Attention-Based Deep Reinforcement Learning for Edge User Allocation

Jiaxin Chang, Jian Wang, *Member, IEEE*, Bing Li, *Member, IEEE*, Yuqi Zhao, and Duantengchuan Li

***Abstract*—Edge computing, a recently developed computing paradigm, seeks to extend cloud computing by providing users minimal latency. In a mobile edge computing (MEC) environment, edge servers are placed close to edge users to offer computing resources, and the coverage of adjacent edge servers may partially overlap. Because of the restricted resource and coverage of each edge server, edge user allocation (EUA), i.e., determining the optimal way to allocate users to different servers in the overlapping area, has emerged as a major challenge in edge computing. Despite the NP-hardness of obtaining an optimal solution, it is possible to evaluate the quality of a solution in a short amount of time with given metrics. Consequently, deep reinforcement learning (DRL) can be used to solve EUA by attempting numerous allocations and optimizing the allocation strategy depending on the rewards of those allocations. In this study, we propose the Dual-sequence Attention Model (DSAM) as the DRL agent, which encodes users using self-attention mechanisms and directly outputs the probability of matching between users and servers using an attention-based pointer mechanism, enabling the selection of the most suitable server for each user. Experimental results show that our method outperforms the baseline approaches in terms of allocated users, required servers, and resource utilization, and its running speed meets real-time requirements.**

***Index Terms*—Edge user allocation, deep reinforcement learning, edge computing.**

I. Introduction

EDGE computing is a novel computing paradigm that processes data at the network's edge. This reduces the distance between the data source and processing, which in turn lowers the latency between end-users and servers. The reduction in latency benefits numerous latency-sensitive applications, such as autonomous driving and online gaming. Mobile edge computing (MEC) is a specific technology that provides computing resources within a radio access network (RAN) close to end devices, allowing users to offload compute-intensive tasks to neighboring edge servers. However, compared to cloud servers, edge servers have restricted coverage and limited resource capacity [1]. Efficient resource allocation in the MEC is critical to support users' task offloading.

Previous studies on computation offloading, such as [2] and [3], mainly focused on resource allocation for multiple users on a single edge server, with the aim of minimizing task completion time and resource demand. Wang et al. [4] considered allocation decisions in a scenario with multiple edge servers deployed with different applications, with the goal of achieving resource and network load balancing. Additionally, Dai et al. [5] explored a coordinated architecture of end-edge-cloud that supports both local computing and computation offloading to the edge or cloud, with the objective of minimizing system energy consumption through offloading decisions. These studies primarily focus on factors such as task completion time, resource or energy consumption, channel selection, and transmission power during data transfer, from the perspective of devices or edge infrastructure providers. In contrast, this paper focuses on computation offloading from the service provider's perspective. We assume that the edge infrastructure providers can offer low-latency edge servers with fixed coverage and limited computing resources (such as CPU, memory, storage, and bandwidth) to service providers. Service providers need to rent edge servers to provide low-latency services to users and must consider how to rent fewer servers to serve more users efficiently. This requirement necessitates an efficient user allocation strategy, which is referred to as the edge user allocation (EUA) problem [6].

The EUA problem typically considers two constraints: coverage limits and resource capacity constraints. In addition, from the service provider's perspective, there are two objectives: enabling more users to connect to edge servers to improve the quality of service (QoS) and renting fewer servers to save costs. It is a multi-objective optimization problem and proved to be NP-hard [6]. In this paper, we study quasi-static scenarios like [6], [7], [8], and [1], where users do not migrate from the coverage of one edge server to another. For example, the users could be intelligent speakers, family webcams, and IoT devices with fixed locations.

Prior studies on the EUA problem have some limitations. Some researchers used mathematical optimization methods such as Lexicographic Goal Programming techniques [6], while others used traditional metaheuristic algorithms such as genetic algorithms [9]. These methods often suffer from the exponential growth of the search space and a heavy computational burden, especially in large-scale scenarios. To provide

Manuscript received 28 October 2022; revised 22 March 2023; accepted 29 June 2023. Date of publication 4 July 2023; date of current version 7 February 2024. This work is supported by the National Natural Science Foundation of China (Nos. 62032016, 61832014, and 61972292). The associate editor coordinating the review of this article and approving it for publication was J. Hwang. *(Corresponding authors: Jian Wang; Bing Li.)*

Jiaxin Chang, Jian Wang, Yuqi Zhao, and Duantengchuan Li are with the School of Computer Science, Wuhan University, Wuhan 430072, China (e-mail: ccjjxx99@whu.edu.cn; jianwang@whu.edu.cn; yuqizhao@whu.edu.cn; dtclee1222@whu.edu.cn).

Bing Li is with the School of Computer Science, Wuhan University, Wuhan 430072, China, and also with Hubei Luojia Laboratory, Wuhan 430072, China (e-mail: bingli@whu.edu.cn).

Digital Object Identifier 10.1109/TNSM.2023.3292272

real-time solutions, some researchers have used heuristic methods such as [1] and [7], which utilize optimized greedy algorithms and can provide solutions in a short time, but there is still a gap between the obtained solutions and the optimal solutions. Currently, no method can provide near-optimal solutions in a short time. Due to the NP-hardness of the EUA problem, it is impracticable to achieve an optimal solution in polynomial time. However, it is relatively easy to assess the quality of a solution. By testing a large number of allocations and utilizing their scores to optimize the allocation policy, we may arrive at a near-optimal allocation policy. Deep reinforcement learning (DRL) has shown significant potential in addressing this problem. Leveraging the powerful fitting ability of deep neural networks, DRL can generate optimal allocation policies when receiving inputs from users and servers. Furthermore, reinforcement learning can optimize the neural network by attempting a large number of allocations with the computed rewards based on various metrics.

Many studies in the field of computation offloading have used DRL, but their methods cannot be directly applied to the EUA problem due to different scenarios with different states and actions. The different inputs and outputs necessitate the redesign of neural networks. Additionally, most traditional DRL agents use combinations of multilayer linear layers (fully connected networks), which have two main drawbacks. First, if all users are encoded through linear layers, the embedding of each user only contains its own information, which cannot provide the neural network with more complete global information, resulting in potentially local optimal allocation policies. Second, the output dimension of the linear layer is also fixed. The conventional practice is to set the number of servers as the output dimension, and the probability of selecting each server is determined by the output value of each dimension. However, this kind of method fails to exploit the dual-sequence nature of the problem and does not directly output the matching degree between users and servers, which may result in unsatisfactory server selection for users. Moreover, these methods necessitate changing the network structure and training a new neural network when the number of servers changes. Therefore, we propose a dual-sequence attention model to solve these two problems. First, we use self-attention to encode users, making each user's embedding incorporate information from related users. Second, when selecting a server for a given user, we use a pointer mechanism (a simplified attention mechanism) to dynamically calculate the matching degree between each server and the user. The introduction of the attention mechanism has tremendous potential in the context of MEC, given its dynamic and flexible nature. This mechanism enables neural networks to effectively handle the variability and adaptability of inputs and outputs. Specifically, when the number or configuration of servers changes, the neural network empowered by the attention mechanism can output the probability of the corresponding server without changing the network structure. In this work, we propose an attention-based deep reinforcement learning method to solve the EUA problem. Our main contributions are summarized as follows:

- We model the EUA problem as a Markov decision process (MDP) and use a DRL approach to solve it, which can obtain near-optimal solutions in a short time.
- We design a dual-sequence attention model (DSAM) as the agent for DRL. DSAM uses a self-attention mechanism to encode users and an attention-based pointer mechanism to output the probability of matching between users and servers. It can handle variable-length user and server sequences and make decisions on which server to allocate each user to.
- We conduct a series of experiments on a real-world edge server dataset to evaluate the performance of our proposed DSAM. The results show that our approach outperforms baseline approaches.

The paper is organized as follows. Section II describes an example of the EUA scenario and formalizes the EUA problem as an MDP. Section III models our proposed DSAM. Section IV introduces the DRL algorithm to train our model. Section V is the experimental part, which evaluates the performance of our approach. Section VI reviews the related work. Finally, Section VII concludes this paper and looks ahead to future work.

II. Problem Statement

A. Motivating Example

One typical application scenario of the EUA problem is the smart traffic management system [10]. Traffic cameras in the city capture a large amount of video data daily. If these data are uploaded to the cloud server for processing, it will occupy a lot of bandwidth and have high latency. Edge computing can solve this problem. Traffic cameras in different locations may connect to edge servers for video processing. When connected to edge servers, the camera requested service will occupy a certain amount of CPU and memory for video processing, use a certain amount of bandwidth to upload videos and occupy much disk storage for recent monitoring records. Since different cameras execute different tasks, the amount of resource requests for corresponding services also varies. Service providers rent edge servers from edge infrastructure providers to offer video processing and analysis services. Each traffic camera (later referred to as a more general term "user") can only connect to one edge server that can cover its location and has enough capacity. The coverage areas of different edge servers overlap, and users in the overlapping areas have multiple choices. However, due to the limited capacity of edge servers, the choice of one user may affect other users. Service providers hope to rent fewer servers to save costs while allowing as many users as possible to connect to edge servers, which requires an efficient allocation strategy. We will use an example to illustrate this scenario next.

In Fig. 1, the user $\boldsymbol{u}_1$ is located in the overlapping area of edge servers $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$. His resource requirements are represented by the values (2, 3, 3, 4) for CPU, memory, storage, and bandwidth, respectively. If we allocate $\boldsymbol{u}_1$ to $\boldsymbol{e}_1$ whose capacity is (10, 12, 9, 15), the remaining capacity of $\boldsymbol{e}_1$ will be (8, 9, 6, 11). Consequently, $\boldsymbol{e}_1$ can only handle either $\boldsymbol{u}_2$ or

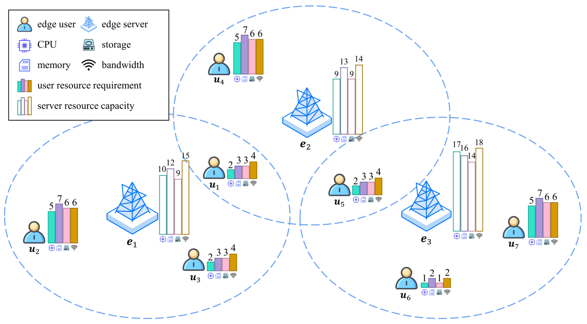


Fig. 1. Example edge computing scenario.

u_3 because the sum of their resource requirements is (7, 10, 9, 10), and the memory and storage capacity of e_1 is insufficient. One of them must connect to a cloud server, leading to higher latency. But if we allocate u_1 to e_2, we can allocate both u_2 and u_3 to e_1. In the meanwhile, u_4 can be allocated to e_2, while u_5, u_6, and u_7 can be allocated to e_3. Thus, all users will be allocated to edge servers, and the proportion of users allocated to edge servers has increased. This demonstrates that the allocation of a single user affects the whole allocation.

In the real world, the scale of the EUA problem involves a much larger number of users and servers, making the search for an optimal strategy much more complex than the example illustrated above. Therefore, solving such a problem is highly challenging and demands a more efficient allocation approach.

B. Problem Formulation

This section formulates the EUA problem and models it as an MDP that is amenable to DRL-based solutions. First, the relevant notations are defined in Table I.

The EUA problem contains two constraints. The first one concerns the server coverage: a user can only be allocated to a server whose coverage encompasses the user's current location:

$$d_{t\pi_t} \leq cov_{\pi_t}, \quad t = 1, 2, \ldots, n \wedge \pi_t \neq -1. \tag{1}$$

The second constraint relates to resource capacities: a user can only be allocated to a server whose capacities can accommodate the user's resource requirements:

$$w_{tk} \leq c^t_{\pi_t k}, \quad t = 1, 2, \ldots, n \wedge \pi_t \neq -1; \forall k = 1, 2, 3, 4. \tag{2}$$

In MDP, an agent plays the role of both a learner and a decision maker [11]. In our approach, the agent refers to a neural network model that must be trained. It continuously interacts with the environment. At each time step, the agent chooses an action according to the current state, and then the environment updates the state accordingly. The environment also gives the reward earned for that action to evaluate its quality. We model EUA as an MDP with the sets of states $\mathcal{S}$, actions $\mathcal{A}$, and rewards $\mathcal{R}$. In EUA, the length of the MDP is the number of edge users. $\mathcal{S}$ is a set of states at each time step:

$$\mathcal{S} = \{\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_t, \ldots \boldsymbol{s}_{n-1}, \boldsymbol{s}_n\}. \tag{3}$$

TABLE I
NOTATIONS

Notation	Description
$\mathcal{U} = \{\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_n\}$	A finite set of edge users
$\mathcal{E} = \{\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_m\}$	A finite set of edge servers
$\boldsymbol{w}_i =< w_i^1, w_i^2, w_i^3, w_i^4 >$	The computing resource requirements of user $\boldsymbol{u}_i$, including four dimensions: CPU, memory, storage, and bandwidth
$\boldsymbol{c}^t_j =< c^t_{j1}, c^t_{j2}, c^t_{j3}, c^t_{j4} >$	The resource capacities of edge server $\boldsymbol{e}_j$ at time step t, including four dimensions: CPU, memory, storage, and bandwidth
$ulo_i,\ ula_i$	The longitude and latitude of the ith user
$elo_j,\ ela_j$	The longitude and latitude of the jth edge server
cov_j	The coverage of edge server $\boldsymbol{e}_j$
act^t_j	An indicator of whether edge server $\boldsymbol{e}_j$ has been activated before time step t
$\boldsymbol{u}_i =< ulo_i, ula_i, w_{i1}, w_{i2}, w_{i3}, w_{i4} >$	The 6-dimensional vector representation of user $\boldsymbol{u}_i$
$\boldsymbol{e}^t_j =< elo_j, ela_j, cov_j, c^t_{j1}, c^t_{j2}, c^t_{j3}, c^t_{j4}, act^t_j >$	The 8-dimensional vector representation of edge server $\boldsymbol{e}_j$ at time step t
d_{ij}	The distance between user $\boldsymbol{u}_i$ and edge server $\boldsymbol{e}_j$
π_t	The id of the server allocated at time step t. We specify $\pi_t = -1$ if there is no suitable server to accommodate the user

Our model allocates one user at each time step. Therefore, the information needed for each time step is the user being allocated and all servers at that time step. The state $\boldsymbol{s}_t$ is defined as:

$$\boldsymbol{s}_t =< \boldsymbol{u}_t, \boldsymbol{e}^t_1, \boldsymbol{e}^t_2, \ldots, \boldsymbol{e}^t_{m-1},\ \boldsymbol{e}^t_m > . \tag{4}$$

When $t = 1$, the initial state is defined according to the problem statement. The next state can be calculated from the current state and action. Action is a set of the server-id chosen at each time step:

$$\mathcal{A} = \{\pi_1, \pi_2, \ldots, \pi_t, \ldots \pi_{n-1},\ \pi_n\}. \tag{5}$$

If $\pi_t = -1$, indicating that no edge server can cover or accommodate u_t, the edge servers in s_{t+1} will remain the same as s_t. Otherwise, the resource capacity of edge server e_{π_t} should be subtracted from the resource requirement of user u_t, while the resource capacity of the other edge servers will stay unchanged.

After allocating all users, the reward can be calculated based on the objectives of the EUA system. Typically, these objectives involve maximizing the percentage of allocated users while minimizing the percentage of required servers. However, incorporating both positive and negative objectives into the reward function can make it challenging to set. In this case, reducing the number of required servers can be accomplished by improving the resource utilization of the rented server. When the resource utilization of each edge server is increased, it leads to a reduction in the number of required servers with the same number of allocated users. Therefore, resource utilization is a more meaningful metric to use in the reward function than the percentage of required edge servers. To account for these objectives, the reward R at the current time step is defined as a weighted sum of the percentage of users allocated and the resource utilization of the servers, as shown in Equation (6):

$$R = (1-\gamma) \times R^u(\mathcal{A}) + \gamma \times R^s(\mathcal{A}), \tag{6}$$

where R^u denotes the percentage of users allocated and R^s denotes the percentage of server resource utilization, and both can be calculated by the allocation actions $\mathcal{A}$. γ is a parameter to balance the weights of the two percentages.

Equation (7) defines the reward set $\mathcal{R}$. It's worth noting that rewards for all previous time steps are set to 0, as rewards can only be calculated when all users have been allocated.

$$\mathcal{R} = \{0, 0, \ldots, 0, R\}. \tag{7}$$

The percentage of users allocated is the ratio of allocated users to the total number of users:

$$R^u(\mathcal{A}) = \frac{|\{\pi_t | \pi_t \neq -1\}|}{n}. \tag{8}$$

The utilization for each resource is defined as the ratio of the total resource requests of all allocated users to the total initial capacity of all activated edge servers. The total server resource utilization is the average of the utilization rates of the four types of resources, as given by Equation (9)

$$R^s(\mathcal{A}) = \frac{1}{4}\sum_{k=1}^{4} \frac{\sum_{i\in\{i|\pi_i \neq -1\}} w_{ik}}{\sum_{j\in\left\{j|\mathrm{act}_j^{n+1}=1\right\}} c_{jk}^1}, \tag{9}$$

where act_j^{n+1} indicates the activation status of the jth server at time step $n+1$, i.e., after all users have been allocated, and c_{jk}^1 refers to the initial capacity of the kth resource at the first time step.

III. Dual-Sequence Attention Model

Section II described a problem instance with state $\mathcal{S}$ and a solution $\mathcal{A} = (\pi_1, \ldots, \pi_t, \ldots, \pi_n)$ as a list of edge servers. At each time step t, user $\boldsymbol{u}_t$ is allocated to server $\boldsymbol{e}_{\pi_t}$, where $\pi_t \in \{1, \ldots, m\}$. To address the EUA problem, this section builds a dual-sequence attention model (DSAM) capable of receiving input states and generating actions. Because the number of users and servers is not fixed for each problem instance, we employ an encoder-decoder that can accept variable-length sequences, analogous to translating sentences of varying lengths. Given a problem instance $\mathcal{S}$, our proposed DSAM defines a stochastic policy $p(\mathcal{A}|\mathcal{S})$ for selecting a solution $\mathcal{A}$. It is factorized and parameterized by θ as follows:

$$p_\theta(\mathcal{A}|\mathcal{S}) = \prod_{t=1}^{n} p_\theta(\pi_t|\boldsymbol{s}_t). \tag{10}$$

DSAM differs from traditional encoder-decoder models in that our proposed model contains two encoders due to the presence of two types of input sequences. The user encoder produces embeddings of all users, while the server encoder produces embeddings of all edge servers. This model can accept a variable number of users and servers, which means that when the number of users changes, the original model can be used without the need for retraining. The decoder produces the list π in n time steps as the solution for the input users and servers. Next, we will introduce DSAM in four subsections. The first three subsections will cover the components of the model, namely, the user encoder, server encoder, and decoder, respectively. The final subsection will describe the overall workflow of these three components together.

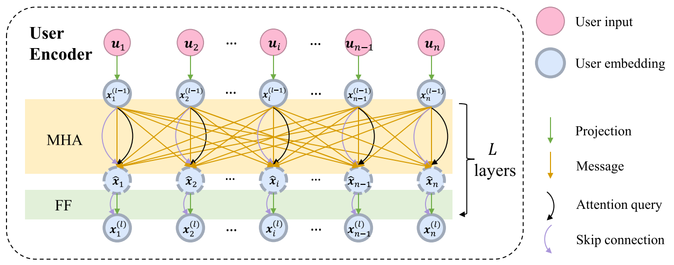


Fig. 2. Structure of the user encoder.

A. Server Encoder

In a typical encoder-decoder model, the encoder is usually an RNN [12] or LSTM [13], because they share in common the ability to convey sequence information. For example, in a translation task, the order of words is essential for accurate translation. For the EUA problem, however, the input order of the edge servers is meaningless. When feeding the servers into the encoder in a different order, the encoder should produce identical vectors for each server. Moreover, at each time step, the server encoder must be executed once to update the server features following each allocation, necessitating a basic structure with minimal calculation requirements to reduce the running and training time. Therefore, we use a linear layer with parameters $\boldsymbol{W}^e$ and $\boldsymbol{b}^e$ instead of an RNN or LSTM in our model to embed the jth input server $\boldsymbol{e}_j^t$ at time step t, as shown in Eq. (11).

$$\boldsymbol{y}_j^t = \boldsymbol{W}^e \boldsymbol{e}_j^t + \boldsymbol{b}^e. \tag{11}$$

B. User Encoder

The user encoder is more complicated than the server encoder. The user encoder is comparable to that of the Transformer architecture [14], which is inspired by the work on tackling the TSP issue [15]. Similar to the server encoder, positional encoding is not used because the input order is irrelevant, which is also why we do not use LSTM. LSTM captures the temporal order of input users, which is unnecessary. Linear layers are not utilized because they encode each user independently and fail to capture information about other users. In contrast, the self-attention mechanism in the Transformer enables each user's vector to incorporate information from other users, making it more comprehensive. This feature is particularly beneficial when allocating users, as it enables the model to select servers with more appropriate capacities. Moreover, the Transformer is parallelizable, resulting in faster training times compared to RNN and LSTM. Fig. 2 shows the user encoder's structure, which consists of a linear layer and L layers of self-attention to capture the latent relations among the input users. When users are fed into the encoder, they are embedded into a vector space by passing through a learned linear layer with parameters $\boldsymbol{W}^u$ and $\boldsymbol{b}^u$: $\boldsymbol{x}_i^{(0)} = \boldsymbol{W}^u\boldsymbol{u}_i + \boldsymbol{b}^u$. Then, $\boldsymbol{x}_i^{(0)}$ will be processed by L sequential layers of self-attention, each consisting of a multi-head attention (MHA) sublayer and a feed-forward (FF) sublayer. Finally, we obtain $\boldsymbol{x}_i^{(L)}$, which contains information of the ith user as well as information about other users with which it is associated.

Self-Attention Layers: Each self-attention layer consists of two sublayers: multi-head attention and feed-forward. Each sublayer includes a skip-connection [16] and batch normalization (BN) citeioffe2015batch. Eqs. (12) and (13) are used in self-attention layers

$$\hat{\boldsymbol{x}}_i = \mathrm{BN}\Big(\boldsymbol{x}_i^{(l-1)} + \mathrm{MHA}_i^l\Big(\boldsymbol{x}_1^{(l-1)}, \ldots, \boldsymbol{x}_i^{(l-1)}, \ldots, \boldsymbol{x}_n^{(l-1)}\Big)\Big), \tag{12}$$

$$\boldsymbol{x}_i^{(l)} = \mathrm{BN}\Big(\hat{\boldsymbol{x}}_i + \mathrm{FF}^l(\hat{\boldsymbol{x}}_i)\Big). \tag{13}$$

Here $\boldsymbol{x}_i^{(l)}$ computed from the previous layer is fed into the next layer, represented as $\boldsymbol{x}_i^{(l-1)}$, until it goes through all L layers. It is important to note that each layer has its own set of parameters, and they are not shared. The final output of the user encoder is $x_i^{(L)}$, which is the user embedding obtained after L layers of self-attention. Since the concept of the layer is only present in the user encoder, in other components, we use x_i to denote the user embedding without the superscript for simplicity, which defaults to the embedding obtained after L layers.

Attention Mechanism: Before discussing multi-head attention, the standard attention mechanism is described. For each node, three vectors, query, key, and value, are calculated by projecting with learned parameters $\boldsymbol{W}^Q$, $\boldsymbol{W}^K$ and $\boldsymbol{W}^V$:

$$\boldsymbol{q}_i = \boldsymbol{W}^Q\boldsymbol{x}_i^{(l-1)}, \quad \boldsymbol{k}_j = \boldsymbol{W}^K\boldsymbol{x}_j^{(l-1)}, \quad \boldsymbol{v}_j = \boldsymbol{W}^V\boldsymbol{x}_j^{(l-1)}. \tag{14}$$

The compatibility of each node pair is then calculated using Eq. (15).

$$b_{ij} = \frac{\boldsymbol{q}_i^T\boldsymbol{k}_j}{\sqrt{d_k}}, \tag{15}$$

where d_k is the dimension of vector $\boldsymbol{k}_i$, which is used to scale the result [14]. Next, through softmax operation, the compatibility can be transformed into attention weight $a_{ij} \in [0, 1]$:

$$a_{ij} = \frac{e^{b_{ij}}}{\sum_{j'} e^{b_{ij'}}}. \tag{16}$$

Finally, vector u_i will be computed by a weighted sum of values v_j:

$$\boldsymbol{x}_i^{(l)} = \sum_j a_{ij}\boldsymbol{v}_j. \tag{17}$$

Muti-head Attention: The calculation procedure of the single-head attention mechanism is depicted above. Muti-head attention enables the model to learn relevant information in different representation subspaces. When the number of heads M is set to eight, the dimensions of $\boldsymbol{q}_i$, $\boldsymbol{k}_i$ and $\boldsymbol{v}_i$ will be divided by eight. Moreover, the query, key, and values of each node will be computed eight times with eight different groups of $\boldsymbol{W}_m^Q$, $\boldsymbol{W}_m^K$ and $\boldsymbol{W}_m^V$ to get $\boldsymbol{q}_{im}$, $\boldsymbol{k}_{im}$ and $\boldsymbol{v}_{im}$, where $m \in [1,\ M]$. Next, with Eqs. (15), (16), and (17), we can obtain $\boldsymbol{x}_{im}$. Finally, the vector of the ith node will be computed by Eq. (18):

$$\begin{aligned}\mathrm{MHA}_i^l&\Big(\boldsymbol{x}_1^{(l-1)}, \ldots, \boldsymbol{x}_i^{(l-1)}, \ldots, \boldsymbol{x}_n^{(l-1)}\Big)\\ &= \boldsymbol{W}_m^O\Big[\boldsymbol{x}_{i1}^{(l)}, \ldots, \boldsymbol{x}_{iM}^{(l)}\Big],\end{aligned} \tag{18}$$

where $\boldsymbol{W}_m^O$ denotes the linear layer parameters and $[\boldsymbol{x}_{i1}^{(l)}, \ldots, \boldsymbol{x}_{iM}^{(l)}]$ means the concatenation of each $\boldsymbol{x}_{im}^{(l)}$.

Feed-forward sublayer: The feed-forward sublayer consists of two linear layers and a *ReLu* activation between them, as shown in Eq. (19).

$$\mathrm{FF}(\hat{\boldsymbol{x}}_i) = \boldsymbol{W}^{\mathrm{ff},1} \cdot \mathrm{ReLu}\Big(\boldsymbol{W}^{\mathrm{ff},0}\hat{\boldsymbol{x}}_i + \boldsymbol{b}^{\mathrm{ff},0}\Big) + \boldsymbol{b}^{\mathrm{ff},1}, \tag{19}$$

where $\boldsymbol{W}^{\mathrm{ff},0}$, $\boldsymbol{W}^{\mathrm{ff},1}$, $\boldsymbol{b}^{\mathrm{ff},0}$, and $\boldsymbol{b}^{\mathrm{ff},1}$ denote the parameters of linear layers.

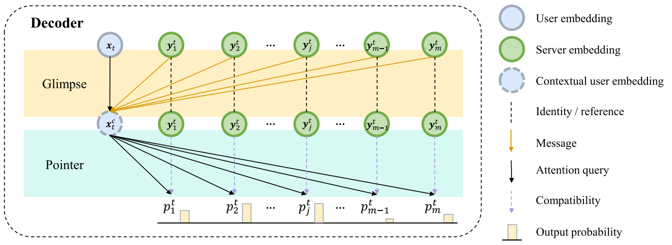


Fig. 3. Structure of the decoder.

C. Decoder

The decoder is responsible for selecting an appropriate server for a user. At each time step, the decoder receives a user embedding and all server embeddings. Fig. 3 shows the structure of the decoder.

There are two components to allocate user $\boldsymbol{u}_t$ to an appropriate edge server for each time step t. First, the glimpse part [17] computes $\boldsymbol{x}_t^c$, the contextual embedding of the t-th user being allocated, with a complete attention mechanism shown in Eqs. (20), (21), (22), and (23). The attention differs from the one in the user encoder in that query $\boldsymbol{q}_t$ is computed by the final user embedding $\boldsymbol{x}_t^{(L)}$ but key $\boldsymbol{k}_j^t$ and value $\boldsymbol{v}_j^t$ are computed by server embedding $\boldsymbol{y}_j^t$. Thus, the glimpse part produces a fusion representation of server information related to this user.

$$\boldsymbol{q}_t = \boldsymbol{W}^Q\boldsymbol{x}_t^{(L)}, \quad \boldsymbol{k}_j^t = \boldsymbol{W}^K\boldsymbol{y}_j^t, \quad \boldsymbol{v}_j^t = \boldsymbol{W}^V\boldsymbol{y}_j^t, \tag{20}$$

$$b_{tj} = \frac{\boldsymbol{q}_t^T\boldsymbol{k}_j^t}{\sqrt{d_k}}, \tag{21}$$

$$a_{tj} = \frac{e^{b_{tj}}}{\sum_{j'} e^{b_{tj'}}}, \tag{22}$$

$$\boldsymbol{x}_t^c = \sum_j a_{tj}\boldsymbol{v}_j. \tag{23}$$

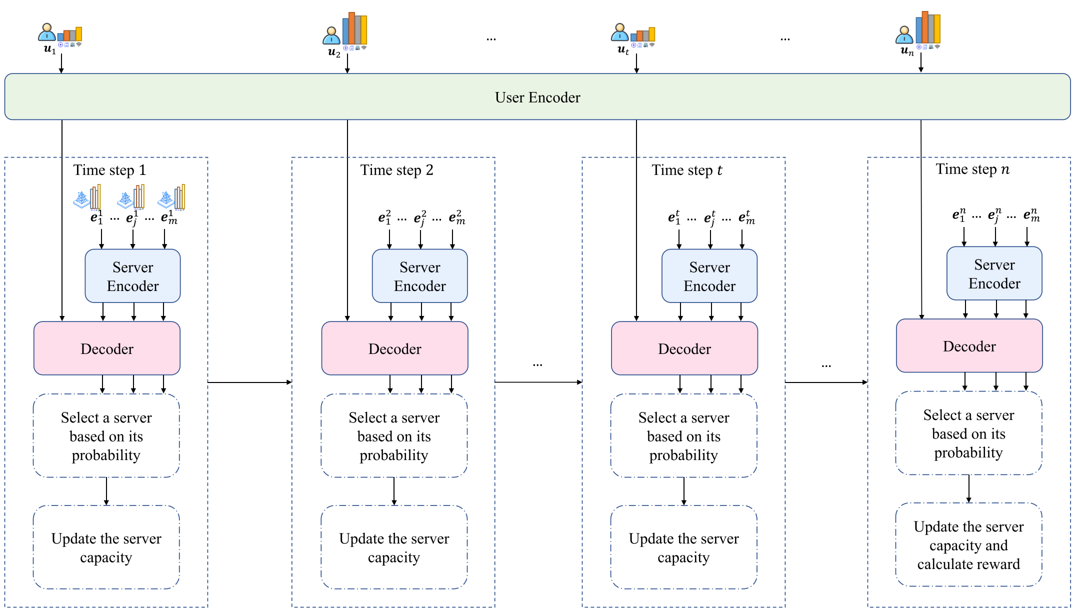

u_1
u_2
u_t
u_n
User Encoder
Time step 1
Time step 2
Time step t
Time step n
e_1^1 … e_j^1 … e_m^1
e_1^2 … e_j^2 … e_m^2
e_1^t … e_j^t … e_m^t
e_1^n … e_j^n … e_m^n
Server Encoder
Decoder
Select a server based on its probability
Update the server capacity
Update the server capacity and calculate reward

Fig. 4. Overall workflow of DSAM.

Then, the pointer part [18] computes the compatibility between this user and each server by Eqs. (24) and (25):

$$\hat{p}_j^t = \begin{cases} C \cdot \boldsymbol{v}^T \tanh\left(\boldsymbol{W}_1 \alpha \boldsymbol{x}_t^c + \boldsymbol{W}_2 \boldsymbol{y}_j\right) & \text{if } d_{tj} \leq cov_j \\ -\infty & \text{otherwise} \end{cases}, \tag{24}$$

$$p_j^t = \frac{e^{\hat{p}_j^t}}{\sum_{j'} e^{\hat{p}_{j'}^t}}, \tag{25}$$

where $\boldsymbol{v}$, $\boldsymbol{W}_1$, and $\boldsymbol{W}_2$ are parameters in the linear layer. C and α are hyperparameters. C is a temperature parameter [19] in softmax that makes the probability smoother and increases the likelihood of the original's negligible ones, so as to expand the exploration capability of reinforcement learning. In addition, α is a scaling parameter used to adjust the size of the contextual user embedding x_i^c. In this case, the value of the user's embedding vector is smaller than that of the server's since the user's resource requirements are significantly less than the server's resource capacities. However, the glimpse part transforms the contextual user embedding into a weighted sum of server embeddings. Therefore, to make the contextual user embedding as small as the original user embedding, the parameter α must be adjusted. The probabilities of allocating a user to a specific server are then estimated by softmax in Eq. (25).

D. *Workflow*

The model takes in all user vectors $u_1, \ldots, u_i, \ldots, u_n$ and all initial server vectors $e_1^1, \ldots, e_j^1, \ldots, e_m^1$. DSAM first sends all user vectors into the user encoder to obtain all user embeddings. Then DSAM starts running step by step, as shown in Fig. 4.

For each time step t, the server encoder first encodes the server vectors of the current time step to obtain server embeddings. Then the decoder receives the user embedding x_t and m server embeddings $y_1^t, \ldots, y_j^t, \ldots, y_m^t$, and generates $p_1^t, \ldots, p_j^t, \ldots, p_m^t$ representing the probability value of allocating x_t to each server. Based on these probabilities, DSAM chooses a server π_t, which may be the server with the highest probability, or randomly sampled according to the probabilities (details will be discussed in Section IV). If the resources of the selected server are insufficient to accommodate the user, π_t is set to -1. Otherwise, the resources of the server are updated to obtain $e_1^{t+1}, \ldots, e_j^{t+1}, \ldots, e_m^{t+1}$, before allocating the next user in the next time step.

After all users are allocated, the reward can be calculated based on the user allocation rate and the server resource utilization rate. With the probabilities of each action generated by the model and the final reward, the model can be trained. The next section will introduce the model's training method.

IV. Self-Critical Sequence Training

As stated in Section III, given a problem instance state $\mathcal{S}$ that contains multiple users and edge servers, the proposed DSAM (with parameters θ) can generate a probability distribution $p_\theta(\mathcal{A}|\mathcal{S})$ from which a solution $\mathcal{A}|\mathcal{S}$ can be sampled. In this section, we train our model using policy gradient [20]. The reward of our DRL model is defined in Section II. Then, we take the expectation of all negative rewards, $l(\mathcal{A}) = -R(\mathcal{A})$, as the loss of all given instances:

$$L(\mathcal{A}|\mathcal{S}) = \mathbb{E}_{p_\theta(\mathcal{A}|\mathcal{S})}[l(\mathcal{A})]. \tag{26}$$

We optimize L by gradient descent, using the REINFORCE [20] algorithm:

$$\nabla L(\theta|\mathcal{S}) = \mathbb{E}_{p_\theta(\mathcal{A}|\mathcal{S})}[(l(\mathcal{A}) - b(\mathcal{S}))\nabla \log p_\theta(\mathcal{A}|\mathcal{S})], \tag{27}$$

where $b(\mathcal{S})$ denotes a baseline function that reduces gradient variance to increase learning speed, and p_θ denotes the policy defined by the model of parameter θ.

Algorithm 1 Self-Critical Sequence Training

```
Input: number of epochs $E$, steps per epoch $T$, batch size $B$
Output: model parameter $\theta$
  Initialize model parameter $\theta$
  for epoch $= 1, \dots, E$ do
    for step $= 1, \dots, T$ do
      $\mathcal{S}_b \leftarrow \mathrm{RandomInstance}()\ \forall b \in \{1, \dots, B\}$
      $\mathcal{A}_b \leftarrow \mathrm{SampleRollout}(\mathcal{S}_b, p_\theta)\ \forall b \in \{1, \dots, B\}$
      $\mathcal{A}_b^{BL} \leftarrow \mathrm{GreedyRollout}(\mathcal{S}_b, p_\theta)\ \forall b \in \{1, \dots, B\}$
      $\nabla L \leftarrow \frac{1}{B}\sum_{b=1}^{B}\left(l(\mathcal{A}_b) - l\left(\mathcal{A}_b^{\mathrm{BL}}\right)\right)\nabla_\theta \log p_\theta(\mathcal{A}_b|\mathcal{S}_b)$
      $\theta \leftarrow \mathrm{Adam}(\theta, \nabla L)$
    end for
  end for
```

Similar to the self-critical training method proposed by Rennie et al. [21], our self-critical sequence training (SCST) method contains a rollout baseline. SCST does not require a critic network since it uses the training model itself as the critic and gets a baseline reward in a different way. The baseline $b(\mathcal{S})$ is the loss (the negative value of the reward) of a solution resulting from a deterministic greedy rollout of the policy defined by the training model. It means that in each batch of the decoder, the model's forward function must execute twice. First, a stochastic server was sampled by its allocation probability for each time step of the decoder. After completing all steps, we can have a sampled solution with its likelihood and loss. The second time, the specific server with the largest allocation probability is selected for each time step of the decoder. Finally, we get a second solution with its loss, which is the required baseline loss. This baseline does not necessitate a critic network, so conserving GPU memory and training time. However, it must repeat the neural network's forward propagation, increasing the running time by 50%. Due to the fact that the two forward methods use the same model and are not sequential, they may be parallelized on two GPUs without increasing training time. Algorithm 1 shows our training method, using Adam [22] as the optimizer.

V. Experiment

We conducted a series of experiments using a real-world dataset to evaluate the performance of our proposed DSAM. Three groups of experiments were designed to answer the three research questions:

- *RQ1:* Does our proposed DSAM outperform the baseline approaches under different data scales?
- *RQ2:* How does the setting of the reward weight parameter γ affect the experimental results?
- *RQ3:* How is the model's convergence affected by the introduction of the attention mechanism and the new training method?
- *RQ4:* Does DSAM meet the real-time requirements?

A. Dataset

Edge servers: To ensure the accuracy and objectivity of our dataset, we extracted all the locations of edge servers from the EUA dataset [6], which covers 125 servers in an area of $1.8km^2$ in the real world. To maintain the consistency of data magnitude, we converted the latitude and longitude in the original dataset to a Cartesian coordinate system. We randomly generated the coverage radius between 100 and 150 meters. As for the resource capacity of edge servers, we take into account a realistic server that consists of a 16-core CPU, 32 GB of memory, 400 GB of storage, and 50 Mbps of bandwidth. To maintain the consistency of the numbers fed to the neural network, we can express the units of each of the four resources as 0.5 core CPU, 1 GB memory, 10 GB storage, and 1 Mbps bandwidth. Consequently, the final resource capacity of the server becomes (32, 32, 40, 50). Since the edge servers in the real world exhibit a heterogeneous nature, we initialize the resource capacities using a normal distribution $N(\mu, \sigma^2)$, where μ is set to 35 based on the real server configuration, and σ is set to 10. The final server data input to the DSAM model includes horizontal and vertical coordinates, coverage, and resource capacity in four dimensions.

Edge users: Due to the uncertainty of the number of users and their random location in the real world, we randomly generate users with various numbers and distributions in the edge scenario. This approach allows for a more objective evaluation of various methods in different situations. To this end, the location of each user is randomly generated within the coverage of all servers, and their resource requirements are randomly selected from {<1, 2, 1, 2>, <2, 3, 3, 4>, <5, 7, 6, 6>}, as referenced in [1]. Similar to their work, our DSAM-DRL also sorts the users in ascending order of their computing resource requirements. The final input data for each user in the DSAM model includes their horizontal and vertical coordinates, as well as their resource requests in four dimensions.

B. Comparison Approaches

To evaluate the proposed model, nine approaches are selected for comparison. The first six did not use deep reinforcement learning.

- *Random*: This method assigns each user to a random edge server as long as the server's coverage and resource capacity are sufficient to satisfy the user.
- *Greedy*: For each user, this approach finds all available edge servers that satisfy the coverage and resource capacity constraints. The user is then assigned to the edge server with the most available resource capacity.
- *MCF (Most-Capacity-First Heuristic) [1]*: It is a state-of-the-art approach. MCF first ranks users in ascending order based on their computing resource requirements. Next, MCF allocates users sequentially, like the Greedy method. MCF differs from Greedy in prioritizing allocating users to active servers, which implies it strives to allocate users to a server that has already been started and is available. MCF initiates a new edge server only when all active servers are out of coverage or have insufficient capacity for allocating users.
- *GA (Genetic Algorithm)*: This is a traditional optimization method. The chromosome is set to the chosen server id corresponding to each user. We use the reward of the EUA problem as the fitness function of the genetic algorithm for optimization.

- *DRoEUA (Decentralized Reactive approach of EUA) [23]*: It employs a fuzzy control mechanism to decide the destination servers for edge users in real-time in a decentralized and reactive way.
- *Optimal [6]*: It models EUA as a variable-sized vector bin packing (VSVBP) problem and solves it with the Lexicographic Goal Programming technique using IBM ILOG CPLEX Optimizer. It can obtain the mathematically optimal solution for the EUA problem.

The remaining three are variants of our DSAM. The first two variants change the user encoder, and the other variant changes the training method.

- *DSAM-Linear-SCST*: This variant uses a linear layer as the user encoder. The model's training approach remains unchanged; it is still SCST.
- *DSAM-LSTM-SCST*: This variant uses an LSTM as the user encoder while keeping SCST as the training approach.
- *DSAM-Transformer-EMA*: It is another variant of our DSAM. The REINFORCE algorithm is used to train the variant with an exponential moving average (EMA) as the baseline, and the model's components remain unchanged.

C. Experiment Settings

Data settings: In each experiment, we evaluate the performance of different approaches with a fixed number of servers and users. As edge infrastructure providers rarely change the location and resource capacity of edge servers once they have been deployed, we keep them fixed for each experiment to simulate the distribution of servers in a specific region. Since the location distribution and resource requests of users are random in the real world, we generated 10,000 sets of test data for each experiment to avoid serendipity in the evaluation results. Each set of test data contains a fixed number of users, but their location distribution and resource requests are newly generated. To obtain the final metrics, such as the user allocation rate for each approach, we average the results of allocation strategies for these 10,000 scenarios. For deep reinforcement learning approaches, we generated 100,000 sets of data for model training.

For the comparison method GA, we set the number of chromosomes to 100, with an initial crossover probability of 0.5 and a mutation probability of 0.05. The crossover and mutation probability are uniformly increased as the number of iterations increases, until it reaches 1. Based on our experimental results, we set the number of iterations to 2000 to achieve good convergence. Please note that due to the long running time required by GA and Optimal, we randomly selected 100 datasets out of 10,000 for testing the performance of these two approaches. This may have an impact on the accuracy of the test results, so the results of GA and Optimal should be considered as a reference only.

For the RQ1, we conducted 30 experiments (Sets #1-3) with different independent variables to evaluate the performance of our approach, as shown in Table II.

Moreover, we conducted five experiments (Set #4) to discuss the settings of weight parameter γ for the RQ2.

For the RQ3, the dataset scale is the default setting; that is, the number of users is 500, the server availability ratio is 50% (i.e., 50% of edge servers in the simulated region are available), and the average server resource capacity is 35. We compare the impact of different models (*DSAM-Linear-SCST* and *DSAM-LSTM-SCST*) and different training methods (*DSAM-Transformer-EMA*: EMA as the baseline of RL training).

For the RQ4, we compared the running time of all approaches under different numbers of users while keeping the number of servers and the average server resource at their default settings.

It should be noted that the length of the input sequences to our model is not fixed. To improve performance, we trained various models for different numbers of users. For example, although two models trained with 800 and 100 users can both handle an input of 790 users, the results obtained with the model developed with 800 users are superior to those obtained with the model trained with 100 users.

Hyperparameter settings: In all experiments with 500 users and 50% available edge servers, we use mini-batches of 100 pieces of data; other hyperparameters are adjusted according to GPU memory limitations. The embedding vectors are all 256-dimensional. In the User Encoder, we use $N = 3$ layers and $M = 8$ heads; the hidden dimension of the feed-forward layer is 512. We use a learning rate $\eta = 1 \times 10^{-4}$ with a decay of 0.98 per epoch. Our code in PyTorch [24] is publicly available on GitHub.[1] Each epoch with the default dataset scale takes 32 minutes with a batch size of 100 on NVIDIA GeForce RTX 3090. Each model is trained for 100 epochs.

D. Results and Analysis

In this section, we will analyze the experimental results in terms of performance comparison, model adjustability, model convergence, and running time.

1) Performance Comparison (RQ1): We conducted three sets of experiments to evaluate the performance of our approach.

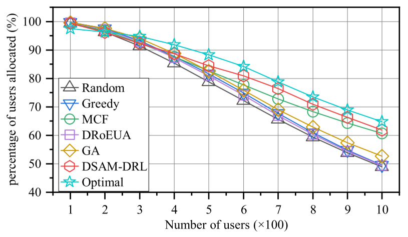
Experiment Set #1: In this set of experiments, the number of edge servers was fixed at 50% of all edge servers in the simulated region, the average server resource capacity was 35, the weight parameter γ is 0.5, and the number of users varied from 100 to 1000 in steps of 100.

Fig. 5(a) depicts how the proportions of allocated users vary as the number of users increases. In general, when the number of users is small (between 100 and 400), the gap between the various approaches is negligible. This is because the resource capacities are sufficient, and almost all users can be easily allocated to an edge server. However, Fig. 5(b) shows that the percentage of servers required for GA, MCF, DSAM-DRL, and Optimal is significantly lower than the baselines when the number of users is small. GA performs well because when there are fewer users, the search space for the solution is smaller, increasing the probability that GA will find a better solution. MCF prioritizes allocating users to active servers, which avoids starting a new server. DSAM-DRL may have

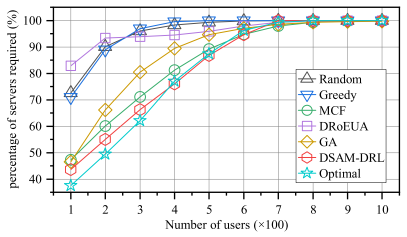
[1]https://github.com/ccjjxx99/DSAM-DRL-EUA

TABLE II
EXPERIMENT DATASET SETTINGS

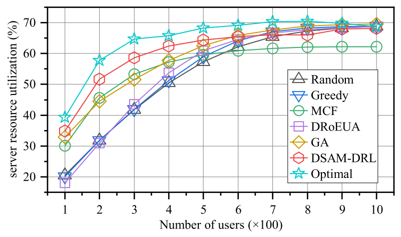
	Number of users	Number of edge servers	Average server resource capacity (μ)	Weight parameter γ
Set #1	100, 200, ...,900, 1000	50%	35	0.5
Set #2	500	10%, 20%, ..., 90, 100%	35	0.5
Set #3	500	50%	30, 35, ..., 70, 75	0.5
Set #4	500	50%	35	0.1, 0.3, 0.5, 0.7, 0.9



(a) Number of users vs. Percentage of users allocated



(b) Number of users vs. Percentage of edge servers required



(c) Number of users vs. Server resource utilization

Fig. 5. Variation of each metric with the increasing number of users.

also learned this during the reinforcement learning process, as we encode the server's information at each time step with a symbolic bit act_j indicating whether the server is active or not. Moreover, DSAM-DRL learns a better strategy, requiring about 5% fewer servers than MCF (at the 100-500 user count). However, DRoEUA always activates more servers, because its strategy is to activate new servers when there are many idle edge servers to ensure load balancing, rather than considering cost savings. This also reduces the resource utilization of edge servers.

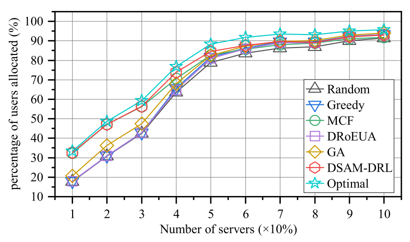
As the number of users increases, the percentages of users allocated show a downward trend. This is reasonable given that the resource capacity of edge servers is limited and that as the number of users increases, there must be an increasing number of users who cannot be assigned to any server. When the number of users is large (more than 700), all methods need almost all edge servers. However, DSAM-DRL and MCF have a distinct advantage over the other approaches in terms of the proportion of users allocated because they sort users in ascending order of resource requirements, which prioritizes users with lower resource requirements. As the number of users increases, the benefits of DSAM-DRL become more evident due to its enhanced ability to capture server characteristics. The performance of GA gradually becomes comparable to the baseline approaches because with the increase of users, the search space of the solution will exponentially expand. The same number of iterations is no longer sufficient to find good enough solutions. However, the performance gap between DSAM-DRL and Optimal remains relatively small even as the number of users increases.

Fig. 5(c) shows the correlation between server resource utilization with the number of users. It has an upward trend as the number of users grows. A possible reason is that the more users there are, the more likely some users can fill the remaining capacity of some edge servers, resulting in a higher server resource utilization ratio. When the number of users is small (between 100 and 500), GA, MCF, and DSAM-DRL perform better than the baselines because they initiate fewer edge servers, decreasing the denominator. As the number of users increases (more than 600), MCF performs worse than other approaches because it first allocates users with fewer resource requirements, but these users occupy the server capacity in the first place, causing users with greater resource requirements to be unable to find edge servers with sufficient capacities. Even though our DSAM-DRL does not completely prevent this issue, it is consistently about 5% higher than MCF due to a well-trained allocation policy. Although DSAM-DRL and MCF have this disadvantage, they still have a much larger percentage of users allocated than other approaches.

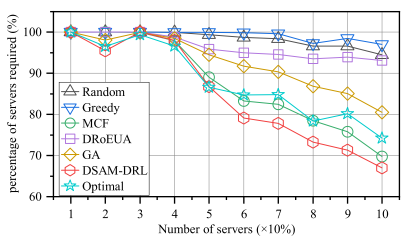
Overall, DSAM-DRL can significantly reduce the percentage of servers required while keeping the percentage of users allocated high enough when the number of users is small. It can also considerably increase the percentage of users allocated while keeping the percentage of required servers to a minimum when the number of users is really high. Moreover, in terms of resource utilization, our approach is almost always the most efficient, with only a small gap compared to Optimal.

Experiment Set #2: In this set of experiments, we fixed the number of users at 500, the average server resource capacity at 35, and the weight parameter γ at 0.5. The number of servers varied in 10% increments from 10% to 100% of all edge servers in the simulated region. Fig. 6 shows the variation in the percentages of allocated users, required edge servers, and server resource utilization. We can observe that the trends of the curves in these figures are almost the opposite of those in Set #1, as an increase in servers with a fixed number of users is comparable to a reduction in users with a fixed number of edge servers.

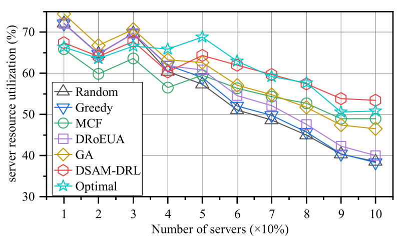
As the number of available edge servers grows, the number of allocated users increases proportionally, as shown in Fig. 6(a). DSAM-DRL is indistinguishable from MCF and Optimal but much better than the baselines when the percentage of edge servers is less than 50%. As the number of edge servers increases, all methods are almost capable of allocating the majority of users (more than 90%) to the edge servers because of the increasing available resource.



(a) Number of servers vs. Percentage of users allocated



(b) Number of servers vs. Percentage of edge servers required



(c) Number of servers vs. Server resource utilization

Fig. 6. Variation of each metric with the increasing number of edge servers.

In Fig. 6(b), as the number of servers increases, the proportion of servers required for DSAM-DRL and MCF decreases rapidly. This is owing to the fact that while the total number of servers increases greatly, the number of servers needed does not increase significantly due to the less significant growth in the percentage of allocated users. Our DSAM-DRL has more apparent advantages over other approaches, even outperforming Optimal. This is because Optimal prioritizes maximizing the user allocation rate, leading to the allocation of more users but also the activation of more edge servers, resulting in less resource efficiency. There is a notable point at the 20% number of edge servers where DSAM-DRL reduces the required servers significantly. A likely explanation is that DSAM is capable of capturing the features of these edge servers and determining that one of them is not required to maintain a high user allocation; hence, this server is deactivated in nearly all of the 10,000 test data. GA and Optimal may have captured this characteristic in some of the test data, and the results have also slightly reduced the server rental rate after averaging.

Fig. 6(c) depicts a declining trend in server resource utilization. As the number of servers increases, the proportion of required servers reduces (as shown in Fig. 6(b)), and the number of allocated users increases (as shown in Fig. 6(a)), resulting in an increase in the actual number of required servers. However, newly allocated users may not utilize the full capacity of newly activated servers, resulting in a decrease in server resource utilization. When the number of edge servers is low, there are some fluctuations because the capacity limitations of a single server may significantly influence the overall resource utilization percentage. In the real scenario, the number of servers will not be so few. When the number of available edge servers exceeds 40%, the resource utilization performance of DSAM-DRL is comparable to that of Optimal.

Experiment Set #3: In this set of experiments, we varied the average capacity of edge servers from 30 to 75 in step 5. The number of users was 500, the number of edge servers was fixed at 50% of all edge servers in the simulated region, and the weight parameter γ was set to 0.5. Fig. 7 depicts the variations in the percentage of users allocated, the percentage of servers required, and the utilization of edge server resources.

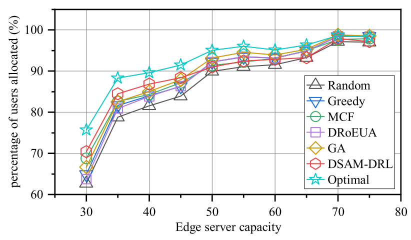
In Fig. 7(a), there is some difference between DSAM-DRL and Optimal, but it outperforms other methods regarding the percentage of users allocated when the edge server capacity is small. However, as the capacity of the edge server increases, the percentage of allocated users rises for all methods because the resource becomes increasingly sufficient. Since practically all users can now be assigned to an edge server, the differentiation between the various strategies is shrinking.

In Fig. 7(b), the baselines consistently utilize nearly all the edge servers. The percentage of servers required for MCF, DSAM-DRL and Optimal rapidly decreases as the edge server capacity increases, because the original users can be accommodated with fewer servers when the resource capacity of each edge server increases. The curve of DSAM-DRL is somewhat upward at points 35 and 70, because during the training of these two models, the DSAM learns to allocate more users according to the indication of the RL reward. We can observe that it allocates more users at the point of 70 than at the point of 75 as the reward for doing so is larger. If we decrease the reward weight of the percentage of users allocated, DSAM-DRL will reduce the percentage of servers required while slightly reducing the percentage of users allocated. This feature will be discussed in the next subsection, which answers RQ2. Moreover, DSAM-DRL consistently outperforms MCF by 5% to 10% and performs comparably to Optimal.

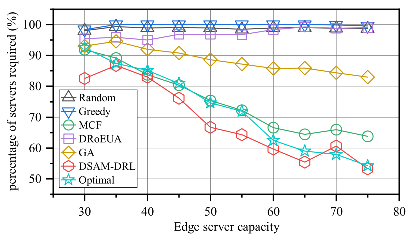
In Fig. 7(c), as the edge server capacity increases, the baselines decline in the server resource utilization because they use nearly all servers with increasing capacity, but the number of users allocated has not changed much. In contrast, MCF and DSAM-DRL fluctuate in resource utilization but maintain an elevated level. Again, our DSAM-DRL performs comparably to Optimal and consistently outperforms MCF by 5% to 10%.

In summary, in all experiments, DRoEUA did not perform well in this scenario because it focuses more on load balancing than cost savings, which lowers the resource utilization of edge servers; GA can almost find the optimal solution when the solution space is small, but with an increase in the number of users, fewer iterations are not enough to find good solutions when the solution space becomes larger; the MCF algorithm designed for the EUA problem performs better than other algorithms; however, due to the use of the Attention mechanism in DSAM and the reinforcement learning training of better policies, DSAM-DRL performs the best in nearly all metrics, especially in terms of cost savings. Compared to MCF, it reduces the number of required servers by 5% to 10% and increases resource utilization by 5% to 10%, approaching the performance of Optimal.

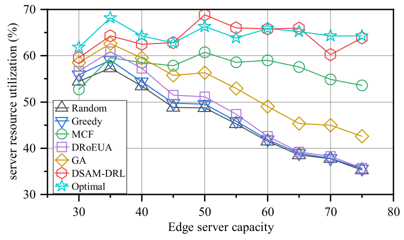
Statistical Tests: When the distribution of users is different, even if the number of users, the number of servers, and the server resources are the same, the results obtained by the



(a) Edge server capacity vs. Percentage of users allocated



(b) Edge server capacity vs. Percentage of edge servers required



(c) Edge server capacity vs. Server resource utilization

Fig. 7. Variation of each metric with the increasing edge server capacity.

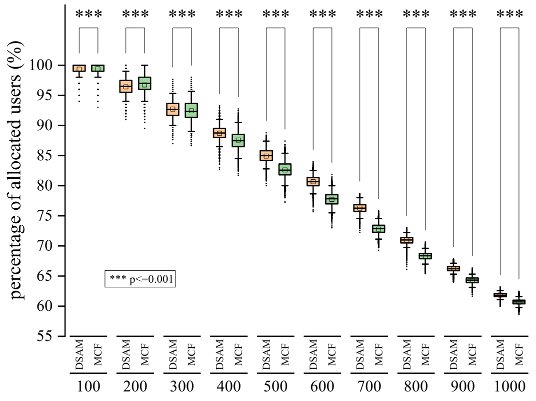


Fig. 8. Significance of the difference between the percentage of users allocated for DSAM-DRL and MCF.

TABLE III
RESULTS OF EXPERIMENT SET#4

Approach	Percentage of users allocated	Percentage of servers required	Edge server resource utilization
Random	78.83%	99.36%	57.35%
Greedy	81.61%	99.98%	58.99%
MCF	82.51%	89.14%	59.54%
DRoEUA	80.92%	95.94%	60.86%
GA	82.51%	94.62%	62.57%
Optimal	88.33%	87.35%	68.26%
DSAM-DRL, $\gamma = 0.1$	86.31%	97.70%	59.12%
DSAM-DRL, $\gamma = 0.3$	85.74%	92.33%	61.87%
DSAM-DRL, $\gamma = 0.5$	84.48%	86.81%	64.32%
DSAM-DRL, $\gamma = 0.7$	83.32%	83.67%	65.39%
DSAM-DRL, $\gamma = 0.9$	82.56%	81.48%	65.83%

same approach will be different. For this reason, we tested each approach with 10,000 independent user distributions for each dataset setting, and the results for each indicator approximately conformed to a normal distribution. Because the difference in the percentage of users allocated between DSAM-DRL and MCF is not readily apparent in the figure, we applied the Bonferroni Correction to analyze the difference between these two approaches at different data scales. Fig. 8 shows the results of the percentage of users allocated for varying numbers of users. The horizontal axis denotes the number of users and the two approaches for each user number.

As shown in Fig. 8, the results are similar in different numbers of edge servers and edge server capacities. The p-value of the Bonferroni Correction indicates the probability that the approach does not affect the experimental results. All p-values are less than 0.001, demonstrating the significance of the effect of different approaches on the percentage of users allocated, which means that our approach has an almost 100% probability of being superior to MCF. Because the number of test data is 10,000, which is large enough to get a representative average value that avoids occasionality, the differences between the two approaches in Figs. 5, 6 and 7 are all statistically significant, even if the differences are small. As for the other two metrics, DSAM-DRL has an advantage of much greater significance.

2) Model Adjustability (RQ2): In addition to performance, we are also concerned about the adjustability of the proposed DSAM-DRL. Consequently, we conducted Experiment Set #4 by fixing the number of users, the number of servers, and the average resource capacity. We varied parameter γ, the weight of the edge server's resource utilization in the RL reward.Table III shows the results.

When γ is small, DSAM-DRL activates almost all edge servers, and the percentage of allocated users reaches a high level due to the small weight given to server resource utilization. However, at this point, DSAM-DRL performs worse than other approaches in terms of the percentage of required servers. If we raise γ, the percentage of allocated users will decrease slightly, but the percentage of servers required and server resource utilization will improve considerably. Moreover, when γ is between 0.5 and 0.9, our DSAM-DRL completely outperforms other methods. Because of the flexibility of our solution, the model users, including infrastructure providers and service providers, can fine-tune the model parameters to determine whether to enable more servers to accommodate more users or sacrifice the experience of a few users to significantly reduce the percentage of servers needed. This also explains why there are special points in the previous set of experiments, such as the 35 and 70 points in Fig. 7(b), because at these points, the model can activate a few more servers to allow a higher percentage of user allocated and thus maximize the reward.

3) Model Convergence (RQ3): Four experiments were conducted to answer RQ3. The number of users was fixed at 500, the number of edge servers was fixed at 50% of all edge servers in the simulated region, the average server resource capacity was 35, and the weight parameter γ was set as 0.5. Our Transformer user encoder and SCST were compared to three approaches with different network architectures and training methods on the convergence. The comparison network model

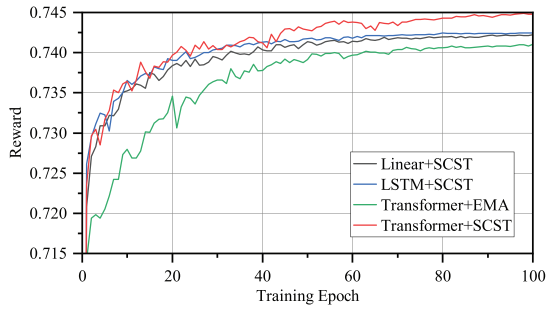


Fig. 9. Convergence curves of different models and training methods.

TABLE IV
RUNNING TIME (SECONDS) OF ALL APPROACHES WITH DIFFERENT NUMBERS OF USERS

Number of users	Random	Greedy	MCF	DRoEUA	GA	Optimal	DSAM-DRL
100	0.04	0.04	0.04	0.15	230.00	0.61	0.11
200	0.08	0.08	0.08	0.29	516.55	3.51	0.24
300	0.12	0.13	0.12	0.43	631.68	8.36	0.32
400	0.16	0.17	0.16	0.58	830.40	14.54	0.44
500	0.20	0.20	0.20	0.72	1011.51	18.06	0.55
600	0.24	0.25	0.25	0.88	1170.36	26.87	0.64
700	0.28	0.29	0.29	1.01	1375.23	45.72	0.73
800	0.32	0.32	0.33	1.14	1502.46	39.57	0.88
900	0.36	0.37	0.37	1.29	1627.91	59.50	0.99
1000	0.40	0.40	0.41	1.43	1781.92	77.04	1.12

employs a linear user encoder and an LSTM user encoder. The comparative training method is REINFORCE with an EMA baseline. The baseline is used to evaluate the difficulty of the problem instance. The accuracy of the evaluation affects the effectiveness of the training model. Although both training methods are based on REINFORCE, different baselines can lead to different results. We trained each model for 100 epochs. Fig. 9 shows the convergence curve of the models.

From the results, we can see that all methods are in the rapid improvement phase at epochs 0 to 40. At epochs 40 to 80, all curves gradually converge and stabilize. After epoch 80, all methods are almost no longer improved. With the same training method, DSAM with the Transformer-based user encoder performs better than the LSTM one because the Transformer's embedding is not affected by the sequence's input order. It also outperforms linear one because its embedding contains information from other users, resulting in more comprehensive decisions. With the same model architecture, SCST outperforms the REINFORCE with an EMA baseline. SCST converges faster, is more stable, and yields better results, due to its more accurate and stable baseline.

4) Running Time (RQ4): Table IV illustrates the running time of various methods on the Intel Core i7-11700 CPU. GA requires the longest time. Even though the Optimal approach can find optimal solutions, it cannot be deployed in practical applications due to real-time requirements. The running time of the remaining methods also increases linearly with the number of users because all methods allocate users one by one, which requires a certain amount of computation for each user's allocation. Random, Greedy, and MCF are the fastest, taking an average of 0.04 s to allocate every 100 users, while DSAM-DRL and DRoEUA require 0.11 s and 0.14 s, respectively. Although DSAM-DRL's running speed is slower than that of MCF, its real-time performance is still within an acceptable range. DSAM-DRL sacrifices only a small amount of speed for a better allocation strategy.

E. Threats to Validity

Threats to Internal Validity: A threat to the internal validity of our approach is whether the results are biased by parameter settings. We performed a set of experiments (Set #4) to discuss the effect of parameter settings on the experimental results. Experiments indicate that our model is effective regardless of input parameters, and different results can be obtained according to user preference, such as allocating more users or saving more servers. As for the parameter settings of the dataset, we conducted multiple sets of experiments with different parameters, including the number of users, the number of edge servers, and the server capacity. Moreover, in the testing phase of all experiments, 10,000 user distributions were randomly sampled to reduce the bias of experimental results. In addition, statistical tests were conducted to show the significance of the results.

Threats to External Validity: A threat to external validity is whether our approach can be generalized to different scenarios. To mitigate this threat, we used a real-world dataset on edge servers, which avoids the possible idiosyncrasies of simulated edge server locations. As for the edge user dataset, in the training phase, we simulated 100,000 distributions to enhance the model's generalization ability. We also simulated 10,000 distributions in the testing phase to avoid the contingency of the results. Moreover, we performed experiments with varying numbers of users, edge servers, and server resource capacities, and the results show that our approach is effective in various scenarios.

Threats to Construct Validity: To mitigate the threats to construct validity, we compared our approach to several baseline and state-of-the-art methods and two variants of our model. Moreover, extensive experiments with different independent variables are conducted to evaluate the performance at different data scales. Therefore, we can validly evaluate our approach against different approaches at different data scales.

VI. RELATED WORK

We discuss the related work from three aspects. In the first aspect, we introduce the work related to the frontiers of computation offloading in MEC and explain the differences between edge user allocation and computation offloading. In the second aspect, we focus on the scenario we are working on, namely, edge user allocation. In the third aspect, we cover deep reinforcement learning for combinatorial optimization problems, the algorithm adopted in this paper.

A. Computation Offloading in MEC

Computation offloading is a crucial technique in MEC that enables users to transfer computational tasks from mobile and IoT devices to edge servers. In recent years, there has been significant research on computation offloading, some of which combine other scenarios, such as mobile prediction and wireless energy transfer. For example, Zaman et al. [25] proposed the LiMPO framework that solves computation offloading combined with mobile prediction. They use a trained lightweight artificial neural network to predict user locations and a multi-objective genetic algorithm to determine the optimal server based on energy consumption, resource utilization, and latency. Zaman et al. [26] also consider user direction, using an LSTM-based model for prediction. Mustafa et al. [27] combined wireless power transfer (WPT) and task offloading in MEC. They provide an overview of recent work on offloading methods to end nodes in MEC and WPT, and they formulate a taxonomy of joint WPT and offloading in MEC. In addition, Tang et al. [28] addressed scenarios where the computational workload exceeds the capacity of the MEC system. To address this challenge, they proposed two offloading algorithms. The first algorithm leverages K-means clustering and genetic algorithms for scenarios with sufficient resources. The other one addresses the multi-knapsack problem for task-overflowed situations. The two algorithms jointly optimize the time and energy of the tasks and penalize the overflowed computations to reduce the task pressure in the next workflow.

Many studies adopt deep reinforcement learning methods in the field of computation offloading. Xiong et al. [2] investigated resource allocation strategies in a single-server, multi-user scenario. They used an improved deep Q-learning (DQN) algorithm to minimize the long-term weighted sum of job completion time and the number of requested resources. Chen et al. [3] allowed for partial task offloading and optimized the same objective using the temporal attentional deterministic policy gradient (TADPG) method, deploying the DRL agent on the device. Wang et al. [4] considered a scenario with multiple edge servers and used DRL to optimize the average service time as well as load balancing of edge server resources and network data links. Dai et al. [5] considered a three-tier architecture consisting of device-edge server-cloud server and used DRL to minimize the energy consumption of the entire system during computation. Qu et al. [29] proposed a Deep Meta Reinforcement learning-based Offloading (DMRO) algorithm to address the issue of poor neural network portability. This framework enables the rapid and flexible learning of optimal edge offloading strategies from dynamic environments.

Despite the relevance of these studies to edge user allocation, there are some differences between computation offloading and our work on edge user allocation. Computation offloading is primarily optimized from the perspective of devices or edge infrastructure providers, considering factors such as task completion time, energy consumption, and channel or link selection. On the other hand, edge user allocation seeks to serve more users with fewer servers, from the perspective of service providers. Due to the different scenarios, the methods in the field of computation offloading cannot be used to solve the problem of edge user allocation.

B. Edge User Allocation

The EUA problem is an essential issue in edge computing. Lai et al. [6] modeled it as a variable-sized vector bin packing problem, which can be solved by the Lexicographic Goal Programming technique. The approach can get the optimal solution but costs too much time. Lai et al. [7] considered the dynamic QoS of app users and presented a heuristic method to quickly yield a sub-optimal solution. He et al. [8] formulated the EUA problem as a potential game and proposed a game-theoretic decentralized approach to solve the EUA problem. Lai et al. [1] proposed a cost-effective heuristic approach named Most Capacity First (MCF) to solve the EUA problem, effectively reducing the number of edge servers required. The above works are methodological improvements to the original EUA problem.

Some works focus on the expansion of the scenario. Peng et al. [30] considered the high mobility of edge users and focused on the reallocation among different edge servers. They designed a mobility-aware and migration-enabled algorithm to allocate edge users in real time. Lai et al. [31] investigated the EUA problem in a multi-cell multi-channel downlink power-domain non-orthogonal multiple access (NOMA) -based mobile edge computing system (MEC) and proposed a decentralized game-theoretic approach. Lai et al. [32] and Wu et al. [23] investigate the online EUA problem of random user arrivals and departures over time. The former proposed an online Lyapunov optimization-based algorithm, and the latter proposed a decentralized reactive approach to allocate users in real time by employing a fuzzy control mechanism. Zou et al. [33] investigated the spatio-temporal edge user allocation (ST-EUA) that considered the decomposability of an edge user's request and the impact of the spatial distance and the temporal dynamics of requests. They proposed a genetic algorithm-based heuristic approach to solve the ST-EUA problem. Panda et al. [34] pointed out that the relationship between resources utilized on an edge server with the number of service requests is usually highly non-linear. They proposed a DRL framework to predict the resource utilization of requests. They can estimate the number of users that an edge server can accommodate for a given latency threshold.

The classical EUA issue is the focus of our work. Although numerous studies have been conducted on this topic in the past, some methods require too much time to execute in large-scale scenarios to meet the real-time requirement; others have attempted to address it by employing heuristic algorithms with a short running time, but the performance of these methods remains unsatisfactory. We utilized its ability to rapidly identify a solution's quality and developed a DRL strategy that performs effectively in a short amount of time.

C. Reinforcement Learning for Combinatorial Optimization Problems

Research on combinatorial optimization problems with deep reinforcement learning has recently been popular. Vinyals et al. [18] first solved TSP with deep learning. They proposed a new neural architecture named Pointer Networks. The input of pointer networks is a variable-length sequence, and the output is the conditional probability of a sequence with elements that are discrete tokens corresponding to positions in the input sequence. They use two RNNs as the encoder and decoder and a simplified Attention Mechanism to calculate the relativities between the decoder state and every encoder state. However, to train the pointer networks with supervised learning, we need optimal labels that are very costly because of the NP-hardness of TSP. Bello et al. [35] train the pointer networks with DRL. They use negative tour length as the reward signal to optimize the parameters of the pointer networks with a policy gradient method and get better results. Nazari et al. [36] investigate DRL for solving the Vehicle Routing Problem (VRP). The VRP is more complicated than the TSP because TSP is a sequence of static nodes, but VRP is a sequence of nodes with dynamic resource requirements. Therefore, they proposed a new architecture that can dynamically update the nodes' resources. They also used the policy gradient method to train the model. Kool et al. [15] proposed a model based on attention layers instead of RNNs in pointer networks. The architecture is similar to the Transformer [14], which can be parallelized. They trained the model using REINFORCE with a simple baseline based on a deterministic greedy rollout, which is more efficient than a value function.

The EUA problem has two sequence inputs, namely edge servers and edge users, in contrast to the previous problems that had only a single sequence input (for instance, a sequence of cities in TSP and locations in VRP). Therefore, the neural networks for these basic sequential decision issues are inapplicable to solving the EUA problem, necessitating the development of a new model. Consequently, we propose DSAM, capable of receiving inputs from two sequences.

VII. Conclusion and Future Work

In this paper, we approach computation offloading from the perspective of service providers and employ deep reinforcement learning (DRL) to solve the EUA problem. We propose a dual-sequence attention model (DSAM) as the DRL agent, which encodes users using self-attention mechanisms and directly outputs the probability of matching between users and servers using an attention-based pointer mechanism. Moreover, we train the model using a policy gradient algorithm with a self-produced baseline. Our approach can produce a good enough solution in real time with a trained model. We evaluated our approach with a real-world dataset and found that DSAM-DRL increases the percentage of users allocated, decreases the percentage of servers required, and significantly increases the resource utilization of the servers against the baseline approaches.

However, there are also some issues that need further consideration. This paper assumes that the edge infrastructure provider can provide edge servers with low enough latency for users within their coverage area, so users have equal weights in terms of latency when selecting an edge server. However, in reality, users can obtain lower latency and higher bandwidth when connecting to servers closer to them. In the future, we will incorporate latency optimization into consideration. In addition, we studied the quasi-static EUA scenario, where users are IoT devices with fixed positions such as smart speakers and traffic cameras. In the future, we will investigate scenarios where users can move among different edge servers, and more factors may need to be incorporated into the state modeling of DRL.

References

[1] P. Lai et al., "Cost-effective app user allocation in an edge computing environment," *IEEE Trans. Cloud Comput.*, vol. 10, no. 3, pp. 1701–1713, Jul.–Sep. 2022.

[2] X. Xiong, K. Zheng, L. Lei, and L. Hou, "Resource allocation based on deep reinforcement learning in IoT edge computing," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1133–1146, Jun. 2020.

[3] J. Chen, H. Xing, Z. Xiao, L. Xu, and T. Tao, "A DRL agent for jointly optimizing computation offloading and resource allocation in MEC," *IEEE Internet Things J.*, vol. 8, no. 24, pp. 17508–17524, Dec. 2021.

[4] J. Wang, L. Zhao, J. Liu, and N. Kato, "Smart resource allocation for mobile edge computing: A deep reinforcement learning approach," *IEEE Trans. Emerg. Topics Comput.*, vol. 9, no. 3, pp. 1529–1541, Jul.–Sep. 2021.

[5] Y. Dai, K. Zhang, S. Maharjan, and Y. Zhang, "Edge intelligence for energy-efficient computation offloading and resource allocation in 5G beyond," *IEEE Trans. Veh. Technol.*, vol. 69, no. 10, pp. 12175–12186, Oct. 2020.

[6] P. Lai et al., "Optimal edge user allocation in edge computing with variable sized vector bin packing," in *Proc. Int. Conf. Serv. Orient. Comput.*, 2018, pp. 230–245.

[7] P. Lai et al., "Edge user allocation with dynamic quality of service," in *Proc. Int. Conf. Serv. Orient. Comput.*, 2019, pp. 86–101.

[8] Q. He et al., "A game-theoretical approach for user allocation in edge computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 3, pp. 515–529, Mar. 2020.

[9] Z. Li and Q. Zhu, "Genetic algorithm-based optimization of offloading and resource allocation in mobile-edge computing," *Information*, vol. 11, no. 2, p. 83, 2020.

[10] G. Liu et al., "Smart traffic monitoring system using computer vision and edge computing," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 8, pp. 12027–12038, Aug. 2022.

[11] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[12] J. L. Elman, "Finding structure in time," *Cogn. Sci.*, vol. 14, no. 2, pp. 179–211, 1990.

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[14] A. Vaswani et al., "Attention is all you need," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 6000–6010.

[15] W. Kool, H. Van Hoof, and M. Welling, "Attention, learn to solve routing problems!" 2018, *arXiv:1803.08475*.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[17] O. Vinyals, S. Bengio, and M. Kudlur, "Order matters: Sequence to sequence for sets," 2015, *arXiv:1511.06391*.

[18] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 2692–2700.

[19] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*.

[20] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, no. 3, pp. 229–256, 1992.

[21] S. J. Rennie, E. Marcheret, Y. Mroueh, J. Ross, and V. Goel, "Self-critical sequence training for image captioning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 1179–1195.

[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[23] C. Wu et al., "Online user allocation in mobile edge computing environments: A decentralized reactive approach," *J. Syst. Archit.*, vol. 113, Feb. 2021, Art. no. 101904.

[24] A. Paszke et al. "Automatic differentiation in PyTorch." 2017. [Online]. Available: https://openreview.net/forum?id=BJJsrmfCZ

[25] S. K. U. Zaman et al., "LiMPO: Lightweight mobility prediction and offloading framework using machine learning for mobile edge computing," *Clust. Comput.*, vol. 26, pp. 99–117, Jan. 2022.

[26] S. K. U. Zaman et al., "COME-UP: Computation offloading in mobile edge computing with LSTM based user direction prediction," *Appl. Sci.*, vol. 12, no. 7, p. 3312, 2022.

[27] E. Mustafa et al., “Joint wireless power transfer and task offloading in mobile edge computing: A survey,” *Clust. Comput.*, vol. 25, no. 4, pp. 2429–2448, 2022.

[28] H. Tang, H. Wu, Y. Zhao, and R. Li, “Joint computation offloading and resource allocation under task-overflowed situations in mobile-edge computing,” *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 2, pp. 1539–1553, Jun. 2022.

[29] G. Qu, H. Wu, R. Li, and P. Jiao, “DMRO: A deep meta reinforcement learning-based task offloading framework for edge-cloud computing,” *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 3, pp. 3448–3459, Sep. 2021.

[30] Q. Peng et al., “Mobility-aware and migration-enabled online edge user allocation in mobile edge computing,” in *Proc. IEEE Int. Conf. Web Serv. (ICWS)*, 2019, pp. 91–98.

[31] P. Lai et al., “Cost-effective user allocation in 5G NOMA-based mobile edge computing systems,” *IEEE Trans. Mobile Comput.*, vol. 21, no. 12, pp. 4263–4278, Dec. 2022.

[32] P. Lai et al., “Dynamic user allocation in stochastic mobile edge computing systems,” *IEEE Trans. Services Comput.*, vol. 15, no. 5, pp. 2699–2712, Sep./Oct. 2022.

[33] G. Zou et al., “ST-EUA: Spatio-temporal edge user allocation with task decomposition,” *IEEE Trans. Services Comput.*, vol. 16, no. 1, pp. 628–641, Jan./Feb. 2023.

[34] S. P. Panda, A. Banerjee, and A. Bhattacharya, “User allocation in mobile edge computing: A deep reinforcement learning approach,” in *Proc. IEEE Int. Conf. Web Serv. (ICWS)*, 2021, pp. 447–458.

[35] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, “Neural combinatorial optimization with reinforcement learning,” 2016, *arXiv:1611.09940*.

[36] M. Nazari, A. Oroojlooy, L. V. Snyder, and M. Takáč, “Reinforcement learning for solving the vehicle routing problem,” in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 9861–9871.

Jian Wang (Member, IEEE) received the Ph.D. degree in computer science from Wuhan University, China, in 2008, where he is currently an Associate Professor with the School of Computer Science. His current research interests include services computing and software engineering.

Bing Li (Member, IEEE) received the Ph.D. degree from the Huazhong University of Science and Technology, Wuhan, China, in 2003. He is currently a Professor with the School of Computer Science, Wuhan University and Hubei Luojia Laboratory, China. His main research areas are services computing, software engineering, artificial intelligence, complex systems, and cloud computing.

Yuqi Zhao received the bachelor’s and master’s degrees from the International School of Software and the School of Computer Science, Wuhan University, where he is currently pursuing the Ph.D. degree. He specializes in building edge-cloud collaboration systems on edge computing environments, user allocation, microservice deployment, service scheduling, and QoS prediction.

Jiaxin Chang received the bachelor’s degree in computer science from Wuhan University, China, in 2021, where he is currently pursuing the master’s degree with the School of Computer Science. His current research interests include edge computing and software engineering.

Duantengchuan Li is currently pursuing the Ph.D. degree with the School of Computer Science, Wuhan University. His research interests include recommendation system, representation learning, natural language processing and their applications in services computing and software engineering.