Enhancing QoE in Collaborative Edge Systems With Feedback Diffusion Generative Scheduling

Changfu Xu, *Graduate Student Member, IEEE*, Jianxiong Guo, *Member, IEEE*, Yuzhu Liang, *Student Member, IEEE*, Haodong Zou, *Student Member, IEEE*, Jiandian Zeng, *Member, IEEE*, Haipeng Dai, *Senior Member, IEEE*, Weijia Jia, *Fellow, IEEE*, Jiannong Cao, *Fellow, IEEE*, and Tian Wang, *Senior Member, IEEE*

***Abstract*—Collaborative edge computing is a promising approach for delivering low-delay services to computation-intensive Internet of Things applications. Deep Reinforcement Learning (DRL) has become an effective way to solve task scheduling decisions in edge systems due to its adaptive learning ability to interact with the environment. However, current DRL-based task scheduling methods still face several challenges, such as limited exploration, sample inefficiency, and performance instability, which can lead to degraded user Quality of Experience (QoE). To address these challenges, we observe that diffusion models, famous for their performance in image generation, exhibit strong exploration, data efficiency, and performance stability. This inspires us to propose FDEdge, a novel feedback diffusion generative scheduling method for enhancing user QoE in collaborative edge systems. We first design an innovative Feedback Diffusion (FDN) model by leveraging historical action probability information during the denoising process. We then incorporate the FDN model into DRL, forming an effective and efficient framework for task scheduling in collaborative edge systems. We also present a probability derivation to ensure the FDEdge's rationality. Extensive experimental results demonstrate that our FDEdge method significantly reduces service delays by 45.42% to 87.57% and speeds up training episode durations by $2.5\times$ times for a higher QoE than state-of-the-art methods.**

***Index Terms*—Edge computing, generative task scheduling, feedback diffusion, deep reinforcement learning.**

Received 6 May 2025; revised 20 June 2025; accepted 3 July 2025. Date of publication 10 July 2025; date of current version 5 November 2025. This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 62172046 and Grant 62372047, in part by the Beijing Natural Science Foundation under Grant 4232028, in part by the Natural Science Foundation of Guangdong Province under Grant 2024A1515011323, in part by Zhuhai Basic and Applied Basic Research Foundation under Grant 2220004002619, in part by the Joint Project of Production, Teaching and Research of Zhuhai under Grant 2220004002686 and Grant 2320004002812, in part by the Science and Technology Projects of Social Development in Zhuhai under Grant 2320004000213, in part by the Supplemental Funds for Major Scientific Research Projects of Beijing Normal University, Zhuhai under Grant ZHPT2023002, in part by the Fundamental Research Funds for the Central Universities, Higher Education Research Topics of Guangdong Association of Higher Education in the 14th Five-Year Plan under Grant 24GYB207, in part by the Beijing Normal University Education Reform Project under Grant jx2024139, in part by the Fundamental Research Funds for the Central Universities, Ministry of Education Industry-University Cooperation Collaborative Education Project under Grant 240904497110437, in part by the Key Laboratory of Equipment Data Security and Guarantee Technology, Ministry of Education, under Grant GDZB2024020100, in part by Interdisciplinary Intelligence Super Computer Center of Beijing Normal University at Zhuhai, and in part by Research Fund of Key Lab of Education Blockchain and Intelligent Technology, Ministry of Education under Grant EBME25-F-15. Recommended for acceptance by M. Amadeo. (*Corresponding Author: Tian Wang.*)

Changfu Xu is with the School of Software and IoT Engineering, Jiangxi University of Finance and Economics, Nanchang 330013, China, and also with the Institute of Artificial Intelligence and Future Networks, Beijing Normal University, Zhuhai 519087, China (e-mail: changfuxu91@outlook.com).

Jianxiong Guo and Weijia Jia are with the Institute of Artificial Intelligence and Future Networks, Beijing Normal University, Zhuhai 519087, China, and also with the Guangdong Key Lab of AI and Multi-Modal Data Processing, Beijing Normal-Hong Kong Baptist University, Zhuhai 519087, China (e-mail: jianxiongguo@bnu.edu.cn; jiawj@bnu.edu.cn).

Yuzhu Liang and Jiandian Zeng are with the Institute of Artificial Intelligence and Future Networks, Beijing Normal University, Zhuhai 519087, China, and also with the Key Lab of Education Blockchain and Intelligent Technology, Ministry of Education, Guangxi Normal University, Guilin 541004, China (e-mail: jiandian@bnu.edu.cn).

Haodong Zou is with the School of Computer Science and Technology, Anhui University, Heifei 230601, China (e-mail: zou_hd@163.com).

Haipeng Dai is with the Department of Computer Science and Technology, Nanjing University, Nanjing 210023, China (e-mail: haipengdai@nju.edu.cn).

Jiannong Cao is with the Department of Computing, Hong Kong Polytechnic University, Hong Kong (e-mail: csjcao@comp.polyu.edu.hk).

Tian Wang is with the Institute of Artificial Intelligence and Future Networks, Beijing Normal University, Zhuhai 519087, China, also with the School of Information Engineering, Huzhou University, Huzhou 313000, China, and also with the College of Computer and Data Scienc, Fuzhou University, Fuzhou 350108, China (e-mail: tianwang@bnu.edu.cn).

We release our open-source code at https://github.com/ChangfuXu/FDEdge.

Digital Object Identifier 10.1109/TMC.2025.3587744

I. Introduction

WITH the rapid proliferation of artificial intelligence in recent years, various modern Internet of Things (IoT) applications, such as Autonomous Driving (AD), Augmented Reality (AR), and Artificial Intelligence-Generated Content (AIGC), have emerged in our daily lives [1], [2], [3]. These applications are usually computationally intensive. For instance, smart vehicles (e.g., Tesla Model X) typically have 8 cameras and 12 ultrasonic radars for continuous large-scale data collection, requiring extensive computational resources [4], [5]. These considerations highlight the importance of providing low-delay service for computation-intensive applications in IoT devices.

Fortunately, edge computing was proposed as a viable solution to meet low-delay service requirements by processing tasks at Edge Servers (ESs) [6], [7]. More precisely, the edge computing technique allocates each arrival task to a suitable ES for processing, providing lower service response time than traditional cloud computing. Furthermore, considering the limitation of a single ES resource, collaborative edge computing has been proposed to further reduce service response time for computation-intensive IoT applications [8], [9].

Current solutions in collaborative edge computing are classified into two categories: traditional heuristic algorithm-based task scheduling methods (e.g., [10], [11]) and Deep Reinforcement Learning (DRL)-based task scheduling methods (e.g., [12], [13]). For example, Fan et al. [14] proposed a heuristic algorithm for service placement, task scheduling, and resource allocation in collaborative edge computing. Although current heuristic algorithms achieve low service delays in edge systems, they often rely on complex handcrafted designs based on rigid assumptions that fall short in addressing practical challenges. For instance, unlike cloud servers, ESs are significantly constrained in computing power and memory capacity. This limitation necessitates the collaborative use of multiple ESs to process computation-intensive tasks, which in turn poses a significant challenge for efficient task scheduling across resource-limited ESs. Additionally, the available computing resources at ESs are uncertain and fluctuate within the dynamic edge environment, complicating efforts to minimize task service delays without prior knowledge of resource availability.

Furthermore, DRL-based task scheduling methods are proposed to address the above challenges, as the DRL technique has several advantages [15], [16]. For example, the DRL methods learn optimal scheduling policies through interaction with the environment, making them better suited to dynamic and uncertain edge computing scenarios. Additionally, unlike heuristic algorithms that rely on fixed rules and assumptions, DRL can generalize across diverse tasks, workloads, and system states without manual redesign. Moreover, DRL optimizes scheduling decisions over the long term by considering cumulative rewards. However, existing DRL-based task scheduling methods, such as those proposed in [17] and [18], still encounter several challenges, often resulting in suboptimal decisions and inconsistent performance in complex edge environments. **First**, current DRL techniques heavily rely on historical experience, which limits their ability to effectively explore optimal solutions. **Second**, current DRL methods often need a large number of interactions with the environment to learn effective policies, making real-time adaptation challenging. **Third**, the model training in DRL techniques can be unstable and sensitive to its hyperparameters. Therefore, in order to overcome these limitations of DRL-based scheduling methods, there is an urgent need to find alternative paradigms with better exploratory, stable, and sampling efficiency.

Furthermore, we observe that the diffusion models [19], [20], which have emerged in generative models in recent years, have good characteristics to tackle these challenges. **First**, diffusion models generate diverse and high-quality action samples, improving exploration efficiency. **Second**, diffusion models make better use of limited training data by generating realistic trajectories or actions, improving sample efficiency. **Third**, the structured denoising process in diffusion models gradually refines the noise samples in a controlled manner, enabling stable policy learning through smoother and more robust updates compared to the abrupt changes typically observed in DRL training. **Fourth**, by modeling complex action distributions, diffusion models have more effective generalization capabilities across varying environments and task patterns. These characteristics inspire us to propose **FDEdge**, a novel feedback diffusion generative task scheduling method for enhancing users' Quality of Experience (QoE) in collaborative edge systems. We first propose a Feedback Diffusion Network (FDN) model based on traditional diffusion models. The FDN is designed by recording historical action probability information and feeding it back to the model input during the denoising process, improving scheduling decisions while reducing training episodes significantly. Furthermore, we use the proposed FDN as an actor and incorporate it into a DRL model, building an effective and efficient framework for task scheduling in collaborative edge systems. Finally, extensive experimental results show that our FDEdge method reduces service delays for a better user QoE than current methods. This paper makes the following contributions:

- We explore the task scheduling problem in collaborative edge systems. Furthermore, we formulate this problem as an online Integer Non-Linear Programming (INLP) problem to minimize the average service delay of task offloading in the entire system. We also prove the INLP problem's NP-hardness.
- We design an innovative FDN model by utilizing historical action probability distribution instead of the random Gaussian noise in the denoising process of diffusion models, improving scheduling decisions and accelerating network training.
- We propose a novel FDEdge method that implements an effective and efficient task scheduling in collaborative edge computing systems by incorporating our FDN model into DRL techniques. Importantly, we also provide a theoretical analysis of action derivation, ensuring the rationality of our FDEdge method.
- We evaluate the effectiveness of our FDEdge method through extensive experiments. The results show that the FDEdge significantly reduces the edge system's service delays by 45.42% to 87.57% for a better QoE and accelerates training episode durations by reaching 2.5× times compared to state-of-the-art methods.

The remainder of this paper is organized as follows. Section II introduces the related work. Section III gives the system model and problem formulation. Section IV describes the proposed method and algorithm with theoretical analysis. Section V evaluates the effectiveness of our method. The conclusions are shown in Section VI.

II. Related Work

In this section, we provide a brief overview of related work, including the areas of heuristic algorithm-based task scheduling, DRL-based task scheduling, and diffusion model-based task scheduling.

A. Heuristic Algorithm-Based Task Scheduling

Traditional heuristic algorithms, such as greedy-based [21], game theory-based [22], and Lyapunov-based [23] optimization, have demonstrated high reliability, which has contributed to their continued study in recent years. For example, Sahni et al. [24] propose a heuristic algorithm to minimize the average completion time of offloading tasks in edge systems. Qin et al. [25] present a distributed threshold-based heuristic algorithm for heterogeneous edge computing systems. However, these methods often rely on greedy optimization, tending to converge to long-term suboptimal solutions. Additionally, game theory approaches, such as those proposed by Jovsilo et al. [26], model task offloading as a multi-user computation offloading game and utilize a static mixed Nash equilibrium algorithm to solve it. Zhou et al. [27] apply Lyapunov optimization to design a two-timescale joint service placement and task offloading method in unmanned aerial vehicle-assisted edge systems, aiming to minimize task processing delay while reducing energy consumption. While these methods reduce service response time in edge systems, they may not be able to adjust or update timely in dynamic edge environments. Additionally, heuristic algorithms often rely on rigid assumptions with intricate handcrafted designs that fall short in addressing efficient task scheduling across resource-limited ESs in and minimizing task service delays without prior knowledge of resource availability.

B. DRL-Based Task Scheduling

The DRL technique [28], [29] has proven effective at capturing complex state-space representations. Therefore, compared to traditional heuristic task scheduling methods, DRL-based task scheduling methods can handle high-dimensional challenges and optimize decision-making in dynamic environments due to their interactive learning framework. For instance, Tang et al. [15] propose a DRL-based task offloading approach in edge computing systems. This approach better exploits the processing capacities of ESs, significantly reducing the failure ratio and service delay of task offloading compared to existing heuristic algorithms. Li et al. [30] investigate a DRL-based online task offloading and workload allocation strategy for collaborative edge computing, reducing computing delays in task offloading. These DRL-based task scheduling methods can efficiently adapt to unforeseen changes in task characteristics, making them highly suitable for task scheduling challenges in real-time edge systems. However, current DRL-based task scheduling methods still suffer from a hard trade-off between exploitation and exploration, a large number of interactions with the environment, and the sensitivity to hyperparameters, thus leading to suboptimal policies that diminish the users' QoE in edge systems.

C. Diffusion Model-Based Task Scheduling

Diffusion models have recently gained popularity in machine learning, particularly for applications such as image generation and video generation [31]. In computer vision, diffusion-based neural networks generate images from the inputs (e.g., text) by reversing the diffusion process, which typically involves introducing Gaussian noise and then gradually removing it to get a high-quality image [19], [32]. This process can be understood to learn a target distribution from Gaussian noise according to the input guidance, exhibiting the strong capabilities of exploration, generalization, and stability. Thus, few works explore the diffusion model for task scheduling in edge systems. For instance, Du et al. [33] present a Deep Diffusion-based Soft-Actor-Critic (D2SAC) algorithm for selecting AIGC service providers, aiming to maximize the human-aware content quality. Zhang et al. [34] design a diffusion-based DRL algorithm for cooperative offloading and resource allocation in edge-enabled metaverse systems. However, these algorithms suffer from performance instability and a large number of training episodes due to their reliance on Gaussian noise in the denoising process.

TABLE I
NOTATIONS FREQUENTLY USED IN OUR FORMULATION

Notation	Description
$\mathcal{T}$	The set of time slot
$\mathcal{B}$	The set of ESs or BSs
$\mathbb{N}_t$	The task set that arrives to BS $b \in \mathcal{B}$ at time slot $t \in \mathcal{T}$
Δ	The length of each time slot $t \in \mathcal{T}$
d_n	The data size of task $n \in \mathbb{N}_t$
ρ_n	The required computation density of offloading task n
$\boldsymbol{\zeta}$	The decision variable of task scheduling
$\boldsymbol{v}$	The transmission rate between BSs
f_b	The CPU or GPU computing capacity of ES $b \in \mathcal{B}$
$T^{\text{serv}}_{t,n,b}$	The service delay of offloading task n that is allocated from BS b to ES b at time slot t
$T^{\text{wait}}_{t,n,b}$	The waiting time of task n in the transmission queue at local BS b and the processing queue at ES b
$q_{t,b}$	The processing queue workload length of ES b at the end of time slot $t-1$
$q^{\text{bef}}_{t,n,b}$	The workload length of the processing queue at ES b before receiving task n at time slot t.

D. Comparative Analysis

Building on the above limitations in current methods, we propose a novel FDEdge method to improve task-scheduling decisions and performance stability in collaborative edge systems. Different from existing methods such as [8], [27], [33], our FDEdge method not only introduces an enhanced diffusion model, FDN, but also integrates it with DRL techniques, forming an effective and efficient task scheduling framework for collaborative edge computing. Moreover, our FDEdge method's reduction of instability and invariance is very suitable for solving the complex edge-edge collaboration problem in real applications. Importantly, our FDEdge method also has rationality by the decision probability derivation as proved in Theorem 2. As a result, our FDEdge method achieves near-optimal scheduling decisions, providing low service response time for enhancing users' QoE in collaborative edge systems.

III. System Model & Problem Formulation

In this section, we first present the system model of our method and then formulate our problem. Table I summarizes the notations frequently used in our formulation.

A. System Model

We focus on a collaborative edge system for processing computation-intensive modern applications, such as AD, AR, and AIGC, as shown in Fig. 1. This system comprises several Base Stations (BSs) with a centralized scheduling framework. Each BS is equipped with an ES of varying computing capacities.

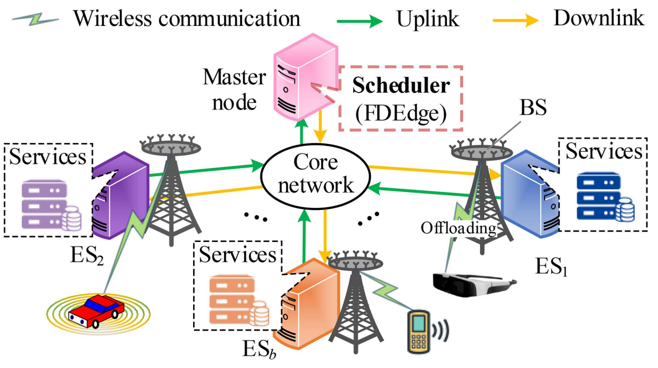


Fig. 1. Overview of our system model, consisting of a set of BSs. Each BS is equipped with an ES. The BSs receive tasks from end devices, such as smart vehicles and mobile phones. Each ES deploys several services. The master node has a global scheduler that deploys our FDEdge method, allocating all arrival tasks to multiple ESs for collaborative processing.

All BSs are connected through a wired core network. Each ES is deployed with several services to provide task processing. The master node deploys a global scheduler to schedule each arrival task using our FDEdge method. Let $\mathcal{B} = \{1, \ldots, B\}$ represent the set of ESs (or BSs), where $b \in \mathcal{B}$ corresponds to the b-th ES, and B denotes the total number of ESs. The system time is divided into consecutive time slots, represented as $\mathcal{T} = \{1, \ldots, |\mathcal{T}|\}$, where $t \in \mathcal{T}$ corresponds to the t-th time slot. Each time slot t has a fixed length of Δ seconds. We now describe the task model, decision variable, and delay model in more detail.

1) Task Model: Let $\mathbb{N}_t = \{1, 2, \ldots, N_t\}$ denote the set of tasks arriving at the master node at time slot t. Notably, this paper aims to propose an effective task scheduling method to reduce the processing of general computation-intensive tasks in collaborative edge systems, regardless of whether CPU or GPU workloads. Thus, we model the task workload using CPU resources in this paper, for example. Specifically, each task $n \in \mathbb{N}_t$ has the properties of data size d_n (in bits) and workload $\rho_n \cdot d_n$ (in CPU cycles), where ρ_n (in CPU cycles/bit) represents the required computation density of task n.

However, for computationally complex tasks handled on GPUs, such as Deep Neural Network (DNN) inference, more detailed modeling of the computational cost in terms of floating-point operations (FLOP) is required. In particular, the workload of DNN task n (e.g., ResNET152 [4]) is modeled by $\sum_{l=1}^{L} 2 \cdot J_{n,l}^{\text{height}} J_{n,l}^{\text{width}} C_{n,l}^{\text{in}} C_{n,l}^{\text{out}} M^{\text{height}} M^{\text{width}}$ (in FLOPs) instead of $\rho_n \cdot d_n$. Here, $J_{n,l}^{\text{height}}$ and $J_{n,l}^{\text{width}}$ represent the height and width of the output feature map, respectively. The $C_{n,l}^{\text{in}}$ and $C_{n,l}^{\text{out}}$ represent the input and output channels, respectively. The M^{height} and M^{width} represent the kernel height and width, respectively. Furthermore, the following formulations remain the same for CPU and GPU workloads.

2) Decision Variable: The master node's scheduler allocates each arrival task to an appropriate ES for processing at each time slot. Let the binary variable $\zeta_{t,n,b} \in \{0, 1\}$ indicate whether the task n arriving at the master node at time slot t is offloaded to an ES $b \in \mathcal{B}$ for processing. This characteristic has a feasible constraint, i.e.,

$$\sum_{b \in \mathcal{B}} \zeta_{t,n,b} = 1, \ \forall n \in \mathbb{N}_t, t \in \mathcal{T}. \tag{1}$$

3) Delay Model: Tasks are processed on a first-come-first-served basis in the processing queue at each ES. This means a task must be processed after all previously arrived tasks in the queue have been completed. The total service delay $T_{t,n,b}^{\text{serv}}$ for task n offloaded from BS b to ES b at time slot t includes several components and is formulated as:

$$T_{t,n,b}^{\text{serv}} = \zeta_{t,n,b} \cdot \left(\frac{d_n}{v_{t,n,b}} + \frac{\rho_n \cdot d_n}{f_b} + T_{t,n,b}^{\text{wait}} \right), \tag{2}$$

where f_b (in CPU cycles/s or FLOPs/s) is the CPU or GPU computing capacity of ES b. $v_{t,n,b}$ (in bits/s) are the transmission rates from the end device to BS b at time slot t. $T_{t,n,b}^{\text{wait}}$ is the waiting time of task n processed by B at time slot t. We consider that the transmission channel of tasks may have fading and interference since these tasks from multiple end devices communicate with the BS on the same channel [35], [36]. Then, according to the Shannon theory [37], the $v_{t,n,b}$ is calculated by

$$v_{t,n,b} = w_{t,n,b} \cdot \log_2 \left(1 + \frac{p_{t,n,b} \cdot h_{t,n,b}}{\delta_{t,n,b}^2} \right), \tag{3}$$

where $w_{t,n,b}$ represents the task n's transmission bandwidth on a wireless channel at time slot t. $p_{t,n,b}$ is the transmission power between task n and BS b at time slot t. $h_{t,n,b}$ is the uplink channel gain between task n and BS b at time slot t, which accounts for both large-scale and small-scale fading using a combination of log-normal and Rayleigh distributions, thereby modeling realistic dynamic wireless environments. The $\delta_{t,n,b}^2$ is the received Gaussian noise power from task n to BS b at time slot t. Furthermore, the $T_{t,n,b}^{\text{wait}}$ is achieved by

$$T_{t,n,b}^{\text{wait}} = \frac{q_{t-1,b} + q_{t,n,b}^{\text{bef}}}{f_b}, \tag{4}$$

where $q_{t,n,b}^{\text{bef}}$ is denoted as the workload length of the processing queue at ES b before receiving task n. Here, the $q_{t,n,b}^{\text{bef}}$ is achieved from system observation. The $q_{t-1,b}$ is denoted as the workload length of the ES b's processing queue at the end of time slot $t-1$. Furthermore, this $q_{t,b}$ is updated by

$$q_{t,b} = \max \left\{ q_{t-1,b} + \sum_{n=1}^{N_t} \zeta_{t,n,b} \cdot g_n - f_b \Delta, \ 0 \right\}, \tag{5}$$

where $\max\{\cdot\}$ presents the maximum function that equals the maximal value in the set. All the initial queue workloads of each ES at initial time slot 0 are set to 0, i.e., $q_{0,b} = 0$ and $q_{0,0,b}^{\text{bef}} = 0$ for $\forall b \in \mathcal{B}$. Here, it is assumed that there is no historical backlog of load for all ES instances when the system is started, and the initial value setting can be adjusted appropriately if there is an initial load.

B. Problem Formulation

Our problem aims to minimize the average service delay of all the tasks offloading in the entire system. Furthermore, our

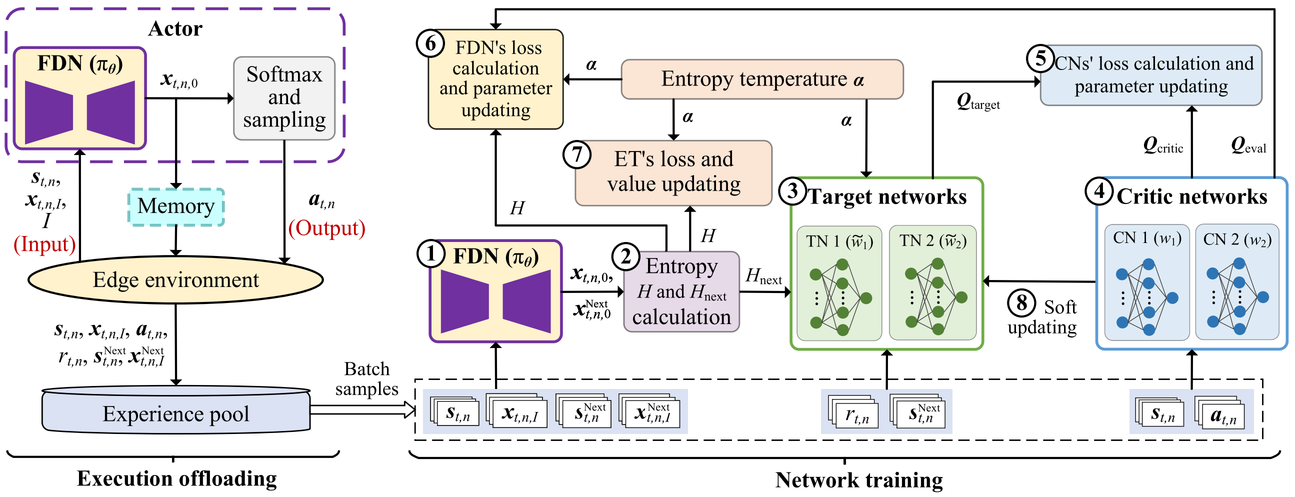

Actor
FDN (π_θ)
$x_{t,n,0}$
Softmax and sampling
$s_{t,n}$, $x_{t,n,I}$, I (Input)
Memory
$a_{t,n}$ (Output)
Edge environment
$s_{t,n}$, $x_{t,n,I}$, $a_{t,n}$, $r_{t,n}$, $s^{Next}_{t,n}$, $x^{Next}_{t,n,I}$
Experience pool
Batch samples
Execution offloading
⑥ FDN's loss calculation and parameter updating
Entropy temperature α
⑦ ET's loss and value updating
⑤ CNs' loss calculation and parameter updating
Q_{target}
Q_{critic}
Q_{eval}
③ Target networks
TN 1 ($\bar{w}_1$)
TN 2 ($\bar{w}_2$)
④ Critic networks
CN 1 (w_1)
CN 2 (w_2)
① FDN (π_θ)
② Entropy H and H_{next} calculation
$x_{t,n,0}$, $x^{Next}_{t,n,0}$
H_{next}
H
⑧ Soft updating
$s_{t,n}$ $x_{t,n,I}$ $s^{Next}_{t,n}$ $x^{Next}_{t,n,I}$
$r_{t,n}$ $s^{Next}_{t,n}$
$s_{t,n}$ $a_{t,n}$
Network training

Fig. 2. The overall architecture of our FDEdge method. First, the system state $s_{t,n}$, historical action probability $x_{t,n,I}$, and denoising step I are input into the actor. Second, the actor outputs the action $a_{t,n}$. Meanwhile, the $x_{t,n,I}$ is updated by equaling the current action probability $x_{t,n,0}$ in memory. Third, according to the $a_{t,n}$, the task n is offloaded to the selected ES for processing. Meanwhile, the reward $r_{t,n}$ is calculated. Fourth, the next system state $s^{\text{next}}_{t,n}$ and next historical action probability $x^{\text{next}}_{t,n,I}$ are observed from the environment and memory, respectively. Finally, the $s_{t,n}$, $x_{t,n,I}$, $a_{t,n}$, $r_{t,n}$, $s^{\text{next}}_{t,n}$, and $x^{\text{next}}_{t,n,I}$ are stored to experience pool. The flow of network training: ① The batch samples $s_{t,n}$, $x_{t,n,I}$, $s^{\text{next}}_{t,n}$, and $x^{\text{next}}_{t,n,I}$ are input into the FDN model to generate $x_{t,n,0}$ and $x^{\text{next}}_{t,n,0}$. ② The entropy H and next entropy H_{next} are calculated according to the $x_{t,n,0}$ and $x^{\text{next}}_{t,n,0}$, respectively. ③ The H_{next}, Entropy Temperature (ET) α, $r_{t,n}$, and $s^{\text{next}}_{t,n}$ are input into Target Networks (TNs) to achieve target Q-value Q_{target}. ④ The $s_{t,n}$ and $a_{t,n}$ are input into the Critic Networks (CNs) to achieve critic Q-value Q_{critic}. ⑤ The CNs' loss is calculated to update their model parameters according to the Q_{target} and Q_{critic}. Furthermore, the $s_{t,n}$ and $a_{t,n}$ are input into the CNs to achieve the evaluation Q-value Q_{eval}. ⑥ The FDN model's loss is calculated to update its model parameters according to the Q_{eval}, H, and α. ⑦ The ET's loss value is calculated to update its value according to the H and α. ⑧ The TNs' model parameters are updated by a soft operation based on the CNs' model parameters.

problem can be formulated as an online INLP optimization problem in the following:

$$\min_{\zeta} \lim_{|\mathcal{T}|\to\infty} \frac{1}{|\mathcal{T}|} \sum_{t\in\mathcal{T}} \sum_{n\in\mathbb{N}_t} \sum_{b\in\mathcal{B}} T^{\text{serv}}_{t,n,b}$$
$$\text{s.t.} \quad \text{Eqn. (1)},$$
$$\zeta_{t,n,b} \in \{0,1\},\ \forall n \in \mathbb{N}_t, t \in \mathcal{T}, b \in \mathcal{B}. \tag{6}$$

Here, the optimization variable of our problem in (6) is task scheduling decision $\zeta = \{\zeta_{t,n,b}\}_{n\in\mathbb{N}_t,t\in\mathcal{T},b\in\mathcal{B}}$. The offline counterpart of this problem is NP-hard as proved by the following Theorem 1.

Theorem 1: The offline counterpart of our problem in (6) is NP-hard.

Proof: The offline counterpart of our problem in (6) is proven to be NP-hard by reducing the multi-knapsack problem to it. Specifically, all tasks are treated as candidate objects in our problem in (6). The task workload is treated as the object weight. The computing capacity of ES corresponds to the weight capacity of the backpack. The task must be assigned exactly to one ES, and its service delay corresponds to the object value. Furthermore, the set $\mathcal{B}$ of ESs is treated as a collection of knapsacks. The tasks offloaded to the ESs to minimize service delay are then viewed as objects placed in multiple knapsacks to maximize object value, subject to the weight capacity constraint f. This setup reduces the multi-knapsack problem to our problem in (6). Since the multi-knapsack problem is known to be NP-hard [38], the offline counterpart of our problem in (6) is also NP-hard. □

IV. Proposed Method & Algorithm

In this section, we describe the details of our proposed method and algorithm. In Subsection IV-A, we present the overall architecture design of our method. Afterward, we provide the implementation and time complexity of the proposed algorithm in Section IV-B.

A. Method Architecture

Considering the several challenges as described in Section I and the NP-hard problem of our problem in (6) as shown in Theorem 1, we propose a novel FDEdge method to achieve better solutions for our problem in this subsection. In particular, we observe that current diffusion models (e.g., [19], [39]) have achieved significant success in the image generation field, which exhibits strong exploration, data efficiency, and performance stability. Inspired by this, we propose the FDEdge method for enhancing user QoE in collaborative edge systems. The FDEdge's overall architecture is given in Fig. 2.

Our FDEdge method's architecture incorporates the denoising diffusion process [39] and the basic framework of Soft-Actor-Critic (SAC) [29], consisting of two primary parts: execution offloading and network training. In the execution offloading part, we design an FDN model based on the traditional diffusion model in [39]. Furthermore, we use the FDN model as an actor and incorporate it into the SAC model. Here, we introduce a feedback strategy that feeds the historical action probability back to the actor's input (see Fig. 3). Moreover, the system state is incorporated into the actor's input using collaborative ES

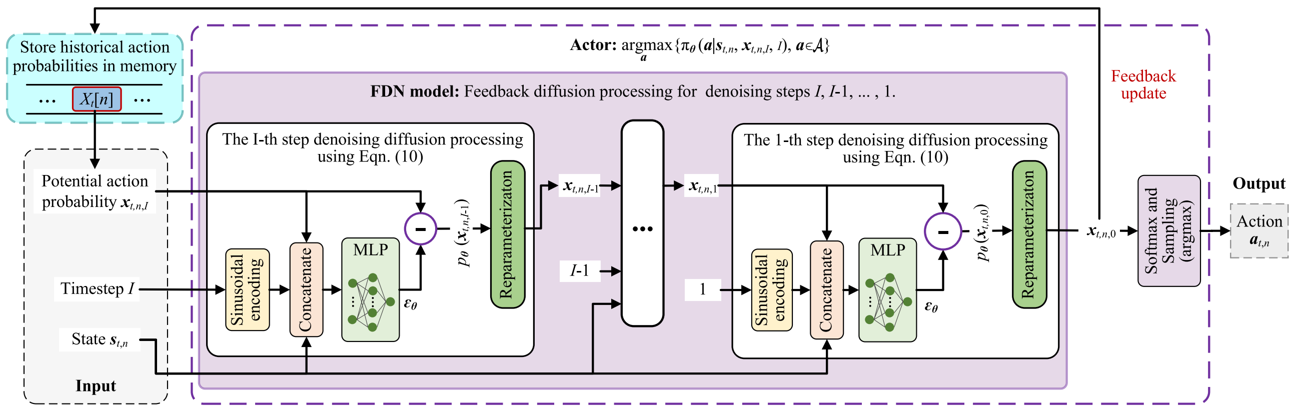

Store historical action probabilities in memory
$X_t[n]$
Actor: $\arg\max_{\boldsymbol{a}} \{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s}_{t,n}, \boldsymbol{x}_{t,n,I}, I), \boldsymbol{a}\in\mathcal{A}\}$
FDN model: Feedback diffusion processing for denoising steps I, I-1, ... , 1.
Feedback update
The I-th step denoising diffusion processing using Eqn. (10)
The 1-th step denoising diffusion processing using Eqn. (10)
Potential action probability $x_{t,n,I}$
Timestep I
State $s_{t,n}$
Input
Sinusoidal encoding
Concatenate
MLP
ε_θ
$p_\theta(x_{t,n,I-1})$
$p_\theta(x_{t,n,0})$
Reparameterizaton
$x_{t,n,I-1}$
I-1
$x_{t,n,1}$
1
$x_{t,n,0}$
Softmax and Sampling (argmax)
Output
Action $a_{t,n}$

Fig. 3. The actor structure with proposed FDN model. The actor input are the timestep I, historical action probability $\boldsymbol{x}_{t,n,I}$, and system state $\boldsymbol{s}_{t,n}$. The output is the action decision $\boldsymbol{a}_{t,n}$. The historical action probability $\boldsymbol{x}_{t,n,0}$ is stored (or updated) into the array $X_t[n]$.

information, such as task size and queue length. Then, the actor will generate task-scheduling decisions based on these inputs. Meanwhile, the historical data of task offloading is stored in an experienced pool. The historical action probability $\boldsymbol{x}_{t,n}$ is recorded and updated in memory. In the network training part, similar to the SAC model, we use two Critic Networks (CNs) and two Target Networks (TNs), which share the same network structure. Then, the FDN model is trained using historical batch samples extracted from the experience pool. The CNs and TNs generate the system's evaluation Q-values $\boldsymbol{Q}_{\text{eval}}$ and target Q-values $\boldsymbol{Q}_{\text{target}}$, respectively. As a result, our FDEdge method not only has a highly effective learning ability for better scheduling decisions but also can accelerate the convergence of network training. The state space, action space, reward function, actor structure, historical action probability feedback strategy, and network training process in our FDEdge method are explained in detail as follows:

1) State Space: The dynamic nature of edge computing systems makes it infeasible to model the available ES computing resources in the state space. Thus, the system state is defined by several factors that influence task scheduling decisions and service delays. Intuitively, the components of the arrival task size, task workload, and processing queue length in the system influence the offloading decision and its associated service delay. Let $\boldsymbol{s}_{t,n}$ denote the system state for task n arrived at the master node at time slot t. Then, $\boldsymbol{s}_{t,n}$ is formulated as

$$\boldsymbol{s}_{t,n} = [d_n,\ \rho_n \cdot d_n,\ \boldsymbol{q}_{t-1}], \tag{7}$$

where $\boldsymbol{q}_{t-1} = [q_{t-1,1}, q_{t-1,2}, \ldots, q_{t-1,B}]$ are obtained by the (5). Let $\boldsymbol{s}^{\text{next}}_{t,n}$ denote the next system state after offloading a task n. Then, $\boldsymbol{s}^{\text{next}}_{t,n}$ can be formulated as

$$\boldsymbol{s}^{\text{next}}_{t,n} = \begin{cases} \boldsymbol{s}_{t,n+1}, & 1 \le n < N_t. \\ \boldsymbol{s}_{t+1,1}, & n = N_t. \end{cases} \tag{8}$$

Here, when the arrival task n is not the last task in the current time slot t, the next task of task n is $n+1$. Otherwise, the next task of task n is the first task in the next time slot $t+1$.

2) Action Space: The action space is usually large since it will significantly increase with the increase of the number of tasks, users, and ES, causing our problem to be hardly solved by traditional heuristic solutions. We use a binary vector $\boldsymbol{a}_{t,n} = [\zeta_{t,n,1}, \zeta_{t,n,2}, \ldots, \zeta_{t,n,B}]$ to represent the action of a task n at time slot t. Using $\mathcal{A}$ represents the set of all action spaces. Then, we have $\mathcal{A} = \{[1,0,0,\ldots,0], [0,1,0,\ldots,0], \ldots, [0,0,0,\ldots,1]\}$. The action $\boldsymbol{a} \in \mathcal{A}$ is determined by the FDN model (denoted as $\pi_{\boldsymbol{\theta}}(\cdot)$) with the inputs of current system state $\boldsymbol{s}_{t,n}$, historical action probability $\boldsymbol{x}_{t,n,I}$, and timestep I. Thus, $\boldsymbol{a}_{t,n}$ is formulated as

$$\boldsymbol{a}_{t,n} = \arg\max_{\boldsymbol{a}} \{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s}_{t,n}, \boldsymbol{x}_{t,n,I}, I), \forall \boldsymbol{a} \in \mathcal{A}\}, \tag{9}$$

where $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s}_{t,n}, \boldsymbol{x}_{t,n,I}, I)$ is the probability of generating the action $\boldsymbol{a}$. $\boldsymbol{\theta}$ is the parameter of FDN model.

3) Reward Function: Obviously, the fewer service delays the more rewards. Thus, the negative task service delay is denoted as the reward. By executing offloading according to the action $\boldsymbol{a}_{t,n}$, we have the reward $r_{t,n}$ as follows:

$$r_{t,n} = -T^{\text{serv}}_{t,n,b}. \tag{10}$$

4) Actor Structure: The actor structure is a multi-step decision-making process that is designed by an FDN model, a softmax unit, and a sampling unit, as shown in Fig. 3. The FDN model performs the diffusion process using I denoising steps. Each denoising step has a sinusoidal encoding unit, a concatenation unit, a Multi-Layer Perceptron (MLP), and a reparameterization unit. The actor's execution process is described below:

- First, the I-th denoising step's input consists of $\boldsymbol{x}_{t,n,I}$, I, and $\boldsymbol{s}_{t,n}$. Here, the $\boldsymbol{x}_{t,n,I}$ represents the initial input, the I is the denoising step index, the $\boldsymbol{s}_{t,n}$ is the system state of the task, and the output of this step is denoted as $\boldsymbol{x}_{t,n,I-1}$.
- Second, the $(I-1)$-th denoising step's input is the I-th denoising step's output, along with $I-1$ and $\boldsymbol{s}_{t,n}$. This process continues through all I denoising steps.
- Third, the output of the final (i.e., $I = 0$) step is the action probability $\boldsymbol{x}_{t,n,0}$. Meanwhile, the $\boldsymbol{x}_{t,n,0}$ is stored or updated in the memory, waiting to be used in the next diffusion process.

- Finally, the action probability distribution $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s}_{t,n}, \boldsymbol{x}_{t,n,I}, I)$ is computed by applying the softmax operation to the output $\boldsymbol{x}_{t,n,0}$. The action $\boldsymbol{a}_{t,n}$ with the highest probability is then selected by sampling from this action probability distribution, as specified by (9). Here, the rationality of the action $\boldsymbol{x}_{t,n,0}$ is derived through the following Theorem 2. Intuitively, Theorem 2 can be viewed as gradually refining the predicted action probability distribution by removing noise in multiple steps, guided by historical decision tendencies and system state. This denoising process smooths the learning dynamics and stabilizes policy updates.

Theorem 2: The rationality of action $\boldsymbol{x}_{t,n,0}$ is derived by the update rule: $\boldsymbol{x}_{t,n,i-1} =$

$$\frac{1}{\sqrt{\lambda_i}}\left(\boldsymbol{x}_{t,n,i} - \frac{\beta_i}{\sqrt{1-\bar{\lambda}}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_{t,n,i}, i, \boldsymbol{s}_{t,n})\right) + \frac{\tilde{\beta}_i}{2}\cdot\boldsymbol{\epsilon}, \quad (11)$$

where $i = I, I-1, \ldots, 1$, $\beta_i = 1 - e^{-\frac{\beta_{\min}}{I} - \frac{2i-1}{2I^2}(\beta_{\max}-\beta_{\min})}$ represents the forward process variance [39], $\tilde{\beta}_i = \frac{1-\bar{\lambda}_{i-1}}{1-\bar{\lambda}_i}\cdot\beta_i$ is a deterministic variance parameter, $\lambda_i = 1-\beta_i$, $\bar{\lambda}_i = \prod_{m=1}^{i}\lambda_m$ is the cumulative production of λ_m over previous m $(m \le i)$ denoising steps, $\lambda_m = 1-\beta_m$, $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_{t,n,I}, i, \boldsymbol{s}_{t,n})$ is the MLP's output, and $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

Proof: First, based on the forward diffusion processing of diffusion models [39], we have

$$\boldsymbol{x}_{t,n,i} = \sqrt{\bar{\lambda}_i}\cdot\boldsymbol{x}_{t,n,0} + \sqrt{1-\bar{\lambda}_i}\cdot\boldsymbol{\epsilon}. \quad (12)$$

Second, similar to [33], by denoting $p(\boldsymbol{x}_{t,n,i-1}|\boldsymbol{s}_{t,n}, i, \boldsymbol{x}_{t,n,i})$ as the transition function from $\boldsymbol{x}_{t,n,i}$ to $\boldsymbol{x}_{t,n,i-1}$, the $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t,n,i-1}|\boldsymbol{s}_{t,n}, i, \boldsymbol{x}_{t,n,i})$ follows a Gaussian distribution presented by $\mathcal{N}(\boldsymbol{x}_{t,n,i-1}; \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}_{t,n,i}, i, \boldsymbol{s}_{t,n}), \tilde{\beta}_i\mathbf{I})$. Here, the $i \in \{1, 2, \ldots, I\}$, $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}_{t,n,i}, i, \boldsymbol{s}_{t,n})$ is the distribution mean which is learned by a deep model, $\tilde{\beta}_i = \frac{1-\bar{\lambda}_{i-1}}{1-\bar{\lambda}_i}\cdot\beta_i$ is a deterministic variance parameter, $\beta_i = 1 - e^{-\frac{\beta_{\min}}{I} - \frac{2i-1}{2I^2}(\beta_{\max}-\beta_{\min})}$ represents the forward process variance controlled by the variational posterior scheduler [39], $\bar{\lambda}_i = \prod_{m=1}^{i}\lambda_m$ is the cumulative production of λ_m over previous m $(m \le i)$ denoising steps, and the $\lambda_m = 1-\beta_m$.

Third, by applying the Bayesian formula and combining the reconstructed sample $\boldsymbol{x}_{t,n,0}$ in (12), we have $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}_{t,n,i}, i, \boldsymbol{s}_{t,n}) =$

$$\frac{1}{\sqrt{\lambda_i}}\left(\boldsymbol{x}_{t,n,i} - \frac{\beta_i}{\sqrt{1-\bar{\lambda}}}\cdot\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_{t,n,i}, i, \boldsymbol{s}_{t,n})\right). \quad (13)$$

Finally, by employing reparameterization technique for the $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t,n,i-1}|\boldsymbol{s}_{t,n}, i, \boldsymbol{x}_{t,n,i})$, we achieve the (11). □

5) Historical Action Probability Feedback Strategy: Current diffusion-based task scheduling methods typically perform the denoising diffusion process from random Gaussian noise, leading to instability in decision-making and degraded performance. To address this issue, we design a historical action probability feedback strategy. Specifically, as shown in Fig. 3, we introduce a historical action probability $\boldsymbol{x}_{t,n,I}$ instead of using random Gaussian noise in the input for each task $n \in \mathbb{N}_t$ at BS $b \in \mathcal{B}$ during time slot $t \in \mathcal{T}$. That is to say, the generative historical action probability of the actor is also fed back to the actor's input in the diffusion process. Here, the historical action probability feedback strategy acts as a soft constraint, narrowing the action distribution over time as the model learns by capturing the inherent periodic information in arrival tasks over a period.

For convenience, we define $N = \max\{N_t\}_{t\in\mathcal{T}}$ that is the maximum number of tasks arriving at the master node in a time slot t. Then, for each task $n \in \mathbb{N}_t$, we create an array X_t of length N to store all the tasks' historical action probabilities at time slot t. Each element $X_t[n]$ in X_t is initialized using a standard Gaussian distribution.

At each time slot t, for each task $n \in \mathbb{N}_t$, we use the corresponding $X_t[n]$ to initialize the historical action probability $\boldsymbol{x}_{t,n,I}$. After applying the diffusion process by I steps, we obtain the output $\boldsymbol{x}_{t,n,0}$. This output is then used to update the historical action probability, i.e., $X_t[n] \leftarrow \boldsymbol{x}_{t,n,0}$.

In addition, we incorporate the historical action probability $\boldsymbol{x}_{t,n,I}$ into the transition tuple to improve network training. Specifically, instead of using the standard tuple $(\boldsymbol{s}_{t,n}, \boldsymbol{a}_{t,n}, r_{t,n}, \boldsymbol{s}_{t,n}^{\text{next}})$, we use the augmented tuple $(\boldsymbol{s}_{t,n}, \boldsymbol{x}_{t,n,I}, \boldsymbol{a}_{t,n}, r_{t,n}, \boldsymbol{s}_{t,n}^{\text{next}}, \boldsymbol{x}_{t,n,I}^{\text{next}})$ in our method.

Furthermore, we initialize the historical action probability $\boldsymbol{x}_{t,n,I}$ with the previous action probability $\boldsymbol{x}_{b,n,t-1,0}$ from the diffusion process's results, enabling our method to leverage historical action probabilities effectively. This strategy allows the FDEdge method to quickly converge to near-optimal decisions, resulting in a significant reduction of service delay compared to current diffusion-based task scheduling methods.

6) Network Training Process: We incorporate the historical action probability $\boldsymbol{x}_{t,n}$ into the transition tuple for network training in our method, ensuring the effectiveness of the diffusion processing. Let $\boldsymbol{\omega}_1$, $\boldsymbol{\omega}_2$, $\tilde{\boldsymbol{\omega}}_1$, and $\tilde{\boldsymbol{\omega}}_2$ represent the parameters of the CNs 1 and 2, and the TNs 1 and 2, respectively. The FDN, CNs, and TNs models are trained using batch samples extracted from the experience pool in a centralized controller. The parameters $\boldsymbol{\theta}$, $\{\boldsymbol{\omega}_j\}_{j=1,2}$, and $\{\tilde{\boldsymbol{\omega}}_j\}_{j=1,2}$ are updated according to the following procedures:

Experience Sampling: First, K samples are drawn from the experience pool: $\{[\boldsymbol{s}_{t,n}^k, \boldsymbol{x}_{t,n}^k, \boldsymbol{a}_{t,n}^k, r_{t,n}^k, \boldsymbol{s}_{t,n}^{k,\text{next}}, \boldsymbol{x}_{t,n}^{k,\text{next}}]\}_{k=1,2,\ldots,K}$. The reward $r_{t,n}^k$, next state $\boldsymbol{s}_{t,n}^{k,\text{next}}$, and next historical action probability $\boldsymbol{x}_{t,n}^{k,\text{next}}$ are then inputted to the FDN and TNs for the target Q-values' calculation, as follows:

$$\begin{aligned} Q_{\text{target}}^k = \; & r_{t,n} + \gamma\cdot\Big[\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_{t,n}^{k,\text{next}}|\boldsymbol{s}_{t,n}^{k,\text{next}}, \boldsymbol{x}_{t,n}^{k,\text{next}}, I)\cdot Q_{\text{target}}^{k,\min} \\ & +\alpha_b\cdot H_{\text{next}}^k(\pi_{\boldsymbol{\theta}})\Big], \end{aligned} \quad (14)$$

where the $\boldsymbol{a}_{t,n}^{k,\text{next}} \sim \pi_{\boldsymbol{\theta}}(\cdot|\boldsymbol{s}_{t,n}^{k,\text{next}}, \boldsymbol{x}_{t,n}^{k,\text{next}}, I)$. The γ is the reward discount factor. The $Q_{t,n}^{k,\min}$ equals $\min_{j=1,2}\{Q_{\tilde{\boldsymbol{\omega}}_j}(\boldsymbol{s}_{t,n}^{k,\text{next}}, \boldsymbol{a}_{t,n}^{k,\text{next}})\}$. The $H_{\text{next}}^k(\pi_{\boldsymbol{\theta}})$ is the entropy of the action distribution at the next time step and equals $-\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_{t,n}^{k,\text{next}}|\boldsymbol{s}_{t,n}^{k,\text{next}}, \boldsymbol{x}_{t,n}^{k,\text{next}}, I)^{\top}\times\log\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_{t,n}^{k,\text{next}}|\boldsymbol{s}_{t,n}^{k,\text{next}}, \boldsymbol{x}_{t,n}^{k,\text{next}}, I)$.

CNs' Parameter Update: The CNs' model parameters $\{\boldsymbol{\omega}_j\}_{j=1,2}$ are updated by minimizing the following CNs' loss functions:

$$\mathcal{L}_Q(\boldsymbol{\omega}_j) = \frac{1}{K}\sum_{k=1}^{K}\left(Q_{\boldsymbol{\omega}_j}(\boldsymbol{s}_{t,n}^k, \boldsymbol{a}_{t,n}^k) - Q_{\text{target}}^k\right)^2. \qquad (15)$$

This ensures that the Q-values predicted by the CNs are consistent with the target Q-values.

FDN's Parameter Update: Furthermore, the FDN's model parameters $\boldsymbol{\theta}$ are updated by minimizing the FDN's loss function $\mathcal{L}_\pi(\boldsymbol{\theta}) =$

$$\frac{1}{K}\sum_{k=1}^{K}[-\alpha \cdot H^k(\pi_{\boldsymbol{\theta}}) - \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_{t,n}^k|\boldsymbol{s}_{t,n}^k, \boldsymbol{x}_{t,n}^k, I)\cdot Q_{\text{eval}}^k]^2, \quad (16)$$

where $Q_{\text{eval}}^k = \min_{j=1,2}\{Q_{\boldsymbol{\omega}_j}(\boldsymbol{s}_{t,n}^k, \boldsymbol{a}_{t,n}^k)\}$ and $H^k(\pi_{\boldsymbol{\theta}})$ is the entropy of the action distribution with the inputs of $\boldsymbol{s}_{t,n}$, $\boldsymbol{x}_{t,n}$, and I. We have $H^k(\pi_{\boldsymbol{\theta}}) = -\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_{t,n}^k|\boldsymbol{s}_{t,n}^k, \boldsymbol{x}_{t,n}^k, I)^\top \times \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_{t,n}^k|\boldsymbol{s}_{t,n}^k, \boldsymbol{x}_{t,n}^k, I)$.

Entropy Temperature Update: The temperature α is updated by minimizing the following entropy loss function:

$$\mathcal{L}(\alpha) = (-H^k(\pi_{\boldsymbol{\theta}}) - \tilde{H})\cdot\alpha, \qquad (17)$$

where $\tilde{H}$ is a hyperparameter representing the target entropy.

TNs' Parameter Update: Finally, the TNs' model parameters $\tilde{\boldsymbol{\omega}}_j$ are updated via the following soft updates:

$$\tilde{\boldsymbol{\omega}}_j = \tau\cdot\boldsymbol{\omega}_j + (1-\tau)\cdot\tilde{\boldsymbol{\omega}}_j, \qquad (18)$$

where τ is a coefficient for controlling the soft update rate.

B. Algorithm Implementation

We implement our FDEdge method as an online algorithm, as shown in Algorithm 1. In this algorithm, the inputs are the tasks data $\{d_n\}_{n\in\mathbb{N}_t,t\in\mathcal{T}}$. The output is the task scheduling decision $\boldsymbol{\zeta}$. The Algorithm 1 works as follows.

Step 1: All historical action probabilities are initialized by setting each element $X_t[n]$ in the array X_t for each $n\in\mathbb{N}_t$ and $t\in\mathcal{T}$ with a standard Gaussian distribution (Line 1). Then, the FDN, CNs, and TNs are initialized with random parameters $\boldsymbol{\theta}$, $\{\boldsymbol{\omega}_j\}_{j=1,2}$, and $\{\tilde{\boldsymbol{\omega}}_j\}_{j=1,2}$, respectively (Line 2). The queue $q_{0,b}$, the action step *count*, and the experience pool $\mathcal{R}$ are set to 0, 0, and empty ($\emptyset$), respectively (Line 3).

Step 2: For each episode $= 1, 2, \ldots, E$, the system environment is reset (Line 5). Then, for each time slot $t\in\mathcal{T}$ and each task $n\in\mathbb{N}_t$ (Lines 6 - 7), the system state $\boldsymbol{s}_{t,n}$ and historical action probability $\boldsymbol{x}_{t,n,I}$ are achieved from the system observation and the value $X_t[n]$, respectively (Line 8). Then, the $\boldsymbol{s}_{t,n}$ and $\boldsymbol{x}_{t,n,I}$, and I are used as input to generate the action $\boldsymbol{a}_{t,n}$ and $\boldsymbol{x}_{t,n,0}$ via the Actor (Line 9). Accordingly, the reward $r_{t,n}$ is calculated using (10) (Line 10). Furthermore, the historical action probability is updated by setting $X_t[n]\leftarrow\boldsymbol{x}_{t,n,0}$ (Line 11). The next system state $\boldsymbol{s}_{t,n}^{\text{next}}$ and historical action probability $\boldsymbol{x}_{t,n,I}^{\text{next}}$ are achieved from the system observation and the value $X_t[n]^{\text{next}}$, respectively (Line 12). Similar to the (8), here $X_t[n]^{\text{next}}$ equals to $X_t[n+1]$ or $X_{t+1}[1]$. The transition tuple $(\boldsymbol{s}_{t,n}, \boldsymbol{x}_{t,n,I}, \boldsymbol{a}_{t,n}, r_{t,n}, \boldsymbol{s}_{t,n}^{\text{next}}, \boldsymbol{x}_{t,n,I}^{\text{next}})$ is stored to the experience pool $\mathcal{R}$ (Line 13). Afterward, the variable *count*, which tracks the number of action steps, is incremented by 1 (Line 14).

Algorithm 1: Online FDEdge Algorithm.

```
   Input: The tasks $\{d_n\}_{n\in\mathbb{N}_t,t\in\mathcal{T}}$.
   Output: The task-scheduling decision $\boldsymbol{\zeta}$.
1  Initialize the $X$ by a standard Gaussian distribution;
2  Initialize the FDNs, CNs, and TNs with random $\boldsymbol{\theta}$,
     $\{\boldsymbol{\omega}_j\}_{j=1,2}$, and $\{\tilde{\boldsymbol{\omega}}_j\}_{j=1,2}$;
3  Initialize the queue $q_{0,b}=0$ for each $b\in\mathcal{B}$, the
     variable count $=0$, and the experience pool $\mathcal{R}=\emptyset$;
4  for episode $\in\{1,2,\cdots,E\}$ do
5      Reset system environment;
6      foreach time slot $t\in\mathcal{T}$ do
7          foreach new arrival task $n$ in master node do
8              Observe $\boldsymbol{s}_{t,n}$ and $\boldsymbol{x}_{t,n,I}\leftarrow X_t[n]$;
9              Generate $\boldsymbol{x}_{t,n,0}$ and $\boldsymbol{a}_{t,n}$ by the Actor;
10             Calculate $r_{t,n}$ using Eqn. (10);
11             Update $X_t[n]\leftarrow\boldsymbol{x}_{t,n,0}$;
12             Observe $\boldsymbol{s}_{t,n}^{\text{next}}$ and $\boldsymbol{x}_{t,n,I}^{\text{next}}\leftarrow X_t[n]^{\text{next}}$;
13             Store the tuple $(\boldsymbol{s}_{t,n}, \boldsymbol{x}_{t,n,I}, \boldsymbol{a}_{t,n}, r_{t,n},$
                 $\boldsymbol{s}_{t,n}^{\text{next}}, \boldsymbol{x}_{t,n,I}^{\text{next}})$ to $\mathcal{R}$;
14             count = count + 1;
15             if $|\mathcal{R}|>500$ and count % $K$ == 0 then
16                 Sample $K$ tuples from $\mathcal{R}$;
17                 Execute network training to update
                     $\{\boldsymbol{\omega}_j\}_{j=1,2}$, $\boldsymbol{\theta}$, $\alpha$, and $\{\tilde{\boldsymbol{\omega}}_j\}_{j=1,2}$ using
                     Eqns. (15) - (18);
18         Update $\boldsymbol{q}_t$ by the Eqn. (5);
```

Step 3: If the size $|\mathcal{R}|$ of experience pool exceeds 500 and is a multiple of batch size K (i.e., after every K executions), K samples from $\mathcal{R}$ are extracted to sequentially update the model parameters $\{\boldsymbol{\omega}_j\}_{j=1,2}$, $\boldsymbol{\theta}$, α, and $\{\tilde{\boldsymbol{\omega}}_j\}_{j=1,2}$ (Lines 15 - 17). The condition $|\mathcal{R}|>500$ ensures that at least 500 history tuples have been stored, allowing for model training based on a sufficient amount of data, as recommended in [29].

Step 4: At the end of each time slot, the queue $\boldsymbol{q}_t$ is updated according to (5) (Line 18).

Finally, the near-optimal task scheduling policy $\pi_{\boldsymbol{\theta}}$ is gradually learned through the above steps.

Theorem 3: For each time slot t, the time complexity of Algorithm 1 is approximately linear, i.e., $O(N_t)$.

Proof: As outlined in Algorithm 1, for each time slot $t\in\mathcal{T}$, the algorithm contains one "for" loop. In this loop, the time complexity is determined by the number N_t of tasks, the time required for ES selection, and the network training time. Since the ES selection is based on the pre-trained FDN model, its time complexity is $O(1)$. The networks are offline trained in a parallel way. Therefore, the overall time complexity of Algorithm 1 for each time slot t approximates $O(N_t)$. □

V. Performance Evaluation

In this section, we present the experimental setup, baseline comparisons, and results, accompanied by insightful analysis.

TABLE II
DEFAULT SETTING OF ENVIRONMENT PARAMETER

Param.	Value	Param.	Value
B	10	$\|\mathcal{T}\|$	100 [15]
N_t	[1, 100]	Δ	1.0 second [15]
ρ_n	[100, 300] cycles/bit [14]	d_n	[10, 40] Mbits [14]
$v_{t,n,b}$	[400, 500] Mbits/s [14]	f_b	[10, 50] GHz [14]

TABLE III
DEFAULT SETTING OF MODEL PARAMETERS

Parameters	Value
Hidden layers of actor, CN, and TN	2 full connection (128, 128)
Learning rate η_a	1e-4
Learning rate η_c	1e-3
Learning rate η_e	3e-4
Reward decay's factor γ	0.95
Soft updating's weight τ	0.005
Batch size K	64
Denoising step I	5
Entropy temperature α	0.05
Target entropy $\tilde{H}$	-1
Experience pool's size $\|\mathcal{R}\|$	10000
Optimizer	Adam
Number of episodes E	16

A. Experimental Setup

Environment Parameters: Following the similar setting in [8], [14], [15], [33], the default environmental parameters used in our experiments are summarized in Table II, unless otherwise specified. Specifically, according to [33], we train the proposed FDN model to schedule tasks to ESs for collaborative processing in a simulation environment with 10 ESs (i.e., $B = 10$). Each ES has a random computing capacity within the range [10, 50] GHz. Our simulations involve N_t tasks from 1 to 100 at each time slot. The data size d_n and computation density ρ_n of tasks are randomly ranged from 10 to 40 Mbits and 100 to 300 CPU cycles per bit, respectively [8]. Here, the task number, task size, and computation density usually meet the real requirements of most network systems. We have recorded 100 time intervals (that is, $|\mathcal{T}| = 100$ represented 100 seconds), and the duration Δ of each time interval is 1 s [14]. Following in [8], [15], the transmission rate $v_{t,n,b}$ between tasks and ESs is uniformly distributed within the range [400, 500] Mbits/s. Here, the parameters B, f_b, $|\mathcal{T}|$, $v_{t,n,b}$, ρ_n, and Δ remain fixed after initialization. The parameters N_t and d_n are randomly generated at each observation.

Model Parameters: Our experiments are implemented using the PyTorch framework. The default model parameters are listed in Table III. For convenience, we use two fully connected hidden layers, and each layer has 128 neurons, to build the FDN (actor) network along with the two CNs and two TNs. Following the settings in [33], the actor's learning rate η_a, the CNs' learning rate η_c, and the temperature α are set to 1e-4, 1e-3, and 3e-4, respectively. The reward decay factor γ, the soft update weight τ, and the batch size K are set to 0.95, 0.005, and 64, respectively. The denoising steps I and the entropy temperature α are set to 5 and 0.05, respectively. The target entropy $\tilde{H}$ and the size $|\mathcal{R}|$ of the experience pool are set to -1 and 10000, respectively. The Adam optimizer is used for model training. The number E of episodes is set to 16, corresponding to a simulation duration of 1600 seconds.

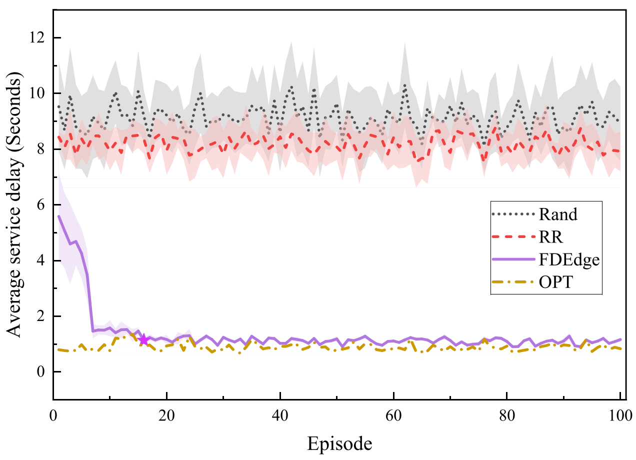
B. Baselines

To comprehensively evaluate the effectiveness of our FDEdge method, we compare it against five baseline methods, including two traditional heuristic approaches, three prominent DRL approaches, a diffusion-based DRL method, and an optimal heuristic method for task offloading. These baselines are described as follows:

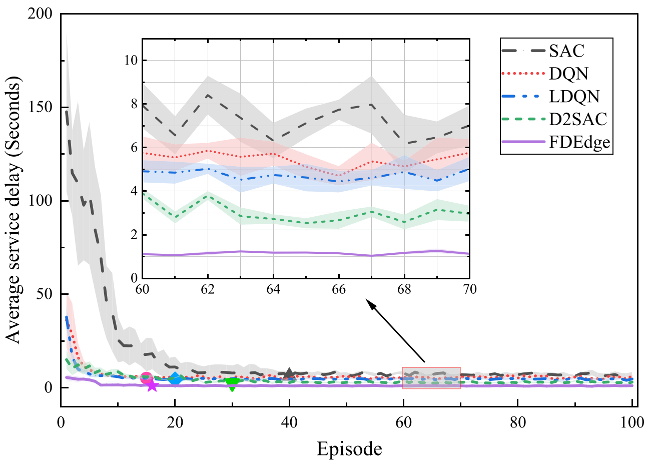
- *Rand:* The Rand method randomly selects an ES to process for each task offloading. This method is a classic offloading solution often used as a comparison in edge computing studies.
- *RR [21]:* The Round Robin (RR) method allocates tasks in cyclical order. This method generates favorable scheduling decisions when tasks are well-balanced, but it does not consider the significant differences among tasks.
- *DQN [28]:* The Deep Q-Network (DQN) is a widely recognized DRL method that has been successfully applied in various domains. In our experiments, we faithfully implement the DQN method for task scheduling as a baseline, ensuring that it uses the same setup and configuration as our proposed method.
- *SAC [29]:* The SAC is a state-of-the-art DRL algorithm known for its stability and efficiency in continuous action spaces. We implement SAC for task scheduling as another baseline, maintaining consistency in the experimental setup for a fair comparison.
- *LDQN [15]:* This LDQN method is based on the DQN model integrated with a Long Short-Term Memory (LSTM) network. This method achieves a state-of-the-art performance in DRL-based edge computing systems.
- *D2SAC [33]:* The D2SAC is an advanced method that integrates the diffusion model with the SAC algorithm for task scheduling in edge networks. As a diffusion-based DRL method, D2SAC is a competitive baseline in our comparison, showcasing the effectiveness of combining diffusion strategies with reinforcement learning.
- *OPT:* We assume that the scheduler knows the available computing and network resources of all ESs before assigning tasks at each time slot. The OPT method selects the best ES for each task at each time slot by exhaustively evaluating all possible action combinations, representing an upper bound on task scheduling performance in our experiments. However, it is impractical in real-world scenarios without the available computing and network resources for each ES before assigning tasks.

C. Results

This section presents extensive experimental results to assess the performance of our FDEdge method compared to the baselines, as shown in Figs. 4–9 and Tables IV and V. We conducted 50 independent experiments to achieve these results.



(a) Comparison of our method against heuristic baselines and the OPT.



(b) Comparison of our method against the baselines of DRL and diffusion.

Fig. 4. The learning process of our FDEdge method and baselines by varying the number E of episodes.

TABLE IV
LEADING PERFORMANCE COMPARISON OF OUR FDEDGE AND BASELINES

Method		Average service delay (Seconds)	Converged episode	Reduced delay	Improved episode
Upper bound	OPT	0.87±0.11	NA	-28.73%	NA
Heuristic	Rand	9.01±1.46	NA	87.57%	NA
	RR	8.22±0.66	NA	86.37%	NA
DRL	SAC	7.10±2.02	40	84.22%	**2.50×**
	DQN	5.58±0.95	15	79.93%	0.93×
	LDQN	4.63±0.57	20	75.81%	1.25×
Diffusion	D2SAC	3.05±0.88	30	**63.28%**	1.87×
	FDEdge	**1.12±0.14**	**16**	NA	NA

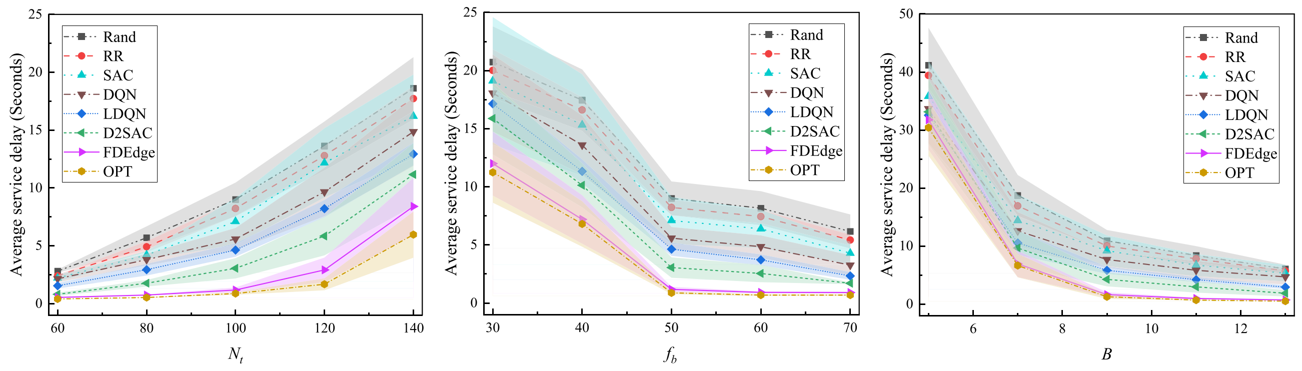
TABLE V
PERFORMANCE AND COMPARISON RESULTS OF THE SAC, SAC-F, FDEDGE-NF, AND FDEDGE METHODS

Method		Average service delay (Seconds)	Converged episode	Reduced delay	Improved episode
DRL	SAC	7.10±2.02	40	8.87%	25%
	SAC-F	6.47±1.71	30	NA	NA
Diffusion	FDEdge-NF	1.83±0.31	25	38.79%	36%
	FDEdge	1.12±0.14	16	NA	NA

1) Performance Comparison: To evaluate the performance of our FDEdge method, we first explore the training result as shown in Fig. 4 and then present a specific comparison of the FDEdge and the seven baselines in terms of average service delay and convergence episodes as shown in Table IV. Furthermore, we evaluate the impact of different environmental parameters on the average service delay of our FDEdge and the baselines, as shown in Fig. 5.

Learning Process: To analyze the learning process of our FDEdge method, we compare the average service delay of our FDEdge and the baselines by varying the number E of training episodes from 1 to 100, as shown in Fig. 4. In this figure, the x-axis represents the number of episodes, and the y-axis denotes the average service delay of all tasks across each episode. From Fig. 4, we observe that the average service delays of the SAC, DQN, LDQN, D2SAC, and FDEdge methods initially drop but gradually stabilize as the number of episodes increases. This behavior occurs because all the methods are based on the DRL technique, which gradually learns optimal scheduling policies. In contrast, the Rand, RR, and OPT methods' average service delays remain stable, as expected. However, the average service delay of our FDEdge is consistently lower than those of the Rand, RR, SAC, DQN, LDQN, and D2SAC methods, while close to the OPT method's delay. Moreover, the proposed FDEdge method converges much faster than the SAC and D2SAC methods. Importantly, the error distribution—represented by the shaded areas in the figure (i.e., standard deviation)—of our FDEdge method is significantly narrower than those of SAC, DQN, LDQN, and D2SAC throughout the entire learning process, demonstrating superior performance stability. These significant performance gaps between our FDEdge method and the baselines evaluate the importance of incorporating the proposed FDN model into DRL techniques for task scheduling.

Leading Performance: We summarize the best performance achieved by our FDEdge and seven baselines in Table IV, in terms of service delay and convergence episode. From Table IV, we can see that the average delays of the Rand, RR, SAC, DQN, LDQN, and D2SAC methods reach 9.01±1.46, 8.22±0.66, 7.10±2.02, 5.58±0.95, 4.63±0.57, and 3.05±0.88, respectively. In contrast, the average delay of our FDEdge method is only 1.12±0.14 seconds, outperforming **87.57%**, **86.37%**, **84.22%**, **79.93%**, **75.81%**, and **63.28%** compared to the Rand, RR, SAC, DQN, LDQN, and D2SAC methods, respectively. This is because our FDEdge method incorporates the proposed FDN model, which achieves better scheduling decisions, thus reducing service delays. On the other hand, our FDEdge converges at episode $E = 16$, whereas SAC, DQN, LDQN, and D2SAC converge at episodes $E = 40$, $E = 15$, $E = 20$, and $E = 30$, respectively. Thus, the FDEdge demonstrates **2.50**×, **1.25**×, and **1.87**× faster convergence than SAC,



(a) Varying the number of tasks. (b) Varying ES's computing capacity. (c) Varying the number of BSs.

Fig. 5. Delay comparisons of our FDEdge method and baselines by varying three environment parameters.

LDQN, and D2SAC, respectively. The rapid convergence of our FDEdge method can be mainly attributed to that the FDEdge leverages the historical action probability for accelerating the policy learning process. These results confirm that our FDEdge method not only outperforms the state-of-the-art approaches significantly in terms of average service delay but also achieves fast convergence.

Effect of the Number of Tasks: Fig. 5(a) shows the impact of varying the number of tasks N_t. As N_t increases from 60 to 140, the average service delays of all methods increase. Obviously, the large number of task arrivals results more workloads in the system, thus leading to longer service delays. However, FDEdge consistently maintains a significantly lower delay compared to the Rand, RR, SAC, DQN, LDQN, and D2SAC, with a slight increase relative to the OPT method. Specifically, the average service delays for Rand, RR, SAC, DQN, LDQN, and D2SAC increase from 2.81±0.14 to 18.61±2.69 seconds, 2.41±0.08 to 17.72±1.09 seconds, 2.21±0.27 to 16.19±3.61 seconds, 2.15±0.30 to 14.85±2.97 seconds, 1.54±0.24 to 12.93±0.98 seconds, and 0.81±0.13 to 11.16±2.73 seconds, respectively, as N_t increases. In contrast, the FDEdge's delay increases from 0.54 to 8.40 seconds, resulting in an improvement around **77.61%**, **75.57%**, **73.08%**, **69.48%**, **62.74%**, and **45.42%** over Rand, RR, SAC, DQN, LDQN, and D2SAC, respectively. These results highlight the effectiveness of our FDEdge method in handling increasing task loads while maintaining low service delays.

Effect of ESs' Computing Capacities: Fig. 5(b) investigates the effect of varying the ES's capacity f_b. From this figure, the average service delays of all methods decrease as f_b increases from 30 to 70, which is expected as the high ES capacities can process tasks more quickly. Our FDEdge consistently outperforms Rand, RR, SAC, DQN, LDQN, and D2SAC, with delays approaching the OPT's result. For instance, when $f_b = 70$, the average service delay of our FDEdge is **0.90±0.07** seconds which is significantly lower than those (i.e., 6.16±1.46, 5.42±0.46, 4.29±0.98, 3.26±0.86, 2.33±0.33, and 1.70±0.12 seconds) of the Rand, RR, SAC, DQN, LDQN, and D2SAC method, nearly identical to the OPT method's delay of 0.68±0.08 seconds. This demonstrates that our FDEdge can effectively leverage increased computational resources to minimize task service delays.

Effect of the Number of BSs: Fig. 5(c) presents the impact of varying the number B of BSs from 5 to 13. As the B increases, the average service delays for Rand, RR, SAC, DQN, LDQN, and D2SAC methods escalate significantly. However, our FDEdge method's service delay exhibits the lowest compared to these six methods while approaching the OPT's results. Moreover, the performance gap between FDEdge and these six methods is always significant as B increases. These results show our FDEdge's robustness in handling increasing numbers of BSs, making it highly scalable for large-scale edge computing systems.

To sum up, the results shown in Fig. 5 provide compelling evidence of our FDEdge's great QoE in reducing task service delays across different edge computing environments. Moreover, the FDEdge method always outperforms the state-of-the-art approaches by a significant margin as the number of tasks, the computing capacity of ES, and the number of BSs increase.

2) Ablation Experiment: To fully evaluate the proposed FDEdge's effectiveness, we perform a series of ablation experiments to explore the performance effects of the DRL and FDEdge methods with or without the proposed feedback diffusion strategy, as shown in Table V. We take the SAC method as an example of the DRL methods. We name SAC-F and FDEdge-NF to represent the SAC method using the proposed feedback diffusion strategy (i.e., the SAC method's system state $\boldsymbol{S}$ incorporates historical action probabilities) and the FDEdge method not using the proposed feedback diffusion strategy, respectively. From Table V, the average service delay and convergence episode of the SAC-F method are lower than those of the SAC method by 8.87% and 25%, respectively. The FDEdge method also outperforms the FDEdge-NF in both service delay and convergence episode by 38.79% and 36%, respectively. These superior achievements can be attributed to the proposed historical action probability feedback strategy. This strategy captures inherent periodic task information to generate better scheduling decisions in edge systems, thus improving service delay. Additionally, the results of the FDEdge-NF and FDEdge methods are significantly better than the results of the SAC and SAC-F methods, respectively, further showing the effectiveness of the proposed FDN model. This is because the FDN model enables better generalization across diverse task scenarios, leading to improved overall results.

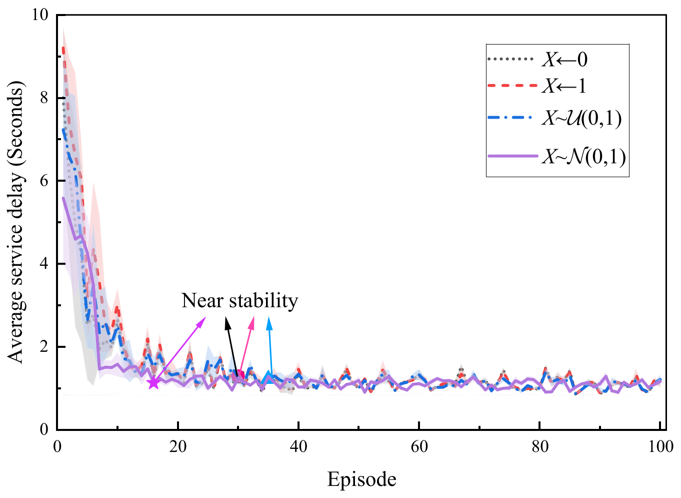


Fig. 6. The average service delay of our FDEdge method by varying initial action probability X.

3) Delay Effects of Varying Initial Action Probabilities: Since the historical action probabilities of consecutive time slots are correlated, we test the average service delay of our FDEdge method by varying the initial action probability as shown in Fig. 6. For generalization, we set the initial action probability with four scenarios: a standard Gaussian distribution $X \sim \mathcal{N}(0,1)$, a uniform distribution $X \sim \mathcal{U}(0,1)$, all values of 0, and all values of 1. According to Fig. 6, the average service delays of the four scenarios gradually converge into a stable value as the number of episodes increases. Meanwhile, the converged delays of the four scenarios are nearly equal and have different convergence episodes. Moreover, the scenario $X \sim \mathcal{N}(0,1)$ converges the fastest compared with the other three scenarios. These results suggest that the initial action probability setting slightly affects the final scheduling decisions of our FDEdge method.

4) Delay Effects of Varying Hyperparameters: To further validate the effectiveness of our FDEdge method, we investigate the effects of six key hyperparameters—actor's learning rate η_a, critic's learning rate η_c, entropy temperature's learning rate η_e, batch size K, reward decay factor γ, and soft updating weight τ—on the service delay, as presented in Fig. 7.

Actor's Learning Rate: From Fig. 7(a), we observe that as the actor's learning rate η_a decreases from 0.1 to 0.00001, the average service delay of our FDEdge method initially decreases and then starts to increase. The FDEdge's optimal performance, in terms of the lowest average service delay, is achieved with 1.1225 seconds when η_a equals 0.0001. Moreover, the delay distribution range is the smallest at this learning rate, indicating consistent performance.

Critic's Learning Rate: Fig. 7(b) is presented by varying the critic's learning rate η_c. Specifically, as the η_c decreases from 0.1 to 0.00001, our FDEdge achieves a significant reduction in average service delay followed by a slight increase. Herein, the lowest average service delay is observed when $\eta_c = 0.001$. Moreover, this result has the smallest distribution range, demonstrating stable performance at this setting.

Entropy Temperature's Learning Rate: In Fig. 7(c), changes in the entropy temperature's learning rate η_e from 0.1 to 0.00001 have a slight impact on the average service delay. Although the effect is limited, the optimal performance is achieved at $\eta_e = 0.0003$, yielding the lowest service delay among these tested values.

Batch Size: Fig. 7(d) reveals that increasing the batch size K from 16 to 256 initially reduces the average service delay significantly and then causes a slight decrease as K grows beyond 64. Notably, though increasing the K to a larger value (e.g., $K = 128$ or 256) leads to improved service delays, this comes at the cost of significantly longer training times, making these settings less practical for real-time applications. These results suggest that setting $K = 64$ can well balance the effectiveness and efficiency of our FDEdge method.

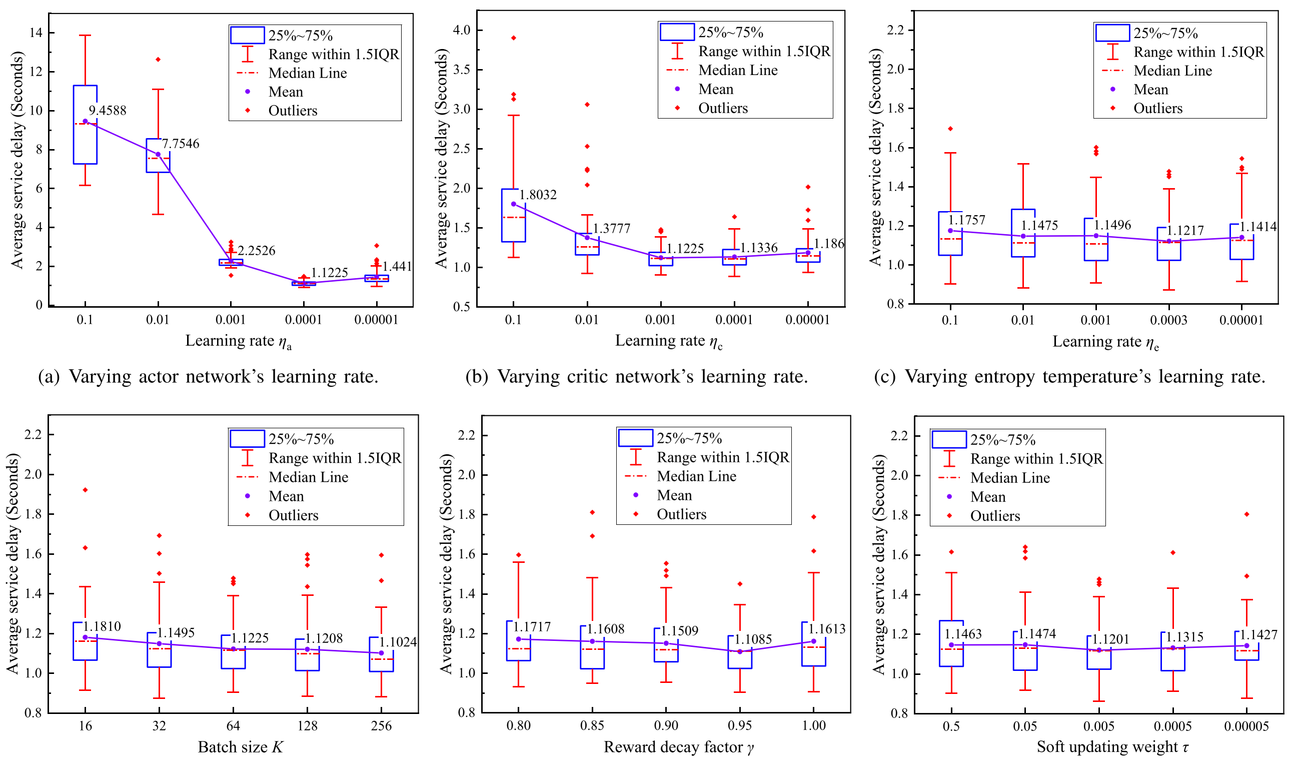
Reward Decay Factor: Fig. 7(e) demonstrate the average service delay of our FDEdge method by varying the reward decay factor γ. From this figure, the variations in γ from 0.80 to 1.0 have little impact on the service delay of our FDEdge method. The γ's optimal value is found to be 0.95 as this setting yields the best performance with minimal delays.

Soft Updating Weight: Fig. 7(f) shows the delay effect of varying soft updating weight τ. In particular, the average service delay of our FDEdge method maintains a small range [0.12, 0.15] as the τ varies from 0.5 to 0.00005. Additionally, when the τ equals 0.005, the FDEdge yields the lowest service delay. This result means that the optimal value of τ is 0.0005 in our FDEdge method.

By summarizing the results presented in Fig. 7(a) to (f), we conclude that the optimal hyperparameter settings for balancing performance and computational cost in our FDEdge method are as follows: an actor's learning rate of 0.0001, a critic's learning rate of 0.001, an entropy temperature's learning rate of 0.0003, a batch size of 64, a reward decay factor of 0.95, and a soft updating weight of 0.005.

5) Delay Effects of Varying Denoising Steps: The denoising step I plays a crucial role in ensuring the quality of image generation in the stable diffusion model [39]. However, increasing I can also lead to longer training and inference times. To explore this trade-off in our FDEdge method, we evaluate the FDEdge's average service delay by varying the denoising step I, as shown in Fig. 8. As depicted in Fig. 8, the average service delay initially decreases and then begins to stabilize with the increase of the denoising step I from 1 to 10. However, when the I is bigger than 5, the service delay is not significantly reduced and may achieve a higher service delay (e.g., $I = 6$ or 8). Meanwhile, the inference and training times of the FDN model will increase as the denoising step I increases [33]. Therefore, setting $I = 5$ strikes a favorable balance between performance and efficiency, making it the recommended configuration for practical applications. These findings indicate that increasing the denoising step helps reduce service delay, but beyond a certain point, the additional improvement in delay does not justify the increased computational cost.

6) Delay Effects of Varying Entropy Temperatures: In the DRL technique, the action entropy regularization term is introduced to balance exploration and exploitation [29], with its



(a) Varying actor network's learning rate. (b) Varying critic network's learning rate. (c) Varying entropy temperature's learning rate.

(d) Varying the size of training batch . (e) Varying the factor of reward decay. (f) Varying the weight of soft updating.

Fig. 7. The average service delay of our FDEdge method by varying six hyperparameters.

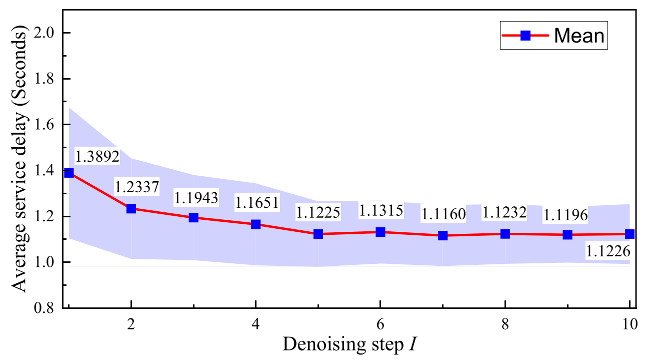


Fig. 8. The average service delay of our FDEdge method by varying the number of denoising steps.

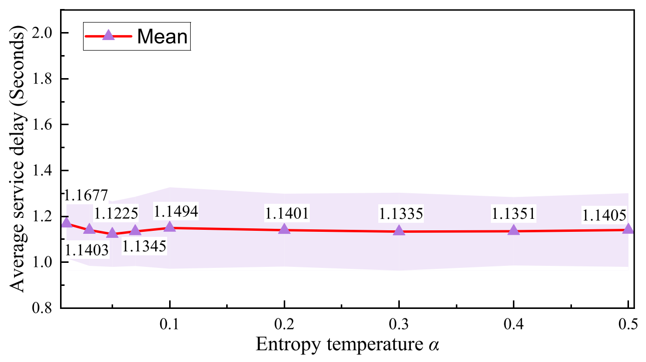


Fig. 9. The average service delay of our FDEdge method by varying the values of entropy temperature.

behavior controlled by the entropy temperature value α. Thus, to assess the entropy temperature's impact on the performance of our FDEdge method, we evaluate the average service delay by varying the α, as shown in Fig. 9. The Fig. 9's results reveal that as the α increases from 0.01 to 0.5, our FDEdge's average service delay first decreases and then begins to rise. Additionally, the optimal value for minimizing service delay is found to be $\alpha = 0.05$, which strikes a balance between exploration and exploitation. More precisely, lower values of α lead to overly greedy behavior, which restricts the agent from exploring actions with higher uncertainty, thus preventing the discovery of more effective actions. On the other hand, higher values of α encourage excessive randomness, potentially hindering the agent's ability to converge on an optimal policy efficiently. These findings suggest that a moderate value of $\alpha = 0.05$ facilitates faster convergence and better performance by promoting an effective exploration-exploitation trade-off. This balance is crucial for ensuring the efficiency and optimality of the task scheduling policy learned by our FDEdge method.

D. Discussion

This section discusses the rationale, scalability, and limitations behind our FDEdge method to provide insights for future directions in the field of edge computing.

1) Computation Overhead: Our experiments are conducted on a machine with an Intel i7 CPU, 32 GB RAM. Training for 16 episodes (convergence point) takes about two minutes in our simulations. Our FDN model requires approximately 1200 MB of CPU memory and < 1 ms per decision at inference. Notably, although we only perform experiments on CPU scenarios, our approach is easily scalable to GPU scenarios by changing the task model as presented in Section III-A.1.

2) Rationale: This paper presents a novel FDEdge method for improving user QoE in collaborative edge computing systems. The experimental results presented in Section V reveal that our FDEdge method significantly outperforms the state-of-the-art methods in average service delay and training episodes. The main reason for these improvements is that, unlike existing DRL methods (e.g. [8], [29], [33]), our FDEdge method not only integrates the diffusion model but also leverages the historical action probability instead of the random Gaussian noise in the denoising process of diffusion models, improving scheduling decisions and accelerating network training. Moreover, the rationality of our FDEdge method is ensured with a theoretical analysis of action derivation (see Theorem 2). As a result, our FDEdge method offers a powerful approach for task scheduling in collaborative edge systems, significantly achieving lower service delays for computation-intensive applications and fewer convergence episodes for network training compared to current state-of-the-art methods.

3) Scalability: Our FDEdge method exhibits scalability. First, the FDEdge could be adapted to other DRL techniques, such as DQN, without significantly changing the core principles. Second, our FDEdge method can also be extended to multi-agent scheduling in distributed systems. Specifically, we can deploy the FDEdge method into each ES for distributed multi-agent scheduling and train the FDN model at each ES using history samples. However, several new challenges should be addressed in distributed systems, such as device mobility, distributed node communication latency and consistency, and local information for global optimization. Third, by retraining our model to suit new scenarios, the FDEdge can be effectively extended for a wide range of modern applications, such as DNN inference, text-to-image, and text-to-video tasks.

4) Limitation: Pricing and energy consumption models are important issues in modern IoT applications [40], [41]. However, our FDEdge method does not consider the problem of service pricing and energy consumption in the system. Additionally, although our simulation experiments account for several real-world edge system factors—such as varying workloads, limited ES resources, and channel fading and interference—the performance gains of our method may be slightly reduced due to the inherent difficulty in fully capturing the complexity of practical edge environments.

VI. Conclusion

In this paper, we address the challenges of the task scheduling problem for enhancing QoE in collaborative edge systems. We first formulate the task offloading problem of collaborative edge computing systems as an online INLP problem with the objective of minimizing service delays. We then design an innovative FDN model and propose a novel FDEdge method for effective task scheduling in collaborative edge systems by incorporating the FDN model into DRL techniques. Furthermore, we implement the FDEdge algorithm to effectively and efficiently solve the INLP problem. Several theoretical analyses are presented to ensure the performance and rationality of our FDEdge method. Finally, many experimental results demonstrate that the proposed FDEdge method significantly reduces the offloading service delays by 45.42% to 87.57%, compared to the state-of-the-art methods. Also, the FDEdge method accelerates network training by $2.5\times$ times. These results highlight the efficiency and effectiveness of the proposed FDEdge method in optimizing task scheduling for edge computing systems.

Future work will focus on developing a more generalized feedback diffusion framework for various DRL architectures. Furthermore, we plan to develop a distributed collaborative edge system prototype using NVIDIA Jetson devices and deploy the proposed FDEdge method for real computation-intensive applications (e.g., text-to-image and image repairing) in the prototype system. In addition, considering pricing and energy optimization in edge systems is another worthwhile work in the future.

References

[1] Y. Siriwardhana, P. Porambage, M. Liyanage, and M. Ylianttila, "A survey on mobile augmented reality with 5G mobile edge computing: Architectures, applications, and technical aspects," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 2, pp. 1160–1192, Second Quarter, 2021.

[2] Y. Cao et al., "A comprehensive survey of AI-generated content (AIGC): A history of generative AI from GAN to ChatGPT," pp. 1–44, 2023, *arXiv:2303.04226*.

[3] Y. Zhao et al., "Joint content caching, service placement, and task offloading in UAV-enabled mobile edge computing networks," *IEEE J. Sel. Areas Commun.*, vol. 43, no. 1, pp. 51–63, Jan. 2025.

[4] 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.

[5] K. Liu, C. Liu, G. Yan, V. C. Lee, and J. Cao, "Accelerating DNN inference with reliability guarantee in vehicular edge computing," *IEEE/ACM Trans. Netw.*, vol. 31, no. 6, pp. 3238–3253, Dec. 2023.

[6] Y. Liang et al., "Latency reduction in immersive systems through request scheduling with digital twin networks in collaborative edge computing," *ACM Trans. Sensor Netw.*, 2024, doi: 10.1145/3701562.

[7] 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 *Proc. IEEE Conf. Comput. Commun.*, 2025, pp. 1–10.

[8] C. Xu, J. Guo, Y. Li, H. Zou, W. Jia, and T. Wang, "Dynamic parallel multi-server selection and allocation in collaborative edge computing," *IEEE Trans. Mobile Comput.*, vol. 23, no. 11, pp. 10523–10537, Nov. 2024.

[9] Y. Liang et al., "Collaborative edge server placement for maximizing QoS with distributed data cleaning," *IEEE Trans. Serv. Comput.*, vol. 18, no. 3, pp. 1321–1335, May/Jun. 2025.

[10] H. Cai, Z. Zhou, and Q. Huang, "Online resource allocation for edge intelligence with colocated model retraining and inference," in *Proc. IEEE Conf. Comput. Commun.*, 2024, pp. 1900–1909.

[11] S. Sthapit, J. Thompson, N. M. Robertson, and J. R. Hopgood, "Computational load balancing on the edge in absence of cloud and fog," *IEEE Trans. Mobile Comput.*, vol. 18, no. 7, pp. 1499–1512, Jul. 2018.

[12] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 26–38, Nov. 2017.

[13] C. Ling, K. Peng, S. Wang, X. Xu, and V. C. M. Leung, "A multi-agent DRL-based computation offloading and resource allocation method with attention mechanism in MEC-enabled IIoT," *IEEE Trans. Serv. Comput.*, vol. 17, no. 6, pp. 3037–3051, Nov./Dec. 2024.

[14] W. Fan et al., “Collaborative service placement, task scheduling, and resource allocation for task offloading with edge-cloud cooperation,” *IEEE Trans. Mobile Comput.*, vol. 23, no. 1, pp. 238–256, Jan. 2024.
[15] M. Tang and V. W. Wong, “Deep reinforcement learning for task offloading in mobile edge computing systems,” *IEEE Trans. Mobile Comput.*, vol. 21, no. 6, pp. 1985–1997, Jun. 2022.
[16] C. Xu et al., “Enhancing AI-generated content efficiency through adaptive multi-edge collaboration,” in *Proc. 44th IEEE Int. Conf. Distrib. Comput. Syst.*, 2024, pp. 1–11.
[17] J. Tan, R. Khalili, H. Karl, and A. Hecker, “Multi-agent distributed reinforcement learning for making decentralized offloading decisions,” in *Proc. IEEE Conf. Comput. Commun.*, 2022, pp. 2098–2107.
[18] P. Dai, K. Hu, X. Wu, H. Xing, and Z. Yu, “Asynchronous deep reinforcement learning for data-driven task offloading in MEC-empowered vehicular networks,” in *Proc. IEEE Conf. Comput. Commun.*, 2021, pp. 1–10.
[19] P. Esser et al., “Scaling rectified flow transformers for high-resolution image synthesis,” in *Proc. 41st Int. Conf. Mach. Learn.*, 2024, pp. 12606–12633.
[20] S. D. Platform. 2024. [Online]. Available: https://stability.ai/news/stable-diffusion-3
[21] R. Beraldi and G. P. Mattia, “Power of random choices made efficient for fog computing,” *IEEE Trans. Cloud Comput.*, vol. 10, no. 2, pp. 1130–1141, Second Quarter, 2022.
[22] L. Chen, S. Zhou, and J. Xu, “Computation peer offloading for energy-constrained mobile edge computing in small-cell networks,” *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1619–1632, Aug. 2018.
[23] C. Xu, J. Guo, J. Zeng, Y. Li, J. Cao, and T. Wang, “Incorporating startup delay into collaborative edge computing for superior task efficiency,” in *Proc. IEEE/ACM 32nd Int. Symp. Qual. Service*, 2024, pp. 1–10.
[24] Y. Sahni, J. Cao, L. Yang, and Y. Ji, “Multi-hop multi-task partial computation offloading in collaborative edge computing,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 5, pp. 1133–1145, May, 2021.
[25] X. Qin, Q. Xie, and B. Li, “Distributed threshold-based offloading for heterogeneous mobile edge computing,” in *Proc. IEEE 43rd Int. Conf. Distrib. Comput. Syst.*, 2023, pp. 202–213.
[26] S. Jošilo and G. Dán, “Decentralized algorithm for randomized task allocation in fog computing systems,” *IEEE/ACM Trans. Netw.*, vol. 27, no. 1, pp. 85–97, Feb. 2019.
[27] R. Zhou, X. Wu, H. Tan, and R. Zhang, “Two time-scale joint service caching and task offloading for UAV-assisted mobile edge computing,” in *Proc. IEEE Conf. Comput. Commun.*, 2022, pp. 1189–1198.
[28] V. Mnih et al., “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[29] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 1861–1870.
[30] M. Li, J. Gao, L. Zhao, and X. Shen, “Deep reinforcement learning for collaborative edge computing in vehicular networks,” *IEEE Trans. Cogn. Commun. Netw.*, vol. 6, no. 4, pp. 1122–1135, Dec. 2020.
[31] F.-A. Croitoru, V. Hondru, R. T. Ionescu, and M. Shah, “Diffusion models in vision: A survey,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 9, pp. 10850–10869, Sep. 2023.
[32] M. Li et al., “Distrifusion: Distributed parallel inference for high-resolution diffusion models,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2024, pp. 7183–7193.
[33] H. Du et al., “Diffusion-based reinforcement learning for edge-enabled AI-generated content services,” *IEEE Trans. Mobile Comput.*, vol. 23, no. 9, pp. 8902–8918, Sep. 2024.
[34] Z. Zhang, J. Wang, J. Chen, H. Fu, Z. Tong, and C. Jiang, “Diffusion-based reinforcement learning for cooperative offloading and resource allocation in multi-UAV assisted edge-enabled metaverse,” *IEEE Trans. Veh. Technol.*, early access, Feb. 25, 2025, doi: 10.1109/TVT.2025.3544879.
[35] Y. Mao, J. Zhang, S. Song, and K. B. Letaief, “Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems,” *IEEE Trans. Wireless Commun.*, vol. 16, no. 9, pp. 5994–6009, Sep. 2017.
[36] Y. Chen, J. Zhao, Y. Wu, J. Huang, and X. Shen, “QoE-aware decentralized task offloading and resource allocation for end-edge-cloud systems: A game-theoretical approach,” *IEEE Trans. Mobile Comput.*, vol. 23, no. 1, pp. 769–784, Jan. 2024.
[37] C. E. Shannon, “A mathematical theory of communication,” *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 379–423, 1948.
[38] L. Gu, D. Zeng, J. Hu, B. Li, and H. Jin, “Layer aware microservice placement and request scheduling at the edge,” in *Proc. IEEE Conf. Comput. Commun.*, 2021, pp. 1–9.
[39] J. Ho, A. Jain, and P. Abbeel, “Denoising diffusion probabilistic models,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 6840–6851.
[40] Z. Chen, Y. Yang, J. Xu, Y. Chen, and J. Huang, “Task offloading and resource pricing based on game theory in UAV-assisted edge computing,” *IEEE Trans. Serv. Comput.*, vol. 18, no. 1, pp. 440–452, Jan./Feb. 2025.
[41] H. Zou, J. Guo, J. Zeng, Y. Li, J. Cao, and T. Wang, “Fine-grained service lifetime optimization for energy-constrained edge-edge collaboration,” in *Proc. IEEE 44th Int. Conf. Distrib. Comput. Syst.*, 2024, pp. 565–576.

Changfu Xu (Graduate Student Member, IEEE) received the BS degree in communication engineering and the MS degree in software engineering from the Jiangxi University of Finance and Economics, in 2015 and 2018, respectively, and the PhD degree in computer science and technology from Hong Kong Baptist University, Hong Kong, China, in 2025. Currently, he is an assistant professor with the School of Software and IoT Engineering, Jiangxi University of Finance and Economics, Nanchang, China. His main research interests include edge computing and AIGC. He has published more than ten papers. He won the Best Paper Runner-up Award of IEEE/ACM IWQoS 2024.

Jianxiong Guo (Member, IEEE) received the BE degree from the School of Chemistry and Chemical Engineering, South China University of Technology, Guangzhou, China, in 2015, and the PhD degree from the Department of Computer Science, University of Texas at Dallas, Richardson, TX, USA, in 2021. He is currently an associate professor with the Advanced Institute of Natural Sciences, Beijing Normal University, Zhuhai, China, and also with the Guangdong Key Lab of AI and Multi-Modal Data Processing, Beijing Normal-Hong Kong Baptist University, Zhuhai, China. He is a member of ACM/CCF. He has published more than 100 peer-reviewed papers and has been a reviewer for many famous international journals/conferences. His research interests include social networks, wireless sensor networks, combinatorial optimization, and machine learning.

Yuzhu Liang (Student Member, IEEE) received the BS and MS degrees from Huaqiao University, Xiamen, China, in 2017 and 2020, respectively, and the PhD degree from Beijing Normal University, Beijing, China, in 2025. Currently, he is a postdoctoral fellow with the Beijing Normal University, Zhuhai, China. His research interests include edge computing, mobile computing, and artificial intelligence.

Haodong Zou (Student Member, IEEE) received the BS degree in software engineering and the MS degree in computer science and technology from Anhui University, China, in 2018 and 2021, respectively, and the PhD degree in computer science and technology from Hong Kong Baptist University, Hong Kong, China, in 2025. Currently, he is a postdoctoral fellow at the Anhui University, Hefei, China. His main research interests include 5G mobile communication, edge computing, and edge intelligence.

Jiandian Zeng (Member, IEEE) received the BSc degree in computer science and technology from Jianghan University, Wuhan, China, in 2014, the MSc degree in computer technology from Huaqiao University, Xiamen, China, in 2018, and the PhD degree in computer and information science from the University of Macau, Macau, China, in 2023. Currently, he is a lecturer with the School of Computer Science and Technology, Beijing Normal University, Zhuhai, China. His research interests include natural language processing and mobile computing.

Haipeng Dai (Senior Member, IEEE) received the BS degree from the Department of Electronic Engineering, Shanghai Jiao Tong University, Shanghai, China, in 2010, and the PhD degree from the Department of Computer Science and Technology, Nanjing University, Nanjing, China, in 2014. His research interests are mainly in the areas of Internet of Things, data mining, and mobile computing.

Weijia Jia (Fellow, IEEE) received the BSc and MSc degrees in computer science from Center South University, China, in 1982 and 1984, respectively and the master of applied sci and PhD degrees in computer science from the Polytechnic Faculty of Mons, Belgium, in 1992 and 1993, respectively. He is currently a chair professor, director with the Institute of AI and Future Networks, Beijing Normal University, and VP for research with Beijing Normal-Hong Kong Baptist University. His contributions have been recognized as optimal network routing and deployment; anycast and QoS routing, sensors networking, AI (knowledge relation extractions; NLP, etc.), and edge computing. He has more than 600 publications in prestigious international journals/conferences, and research books.

Jiannong Cao (Fellow, IEEE) received the BSc degree in computer science from Nanjing University, China, in 1982, and the MSc and PhD degrees in computer science from Washington State University, USA, in 1986 and 1990, respectively. He is currently the Otto Poon Charitable Foundation professor in data science and the chair professor of distributed and mobile computing with the Department of Computing, Hong Kong Polytechnic University, Hong Kong. He is also the director of the Internet and Mobile Computing Lab and the associate director with the University Research Facility in Big Data analytics. His research interests include parallel and distributed computing, wireless networks and mobile computing, Big Data and cloud computing, pervasive computing, and fault-tolerant computing. He has coauthored five books in mobile computing and wireless sensor networks, co-edited nine books, and published more than 600 papers in major international journals and conference proceedings. He is a distinguished member of ACM and a senior member of the China Computer Federation (CCF).

Tian Wang (Senior Member, IEEE) received the BSc and MSc degrees in computer science from Central South University, in 2004 and 2007, respectively, and the PhD degree in computer science from the City University of Hong Kong, in 2011. Currently, he is a professor with the Institute of Artificial Intelligence and Future Networks, Beijing Normal University. His research interests include Internet of Things, edge computing, and mobile computing. He has 30 patents and has published more than 300 papers in high-level journals and conferences. He was a co-recipient of the Best Paper Runner-up Award of IEEE/ACM IWQoS 2024. He has more than 16000 citations, according to Google Scholar. His H-index is 74.