, it is also input into the encoder. Then the final hidden layer state h of the encoder is obtained, which is input into a deep neural network (DNN). The input layer of the deep neural network has the same number of neurons as the number of hidden layers, which corresponds to the length of the input service requests sequence. The output layer of the DNN consists of a single neuron, representing the predicted value of the critic network. This predicted value serves as the baseline function in the strategy gradient descent algorithm, assisting the Actor-network during training. The Critic-network is trained independently using the stochastic gradient descent algorithm.

IV. Experiments

In this section, we conducted experiments using three real-world datasets. We designed three sets of experiments to address three specific research questions.

- RQ1: Is there a correlation between different optimization objectives?
- RQ2: Can the proposed RSNet perform better than current methods at various data scales?
- RQ3: Can PSNet effectively solve the multi-objective optimization problem of edge service requests scheduling?

In order to investigate the relationship between the three optimization objectives mentioned in this problem, we formulated RQ1 to explore their interrelationships through a two-by-two comparison experiment. Building upon the insights gained from RQ1, we proceeded to conduct experiments using three distinct real-world datasets. The objective was to validate the effectiveness of our proposed method by comparing it with other existing methods. The results obtained from RQ1 and RQ2 informed our investigation into whether the proposed PSNet method is applicable and beneficial for solving multi-objective optimization problems, which constituted RQ3.

A. Data Sets

We constructed experiments on three publicly available real data sets to validate the questions above. The data marked with * in the Table I indicates the data constructed from the existing data combined with the real environment; the data marked with # indicates the data extracted and calculated from the existing data.

EUA-dataset [18]: In the Melbourne Central Business District, we utilized the EUA-dataset, which provides the geographic positions of 816 mobile users and 125 base stations. To simulate real-world scenarios, we generated 500,000 service requests from the 816 end-users included in the dataset. These service requests were directed towards the 125 base stations, which served as edge servers for receiving and processing the requests. The simulation took into account the end-user and base station information available in the dataset, as well as the characteristics of service requests in the actual environment.

TABLE I
EXPERIMENTAL DATA SETS AND PROCESSING

Data	#Users	#Edge Servers	#Trace	#Requests	#Records
EUA	816	125	0	500,000*	500,000
Google Cluster Trace	500*	12,500 (100#)	1,000,000	400,000#	400,000
Alibaba Cluster Trace	500*	4,000 (100#)	1,000,000	400,000*	400,000*

Google Cluster Trace [19]: This dataset originates from an edge-cloud cooperation system, which was initially designed for cloud environments but later expanded to incorporate data from edge contexts. The trace information includes over 1,000,000 records from 12,500 machines within the Google cloud cluster. To focus on edge servers, we randomly sampled 100 out of the 12,500 machines. We extracted all the records marked as successful in the trace and considered them as service requests. This resulted in a total of 400,000 service request records. Finally, we selected 500 end users to send service requests based on the relevant characteristics of the successful service request records.

Alibaba Cluster Trace [20]: The dataset used in this study contains information about the number of CPU cores, RAM size, and disk size of multiple edge servers. Additionally, it includes dynamic server information, such as the resource utilization rate at different timestamps. Specifically, the Alibaba Cluster Trace dataset logs the load of 4,000 machines within the Alibaba cluster over an 8-day period, resulting in over 1,000,000 records. To simplify the dataset, we randomly selected 100 machines from the original 4,000 machines to represent the edge servers. By combining the machine load data with the characteristics of service requests in a real-world environment, we generated a dataset consisting of 400,000 service request records. Furthermore, we selected 500 end users based on the combined characteristics of the machine load and the service request to send service requests.

B. Baseline Approaches

The five alternatives listed below are chosen for comparison with our proposed PSNet. The UNIX operating system employs the multi-level feedback queue scheduling technique, which is a standard CPU processor scheduling mechanism.

- FCFS [21] (First-Come, First-Served): The edge server processes all incoming service requests in the order they are received. To put it another way, service requests that arrive initially are processed immediately, while those that come later are processed after that.
- HRRN [22] (High Response Ratio Next): The greater the response ratio, the higher the priority it will be given in terms of waiting time and running time.
- OnPQ [19]: The scheduling process is depicted as a Markov decision-making process. Energy usage and average wait times for service requests are both lowered by using the Q-Learning algorithm.
- OnDisc [23] This sensitivity to delay is shown as a weighted reaction time. The notion of greatest residual density first (HRDF) is used during scheduling.
- RLPNet [16]: A multi-objective optimization model based on pointer networks to solve the scheduling of service requests problem. The WLC model reduces the complex multi-objective optimization issue to a simpler one.

We compared with the FCFS and HRRN algorithms, since the subject presented in this work is also fundamentally a service requests scheduling problem. OnPQ, OnDisc, and RLPNet, the three most recent machine learning-based techniques, were also used for comparison.

C. Parameter Settings

We have set some essential parameters in for the experimental process. For the training part of the network, the network consists of Actor network and Critic network, and both Actor and Critic are RNN networks with LSTM as the unit. The embedding layer and hidden layer of the LSTM have 128 neurons each, and the softmax temperature is set to 1.0. The optimizer used is Adam, which combines the strengths of Adagrad and RMSprop to handle sparse gradients and non-smooth targets, respectively. Adam also adjusts the learning rate automatically and converges faster, making it suitable for complex networks. For the sampling strategy, a greedy approach is employed. The learning rate determines the size of each step, and an appropriate value needs to be chosen for tuning. The initial learning rate is 0.0001, with a decay coefficient of 0.96 and a decay period of 1,000. To prevent large jitter during the training process, the $L2$ regularization method is applied. Due to the large amount of data, a smaller batch size of 128 is chosen to avoid memory overflow. The learning step is 20,000.

D. Optimization of Objective Correlation Analysis (RQ1)

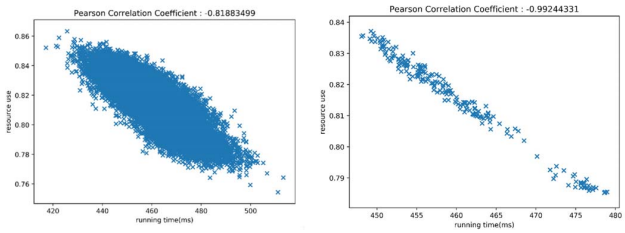
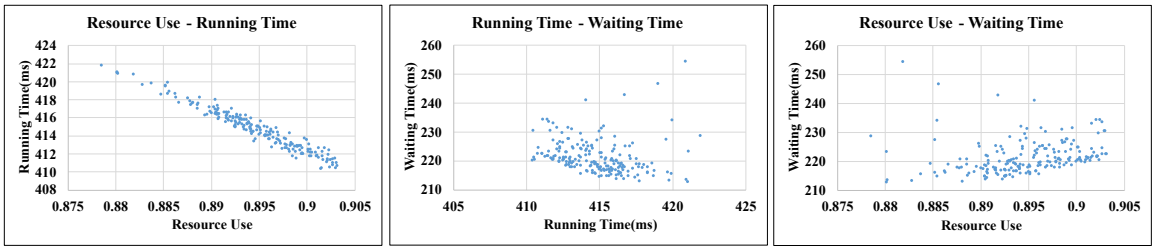


Fig. 4. Correlation verification of resource utilization with total running time

Through the analysis of service request scheduling problems in real environments, we observe the following patterns: as the resource utilization increases, the running time tends to decrease accordingly, while as the resource utilization decreases, the running time tends to increase accordingly. The analogy can be made with the classical boxing problem. When the total amount of goods is fixed, there is a correlation between space utilization inside the box and the number of required packages. Specifically, higher space utilization leads to a smaller number



(a) (b) (c)

Fig. 5. Experimental comparison results of the correlation between optimization objectives

of required packages, while lower space utilization necessitates a larger number of boxes.

To provide empirical evidence for our study, we investigated the correlation between two optimization objectives: resource utilization and total running time. In order to establish a clear focus, we modified the original reward formula to solely consider the optimization objective of total running time. This modification enabled RLPNet to undergo independent training with a specific emphasis on this objective. Throughout the training process, we retained both the resource utilization data and the total running time data derived from each training iteration.

There are several methods to analyze the correlation of parameters, the most commonly used is the Pearson correlation coefficient, who measures the linear correlation between two variables (e.g., α and β). It is also called a parametric correlation test because it depends on the distribution of the data. This method can only be applied when both α and β are sampled from normal distributions. The plot of $\beta = f(\alpha)$ is called a linear regression curve.

The correlation of parameters is analyzed using the Pearson correlation, which is defined as:

$$r = \frac{\sum(\alpha - m_\alpha)(\beta - m_\beta)}{\sqrt{\sum(\alpha - m_\alpha)^2 \sum(\beta - m_\beta)^2}}, \tag{13}$$

where α and β are two vectors of length n, and m_α and m_β correspond to the mean values with α and β, respectively. The *p*-value of the correlation can be determined by Equation (14). We first queried the correlation coefficient table, where the degrees of freedom are: $df = n-2$ and n is the number of observations (length) in the α and β variables. Next, we calculated the ζ-value using the following approach: the corresponding *p*-value is determined by referring to the ζ-distribution table.

$$\zeta = \frac{r}{\sqrt{1-r^2}}\sqrt{n-2}. \tag{14}$$

As is commonly known, when the *p*-value is closer to 1 or -1, it indicates a higher correlation between the two variables. On the other hand, when the *p*-value is closer to 0, it suggests a lower correlation between the variables.

In Fig. 4, we can see the results of the correlation verification of the resource utilization with the other two optimization objectives. The Pearson correlation coefficient for the original result is -0.8188. However, when we average the results every 100 rounds out of the 20,000 rounds, the Pearson correlation coefficient becomes -0.9924.

Additionally, in our correlation experiment, PSNet undergoes independent training consisting of 20,000 rounds. Throughout the training process, we save the data for resource utilization, total running time, and average waiting time from each round. We then utilize this data to generate a scatter diagram, as depicted in Fig. 5 (a), showcasing the resource utilization rate and total running time for every 100 rounds. Moreover, we conduct Pearson correlation analysis based on Equation (15).

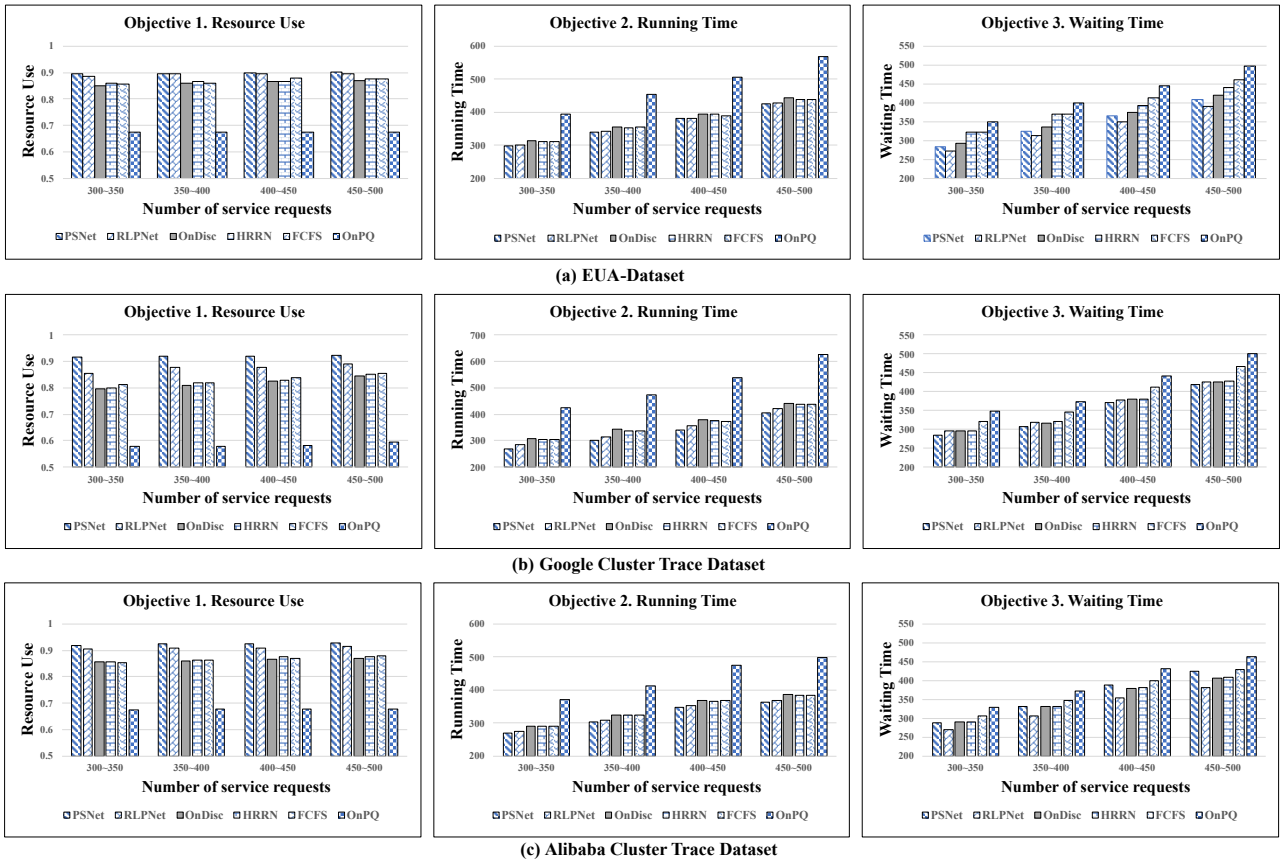
$$\rho(X,\ Y) = \frac{n\sum_1^n XY - \sum_1^n XY}{\sqrt{(n\sum_1^n X^2 - (\sum_1^n X)^2)(n\sum_1^n Y^2 - (\sum_1^n Y)^2)}}, \tag{15}$$

where X and Y are the two variables to be analyzed. The resulting Pearson correlation coefficient is -0.97885. We took out the data of resource utilization rate and average waiting time to draw a scatter diagram, as shown in Fig. 5(b), conducted Pearson correlation analysis, and got a Pearson correlation coefficient of -0.00805. We took the data of total running time and average waiting time to draw a scatter diagram, as shown in Fig. 5(c), and performed Pearson correlation analysis. The Pearson correlation coefficient is 0.032777.

Through the aforementioned experiments, we observed a negative correlation between running time and resource utilization. Specifically, as the server's running time increases, the utilization of resources decreases. This finding implies that optimizing both metrics can be achieved by enhancing resource utilization or reducing running time. As a result, our optimization objective becomes more precise and concise.

E. PSNet Performance on Different Data Sets (RQ2)

In terms of resource consumption, running time, and waiting time, RSPNet outperforms competing algorithms regardless of the value of n or m, as shown by the experimental findings. Because OnDisc seeks to reduce weighted response time, its latency performance is often superior to that of other algorithms, except for RLPNet. However, the OnDisc's success with other objectives is somewhat subpar. Since OnPQ must



(a) EUA-Dataset

(b) Google Cluster Trace Dataset

(c) Alibaba Cluster Trace Dataset

Fig. 6. Experimental results of controlling the number of edge servers $m = 5$

consolidate numerous service requests into a single batch before passing it to the edge server for processing, the next batch may only be processed after the previous batch has been completed. Since the execution time needed by specific service requests varies, an execution strategy will result in more fantastic idle time inside an edge server, resulting in poor performance of OnPQ for each optimization target. The performance of HRRN and FCFS, two conventional work scheduling algorithms inside operating systems, is pretty excellent and within the range of expectations. Regarding waiting time, HRRN is marginally superior to FCFS owing to the consideration of the response ratio. First, the number of edge servers is controlled at m and the number of service requests is divided into four groups: (300∼350),(350∼400), (400∼450), and (450∼500). We verify the performance of different algorithms under each optimization objective, as shown in Fig. 6. Among them, Fig. 6(a) shows the performance effect under the EUA dataset; Fig. 6(b) shows the performance effect under the Google Cluster Trace dataset; And the Fig. 6(c) shows the performance effect under the Alibaba Cluster Trace dataset.

As can be observed from the figure, the level of resource use for each of the various approaches steadily grows along with the number of service requests. The approach that we have proposed for PSNet is the one that has the highest resource usage on all three different datasets, basically above 90%. The amount of time it takes for the system to run increases proportionately as the number of service requests increases from 300 to 500; our PSNet takes the least amount of time for varying data sets containing the same number of tasks. The amount of time that customers have to wait in line is gradually growing longer as the number of services available rises. Our method performs the best on the Google trace dataset with the shortest average waiting time, but the RLPNet's method performs better on the EUA dataset and the Alibaba trace dataset because we add the scoring mechanism to the pointer network. The approach used by RLPNet in synthesizing the multi-objective optimization results is WLC, and although as many as twenty-seven enumeration experiments are conducted in selecting the three weight coefficients, there is still no way to guarantee that the final results are optimal. Our proposed PSNet is better than RLPNet in terms of the resource utilization and running time on the Alibaba dataset, except for the waiting time. In fact, this is a normal phenomenon and it precisely shows that RLPNet has some limitations in multi-objective optimization by weighting parameters, and the

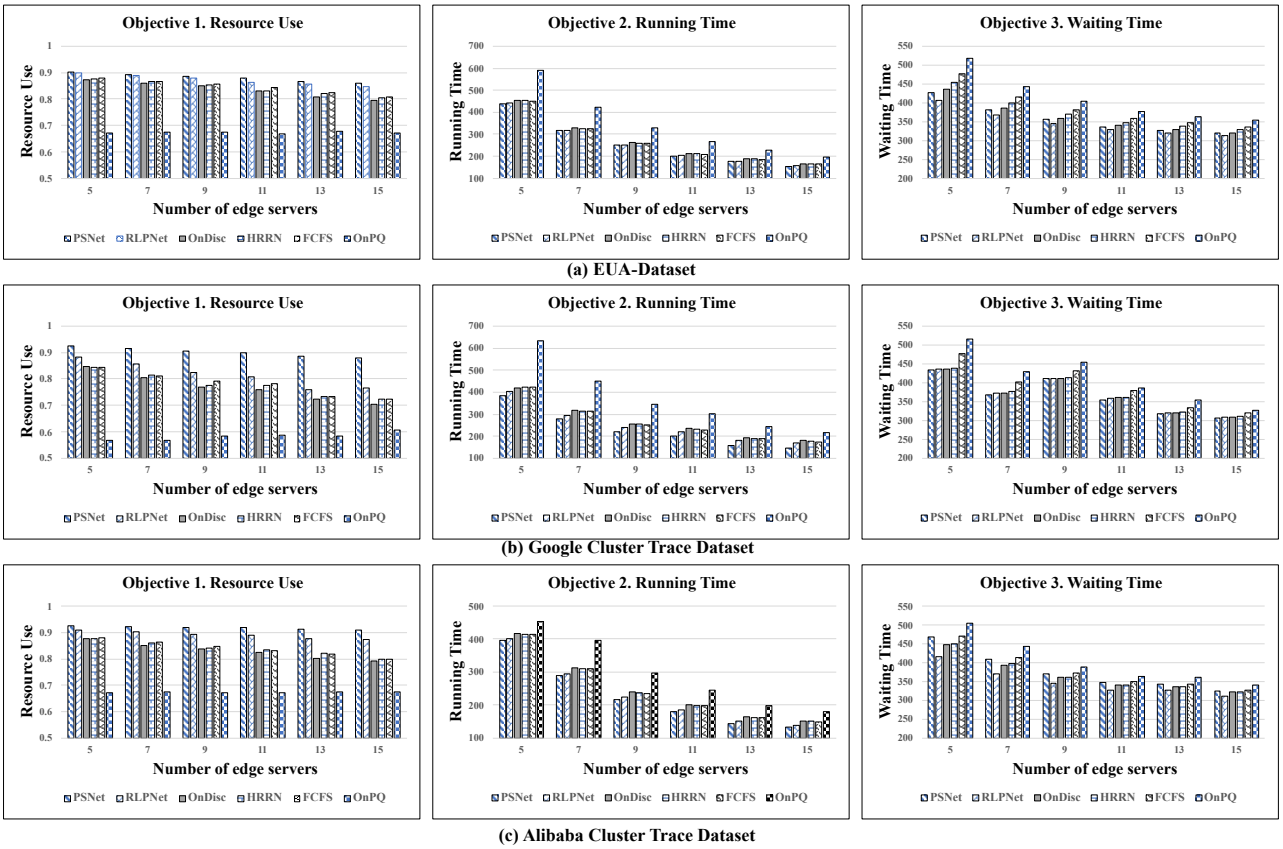


Fig. 7. Experimental results for controlling the number of service requests $n = 500$

selected set of metrics is over-emphasized.

Based on the observation from Fig.6, it is evident that PSNet exhibits a shorter waiting time in comparison to OnDisc, OnPQ, HRRN, and FCFS approaches. However, it has a longer waiting time compared to the RLPNet approach. Both PSNet and RLPNet utilize deep reinforcement learning and pointer network methods, demonstrating the effectiveness of these techniques. The reason for PSNet's longer waiting time is due to its increased use of pointer network computing processes. Although this results in improved resource utilization and reduced running time, it also lengthens the waiting time in comparison to RLPNet.

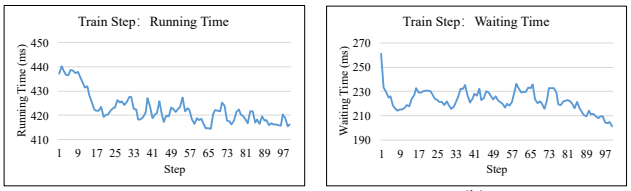
To validate the algorithm performance under different numbers of edge servers, we have designed the second set of experiments. This experiment involves testing the algorithm using varying numbers of edge servers ($m = 5, 7, 9, 11, 13, 15$) and comparing their performance. The reason for choosing $m = 500$ is that this provides a large enough sample for a more adequate training of our method, while we also chose other values and conducted experiments with the same conclusion. The performance of different algorithms under each optimization objective is verified, as shown in Fig. 7. Fig. 7(a) shows the performance effect under the EUA dataset, Fig. 7(b) shows the performance effect under the Google Cluster Trace data set, and Fig. 7(c) shows the performance effect under the Alibaba Cluster Trace data set.

We have formulated the service request scheduling problem in the edge environment as a serialized multi-objective optimization problem and proposed a new pointer network model, named PSNet, to tackle this problem. Our experiments show that PSNet outperforms RLPNet in each optimization objective on the Google Cluster Trace dataset. Moreover, PSNet also outperforms RLPNet on the EUA dataset and the Alibaba dataset, exhibiting superior performance in terms of resource utilization and total running time, but slightly weaker performance on average waiting time. These findings suggest that PSNet has the potential to be a promising solution for service request scheduling problems in the edge. However, it should be noted that the effectiveness of the proposed model may depend on the specific dataset and problem characteristics.

F. Analysis of The Effectiveness of Multi-objective Optimization (RQ3)

In order to verify the effectiveness of PSNet for solving multi-objective optimization problems, we designed a set of independent experiments: PSNet was trained using the Google dataset with a batch size of 128 and 10,000 rounds. The values of the total running time and the average waiting time after

each round of training were counted and averaged every 100 rounds to form a training curve, as shown in Fig 8. The training curve of the total running time is shown in Fig. 8(a), and the training curve of the average waiting time is shown in Fig. 8(b). As can be seen in Fig. 8(a), when the number of



(a) (b)

Fig. 8. The training curve of PSNet

training rounds is relatively small, the overall running time of the system is still relatively high, with the highest value around 440. As the training proceeds, it decreases rapidly in the range of steps taken from 1-17, fluctuates down from inside the range of 21-97, and finally stabilizes at around 415.

Fig. 8(b) shows that as the training steps increase, the overall waiting time is decreasing, especially in the first ten rounds at the very beginning, and his decrease is particularly obvious. Subsequently, between 10 and 80, the average waiting time fluctuates but is stable overall. From 81 to 100, the average wait time decreases again.

The experimental results show that PSNet can successfully optimize the two metrics of total running time and average waiting time after training, which shows the effectiveness of PSNet on the multi-objective optimization problem.

V. Related work

Task scheduling in edge computing has been a hot topic in the past years. The works in this area can be categorized into two types: strategy-based and deep learning-based approaches.

A. Strategy-based Approaches

In edge computing, strategy-based algorithms [4]–[7], [24], [25] have been widely used in job scheduling. For example, the multi-service task computing offloading algorithm (MTCOA) proposed by Song *et al.* [5], contemplating computational cost and resource use, establishes optimal solutions to offload multi-service tasks. Wang *et al.* [6] suggested a task processing delay-based simulated annealing fusion approach for edge computing resource efficiency, processing delays, and uneven system load. In [7], Liao *et al.* characterized the dependence linkages between application activities for mobile edge computing settings as directed acyclic networks and resolved prioritization-based application assignments and scheduling issues for the timeout rate of jobs using an online method. However, this strategy only focuses on optimizing the workflow completion time. FCFS [21] is a method where all incoming service requests are processed in the order they were received by the edge server. In [22], Tan *et al.* prioritize jobs with higher response ratios regarding waiting and running times.

Overall, the above-mentioned strategy-based approaches have contributed significantly to advancing task scheduling research in edge computing. However, their analysis is often limited to optimization problems with narrow objectives, overlooking the optimization of multiple objectives. Due to their longer iteration times, strategy-based algorithms may also be incompatible with low-latency service requests at the edge.

B. Deep Learning-based Approaches

In recent years, the applications of deep learning to resource scheduling and service request scheduling in the context of edge computing and the IoT have received increasing attention from researchers. A number of studies have demonstrated that deep learning-based scheduling algorithms can outperform traditional strategy-based algorithms in terms of performance. For example, Wang *et al.* [26] utilized tools from matches theory and deep reinforcement learning to create a two-timescale mechanism for resource scheduling. They proposed A3c-do, a residual recurrent neural network that can allocate resources in the IoT efficiently. Cai *et al.* [11] suggested using distributed convolutional neural networks to optimize the interaction of edge network devices. Zou *et al.* [14] further refined this model by utilizing the Markov decision process and Asynchronous-Advantage-Actor-criticA3C deep reinforcement learning. In addition to these studies, Zheng *et al.* [27] used Deep-Q-Network (DQN) techniques to handle a high dimensional and complex workload scheduling problem. Some studies have proposed novel approaches to address the challenges of service request scheduling in edge settings. For instance, OnPQ ([19]) uses the Q-Learning algorithm to reduce energy usage and average wait times for service requests by considering the sensitivity to delay as a weighted reaction time. Another novel approach proposed by Zhao *et al.* [16] uses pointer networks to solve the multi-objective optimization problem of scheduling service requests. The weighted linear combination (WLC) model is used to reduce the complex multi-objective optimization issue to a simpler one. Overall, these studies have demonstrated the potential of deep learning techniques to improve resource scheduling and service request scheduling in edge computing and IoT environments.

In summary, traditional multi-objective optimization algorithms have limitations in addressing the correlation between multiple optimization objectives, and they often fail to identify a single best parameter configuration in the generated Pareto optimal solution sets. For example, in the case of RLPNet, the optimization objectives are treated equally, and parameter experiments are conducted to select a balanced set of parameters. However, this approach does not fully cover the research landscape, and selecting appropriate parameter ranges still requires domain expertise. To overcome these challenges, we propose a deep reinforcement learning method that incorporates multiple pointer networks and scoring mechanisms for optimal scheduling. Our approach addresses the complexity of multi-objective optimization problems by optimizing different metrics in the pointer network model.

VI. Conclusion and Future Work

In this paper, we propose PSNet as a solution to the service request task scheduling problem in edge computing. PSNet aims to find the optimal parameter ratio by utilizing a linear weighting approach to address the multi-objective optimization problem. To accommodate the characteristics of the multi-objective optimization problem, we divide the pointer network model into multiple networks and optimize them for different metrics. Additionally, we introduce a scoring mechanism that considers different optimization objectives, leading to more insightful outcomes. Our experiments conducted on three real-world datasets demonstrate that our method outperforms several state-of-the-art approaches.

In the future, we intend to validate our approach across a broader range of multi-objective optimization scenarios and explore a service request scheduling system that effectively coordinates between the edge and the cloud, addressing the pressing challenge in edge-cloud collaborative systems.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (Nos. 62032016, 61832014, and 61972292) and the Key Research and Development Program of Hubei Province (No. 2021BAA031). Bing Li and Lei Fu are corresponding authors of the paper.

References

[1] B. Han, V. Sciancalepore, Y. Xu, D. Feng, and H. D. Schotten, "Impatient queuing for intelligent task offloading in multiaccess edge computing," *IEEE Trans. Wirel. Commun.*, vol. 22, no. 1, pp. 59–72, 2023.

[2] J. Deng, B. Li, J. Wang, and Y. Zhao, "Microservice pre-deployment based on mobility prediction and service composition in edge," in *2021 IEEE International Conference on Web Services, ICWS*, 2021, pp. 569–578.

[3] L. Ai, B. Tan, J. Zhang, R. Wang, and J. Wu, "Dynamic offloading strategy for delay-sensitive task in mobile-edge computing networks," *IEEE Internet Things J.*, vol. 10, no. 1, pp. 526–538, 2023.

[4] X. Xia, H. Qiu, X. Xu, and Y. Zhang, "Multi-objective workflow scheduling based on genetic algorithm in cloud environment," *Inf. Sci.*, vol. 606, pp. 38–59, 2022.

[5] S. Song, S. Ma, J. Zhao, F. Yang, and L. Zhai, "Cost-efficient multi-service task offloading scheduling for mobile edge computing," *Appl. Intell.*, vol. 52, no. 4, pp. 4028–4040, 2022.

[6] S. Wang, W. Wang, Z. Jia, and C. Pang, "Flexible task scheduling based on edge computing and cloud collaboration," *Comput. Syst. Sci. Eng.*, vol. 42, no. 3, pp. 1241–1255, 2022.

[7] H. Liao, X. Li, D. Guo, W. Kang, and J. Li, "Dependency-aware application assigning and scheduling in edge computing," *IEEE Internet Things J.*, vol. 9, no. 6, pp. 4451–4463, 2022.

[8] N. Srinivas and K. Deb, "Muiltiobjective optimization using nondominated sorting in genetic algorithms," *Evolutionary Computation*, vol. 2, no. 3, pp. 221–248, 1994.

[9] X. Zhao, X. Guo, Y. Zhang, and W. Li, "A parallel-batch multi-objective job scheduling algorithm in edge computing," in *IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2018, pp. 510–516.

[10] S. Tuli, S. Ilager, K. Ramamohanarao, and R. Buyya, "Dynamic scheduling for stochastic edge-cloud computing environments using A3C learning and residual recurrent neural networks," *IEEE Trans. Mob. Comput.*, vol. 21, no. 3, pp. 940–954, 2022.

[11] S. Cai, D. Wang, H. Wang, Y. Lyu, G. Xu, X. Zheng, and A. V. Vasilakos, "Dynacomm: Accelerating distributed CNN training between edges and clouds through dynamic communication scheduling," *IEEE J. Sel. Areas Commun.*, vol. 40, no. 2, pp. 611–625, 2022.

[12] H. Dai, J. Wu, Y. Wang, and C. Xu, "Towards scalable and efficient deep-rl in edge computing: A game-based partition approach," *J. Parallel Distributed Comput.*, vol. 168, pp. 108–119, 2022.

[13] S. Ide and B. O. Apduhan, "A framework of an rl-based task offloading mechanism for multi-users in edge computing," in *Computational Science and Its Applications - ICCSA 2022 Workshops - Malaga, Spain, July 4-7, 2022, Proceedings, Part III*, ser. Lecture Notes in Computer Science, O. Gervasi, B. Murgante, S. Misra, A. M. A. C. Rocha, and C. Garau, Eds., vol. 13379. Springer, 2022, pp. 103–114.

[14] J. Zou, T. Hao, C. Yu, and H. Jin, "A3c-do: A regional resource scheduling framework based on deep reinforcement learning in edge scenario," *IEEE Transactions on Computers*, vol. 70, no. 2, pp. 228–239, 2021.

[15] T. Zheng, J. Wan, J. Zhang, and C. Jiang, "Deep reinforcement learning-based workload scheduling for edge computing," *J. Cloud Comput.*, vol. 11, p. 3, 2022.

[16] Y. Zhao, B. Li, J. Wang, D. Jiang, and D. Li, "Integrating deep reinforcement learning with pointer networks for service request scheduling in edge computing," *Knowledge-Based Systems*, vol. 258, p. 109983, 2022.

[17] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems*, 2015, pp. 2692–2700.

[18] P. Lai, Q. He, M. Abdelrazek, F. Chen, J. G. Hosking, J. C. Grundy, and Y. Yang, "Optimal edge user allocation in edge computing with variable sized vector bin packing," in *Service-Oriented Computing - 16th International Conference, ICSOC*, vol. 11236, 2018, pp. 230–245.

[19] J. Lu, X. Guo, X. Zhao, and H. Zhou, "A parallel tasks scheduling algorithm with markov decision process in edge computing," in *Green, Pervasive, and Cloud Computing - 15th International Conference, GPC*, vol. 12398, 2020, pp. 362–375.

[20] M. Xu, Z. Fu, X. Ma, L. Zhang, Y. Li, F. Qian, S. Wang, K. Li, J. Yang, and X. Liu, "From cloud to edge: a first look at public edge platforms," in *IMC '21: ACM Internet Measurement Conference, Virtual Event, USA*, 2021, pp. 37–53.

[21] J. P. Champati, H. Al-Zubaidy, and J. Gross, "Statistical guarantee optimization for aoi in single-hop and two-hop FCFS systems with periodic arrivals," *IEEE Trans. Commun.*, vol. 69, no. 1, pp. 365–381, 2021.

[22] R. Chen, L. Cui, Y. Zhang, J. Chen, K. Yao, Y. Yang, C. Yao, and H. Han, "Delay optimization with FCFS queuing model in mobile edge computing-assisted UAV swarms: A game-theoretic learning approach," in *2020 International Conference on Wireless Communications and Signal Processing (WCSP)*, 2020, pp. 245–250.

[23] H. Tan, Z. Han, X. Li, and F. C. M. Lau, "Online job dispatching and scheduling in edge-clouds," in *2017 IEEE Conference on Computer Communications, INFOCOM*, 2017, pp. 1–9.

[24] T. Li, J. Shi, W. Deng, and Z. Hu, "Pyramid particle swarm optimization with novel strategies of competition and cooperation," *Appl. Soft Comput.*, vol. 121, p. 108731, 2022.

[25] J. Sun, L. Yin, M. Zou, Y. Zhang, T. Zhang, and J. Zhou, "Makespan-minimization workflow scheduling for complex networks with social groups in edge computing," *J. Syst. Archit.*, vol. 108, p. 101799, 2020.

[26] Z. Wang, Y. Wei, Z. Feng, F. R. Yu, and Z. Han, "Resource management and reflection optimization for intelligent reflecting surface assisted multi-access edge computing using deep reinforcement learning," *IEEE Trans. Wirel. Commun.*, vol. 22, no. 2, pp. 1175–1186, 2023.

[27] X. Kong, G. Duan, M. Hou, G. Shen, H. Wang, X. Yan, and M. Collotta, "Deep reinforcement learning-based energy-efficient edge computing for internet of vehicles," *IEEE Trans. Ind. Informatics*, vol. 18, no. 9, pp. 6308–6316, 2022.