# Efficient and Green Service Function Chain Deployment and Scheduling for Mobile Edge Computing

Xuejian Chi, Yifei Zou, *Member, IEEE*, Congwei Zhang, Yapu Zhang, Jiguo Yu, *Fellow, IEEE*, Xiuzhen Cheng, *Fellow, IEEE*, Falko Dressler, *Fellow, IEEE*, and Dongxiao Yu, *Senior Member, IEEE*

***Abstract*—Service Function Chain (SFC) decomposes a user's overall service into multiple service components and links them in an orderly manner, providing more flexible, efficient, and responsive services for compute-intensive applications in mobile edge computing. Existing research primarily focuses on optimizing the deployment cost and system efficiency of SFC. However, the dynamic nature of user trajectories, along with the absence of adaptive scheduling and service component sharing in complex and heterogeneous environments, hinders existing approaches from achieving optimal resource efficiency. To achieve adaptive optimization of SFC deployment and efficient component sharing across multiple SFCs, we propose an SFC Deployment and Scheduling strategy based on the Soft Actor-Critic (SAC) algorithm, named SAC-DS. The proposed SAC-DS strategy first employs a Long Short-Term Memory (LSTM) model to predict user trajectories, thereby accurately forecasting future service demands. Based on these predictions, an SAC-based agent is trained to dynamically optimize SFC deployment and scheduling policies in real time. SAC-DS integrates SFC deployment, component sharing, and scheduling into a unified framework, effectively improving system throughput while minimizing overall deployment costs, thereby enhancing user experience. Compared with existing baseline methods, SAC-DS improves network throughput by 3.1%–18.1% and reduces average cost by 1.8%–10.0%.**

***Index Terms*—Service function chain, mobile edge computing, adaptive scheduling, service components sharing.**

## I. Introduction

The Service Function Chain (SFC) deployment problem in mobile edge computing has recently attracted increasing attention from researchers [1]. This problem involves decomposing application services into smaller components, which can be pre-deployed on multiple resource-constrained edge servers and linked together in an orderly manner to form various services [2]. Compared with traditional methods of overall service deployment, this deployment offers a more flexible, automated, and orchestratable approach to managing network service paths in heterogeneous and resource-constrained environments, effectively reducing service latency and improving service quality. Therefore, SFC provides more efficient and responsive services for latency-sensitive and computation-intensive applications such as Augmented Reality (AR) and Virtual Reality (VR) [3].

Due to its significance, related studies have been conducted on resource scheduling and task offloading for edge-based SFC, with a primary focus on optimizing deployment cost*, system stability and network throughput efficiency [4]–[16]. However, the inherent mobility of user, heterogeneity, network connectivity instability and resource constraints in the edge computing environment still lead to four core problems. Firstly, the dynamic nature of user trajectory means that static SFC deployment strategies often fail to cover the entire service cycle of a user, potentially resulting in severe processing delays or even service interruptions [4]–[6]. Secondly, traditional methods rely on approximation and heuristic algorithms to obtain suboptimal solutions, which are derived from simplified modeling of the complex relationships between deployment cost and service latency among different types of SFCs in real mobile edge computing environments [7]–[9]. Therefore, in complex and time-varying network environments, such methods struggle to adaptively determine optimal deployment and scheduling strategies. Thirdly, the resilience of the overall mobile edge computing based service scheduling needs to be taken into consideration so that SFCs can be reestablished if needed [17], [18]. Finally, since users may request different types of services, deploying a complete SFC independently for each user can lead to inefficient resource utilization [10]–[12]. For instance, both video analytics services and VR applications typically involve object detection and recognition modules. Deploying these modules separately for each service request results in redundant deployments, wasting both computational and storage resources [13], [14]. Therefore, there is an urgent need for a new approach that can simultaneously consider user trajectory prediction, adaptive deployment in complex environments, and cross-service component sharing.

To address the above issues, we investigate the problem of SFC deployment and scheduling with user trajectory prediction and service component sharing mechanisms. An example of the shareable service function chain in this problem is shown in Fig. 1, User 1 requests a video object detection service, while User 2 requests a VR rendering service. These

Xuejian Chi, Yifei Zou, Congwei Zhang, and Xiuzhen Cheng are with the School of Computer Science and Technology, Shandong University, and also with Shandong Provincial Key Laboratory of Computing-Network Integration (Shandong University), Qingdao, Shandong 266237, China. E-mail: {chixuejian, xczhw}@mail.sdu.edu.cn, {yfzou, xzcheng}@sdu.edu.cn

Dongxiao Yu is with the School of Cryptologic Science and Engineering, Shandong University, Jinan, Shandong 250101, China. E-mail: dxyu@sdu.edu.cn

Yapu Zhang is with the Institute of Operations Research and Information Engineering, Beijing University of Technology, Beijing, 100081, P. R. China. E-mail: zhangyapu@bjut.edu.cn

Jiguo Yu is with School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, 611731, China, and also with the Big Data Institute, Qilu University of Technology, Jinan, 250353, China, E-mail: jiguoyu@sina.com

Falko Dressler is with the School of Electrical Engineering and Computer Science, TU Berlin, Berlin, 10587, Germany. E-mail: dressler@ccs-labs.org

(Corresponding author: Yifei Zou.)

*Deployment cost refers to the one-time overhead of initializing a new service component on an edge server, including image pulling, resource allocation, container startup, and component configuration.

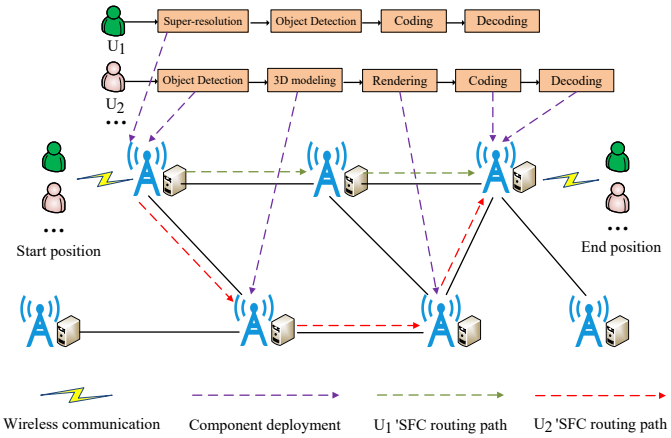



Fig. 1: An example of SFC deployment and scheduling.

two services can be decomposed into different service function chains, with some components (e.g., encoding, decoding, and object detection modules) being shared. Based on user trajectory prediction and service component sharing, we can dynamically and adaptively determine the optimal deployment, scheduling. By maximizing network throughput (i.e., the total number of user requests that the system can process within a given time period) while minimizing deployment costs, we provide users with a seamless and low-latency service experience. However, the adaptive and efficient deployment and scheduling of SFC still face the following three challenges.

Firstly, user mobility complicates SFC deployment decisions, making it critically important to accurately predict user trajectory in order to guide subsequent SFC deployment [19], [20]. When users move beyond the coverage area of the current server, if service components are not migrated in time or pre-deployed to other servers, task execution failures or service interruptions may occur [21], [22]. Intuitive solutions typically pre-deploy all user service components on every server to ensure seamless service delivery and reduce waiting latency. However, this approach may result in significant resource waste on servers that are never accessed by some users, thereby reducing overall resource utilization efficiency.

Secondly, the time-varying characteristics of the network environment, along with differences in deployment costs and service latency among different types of SFCs, make the adaptive deployment and scheduling of SFCs highly complex. Due to the intricate temporal dependencies and resource competition among SFCs, traditional optimization methods fail to leverage historical information or temporal correlations, making it difficult to accurately capture and depict these complex characteristics. As a result, they are unable to dynamically determine the optimal SFC deployment and scheduling strategy for each user.

Finally, to maximize resource savings, the framework proposed in this paper encourages different users to share certain service components as much as possible. As illustrated in Fig. 1, although User 1 and User 2 request distinct service function chains, three components can still be shared between them. However, due to the dynamic variation in user locations and the complex relationships involved in sharing service components among users, adaptively determining the service component sharing strategy becomes essential.

To address the aforementioned challenges, this paper proposes **SAC-DS**, an efficient and green[†] SFC **D**eployment and **S**cheduling algorithm via Soft Actor-Critic (**SAC**) strategy. First, the proposed method leverages a Long Short-Term Memory (LSTM) network to predict user trajectories, enabling accurate prediction of future service demands. Subsequently, SAC-DS designs a reward function that balances deployment cost and network throughput, and incorporates an exploration mechanism to train a deep reinforcement learning agent based on the SAC algorithm. This agent is used to model the complex nonlinear relationships between deployment cost and service latency across different types of SFCs. This enables the SAC-DS to adaptively determine optimal deployment locations and migration strategies for each requested SFC, while simultaneously optimizing the service component sharing mechanism among multiple SFCs, thereby enhancing overall network performance and user experience.

This paper makes the following contributions:

- We investigate the problem of SFC deployment and scheduling with user trajectory prediction and service component sharing mechanisms, and prove its NP-hardness. Compared with previous works that primarily focus on static deployment and lack SFC adaptive sharing mechanisms, our approach can predict user trajectories, pre-deploy the user's SFC to the corresponding servers, and achieve efficient resource utilization through the sharing of certain components.
- To address the proposed problem, we formulate a mathematical optimization model and formalize it as a Markov Decision Process (MDP), with the objective of maximizing network throughput while minimizing total system cost. An efficient SFC deployment and scheduling strategy based on the SAC algorithm, named SAC-DS, is proposed. This strategy integrates LSTM-based user trajectory prediction with deep reinforcement learning, thereby enabling adaptive determination of optimal strategies for SFC sharing, deployment, and scheduling.
- Extensive experiments are conducted to evaluate the performance of the proposed SAC-DS strategy. Compared with existing baseline methods, SAC-DS improves network throughput by 3.1%–18.1% and reduces average cost by 1.8%–10.0%.

The remainder of this paper is organized as follows. Section II reviews the related work on SFC deployment and scheduling. Section III presents the system model and formally formulates the optimization problem. In Section IV, we propose the SAC-DS strategy, which optimally guides the deployment and scheduling of service components, aiming to maximize network throughput while minimizing cost. Section V evaluates the performance of the proposed approach. Finally, Section VII concludes the paper.

[†]Green refers to the saving of edge server resources, including computing, storage, and bandwidth resources.

TABLE I: Feature Comparison of SFC Deployment and Scheduling Methods.

| Work | Method | Optimality | Online Decision Making | Service Migration | User Trajectory Prediction | Service Component Sharing |
|---|---|---|---|---|---|---|
| [5] | Heuristic and Approximation Methods | ✗ | ✓ | ✗ | ✗ | ✓ |
| [6] | Heuristic and Approximation Methods | ✗ | ✓ | ✗ | ✗ | ✗ |
| [7],[8] | Heuristic and Approximation Methods | ✗ | ✗ | ✓ | ✓ | ✗ |
| [9]-[11] | Exact Optimization Methods | ✓ | ✗ | ✓ | ✗ | ✗ |
| [12] | Exact Optimization Methods | ✓ | ✗ | ✓ | ✗ | ✓ |
| [13]-[15] | Intelligent Learning-Based Methods | ✓ | ✓ | ✓ | ✗ | ✗ |
| [16],[17] | Intelligent Learning-Based Methods | ✓ | ✓ | ✗ | ✗ | ✓ |
| Ours | SAC-DS | ✓ | ✓ | ✓ | ✓ | ✓ |

• Optimality (✓/✗): can / cannot obtain the optimal solution.
• Online Decision Making (✓/✗): support / does not support online decision making.
• Service Migration (✓/✗): service components can / cannot be dynamically migrated.
• User Trajectory Prediction (✓/✗): able / unable to predict user mobility for proactive deployment.
• Service Component Sharing (✓/✗): support / does not support reuse of components across service chains.

## II. Related Work

In SFC deployment and scheduling, existing studies can be broadly divided into three categories. The first category includes heuristic and approximation methods, which rely on problem structures or engineering rules to quickly generate feasible solutions under real-time and resource constraints. For example, Yue *et al.* considered component shareability to improve resource utilization [4]; Thiruvas *et al.* used complex-network-based heuristics to mitigate service degradation [5]; and Li *et al.* designed an approximation algorithm for service cost minimization and dynamic admission control [7].

The second category includes exact optimization methods, such as Integer Linear Programming (ILP) and dynamic programming, which explicitly model resource constraints, service paths, and network topology to obtain optimal or near-optimal solutions. For example, Liu *et al.* formulated joint service placement and routing as an ILP problem to improve request acceptance [8]; Xia *et al.* designed a dynamic programming-based instance consolidation strategy for availability and fault tolerance [9]; and Yang *et al.* proposed a redundancy optimization algorithm to reduce service disruption risks in dynamic 5G networks [10].

The third category is based on intelligent decision-making, with deep reinforcement learning (DRL) as its core. These methods emphasize learning-based and adaptive mechanisms to cope with complex and dynamic network environments. Unlike traditional approaches, they do not rely on explicit modeling. Instead, agents interact with the environment to learn deployment and scheduling strategies, making them particularly suitable for scenarios with highly dynamic traffic and time-varying resource availability. For example, He *et al.* proposed a DRL algorithm based on policy gradients and Kronecker-factored trust region methods to optimize service function placement and service quality in IoT scenarios [12]. Wang *et al.* leveraged multi-agent deep reinforcement learning to coordinate deployment and routing tasks, introducing a joint reward mechanism to achieve optimal performance [13]. Pei *et al.* formulated the component placement problem as a binary integer programming task and applied a double Deep Q Network-based method (DQN-based) to improve the reliability of deployment [14].

TABLE II: List of Main Notations.

| Notation | Description |
|---|---|
| $U$ | The set of users requesting SFC |
| $A$ | The set of Access Points |
| $L$ | The set of physical links |
| $u_i$ | The i-th user in mobile edge computing |
| $e_n$ | The n-th edge server in mobile edge computing |
| $f_{j_m}$ | The types of service components |
| $S_i^t$ | The SFC requested by user $u_i$ at time $t$ |
| $r_i$ | The packet rate sent by user $u_i$ |
| $D_i^{req}$ | The latency constraints of user $u_i$ |
| $R_n^{COM}$ | The available computational resources of edge server $e_n$ |
| $R_n^{STO}$ | The available storage resources of edge server $e_n$ |
| $R_{n,n'}^{BW}$ | The available bandwidth resources between $e_n$ and $e_{n'}$ |
| $c_{j,n}$ | The cost of deploying a component of type $f_j$ on $e_n$ |
| $c_{i,n}^{com}$ | The processing cost for $u_i$ per unit data rate on $e_n$ |
| $c_{i,(n,n')}^{ban}$ | The communication cost per unit data rate between $e_n$ and $e_{n'}$ |
| $\Omega_i^t$ | Whether the SFC request issued by $u_i$ is accepted at time $t$ |
| $d_{j,n}$ | The deployment delay of a component of type $f_j$ on $e_n$ |
| $d_{i,n}^{com}$ | The processing delay for $u_i$ per unit data rate on $e_n$ |
| $d_{i,(n,n')}^{ban}$ | The communication delay per unit data rate between $e_n$ and $e_{n'}$ |
| $q_{j,n}^t$ | The queueing delay of component $f_j$ on edge server $e_n$ at time $t$ |
| $C^t$ | The total cost of fulfilling user requests |
| $D_i^t$ | The total delay required to complete the SFC for user $u_i$ |

However, SFC deployment and scheduling remains challenging due to resource limitations, performance requirements, reliability guarantees, and service function ordering constraints [23]–[25]. Heuristic and exact optimization methods often rely on simplified models and overlook historical experience, while existing learning-based methods usually ignore user trajectory variations and adaptive service component sharing in MEC scenarios. As shown in TABLE I, SAC-DS addresses these limitations by integrating user trajectory prediction and adaptive component sharing, enabling the agent to make deployment and scheduling decisions based on predicted service demands. This improves network throughput and reduces deployment costs.

## III. System Overview and Problem Modeling

This section provides an overview of the efficient SFC deployment and scheduling system based on deep reinforcement learning. Additionally, it offers a detailed description of the

various components of the problem model. TABLE II presents the key symbols that will be used throughout this paper.

### A. Service Function Chain Network Model

The physical network is modeled as an undirected graph $W = (U, A, L)$, where $U$ denotes the user set requesting the service function chains, $A$ denotes the collection of Access Points (APs), and $L$ denotes the physical link set. Each AP is associated with an edge server equipped with certain computation and storage resources. Therefore, in the following discussion, no further distinction is made between APs and edge servers. The edge servers are represented by a set $E = \{e_1, e_2, \cdots, e_n, \cdots, e_N\}, n \in [1, \cdots, N]$, where the edge servers are heterogeneous. The parameter $R^{BW}_{n,n'}$ denotes the available bandwidth resources between node $e_n$ and node $e_{n'}$; $R^{COM}_n$ denotes the available computational resources at the edge server $e_n$; and $R^{STO}_n$ denotes the available storage resources at the edge server $e_n$. The network system operates continuously along a time axis, where time begins at $t = 0$ and increases incrementally in a sequential manner. Each time slot has a fixed duration, denoted in $\tau$ seconds.

### B. Service Function Chain Request Model

We represent the user request set as $U = \{u_1, u_2, \cdots, u_i, \cdots, u_I\}$, $i \in [1, 2, \cdots, I]$. The network function service is provided by the network operator, and the type of service component is defined by $F = \left\{f_1, f_2, \cdots, f_j, \cdots, f_J\right\}, j \in [1, \cdots, J]$. Each specific user request is defined as $u_i = (S^t_i, r_i, D^{req}_i)$, where $S^t_i = \left\{f^t_{j_1} \rightarrow f^t_{j_2} \rightarrow \cdots \rightarrow f^t_{j_m}\right\}$ denotes the SFC requested by $u_i$ at time $t$, $r_i$ denotes the packet rate sent by $u_i$, which affects the resources and latency consumed by the server in processing user requests, and $D^{req}_i$ denotes the specific latency constraints of $u_i$.

### C. Matrix Definition

To provide a clearer representation of the deployment and scheduling of service components within the mobile edge computing network, we define the following four matrices:

*1) Deployment Matrix:* $\Psi = \left[\left\{\psi^t_{j,n}\right\}_{J\times N}\right]$. If a service component of type $f_j$ is to be deployed on edge server $e_n$ at time $t$, then $\psi^t_{j,n} = 1$; otherwise, $\psi^t_{j,n} = 0$. Note that if an edge server has already deployed a service component of type $f_j$ prior to time $t$, there is no need for further deployment of the same component. The deployment state of service components in the mobile edge computing network at time $t$ is represented by the matrix $\overline{\Psi} = \left[\left\{\overline{\psi}^t_{j,n}\right\}_{J\times N}\right]$.

*2) Scheduling Matrix:* $\Phi_i = \left[\left\{\varphi^t_{(i,j),n}\right\}_{J\times N}\right]$. If a service component of type $f_j$ deployed on edge server $e_n$ at time $t$ is scheduled to serve user $u_i$, then $\varphi^t_{(i,j),n} = 1$; otherwise, $\varphi^t_{(i,j),n} = 0$.

*3) Routing Matrix:* $L_i = \left[\left\{l^t_{i,(n,n')}\right\}_{N\times N}\right]$. If the data sent by user $u_i$ traverses the link between edge server $e_n$ and $e_{n'}$ at time $t$, then $l^t_{i,(n,n')} = 1$; otherwise, $l^t_{i,(n,n')} = 0$. Additionally, if $n = n'$, then $l^t_{i,(n,n')} = 0$.

*4) Ergodic Matrix:* $X_i = \left[\left\{\chi_{i,j_m}\right\}_{I\times M}\right]$. This matrix represents the completion order of service components in the SFC requested by user $u_i$. For example, if $f^t_{j_1}$ is the first component to be completed in the actual execution, then $\chi_{i,j_m} = 1$; if $f^t_{j_1}$ is the second component to be completed, then $\chi_{i,j_m} = 2$, and so on.

### D. Cost and Delay Model

The total cost of the SFC deployment and scheduling system primarily focuses on three key aspects: deployment cost, processing cost, and bandwidth cost [26], [27]. Deployment cost pertains to the initial setup of servers and service components. Processing cost is related to the consumption of computational resources during the operation of service components, such as CPU utilization. Bandwidth cost denotes the network bandwidth resources required for data transmission. In comparison to these costs, storage cost is generally considered relatively low in edge computing environments and, therefore, is not considered in the scope of this discussion. Therefore, the total cost of fulfilling user requests in the mobile edge computing network is defined in Eq. (1).

$$\begin{aligned} C^t = &\sum_{j=1}^{J}\sum_{n=1}^{N} c_{j,n}\cdot\psi^t_{j,n} + \sum_{i=1}^{I}\sum_{n=1}^{N} c^{com}_{i,n}\cdot r_i\cdot\varphi^t_{(i,j),n}\cdot\Omega^t_i \\ &+ \sum_{i=1}^{I}\sum_{n=1}^{N}\sum_{n'=1}^{N} c^{ban}_{i,(n,n')}\cdot r_i\cdot l^t_{i,(n,n')}\cdot\Omega^t_i, \end{aligned} \tag{1}$$

the three terms in Eq. (1) correspond to deployment cost, processing cost, and bandwidth cost, respectively. $c_{j,n}$ denotes the cost of deploying a service component of type $f_j$ on edge server $e_n$, $c^{com}_{i,n}$ denotes the processing cost for user $u_i$ per unit data rate on edge server $e_n$, and $c^{ban}_{i,(n,n')}$ denotes the bandwidth cost per unit data rate for user $u_i$ over the link between $e_n$ and $e_{n'}$. Additionally, $\Omega^t_i \in [0, 1]$ indicates whether the mobile edge computing network accepts the SFC request issued by user $u_i$ at time $t$.

In dynamic service environments, deployment latency becomes a critical factor that cannot be overlooked. When services require the dynamic addition or removal of service components, deployment latency arises from activities such as container image pulling, container creation and startup, as well as the configuration and initialization of components. Additionally, data processing latency, communication latency, and queueing latency are also pivotal factors. As the number of service components fluctuates, the data processing load may vary, potentially causing variations in processing latency. The data transfer between service components is dependent on bandwidth resources, which introduces communication latency. Furthermore, when multiple users are scheduled to the same shared service component instance, later requests need to wait in the processing queue, which introduces queueing latency. Therefore, the total delay required to complete the

SFC for user $u_i$ is defined in Eq. (2).

$$D_i^t = \sum_{\forall f_j \in S_i^t} \sum_{n=1}^{N} d_{j,n} \cdot \psi_{j,n}^t + \sum_{\forall f_j \in S_i^t} \sum_{n=1}^{N} d_{i,n}^{com} \cdot r_i \cdot \varphi_{(i,j),n}^t + \sum_{\forall f_j \in S_i^t} \sum_{n=1}^{N} \sum_{n'=1}^{N} d_{i,(n,n')}^{ban} \cdot r_i \cdot l_{i,(n,n')}^t + \sum_{\forall f_j \in S_i^t} \sum_{n=1}^{N} q_{j,n}^t \cdot \varphi_{(i,j),n}^t, \tag{2}$$

where $d_{j,n}$ denotes the latency of deploying a service component of type $f_j$ on edge server $e_n$, $d_{i,n}^{com}$ denotes the processing latency for user $u_i$ per unit data rate on edge server $e_n$, and $d_{i,(n,n')}^{ban}$ denotes the communication latency per unit data rate for user $u_i$ over the link between $e_n$ and $e_{n'}$. Moreover, $q_{j,n}^t$ denotes the waiting time of the queue associated with service component $f_j$ on edge server $e_n$ at time slot $t$. The last term in Eq. (2) captures the waiting time introduced when multiple users are scheduled to the same shared service component instance. This queueing delay reflects the time-isolated processing mechanism of shared components, where a shared component processes one user's request at a time and other requests wait in the queue. For users sharing the same service component, SAC-DS adopts a first-in-first-out (FIFO) rule across different time slots, which prevents later-arriving requests from bypassing earlier requests in the same shared component queue. For users arriving in the same time slot, SAC-DS orders their requests according to their latency constraints in ascending order, so that requests with stricter latency requirements are processed earlier. This scheduling rule helps prevent starvation caused by arbitrary priority assignment or repeated preemption, while reducing QoS violations for delay-sensitive users. From a security perspective, shared components are reused only through service interfaces; after each request is completed, request-specific contexts such as intermediate data, temporary cache, and request metadata are cleared, and practical deployments can further use sandboxed or hardened container runtimes such as Kata Containers or gVisor to strengthen tenant isolation [28], [29].

### *E. Constraints*

During the operation of edge servers, a series of resource constraints need to be met. Given the limited computational and storage resources available on edge servers, these resources must be allocated in a rational and efficient manner to enable the parallel processing of multiple user requests [30]. Additionally, due to limited network bandwidth, edge servers must account for this constraint during data transmission to ensure effective and stable communication. An edge server can process multiple service components at the same time, as long as the total computational, storage, and bandwidth resources consumed by these components do not exceed the server's resource capacities. Therefore, parallel processing is supported through the computation, storage, and bandwidth constraints in Eqs. (3)−(5), respectively.

$$\sum_{i=1}^{I} r_{n,j}^{COM} \cdot r_i \cdot \varphi_{(i,j),n}^t \le R_n^{COM}, \forall n \in [1, \cdots, N], \tag{3}$$

$$\sum_{i=1}^{I} r_{n,j}^{STO} \cdot r_i \cdot \varphi_{(i,j),n}^t \le R_n^{STO}, \forall n \in [1, \cdots, N], \tag{4}$$

$$\sum_{i=1}^{I} r_{n,j}^{BW} \cdot r_i \cdot l_{i,(n,n')}^t \le R_n^{BW}, \forall n \in [1, \cdots, N], \tag{5}$$

where $r_{n,j}^{COM}$ denotes the computational resources required by edge server $e_n$ to process a service component of type $f_j$ per unit data rate, $r_{n,j}^{STO}$ denotes the storage resources required by edge server $e_n$ for processing a service component of type $f_j$ per unit data rate, and $r_{n,j}^{BW}$ denotes the bandwidth resources needed to transmit a service component of type $f_j$ per unit data rate between node $e_n$ and node $e_{n'}$.

We employ container-based virtualization technology, enabling flexible resource allocation to efficiently handle user requests in each time slot. This implies that a service component of the same type needs to be deployed only once on a given edge server. Furthermore, if a service component of type $f_j$ is to be deployed on edge server $e_n$ at time $t$, it must not have been deployed on the same server at time $t-1$:

$$\sum_{t=1}^{\infty} \psi_{j,n}^t + \overline{\psi}_{j,n}^t \le 1, \forall j \in [1, \cdots, J], n \in [1, \cdots, N], \tag{6}$$

$$\sum_{t=1}^{\infty} \sum_{n=1}^{N} \psi_{j,n}^t + \overline{\psi}_{j,n}^t \le N, \forall j \in [1, \cdots, J], \tag{7}$$

$$\psi_{j,n}^t = \overline{\psi}_{j,n}^t - \overline{\psi}_{j,n}^{t-1} \le N, \forall j \in [1, \cdots, J], n \in [1, \cdots, N]. \tag{8}$$

Additionally, the scheduling of a service component of type $f_j$ deployed on edge server $e_n$ at time $t$ to a user is contingent upon the condition that the service component of type $f_j$ is either deployed on edge server $e_n$ at time $t$, or was previously deployed on edge server $e_n$ prior to time $t$:

$$\varphi_{(i,j),n}^t = \begin{cases} 1, & \overline{\psi}_{i,n}^t = 1, \\ 0, & otherwise. \end{cases} \tag{9}$$

The lifecycle of a deployed service component is dynamically managed. When a component is not currently used by any user request, it is marked as inactive rather than being immediately removed from the edge server. This design allows future requests to reuse inactive components and avoids unnecessary repeated deployment overhead. Component eviction is triggered only when a new component needs to be deployed but the storage resource of the target edge server is insufficient. In this case, SAC-DS selects components from the inactive component set and evicts them according to the Least Recently Used (LRU) replacement strategy. Active components that are currently serving user requests are not evicted.

Considering the presence of user mobility and the need to ensure service quality, edge servers must ensure that user requests are processed within the specified latency constraints when handling their SFCs. If the time taken exceeds the latency constraint, it indicates that the user request cannot be serviced by the edge server. To account for this, a latency sensitivity coefficient $\phi$ is introduced, and the latency constraint is given in Eq. (10). For latency-sensitive users, $\phi = 1$; for non-latency-sensitive users, $\phi$ can take a value greater than 1:

$$D_i^t \le \phi \cdot D_i^{req}, \forall i \in [1, \cdots I], \tag{10}$$

Eq. (10) ensures that the total delay of each accepted request does not exceed its latency requirement.

For an SFC, the order dependency constraints between service components are critical. Each service component is responsible for processing a specific portion of the network traffic, and any disruption or omission in the execution sequence of these components may lead to service interruptions or errors. Furthermore, the complete execution of the SFC requested by the user is essential. This implies that once the network decides to accept the request issued by user $u_i$, it must ensure that all service components in the sequence $S_i^t$ are successfully processed in their entirety:

$$\chi_{i,j_m} \cdot \Omega_i^t = \begin{cases} m, & \Omega_i^t = 1, \\ 0, & \Omega_i^t = 0, \end{cases} \tag{11}$$

where $m$ denotes the total number of service components in the SFC requested by the user.

In order to maintain the sequential integrity of the data flow sent by the user, it is essential not only to ensure that the data flows through each service component in the predefined order, but also to guarantee the continuity between the edge servers and the links. This means that, when data is transmitted from one service component to the next, it must seamlessly traverse the links connected to the edge servers, avoiding any disruption or unnecessary communication delay. The routing continuity constraint is given in Eq. (12). If $\varphi^t_{(i,j),n} = 1$ and $f^t_{j_m} = f_j$, then:

$$l^t_{i,(n,n')} = \begin{cases} 1, & \varphi^t_{(i,j'),n'} = 1, f^t_{j_{m+1}} = f_{j'}, \\ 0, & \varphi^t_{(i,j'),n'} = 1, f^t_{j_{m+1}} \neq f_{j'}, \\ 0, & \varphi^t_{(i,j'),n'} = 0, \end{cases} \tag{12}$$

this equation indicates that the integrity of the SFC link can only be ensured when a service component of type $f_{j'}$ is deployed on server $e_{n'}$ and can be scheduled to user $u_i$.

Finally, although the mobile edge computing network is designed to efficiently handle user requests, due to the limitations of network resources, the stringent latency constraints of users, and other objective factors, it is not always possible to process all user requests in real-time. Therefore, not all requests can be admitted:

$$\sum_{i=1}^{I} \Omega_i^t \leq I. \tag{13}$$

### F. Problem Definition

The optimization objective is to minimize the cost of completing user requests while maximizing the throughput of the mobile edge computing network. The coefficients $\eta$ and $\beta$ are used to normalize cost and throughput, eliminating the effect of dimensional inconsistency. $\delta$ is a weighting factor, allowing us to dynamically adjust cost and throughput based on actual needs and network conditions. The smaller the $\delta$ value, the higher the throughput of the system, which can minimize the cost of user requests while ensuring a certain network throughput, and vice versa.

$$P1: \quad \min \ \eta \cdot \delta \cdot C^t - \beta \cdot (1-\delta) \cdot \sum_{i=1}^{I} \Omega_i^t \tag{14}$$

$$s.t. \quad (3), (4), (5), (6), (7), (8), (9), (10), (11), (12), (13).$$

### G. Computational Complexity

The proposed SFC deployment and scheduling problem involves joint service component deployment, user scheduling, routing, and component sharing decisions under resource and latency constraints. These coupled decisions make the problem computationally challenging. We formally establish its computational hardness as follows.

**Lemma 1.** *The P1 problem is NP-hard.*

The proof is based on a polynomial-time reduction from the Multiple Facility Location Problem (MFLP), which is a classical NP-hard combinatorial optimization problem [31]. In the reduction, facilities are mapped to edge servers, customers are mapped to user requests, facility opening costs are mapped to service component deployment costs, and customer-facility service costs are mapped to the cost of assigning SFC requests to edge servers. Thus, solving P1 would also solve the corresponding MFLP instance. The detailed proof is provided in Appendix A.

According to Lemma 1, the SFC deployment and scheduling problem is unlikely to be solved optimally in polynomial time. This motivates the design of an adaptive learning-based strategy for large-scale and dynamic SFC deployment and scheduling, which will be introduced in the next section.

## IV. Efficient SFC Deployment and Scheduling

This section first employs the Long Short-Term Memory (LSTM) network to predict user trajectories, and the predicted results are then fed into the deep reinforcement learning Soft Actor-Critic (SAC) algorithm. Subsequently, the SAC-DS strategy is employed to achieve efficient and stable SFC deployment and scheduling within a complex mobile edge computing network environment. Through this comprehensive framework, the deployment and scheduling strategies can be dynamically adjusted and optimized based on the actual user trajectory, thereby maximizing network throughput, minimizing user request costs, and ultimately improving overall network performance and quality of service. The overall workflow of the proposed method is illustrated in Fig. 2.

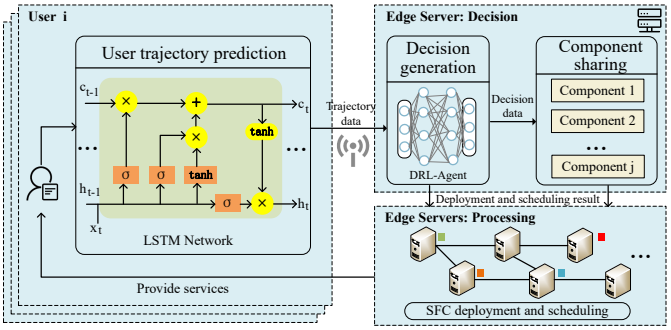



Fig. 2: The overview of our proposed method.

### A. LSTM Network

To address the limitation of fixed source and aggregation access point assumptions in existing SFC deployment research,

we propose a dynamic strategy that leverages user trajectory prediction. Specifically, we utilize an LSTM network to predict the user's trajectory during the service period, enabling adaptive determination of the source and aggregation access points. First, historical user trajectory data is collected. Each data entry includes the user $u_i$'s starting position $p_i^s$, request initiation time $t_i^s$, request duration $t_i^e$, endpoint position $p_i^e$, and setting $t_i^e = D_i^{req}$. The starting position coordinates are discretized using an equidistant grid (e.g., dividing the physical space into cells of $500m$ or $1000m$ scales) [32].

From each user trajectory record, the following features are extracted as inputs for the LSTM model: $\langle p_i^s, t_i^s, t_i^e \rangle$. The LSTM's output is the user's predicted trajectory, specifically the sequence of predicted locations (e.g., grid cell IDs) for future time steps across the entire service period $[t_i^s, t_i^e]$. The LSTM network can effectively capture long-term dependencies in time-series mobility data through its gating mechanism, which makes it suitable for predicting user trajectories in mobile edge computing scenarios. To keep the main text concise, the detailed gate-level formulation of the LSTM prediction model is provided in Appendix B.

During online scheduling, the LSTM-based trajectory prediction is refreshed at the beginning of each time slot using the user's latest observed mobility information. When multiple candidate trajectories are generated, SAC-DS selects the trajectory with the highest prediction probability as the input to the SAC agent, which keeps the state representation compact. The LSTM model is trained using Mean Squared Error (MSE) [33] to measure the distance between predicted and actual positions, and optimized by Adam. For evaluation, we further report the grid-level prediction accuracy, defined as the proportion of time steps in which the predicted grid cell matches the user's actual grid cell.

### B. Deep Reinforcement Learning Algorithm

Deep reinforcement learning integrates the principles of deep learning and reinforcement learning, aiming to solve complex decision-making problems by approximating value functions or policies through neural networks [34], [35]. At each time step, the agent observes the current state $s_t$ from the environment and uses its policy network to select an action $a_t$. After the action is executed, the agent receives a reward $r_t$ and observes the new state $s_{t+1}$. The agent then uses this experience tuple to update its network parameters, typically by minimizing the temporal-difference error between predicted and target values. This process is repeated iteratively, enabling the agent to learn an optimal policy that maximizes the expected cumulative reward through continuous interaction with the environment.

We formulate the SFC deployment and scheduling problem in the mobile edge computing network as a Markov Decision Process (MDP) with infinite states, discount costs, and throughput [36]. The Markov Decision Process is defined as $<S, A, P, R, \gamma>$, where $S$ denotes the system state space, $A$ denotes the discrete action set, $P$ denotes the system state transition probabilities, $R$ is the reward function, and $\gamma$ is the discount factor.

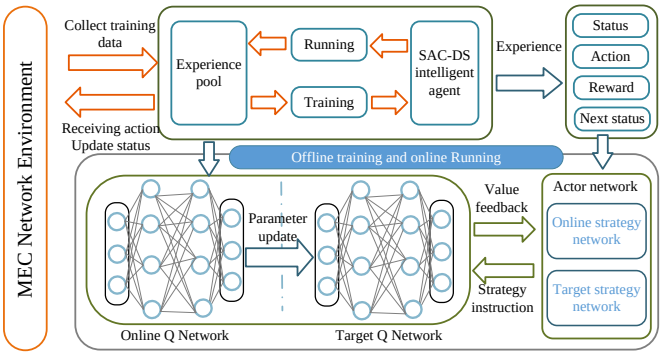



Fig. 3: SAC-DS framework.

*1) System state S:* At time slot $t$, the state is represented as $s_t = \left(\hat{\tau}^t, \overline{\Psi}^t, R_{re}^{COM,t}, R_{re}^{STO,t}, R_{re}^{BW,t}\right)$, where $\hat{\tau}^t$ is the predicted user trajectory, $\overline{\Psi}^t$ is the current deployment state of service components, $R_{re}^{COM,t}$ and $R_{re}^{STO,t}$ are the remaining computational and storage resources of edge servers, and $R_{re}^{BW,t}$ is the remaining bandwidth resource state. The trajectory state $\hat{\tau}^t$ is obtained from the LSTM predictor. Since the LSTM may generate multiple candidate future trajectories with different probabilities, directly including all candidates would significantly enlarge the state space. Therefore, SAC-DS adopts a greedy top-1 strategy and uses the trajectory with the highest prediction probability as the trajectory input. This keeps the state representation compact and improves training efficiency, while the possible resource waste caused by prediction errors is limited according to our trajectory prediction results.

The edge server state records both the deployed service components and the remaining server resources. Specifically, $\overline{\Psi}^t = [\{\overline{\psi}_{j,n}^t\}_{J\times N}]$ indicates whether component $f_j$ has already been deployed on edge server $e_n$, while $R_{re}^{COM,t}$ and $R_{re}^{STO,t}$ record the remaining computational and storage resources, respectively. The bandwidth state $R_{re}^{BW,t}$ records the remaining bandwidth resources of physical links.

*2) Action A:* The action consists of two parts: the deployment action and the scheduling action of service components. At time slot $t$, the action can be represented as $a_t = (\Psi^t, \Phi^t)$, where $\Psi^t = \left[\left\{\psi_{j,n}^t\right\}_{J\times N}\right]$ denotes the deployment action decided by the agent. If a service component of type $f_j$ is newly deployed on edge server $e_n$ at time $t$, then $\psi_{j,n}^t = 1$; otherwise, $\psi_{j,n}^t = 0$. Let $\Phi^t = \left[\left\{\varphi_{(i,j),n}^t\right\}_{I\times J\times N}\right]$ denote the scheduling action. If the service component of type $f_j$ on edge server $e_n$ is scheduled to serve user $u_i$, then $\varphi_{(i,j),n}^t = 1$; otherwise, $\varphi_{(i,j),n}^t = 0$.

Service component sharing is encoded through the joint deployment and scheduling action. Specifically, if a required component $f_j$ has already been deployed on server $e_n$, i.e., $\overline{\psi}_{j,n}^t = 1$, and the agent sets $\varphi_{(i,j),n}^t = 1$ without triggering a new deployment action $\psi_{j,n}^t = 0$, then the request of user $u_i$ reuses the existing component instance on $e_n$. In this case, multiple users can be scheduled to the same deployed component instance. Therefore, component sharing is not introduced as an additional hard-coded action variable; instead, it is implicitly represented by scheduling users to already deployed components when such reuse reduces deployment

cost without violating resource and latency constraints.

*3) Reward and objective function:* The objective function of maximum entropy reinforcement learning consists of two components: one is the reward function, which depends on the current state and the action taken; the other is the entropy regularization term, which is related to the entropy of the policy. The purpose of the entropy regularization term is to encourage the agent to explore a broader range of actions, thereby preventing premature convergence to a local optimum and enhancing the diversity and exploration of the policy.

$$J(\pi) = \sum_{t=0}^{T} [r(s_t, a_t) + \alpha \cdot H(\pi(\cdot \mid s_t))], \tag{15}$$

where $r(s_t, a_t)$ denotes the traditional immediate reward function, defined as $r(s_t, a_t) = -(\eta \cdot \delta \cdot C^t - \beta \cdot (1-\delta) \cdot \sum_{i=1}^{I} \Omega_i^t)$, which depends on the current state $s_t$ and the action $a_t$ taken. $H(\pi(\cdot \mid s_t))$ denotes the entropy of the policy $\cdot$ at state $s_t$. Entropy is a measure of the uncertainty of a random variable, and in this context, it is used to encourage stochasticity and promote exploration in the policy.

In a discrete action space, the entropy can be defined as $H(\pi(\cdot \mid s_t)) = -\log \pi(a_t \mid s_t)$, where $\pi(a_t \mid s_t)$ represents the probability of taking action $a_t$ in state $s_t$, which serves as a weight for computing the weighted average. $\alpha$ is the coefficient of the entropy regularization term, also referred to as the temperature coefficient, which controls the relative importance of the entropy term in the overall reward function. When the temperature coefficient $\alpha$ is large, the agent is encouraged to explore a wider range of actions; conversely, when $\alpha$ is small, the agent tends to exploit the known information more, reducing exploration.

### C. SAC-DS Network Architecture

As illustrated in Fig. 3, SAC-DS follows a maximum-entropy Actor-Critic architecture, which consists of an Actor network, Critic networks, target Q-networks, and an experience replay buffer. Given the current system state, the Actor network outputs a probability distribution over the discrete action space through a Softmax layer, thereby generating SFC deployment and scheduling decisions. The Critic networks estimate the state-action value, i.e., the Q-value, to evaluate the long-term return of the selected action. The target Q-networks are updated through a soft-update mechanism to stabilize training, while the experience replay buffer stores historical interaction samples for off-policy learning.

During training, the Actor network is updated to maximize the entropy-regularized expected return, and the Critic networks are updated to reduce the temporal-difference error of Q-value estimation. In addition, the temperature coefficient $\alpha$ is adaptively tuned to balance exploration and exploitation: a larger $\alpha$ encourages more diverse deployment and scheduling actions, while a smaller $\alpha$ makes the policy more exploitative. The detailed loss functions for the Actor network, Critic networks, and temperature coefficient update are provided in Appendix C.

The training and online decision process of SAC-DS follows the standard interaction-update loop of maximum-entropy reinforcement learning. At each time slot, the agent observes the current MEC state, selects a deployment and scheduling action, receives the corresponding reward, and stores the transition in the experience replay buffer. The detailed pseudocode of SAC-DS is provided in Appendix C.

## V. Performance Evaluation

This section demonstrates the superior performance of the proposed SAC-DS strategy through comparison with seven benchmark strategies. The section first presents the simulation parameter settings, followed by an overview of the seven benchmark methods used for comparison. Finally, based on the simulation results, the impact of different parameter configurations on key performance metrics, such as total cost and throughput, in fulfilling user requests is analyzed.

### A. Simulation Parameter Setup

We conduct experiments using a custom Python-based discrete-event simulator. The user mobility traces are obtained from the Geolife dataset [37], and the experiments are conducted in a $5km \times 5km$ urban area. A network topology with 100 access points and 1200 links is generated by GT-ITM, and the user mobility traces are mapped onto this logical edge network. In the simulator, mobile users, terminal devices, edge servers, service components, and network links are modeled as logical entities. A total of 500 high-mobility users are selected from the dataset to construct the mobility trace pool, and service requests are generated according to the workload settings of different experiments. Each experiment is independently repeated 100 times with different random seeds, and the reported results are averaged over these runs.

The main simulation parameters are as follows. The SFC length is randomly drawn from $[4, 8]$, the number of SFC types is set to 10, and the deployment cost of each service component is randomly chosen from $[0.1, 0.5]$. The latency constraint of each user follows a uniform distribution in $[10, 30]$ seconds. The processing delay and communication delay are randomly sampled from $[0.1, 0.5]$ and $[0.01, 0.5]$ seconds per unit data rate, respectively [38]. The communication delay includes both user-edge and inter-edge-server transmission latency. The link bandwidth resources are uniformly distributed in $[100, 600]$ Mbps [39].

For SAC training, the learning rates of the Q-network ($\lambda_Q$) and policy network ($\lambda_\pi$) are both set to $10^{-3}$. The soft-update coefficient $\mu$, discount factor $\gamma$, and reward weighting factor $\delta$ are set to 0.005, 0.9, and 0.2, respectively. The hidden layer size is 256, the replay buffer size is $1 \times 10^4$, and the update frequency ratio between the Critic and Actor networks is 2:1. During training, 10 testing cycles are conducted after every 10 training cycles, and training stops when both reward and loss values become stable.

### B. Benchmarks

The selected baselines are designed to cover the major paradigms of recent SFC deployment and scheduling methods. Specifically, DQN and DDQN represent value-based DRL methods, A3C and PPO represent policy-gradient and

TABLE III: Proportion of requests and service success rate under varying user request volumes and cost ranges.

| Number of user requests | Cost range of user requests | Proportion of requests | Service success rate |
|---|---|---|---|
| 250 | (0-4) | 40% | 100% |
| | [4-6) | 30% | 100% |
| | [6-8) | 30% | 100% |
| 350 | (0-4) | 40% | 88% |
| | [4-6) | 30% | 85% |
| | [6-8) | 30% | 83% |
| 450 | (0-4) | 40% | 95% |
| | [4-6) | 30% | 72% |
| | [6-8) | 30% | 69% |

actor-critic DRL methods, GNN represents topology-aware learning-based scheduling, DL-TPA represents hybrid deep learning-based optimization, and KPMST-H represents traditional heuristic optimization. Therefore, the comparison covers both learning-based and non-learning-based approaches.

- **Deep Q-Network (DQN) [15]**: DQN is a classical value-based DRL method for discrete decision-making and is used to learn SFC deployment and scheduling policies.
- **Deep Learning-based Two-Phase Algorithm (DL-TPA) [16]**: DL-TPA is a hybrid learning-based method that decomposes SFC placement and link allocation into two sequential optimization phases.
- **Key-node Preferred Minimum Spanning Tree based Heuristic (KPMST-H) [6]**: KPMST-H is a traditional heuristic method that constructs routing trees and greedily performs component deployment and routing.
- **Graph Neural Network (GNN) [40]**: GNN is a topology-aware learning method that encodes the MEC network as a graph and generates SFC deployment decisions based on learned graph representations.
- **Proximal Policy Optimization (PPO) [41]**: PPO is a policy-gradient method that uses clipped policy updates to improve training stability.
- **Double Deep Q-Network (DDQN) [42]**: DDQN is an enhanced value-based DRL method that reduces Q-value overestimation by decoupling action selection and action evaluation.
- **Asynchronous Advantage Actor-Critic (A3C) [43]**: A3C is an asynchronous actor-critic method that uses multiple parallel workers to improve exploration and training efficiency.

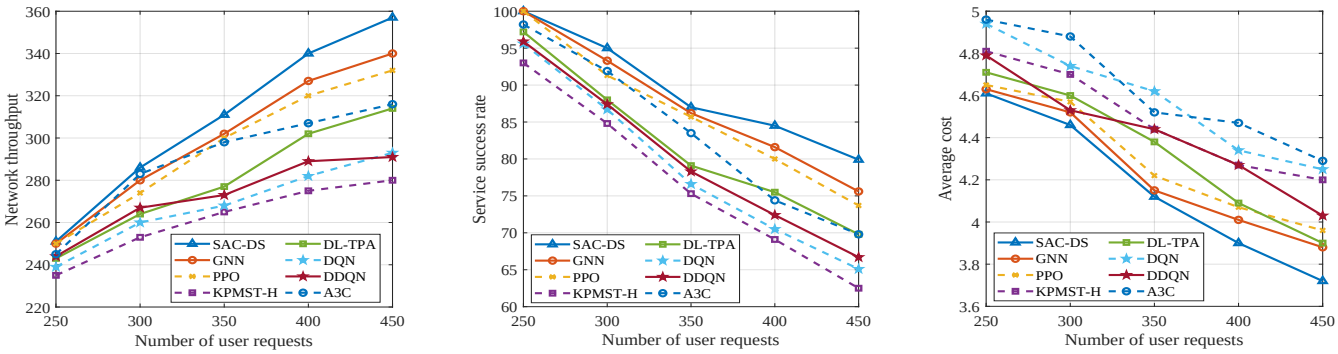
### C. Comparison with Benchmarks

*1) The impact of the number of user requests:* Network Throughput. We define network throughput as the number of SFC user requests that can be successfully processed by the system within a given time period. The number of user requests for SFCs is set to 250, 300, 350, 400, and 450. As shown in Fig. 4(a), when the number of user requests is 250, system resources are sufficient, and the network throughput achieved by the SAC-DS strategy is slightly higher than that of the other seven strategies. As the number of requests gradually increases and system resources become limited, the network throughput of the SAC-DS strategy becomes significantly higher than the other seven strategies. On average, SAC-DS achieves approximately 3.1%–18.2% higher network throughput compared to other strategies.

Service success rate. We define service success rate as the ratio of successfully served user requests to the total number of user requests within a time period. As shown in Fig. 4(b), the service success rate of all strategies decreases as the number of user requests increases. However, SAC-DS consistently outperforms the other seven strategies. When the number of user requests reaches 450, the service success rate of SAC-DS is approximately 5.7%, 8.4%, 14.5%, 14.5%, 19.8%, 22.7%, and 27.8% higher than that of GNN, PPO, DL-TPA, A3C, DDQN, DQN, and KPMST-H, respectively. The reason for the gradual decline in service success rate with increasing user requests is the limited availability of network resources, which prevents the system from serving all requests. As a result, SAC-DS tends to prioritize requests with lower resource consumption while ignoring or delaying those with higher demands. This explanation is supported by TABLE III. When the number of user requests is 250, all requests achieve a 100% service success rate due to sufficient resources. However, as the number of requests increases to 350, resource constraints lead to a decline in the service success rates across all request types. Nevertheless, the success rate for low-cost requests (cost range (0-4)) remains higher than that for high-cost ones (cost ranges [4–6) and [6–8)), indicating that SAC-DS tends to prioritize low-cost requests under limited resource conditions. When the request volume further increases to 450, this trend becomes more pronounced: the service success rate of low-cost requests significantly surpasses that of high-cost requests.

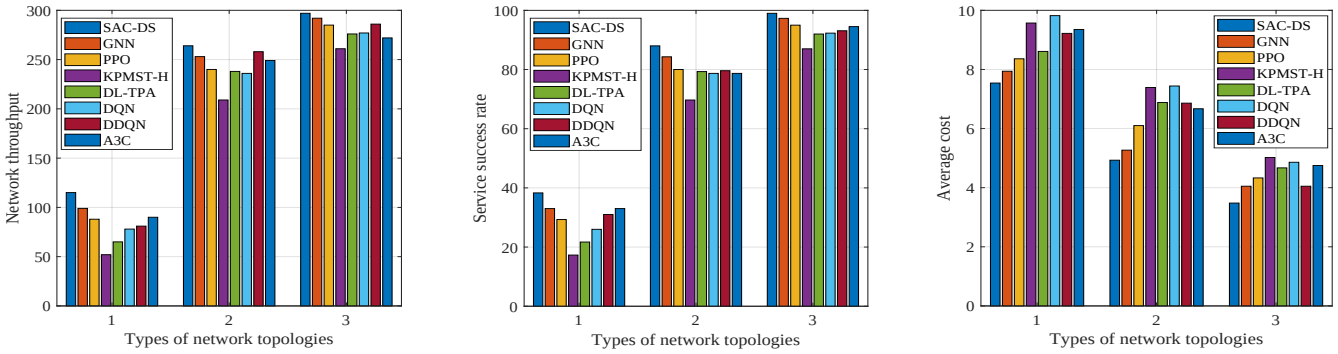
Average Cost. As shown in Fig. 4(c), with increasing number of user requests, the average cost of fulfilling these requests shows a consistent downward trend. The rise in user requests leads the strategy to potentially abandon some high-resource, high-cost requests in favor of accepting lower-resource, lower-cost ones (refer to TABLE III). This tendency reflects the strategy's focus on resource utilization efficiency. By prioritizing low-cost requests, the strategy can achieve more efficient resource allocation, thereby effectively reducing the average cost. On average, SAC-DS achieves approximately 1.8%–10.0% lower average cost compared to other strategies.

*2) The impact of the network topology:* To evaluate the impact of network topology on system performance, we fixed the number of user requests at 300 and considered three types of network topologies: star, mesh, and fully connected. In the star topology, all edge servers are connected to a central server via dedicated links, and communication between any two servers is relayed through the center. The mesh topology features partial direct interconnections among servers, where communication can occur through multi-hop forwarding. Eight different schemes were tested in terms of network throughput, service success rate, and average cost, as shown in Fig. 5. The results indicate that, compared with the other seven schemes, SAC-DS improves network throughput and service success rate by approximately 5.0%–29.5%, while reducing the average cost by about 7.6%–27.9%. These findings demonstrate that SAC-DS maintains high system performance and



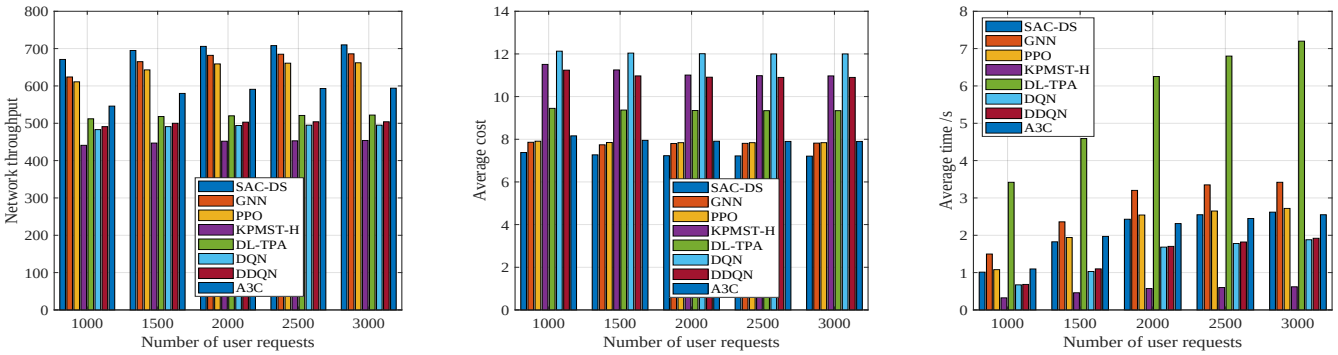

(a) Network throughput. (b) Service success rate. (c) Average cost.

Fig. 4: The impact of the number of user requests on system performance.




(a) Network throughput. (b) Service success rate. (c) Average cost.

Fig. 5: The impact of the network topology on system performance.

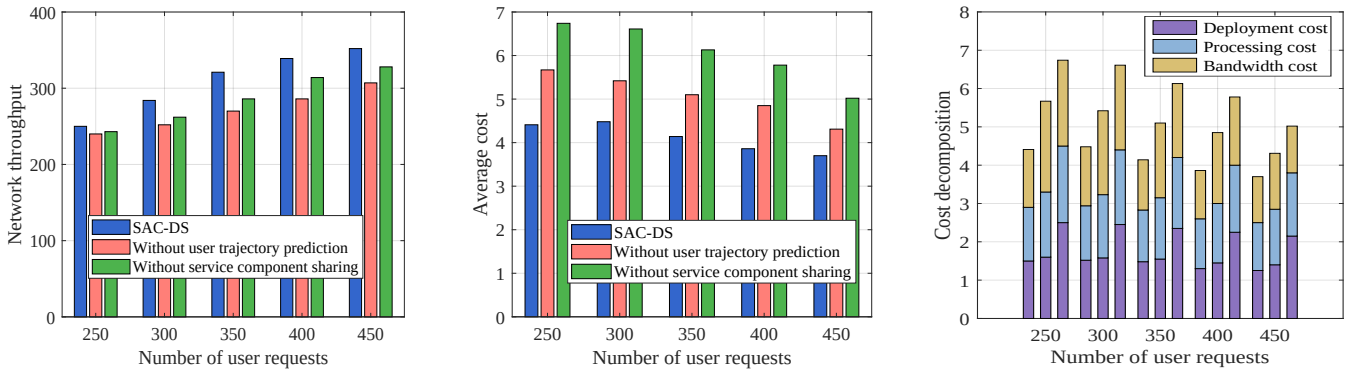



(a) Network throughput. (b) Average cost. (c) Average time.

Fig. 6: Large-scale validation and computational complexity assessment.

stability under various network topologies, highlighting its strong adaptability and robustness.

*3) Large-scale validation and computational complexity assessment:* In large-scale networks, as the number of user requests increases, SAC-DS achieves precise service chain deployment through user trajectory prediction, thereby avoiding inefficient resource allocation. Moreover, the service component sharing mechanism further enhances the overall resource utilization efficiency of the system. To evaluate the scalability and computational complexity of SAC-DS in large-scale network scenarios, we designed two sets of experiments. The first set focused on large-scale validation, where the number of user requests was set between 1000 and 3000, and the network throughput and average cost of eight schemes were tested. The second set aimed to assess computational complexity, measuring the average time required for each scheme to generate SFC deployment and scheduling strategies under the same range of user requests. The experimental results are shown in Fig. 6. Compared with the other seven schemes, SAC-DS improved network throughput by approximately 4.4% to 55.3% and reduced average cost by approximately 7.0% to 39.7% in large-scale scenarios. Moreover, when the number of user requests increases from 2000 to 3000, the network throughput and average cost curves gradually become stable, indicating that the




(a) Ablation study on network throughput. (b) Ablation study on average cost. (c) Finer-grained decomposition of cost.

Fig. 7: Ablation study of SAC-DS and its variants. (a) Network throughput. (b) Average cost. (c) Cost decomposition, where each group of bars corresponds to SAC-DS, Variant 1, and Variant 2, respectively.

TABLE IV: Contribution of User Trajectory Prediction.

| Schemes | Ratio of $D_i^t > D_r^{req}$ | Ratio of SFC components scheduled |
|---|---|---|
| SAC-DS | 11.1% | 8.5% |
| Variant 1 | 32.4% | 45.7% |

TABLE V: Contribution of Service Component Sharing.

| Schemes | Number of deployed SFC components | Total redundancy ratio of various SFC components |
|---|---|---|
| SAC-DS | 1327 | 15.9% |
| Variant 2 | 2054 | 55.4% |

system approaches a stable operating region under fixed edge resource capacity. The average execution time of SAC-DS for a single SFC deployment and scheduling task is comparable to that of A3C and PPO, lower than that of GNN and DL-TPA, and higher than that of KPMST-H, DQN, and DDQN. In practical ultra-large-scale MEC environments, SAC-DS can be deployed regionally according to geographical areas, base-station clusters, or edge-server groups, so that each SAC-DS instance only handles a bounded local state and action space with lightweight inter-region information exchange. These results indicate that SAC-DS maintains excellent performance in large-scale network scenarios and maintains computational complexity comparable to SOTA methods.

### *D. Impact of Hyperparameters and Ablation Study*

*1) Impact of Hyperparameters:* We further evaluate the convergence behavior of SAC-DS under different discount factors $\gamma$ and learning rates $\lambda$. The discount factor controls the importance of future rewards, while the learning rate determines the update step size of network parameters. The results show that SAC-DS achieves stable convergence when $\lambda = 0.001$ under both $\gamma = 0.5$ and $\gamma = 0.9$. In contrast, larger learning rates such as $\lambda = 0.01$ and $\lambda = 0.1$ may lead to local optima or unstable oscillations due to overly aggressive parameter updates. The detailed convergence curves of network throughput, average cost, and cumulative reward are provided in Appendix D.

*2) Ablation Study:* SAC-DS mainly relies on two core components: user trajectory prediction and service component sharing. To evaluate their contributions, we compare SAC-DS with two variants: Variant 1 removes user trajectory prediction, and Variant 2 removes service component sharing. Fig. 7(a) and Fig. 7(b) report the network throughput and average cost results, while Fig. 7(c) further decomposes the average cost into deployment cost, processing cost, and bandwidth cost.

Compared with Variant 1, SAC-DS improves average network throughput by about 14.1% and reduces average cost by about 18.8%. As shown in TABLE IV, when the number of user requests is 450, Variant 1 has a higher ratio of requests violating the latency constraint ($D_i^t > D_r^{req}$) and a higher ratio of scheduled service components. This indicates that, without trajectory prediction, the system tends to perform reactive rescheduling after users move, leading to more request timeouts and higher cost. The cost decomposition in Fig. 7(c) further shows that removing trajectory prediction mainly increases bandwidth cost (from 1.20–1.51 under SAC-DS to 1.46–2.37 under Variant 1), because less accurate service decisions lead to more frequent component migrations and higher communication overhead. Thus, user trajectory prediction enables proactive deployment along users' future mobility paths and reduces migration-induced bandwidth consumption.

Compared with Variant 2, SAC-DS improves average network throughput by about 7.9% and reduces average cost by about 32%. As shown in TABLE V, Variant 2 deploys more service components and produces a higher redundancy ratio when the number of user requests is 450. This indicates that, without component sharing, the system repeatedly deploys identical components for different SFCs, resulting in unnecessary resource consumption. The cost decomposition in Fig. 7(c) further shows that removing component sharing mainly increases deployment cost (from 1.25–1.50 under SAC-DS to 2.15–2.50 under Variant 2), confirming that component sharing reduces redundant service instantiation across edge servers. Thus, service component sharing reduces redundant deployments and improves resource utilization.

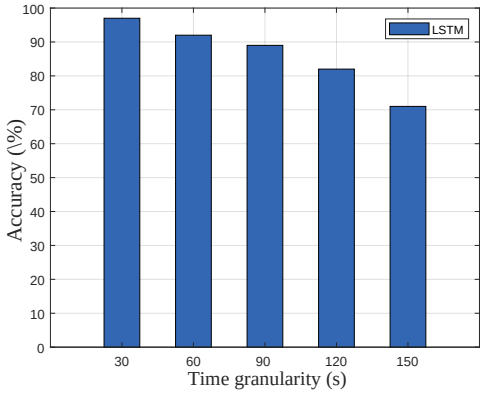



Fig. 8: LSTM prediction accuracy.

TABLE VI: Robustness evaluation under dynamic mobility scenarios.

| Scenario | Avg. Acc. | Min. Acc. | Succ. Rate | Avg. Cost |
|---|---|---|---|---|
| Normal mobility | 97.1% | 95.4% | 94.9% | 4.47 |
| High-speed mobility | 92.4% | 84.7% | 92.5% | 4.88 |
| Abrupt movement | 87.3% | 79.4% | 89.5% | 5.22 |

*3) LSTM Model Prediction:* We evaluate the LSTM prediction performance under different time granularities, including 30s, 60s, 90s, and 120s. As shown in Fig. 8, the LSTM achieves the highest grid-level prediction accuracy of 97.32% when the time-slot length is set to $\tau = 30s$. Therefore, we set $\tau = 30s$ in the following experiments to balance prediction accuracy and scheduling responsiveness.

To further evaluate the robustness of the LSTM module and the overall SAC-DS framework under dynamic mobility conditions, we conduct additional experiments under three mobility scenarios, as shown in TABLE VI. The experiments are conducted with 300 user requests. In the TABLE VI, Avg. Acc. denotes the average grid-level prediction accuracy over all evaluated time slots, while Min. Acc. denotes the minimum grid-level prediction accuracy observed among these time slots. Besides the normal mobility setting, we construct two stress-test scenarios: high-speed mobility with doubled user speed, and abrupt movement with random direction changes.

As shown in TABLE VI, the prediction accuracy decreases as the mobility pattern becomes more dynamic, but the degradation remains bounded. Under high-speed mobility and abrupt movement, the average prediction accuracy remains 92.4% and 87.3%, respectively, while the minimum accuracy remains 84.7% and 79.4%. More importantly, SAC-DS remains stable under sudden mobility changes: the service success rate only decreases from 94.9% to 92.5% and 89.5%, while the average cost increases from 4.47 to 4.88 and 5.22. This is because SAC-DS operates as a closed-loop decision-making system, where updated user locations and reward feedback guide subsequent deployment and scheduling decisions to mitigate the impact of prediction errors.

*4) Sensitivity Analysis:* We conduct sensitivity analysis on the reward weighting factor $\delta$, resource capacity, and request similarity to evaluate the robustness of SAC-DS. The results show that $\delta$ provides a controllable trade-off between throughput and cost, SAC-DS maintains a higher service success rate under different resource capacities, and higher request similarity further improves the benefit of component sharing. Detailed results and curves are provided in Appendix D.

## VI. Acknowledge

This work was supported in part by the National Key R&D Program of China (No. 2023YFB2703600), National Natural Science Foundation of China under Grant 62572280, U24A20244.

## VII. Conclusion

This paper investigates the problem of SFC deployment and scheduling in mobile edge computing networks and proposes an efficient approach, named SAC-DS. Compared with existing static and non-sharing SFC approaches, SAC-DS can proactively predict user trajectories and formulate the optimal strategies. SAC-DS addresses trajectory prediction using an LSTM network and trains a deep reinforcement learning agent to adaptively determine strategies for SFC deployment, scheduling, and component sharing, thereby achieving seamless integration from user trajectory prediction to service deployment and scheduling. Compared with existing baseline methods, SAC-DS improves network throughput by 3.1%–18.1% and reduces average cost by 1.8%–10.0%. Finally, considering the deployment and scheduling of SFC under privacy protection will be our work in the future.

## References

[1] P. Jin, X. Fei, Q. Zhang, F. Liu, and B. Li, "Latency-aware VNF Chain Deployment with Efficient Resource Reuse at Network Edge," in *Proc. of IEEE INFOCOM*, 2020, pp. 267–276.

[2] H. Huang, J. Tian, H. Yin, G. Min, D. Wu, and W. Miao, "RQAP: Resource and QoS Aware Placement of Service Function Chains in NFV-Enabled Networks," *IEEE Transactions on Services Computing*, vol. 16, no. 6, pp. 4526–4539, 2023.

[3] Z. Ni, H. Chen, B. Gao, K. Lin, L. Wu, and J. Yu, "Reward-Oriented Task Offloading in Energy Harvesting Collaborative Edge Computing Systems," *IEEE Transactions on Mobile Computing*, vol. 23, no. 12, pp. 14 414–14 426, 2024.

[4] Y. Yue, B. Cheng, M. Wang, B. Li, X. Liu, and J. Chen, "Throughput Optimization and Delay Guarantee VNF Placement for Mapping SFC Requests in NFV-Enabled Networks," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4247–4262, 2021.

[5] P. K. Thiruvasagam, A. Chakraborty, A. Mathew, and C. S. R. Murthy, "Reliable Placement of Service Function Chains and Virtual Monitoring Functions With Minimal Cost in Softwarized 5G Networks," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1491–1507, 2021.

[6] O. Alhussein, P. T. Do, J. Li, Q. Ye, W. Shi, W. Zhuang, X. Shen, X. Li, and J. Rao, "Joint VNF Placement and Multicast Traffic Routing in 5G Core Networks," in *Proc. of IEEE Global Communications Conference (GLOBECOM)*, 2018, pp. 1–6.

[7] J. Li, S. Guo, W. Liang, Q. Chen, Z. Xu, W. Xu, and A. Y. Zomaya, "Digital Twin-Assisted, SFC-Enabled Service Provisioning in Mobile Edge Computing," *IEEE Transactions on Mobile Computing*, vol. 23, no. 1, pp. 393–408, 2024.

[8] L. Liu, S. Guo, G. Liu, and Y. Yang, "Joint Dynamical VNF Placement and SFC Routing in NFV-Enabled SDNs," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4263–4276, 2021.

[9] R. Xia, H. Dai, J. Zheng, R. Gu, X. Wang, W. Wang, and G. Chen, "Safe: Service availability via failure elimination through vnf scaling," *IEEE/ACM Transactions on Networking*, vol. 31, no. 5, pp. 2042–2057, 2023.

[10] L. Yang, J. Jia, H. Lin, and J. Cao, "Reliable dynamic service chain scheduling in 5g networks," *IEEE Transactions on Mobile Computing*, vol. 22, no. 8, pp. 4898–4911, 2023.

[11] G. Li, H. Chen, L. Wu, X. Chi, J. Yao, F. Xia, and J. Yu, "Efficient Deployment and Scheduling of Shared VNF Instances in Mobile Edge Computing Networks," *IEEE Internet of Things Journal*, vol. 11, no. 19, pp. 32 259–32 271, 2024.

[12] B. He, J. Wang, Q. Qi, H. Sun, and J. Liao, "Towards Intelligent Provisioning of Virtualized Network Functions in Cloud of Things: A Deep Reinforcement Learning Based Approach," *IEEE Transactions on Cloud Computing*, vol. 10, no. 2, pp. 1262–1274, 2022.

[13] S. Wang, C. Yuen, W. Ni, Y. L. Guan, and T. Lv, "Multiagent Deep Reinforcement Learning for Cost- and Delay-Sensitive Virtual Network Function Placement and Routing," *IEEE Transactions on Communications*, vol. 70, no. 8, pp. 5208–5224, 2022.

[14] J. Pei, P. Hong, M. Pan, J. Liu, and J. Zhou, "Optimal VNF Placement via Deep Reinforcement Learning in SDN/NFV-Enabled Networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 263–278, 2020.

[15] W. Lv, Q. Wang, P. Yang, Y. Ding, B. Yi, Z. Wang, and C. Lin, "Microservice Deployment in Edge Computing Based on Deep Q Learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 11, pp. 2968–2978, 2022.

[16] J. Pei, P. Hong, K. Xue, D. Li, D. S. L. Wei, and F. Wu, "Two-Phase Virtual Network Function Selection and Chaining Algorithm Based on Deep Learning in SDN/NFV-Enabled Networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1102–1117, 2020.

[17] D. Ergenç, A. Memedi, M. Fischer, and F. Dressler, "Resilience in Edge Computing: Challenges and Concepts," *Foundations and Trends in Networking*, vol. 14, no. 4, pp. 254–340, 5 2025.

[18] F. Dressler, C. F. Chiasserini, F. H. P. Fitzek, H. Karl, R. Lo Cigno, A. Capone, C. E. Casetti, F. Malandrino, V. Mancuso, F. Klingler, and G. A. Rizzo, "V-Edge: Virtual Edge Computing as an Enabler for Novel Microservices and Cooperative Computing," *IEEE Network*, vol. 36, no. 3, pp. 24–31, 5 2022.

[19] X. Chi, H. Chen, G. Li, Z. Ni, N. Jiang, and F. Xia, "EDSP-Edge: Efficient Dynamic Edge Service Entity Placement for Mobile Virtual Reality Systems," *IEEE Transactions on Wireless Communications*, vol. 23, no. 4, pp. 2771–2783, 2024.

[20] B. Gao, Z. Zhou, F. Liu, and F. Xu, "Winning at the Starting Line: Joint Network Selection and Service Placement for Mobile Edge Computing," in *Proc. of IEEE INFOCOM*, 2019, pp. 1459–1467.

[21] Y. Zhao, L. Zuo, D. Yu, R. Pan, L. Fu, Y. Lu, and H. Zhang, "Mobile Systems for Healthcare 5.0: Applications, Challenges and Opportunities," *High-Confidence Computing*, vol. 6, no. 2, p. 100392, 2026.

[22] J. N. Njoku, E. C. Nkoro, R. M. Medina, P. M. Custodio, C. I. Nwakanma, J.-M. Lee, and D.-S. Kim, "Digital Twin and Metaverse-Enhanced Battery Management for Electric Vehicles," *High-Confidence Computing*, vol. 6, no. 1, p. 100358, 2026.

[23] H. Sun, H. Chen, Z. Ni, X. Fan, G. Li, and F. Xia, "Computing While Navigating: A Novel Task Offloading and Trajectory Planning Scheme for UAV-Assisted MEC System," *IEEE Transactions on Vehicular Technology*, pp. 1–11, 2025.

[24] S. Qi, H. Ma, Y. Zou, Y. Yuan, P. Li, and D. Yu, "Fed-MS: Fault Tolerant Federated Edge Learning with Multiple Byzantine Servers," in *2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS)*, 2024, pp. 982–992.

[25] C. Zhang, Y. Zou, Z. Zhang, D. Yu, J. T. Gómez, T. Lan, F. Dressler, and X. Cheng, "Distributed age-of-information scheduling with noma via deep reinforcement learning," *IEEE Transactions on Mobile Computing*, 2024.

[26] R. Wang, X. Yu, Q. Wu, C. Yi, P. Wang, and D. Niyato, "Efficient Deployment of Partial Parallelized Service Function Chains in CPU+DPU-Based Heterogeneous NFV Platforms," *IEEE Transactions on Mobile Computing*, vol. 23, no. 10, pp. 9090–9107, 2024.

[27] X. Chi, H. Chen, Z. Ni, H. Sun, P. Sun, and D. Yu, "H-STEP: Heuristic Stable Edge Service Entity Placement for Mobile Virtual Reality Systems," *IEEE Transactions on Mobile Computing*, vol. 24, no. 8, pp. 7377–7388, 2025.

[28] Y. Yin, H. Chen, Y. Gao, P. Sun, L. Wu, Z. Li, and W. Liu, "FFCBA: Feature-based Full-target Clean-label Backdoor Attacks," in *Proceedings of the 33rd ACM International Conference on Multimedia*, ser. MM '25, 2025.

[29] M. Alshammari *et al.*, "Service Function Chaining security survey: Addressing security challenges and threats," *Computer Networks*, vol. 221, p. 109490, 2023.

[30] T. Liu, S. Ni, X. Li, Y. Zhu, L. Kong, and Y. Yang, "Deep Reinforcement Learning Based Approach for Online Service Placement and Computation Resource Allocation in Edge Computing," *IEEE Transactions on Mobile Computing*.

[31] G. Yan-cheng, "Multi-facility location model with timeliness constrains of logistics distribution centers," in *2008 IEEE International Conference on Automation and Logistics*, 2008, pp. 2533–2536.

[32] L. Wang, S. Chen, F. Chen, Q. He, and J. Liu, "B-Detection: Runtime Reliability Anomaly Detection for MEC Services With Boosting LSTM Autoencoder," *IEEE Transactions on Mobile Computing*, vol. 23, no. 4, pp. 2599–2613, 2024.

[33] E. Saleh, M. Alsmadi, and S. Ikki, "FD MU-MIMO Systems: Performance Analysis in the Presence of Imperfect CSI and Non-Ideal Transceivers," *IEEE Transactions on Mobile Computing*, vol. 24, no. 1, pp. 422–434, 2025.

[34] Y. Hao, S. Yang, F. Li, Y. Zhang, S. Wang, and X. Ren, "Edgetimer: Adaptive multi-timescale scheduling in mobile edge computing with deep reinforcement learning," in *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications*, 2024, pp. 671–680.

[35] L. Lu, Q. Li, D. Zhao, Y. Yang, Z. Luan, J. Zhou, Y. Jiang, and M. Xu, "Hawkeye: A Dynamic and Stateless Multicast Mechanism with Deep Reinforcement Learning," in *Proc. of IEEE INFOCOM*, 2023, pp. 1–10.

[36] T. Zhao, S. Zhou, Y. Sun, and Z. Niu, "A Predictive Frame Transmission Scheme for Cloud Gaming in Mobile Edge Cloudlet Systems," *IEEE Transactions on Mobile Computing*, vol. 22, no. 7, pp. 3774–3789, 2023.

[37] Y. Zheng, L. Wang, R. Zhang, X. Xie, and W.-Y. Ma, "GeoLife: Managing and Understanding Your Past Life over Maps," in *Proc. of The Ninth International Conference on Mobile Data Management (mdm 2008)*, 2008, pp. 211–212.

[38] Y. Ma, W. Liang, M. Huang, W. Xu, and S. Guo, "Virtual Network Function Service Provisioning in MEC Via Trading Off the Usages Between Computing and Communication Resources," *IEEE Transactions on Cloud Computing*, vol. 10, no. 4, pp. 2949–2963, 2022.

[39] G. Li, H. Chen, L. Wu, X. Chi, J. Yao, F. Xia, and J. Yu, "Efficient Deployment and Scheduling of Shared VNF Instances in Mobile Edge Computing Networks," *IEEE Internet of Things Journal*, vol. 11, no. 19, pp. 32 259–32 271, 2024.

[40] K. Huang, C. Qiu, C. Hou, X. Li, and X. Wang, "HyperJet: Joint Communication and Computation Scheduling for Hypergraph Tasks in Distributed Edge Computing," in *IEEE INFOCOM 2025*, 2025, pp. 1–10.

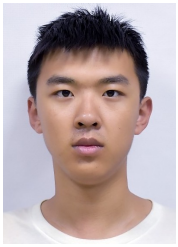
[41] B. Hou, S. Yang, F. A. Kuipers, L. Jiao, and X. Fu, "EAVS: Edge-assisted Adaptive Video Streaming with Fine-grained Serverless Pipelines," in *IEEE INFOCOM 2023*, pp. 1–10.

[42] J. Chi, X. Zhou, F. Xiao, Y. Lim, and T. Qiu, "Task Offloading via Prioritized Experience-Based Double Dueling DQN in Edge-Assisted IIoT," *IEEE Transactions on Mobile Computing*, vol. 23, no. 12, pp. 14 575–14 591, 2024.

[43] P. K. Mu, J. Zheng, T. H. Luan, L. Zhu, Z. Su, and M. Dong, "AMIS-MU: Edge Computing Based Adaptive Video Streaming for Multiple Mobile Users," *IEEE Transactions on Mobile Computing*, vol. 23, no. 1, pp. 117–134, 2024.


**Xuejian Chi** received the B.E. degree in Automation in 2022 and M.E. degree in Control Science and Engineering in 2024 both from China University of Petroleum. He is currently pursuing a Ph.D. degree in the School of Computer Science and Technology in Shandong University, China. His current research interests include edge computing and edge intelligence.

**Yifei Zou** (Member, IEEE) received the BEng degree from Wuhan University, China, in 2016, and the Ph.D. degree from The University of Hong Kong, China, in 2020. He is currently an associate professor at the School of Computer Science and Technology, Shandong University, Qingdao, China. His research interests include edge intelligence, Internet of agents, and distributed computing.



**Congwei Zhang** received the B.E. degree from the Computer Science Program of Taishan College, Shandong University, in 2021. He is currently pursuing the Ph.D. degree with the School of Computer Science and Technology, Shandong University. His research interests include machine learning for network, wireless networks, and distributed computing.

**Yapu Zhang** received the B.S. degree in mathematics and applied mathematics from Northwest University, Shaanxi, China, in 2016, and the Ph.D. degree in operational research from the University of Chinese Academy of Sciences, Beijing, in 2021. She is currently a lecturer with the Institute of Operations Research and Information Engineering, Beijing University of Technology, Beijing, China. Her research interests include social networks, combinatorial optimization, and algorithm design.

**Jiguo Yu** received the Ph.D. degree from the School of Mathematics, Shandong University, in 2004. He was a Full Professor with the School of Computer Science, Qufu Normal University, Shandong, China, in 2007. He is currently a full professor with University of Electronic Science and Technology of China, Chengdu, Sichuan, China, and a joint professor at Qilu University of Technology, Jinan, Shandong, China. His main research interests include privacy-aware computing, blockchain, intelligent IoT, AI and cyber security, distributed computing and graph theory. He is a Fellow of IEEE, a member of ACM and a senior member of the China Computer Federation (CCF).

**Xiuzhen Cheng** received her M.S. and Ph.D. degrees in computer science from the University of Minnesota Twin Cities in 2000 and 2002, respectively. She is a professor in the School of Computer Science and Technology, Shandong University, Qingdao, China. She has published more than 170 peer-reviewed papers. Her current research interests include cyber physical systems, wireless and mobile computing, sensor networking, wireless and mobile security, and algorithm design and analysis. She worked as a professor with the Department of Computer Science, the George Washington University, Washington, DC, USA, from 2013 to 2017. She worked as a program director for the US National Science Foundation (NSF) from April to October in 2006 (full time), and from April 2008 to May 2010 (part time). She received the NSF CAREER Award in 2004. She is Fellow of IEEE and a member of ACM.

**Falko Dressler** received his M.Sc. and Ph.D. degrees from the Dept. of Computer Science, University of Erlangen in 1998 and 2003, respectively. He is full professor and Chair for Telecommunication Networks at the School of Electrical Engineering and Computer Science, TU Berlin. Dr. Dressler has been associate editor-in-chief for IEEE Trans. on Network Science and Engineering, IEEE Trans. on Mobile Computing and Elsevier Computer Communications as well as an editor for journals such as IEEE/ACM Trans. on Networking, Elsevier Ad Hoc Networks, and Elsevier Nano Communication Networks. He has been chairing conferences such as IEEE INFOCOM, ACM MobiSys, ACM MobiHoc, IEEE VNC, IEEE GLOBECOM. He authored the textbooks Self-Organization in Sensor and Actor Networks published by Wiley & Sons and Vehicular Networking published by Cambridge University Press. He has been an IEEE Distinguished Lecturer as well as an ACM Distinguished Speaker. Dr. Dressler is an IEEE Fellow, an ACM Fellow, and an AAIA Fellow. He is a member of the German National Academy of Science and Engineering (acatech). He has been serving on the IEEE COMSOC Conference Council and the ACM SIGMOBILE Executive Committee. His research objectives include next generation wireless communication systems in combination with distributed machine learning and edge computing for improved resiliency. Application domains include the internet of things, cyber-physical systems, and the internet of bio-nano-things.

**Dongxiao Yu** (Senior Member, IEEE) received the B.S. degree in 2006 from the School of Mathematics, Shandong University and the Ph.D degree in 2014 from the Department of Computer Science, The University of Hong Kong. He became an associate professor in the School of Computer Science and Technology, Huazhong University of Science and Technology, in 2016. He is currently a professor in the School of Cryptologic Science and Engineering, Shandong University. His research interests include wireless networks, distributed computing and graph algorithms.