Dynamic Neural Network-Based Resource Management for Mobile Edge Computing in 6G Networks

Longfei Ma, *Student Member, IEEE,* Nan Cheng, *Senior Member, IEEE,* Conghao Zhou, *Member, IEEE,* Xiucheng Wang, *Student Member, IEEE,* Ning Lu, *Member, IEEE,* Ning Zhang, *Senior Member, IEEE,* Khalid Aldubaikhy, *Member, IEEE,* Abdullah Alqasir, *Member, IEEE*

***Abstract*—Mobile edge computing (MEC) can be used to reduce the task delay for users with limited computing resources. However, in 6G networks, the diversity of tasks is greatly increased. For those extremely delay-sensitive small-size computing tasks, the inference delay of neural network (NN)-based algorithms such as resource allocation and task offloading cannot be ignored. As a hyperparameter, the inference cost of NN is usually difficult to adjust. Dynamic neural network (DyNN) is an emerging technique that improves the model efficiency by adjusting the network architecture on-demand according to the sample characteristics during inference. In this paper, we propose a DyNN-based resource management method for MEC that dynamically adjusts the depth and width of the NN according to the features of the task, improving computational efficiency and achieving a balance between inference delay and the management performance of computational and communication resources. Furthermore, to reduce the training cost of DyNN, a new training method is proposed in this paper, where all the blocks in DyNN are gradually trained in the order of size. Simulation results demonstrate that the proposed DyNN-based resource management method outperforms the traditional optimization algorithm and the static-NN-based method.**

***Index Terms*—Mobile edge computing, inference delay, dynamic neural network, training cost.**

I. Introduction

In recent years, mobile edge computing (MEC) has attracted soaring attention to relieve the backhaul burden by moving the caching and processing of the task from cloud to the network edge [2]. This means that MEC significantly reduces the mobile computing task delay, which is attractive for meeting the stringent delay requirements of high urgency applications, and thus is extensively used in augmented reality (AR), Internet-of-Things (IoT), connected vehicles and so on [3]–[7]. However, with the development of 6G, the increasingly complexity and dynamics pose challenge on MEC resource allocation and task scheduling. The optimization problem of MEC system can be seen as a generalization of the classical bin packing problem and is therefore an NP-hard problem [8], [9]. For this kind of problem, although the traditional optimization algorithm can give a satisfactory solution, it is very time-consuming.

As outlined by the International Telecommunication Union (ITU) [10], 6G networks need to offer unparalleled application diversity, catering to a wide range of user demands. To effectively address the varied requirements of different applications, 6G networks must incorporate highly flexible technologies and provide customizable solutions. In this regard, AI emerges as a critical component, surpassing the capabilities of traditional approaches and enabling the seamless integration of intelligence into future networks. As a representative technique of AI, deep learning (DL) is considered as a potential solution, since it can extract the hidden information contained in massive data and learn the complex interrelationships that are difficult to be discovered. In particular, neural network (NN) models are widely used to perform various computing tasks in mobile networks and greatly improve the quality of service [11], [12].

There are various resource allocation problems in MEC, such as bandwidth, CPU frequency, offloading decisions, and so on. The NN models can extract optimization problems from a large number of application scenarios, and then iterate using appropriate gradient descent methods to improve the decision performance. To improve the network utility of vehicular edge computing networks, a deep reinforcement learning-based computational offloading and resource allocation method is proposed [13]. Xu *et al.* propose an efficient reinforcement learning-based resource management algorithm for renewable energy scenarios to instantly learn the optimal policy for workload offloading and server configuration to minimize the long-term system cost [14].

Despite the success of DL in wireless networks, there has been limited research on inference delay, which refers to the

This work was supported by the National Key Research and Development Program of China (2020YFB1807700), the National Natural Science Foundation of China (NSFC) under Grant No. 62071356, the Fundamental Research Funds for the Central Universities, and the Innovation Fund of Xidian University (YJSJ23012). This article was presented in part at the IEEE ICC 2022 [1]. *(Corresponding Author: Nan Cheng.)*

Longfei Ma, Nan cheng, and Xiucheng Wang are with the State Key Laboratory of ISN and School of Telecommunications Engineering, Xidian University, Xi'an 710071, China (e-mail: lfma@stu.xidian.edu.cn; dr.nan.cheng@ieee.org; xcwang_1@stu.xidian.edu.cn).

Conghao Zhou is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, N2L 3G1, Canada (e-mail: c89zhou@uwaterloo.ca).

Ning Lu is with the Department of Electrical and Computer Engineering, Queen's University, Kingston, ON K7L 3N6, Canada (e-mail: ning.lu@queensu.ca).

Ning Zhang is with the Department of Electrical and Computer Engineering, University of Windsor, Windsor, ON, N9B 3P4, Canada (e-mail: ning.zhang@uwindsor.ca).

Khalid Aldubaikhy and Abdullah Alqasir are with the Department of Electrical Engineering, Qassim University, Buraydah 52571, Saudi Arabia (e-mail: khalid@qec.edu.sa; a.alqasirg@qec.edu.sa).

delay incurred by NNs during inference, particularly for delay-sensitive tasks. A typical MEC task's service delay consists of inference delay and processing delay. Inference delay is the time for executing task scheduling or resource orchestration algorithms, while processing delay is the time for task data communication and execution. Processing delay relies on task scheduling, resource allocation, and algorithm performance, such as DL model accuracy. In contrast, inference delay is influenced by algorithm computational complexity and device computing capability. Although conventional wireless resource allocation problems often neglect inference delay, it becomes crucial in ultra-low latency B5G/6G MEC services. Microsecond-level service delay is demanded by various 6G applications [15], highlighting the significance of minimizing inference delay. Services with very small data and workload per decision slot emphasize reducing inference delay, while those with substantial data and workload prioritize processing delay, categorizing them as inference delay-sensitive and processing delay-sensitive services, respectively. Recognizing the ITU's assertion regarding of 6G network applications [10], it is evident that 6G networks must possess highly flexible technologies and offer tailored solutions to effectively accommodate diverse application scenarios. For an MEC server, it has to continuously adjust the resource management algorithms since both two categories of services can be requested by the users it serves. However, it is very challenging since inference delay and processing delay are often a tradeoff. Generally, the NN model inference performance (for MEC resource management, the NN model performance determines the processing delay) is proportional to the size of the NN model. On the other hand, the inference delay is also proportional to the size of the NN model, since a larger model requires a larger amount of computing. Worse still, due to the non-linearity of the NN models, usually it is difficult to derive direct mathematical relation between inference performance and inference time.

In recent years, a new NN structure called dynamic neural network (DyNN) has attracted much attention, which can dynamically change the structure or parameters of the NN to achieve specific operations during the inference process, thereby significantly improving the computational efficiency [16]. Moreover, the dynamic nature of DyNN can be exploited to achieve a trade-off between model performance and inference delay. Specifically, when the task size is small, the inference delay is comparable to the transmission and computation delay, and then DyNN can reduce the inference delay by dynamically scaling down the depth and width of the NN. Conversely, when the task size is large, the inference delay has less impact on the service delay, and DyNN can obtain a high-performance resource allocation model for inference by using a deeper and wider NN architecture in that case. DyNN has great potential to be used in wireless network management, especially when tasks have various features and some of them are extremely delay-sensitive [1]. Although DyNN has many advantages, the dynamic nature of DyNN requires that the different sub-NNs in it should all have acceptable inference performance. Therefore, all these sub-NNs should be fully trained instead of only requiring end-to-end optimization for a single model as in traditional static NNs, making the training of DyNNs more challenging. Thus, the training method should also be optimized to accelerate the training speed and reduce the costs of implementing the DyNN.

In this paper, the DyNN is used to manage the resource in MEC for supporting tasks with different sizes and delay requirements. In addition, we propose a new DyNN training method in order to improve training efficiency. Specifically, we first train the neural network with the biggest size and then gradually train the smaller sub-NNs in the DyNN until all blocks of DyNN are trained. Thus, the training cost can be reduced greatly than the traditional training method. The main contribution of this paper is as follows.

1) In delay-sensitive MEC networks, we formulate the resource allocation and offloading problem considering inference delay. To solve the problem, we designed an efficient dynamic neural network whose architecture can be dynamically changed according to the task characteristics, effectively balancing inference delay and model performance.
2) In order to reduce the deployment cost of the DyNN model, we also propose a new DyNN training method, which first trains the largest architecture of DyNN, and then gradually trains the smaller sub-NNs in DyNN until all blocks of DyNN are trained. Compared with traditional training methods, this method can shorten the training time significantly.
3) We design a new loss function to train DyNN based on the optimization objective of MEC networks, thereby eliminating the need for labels and achieving unsupervised learning, which further enhances the practicality of the algorithm.
4) Through simulation, we demonstrate the performance benefits of the proposed algorithm across various aspects. Compared with the benchmark methods, the proposed algorithm greatly reduces the service delay under different network conditions, thus improving service quality.

The remainder of the paper is organized as follows. In Section II, related work is presented. The system model and problem formulation are discussed in Section III. We propose a DyNN-based resource management algorithm to solve the problem in Section IV followed by the extensive simulation results presented in Section V. Finally, Section VI concludes this paper.

II. Related Work

By deploying computing resources at the network edge, MEC enhances the quality of service for mobile users. In this section, we extensively explore the use of DL in MEC systems.

Reducing processing delay and energy consumption to improve the quality of service is the main optimization goal for MEC systems. To minimize the normalized energy consumption, a DL architecture is proposed in [17] that uses a digital twin of the real network to train DL algorithms offline on a central server. Considering the high variability of delay and the trust risk in cooperation, an online learning-aided cooperative offloading mechanism is proposed to accommodate delay variations, where social trust is used to organize the computation offloading [18]. In addition, considering the distributed deployment requirements in practical situations, a decentralized computation offloading algorithm is proposed to minimize the average delay of a pervasive edge computing network based on game theory and adversarial imitation learning [19]. All the above work shows that DL can significantly improve the performance of MEC networks. However, the deployment and implementation overheads of these DL schemes are not considered.

Among various DL categories, deep reinforcement learning is widely used in MEC because it can effectively cope with environmental changes and does not require labels during training. For the multi-user multi-edge node computation offloading problem, a model-free reinforcement learning offloading mechanism is proposed based on the unknown payoff game framework [20]. For the resource allocation problem of collaborative MEC networks, an intelligent resource allocation algorithm based on multi-task reinforcement learning is proposed in [21] to learn the network environment in a self-supervised learning manner. Similarly, although deep reinforcement learning can cope with complex dynamic network environments, the execution cost of the algorithms remains a pressing issue.

The work in [1] presented preliminary results, focusing on the trade-off between inference delay and transmission delay. To address the operational process of the task more effectively, a new DyNN is proposed to optimize computing resources while considering the overheads from inference, transmission, and operations. It is worth noting that there has been work applying DyNN to wireless communications, but it is aimed at MIMO detection, not delay-sensitive systems such as MEC [22].

III. System Model and Problem Formulation

A. Task Processing Delay

Consider an MEC system with N users and one server as shown in Fig. 1. We assume that at the beginning of each delay slot. User i generates a task, denoted by o_i, with the size of s_i and different urgency. Since all tasks are assumed as extremely delay-sensitive, all tasks should be processed before the deadline d_i. In addition, user i is equipped with a central processing unit (CPU) with a different computing frequency f_i^{loc}, while all users' CPU frequency is much lower than that of the server's f^{ser}. The task can be processed locally or in the server by offloading. Inspired by the success of the 5G network of orthogonal frequency division multiplexing (OFDM), we assume that the OFDM is used for task uploading and downloading and the total bandwidth resource for all users in the system is limited with the size of $b_{\max}$, each user can be allocated with different bandwidth on the demand of the size and urgency of the task. Similar to [23], the transmission rate for user i is mainly determined by the allocated bandwidth and the distance between users and server, which are various for different users. As a consequence, if user i determines to offload the task to the server, the transmission delay $T_{\mathrm{tra},i}$ is

$$T_i^{\mathrm{tra}} = \frac{s_i}{b_i \log_2\left(1+\frac{g_0 (l_0/l_i)^h p}{b_i \sigma_0}\right)}, \tag{1}$$

where b_i is the bandwidth allocated to user i, g_0 is the path loss constant, h is the path loss exponent, l_0 is the reference distance, l_i is the distance between user i and the server, p and σ_0 are the power of transmitting and noise. In order to

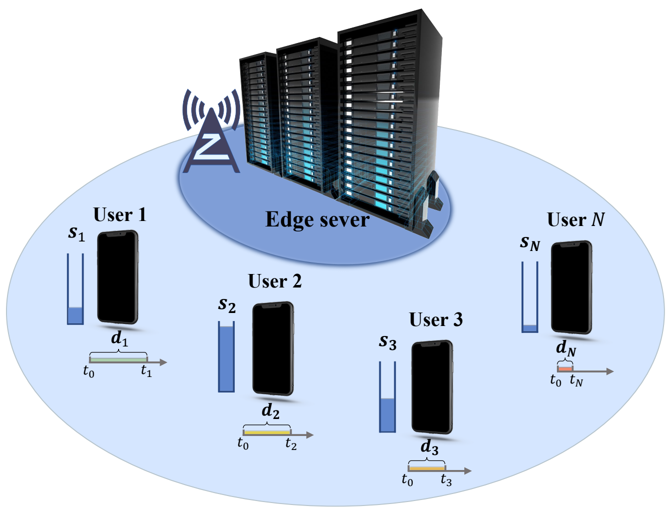


Fig. 1. MEC system with delay-sensitive tasks.

enable the server can process multiple tasks from different users simultaneously, the computing resource, that is, the CPU frequency of the server is assumed to be allocatable. Therefore, if user i offloads the task to the server, and f_i^{ser} computing resource is allocated to it, the server processing delay is

$$T_i^{\mathrm{com}} = \frac{c_i s_i}{f_i^{\mathrm{ser}}}, \tag{2}$$

where c_i (in CPU cycles/bit) is the workload determined by the task complexity. Assuming when part of the computing resource is allocated to a specific user it is reserved, thus the queuing delay is ignored in this paper, and the total delay of offloading is

$$T_i^{\mathrm{edge}} = T_i^{\mathrm{tra}} + T_i^{\mathrm{com}}. \tag{3}$$

However, since the computing resources of the server are limited, offloading all tasks to the server may defect the computing efficiency of the server and increase the probability of missing the deadline. As a result, users can also process tasks locally when there are few computing resources in the server or the distance between users and the server is too long that the transmission delay is large. Since the transmission delay is zero when processing locally, the delay of local computing is

$$T_i^{\mathrm{loc}} = \frac{c_i s_i}{f_i^{\mathrm{loc}}}. \tag{4}$$

B. Energy Consumption Model

Besides the task processing delay, energy consumption is an important factor, since the batteries of IoT devices are energy-limited. According to [23], the energy consumed by the edge server is ignored since it can always be charged by wire. Therefore, if users offload tasks to the server, the energy consumption of transmission is

$$E_i^{\text{edge}} = pT_i^{\text{tra}}. \tag{5}$$

Otherwise, the task is processed locally, and the energy consumption is [24]

$$E_i^{\text{loc}} = \mu c_i s_i (f_i^{\text{loc}})^2, \tag{6}$$

where μ is a constant value of energy efficiency. Considering the information of task, device, and location together, we define the feature vector u_i of user i as $\{s_i, c_i, d_i, l_i, f_i^{\text{loc}}\}$, thus the set of user features can be expressed as

$$\mathcal{U} = \{u_1, u_2, \ldots, u_N\}. \tag{7}$$

C. Inference Delay Model

The algorithm inference delay, which is often overlooked, is a critical factor in delay-sensitive scenarios. It is well known that convex optimization relies on multiple iterations to get the solution to the problem. In each iteration, there are a large number of addition and multiplication and differentiation calculations. As a result, the inference delay of convex optimization is huge and cannot be simply ignored in extremely delay-sensitive scenarios. Moreover, it is challenging to adjust the inference delay of convex optimization on demand, since its computational complexity depends on the algorithm architecture. The inference delay of the DL-based method is determined by the depth and width of the NN architecture. For example, a multilayer perceptron (MLP) requires multiple matrix multiplications, the number of which is the depth of NN and the size of the matrix is equal to the width of NN [25]. Therefore, the inference delay of the DL-based method can be adjusted by changing the depth and width of NN. In this paper, we focus on the inference delay of the DL-based method. To accurately represent the amount of operations, we use Multiply-Accumulate Operations (MACs) to measure the number of model operations [26], and the computational complexity of the i-th layer in NN is given by

$$\mathcal{M}_i = \zeta \mathcal{D}_{\text{in}} \mathcal{D}_{\text{out}} + \iota, \tag{8}$$

where $\mathcal{D}_{\text{in}}$ and $\mathcal{D}_{\text{out}}$ denote the input and output dimensions, respectively, and the coefficients ζ and bias ι are determined by the structure of the NN. Therefore, the inference delay of an MLP can be calculated as

$$T^{\text{MLP}} = \lambda \sum_i \eta_i \frac{\mathcal{M}_i}{f}, \tag{9}$$

where η_i is the binary value that controls whether the i-th layer is used or not, and η_i equals 1 means that the corresponding layer is executed, otherwise it is not. f is the CPU frequency of the processing device, λ is the computational efficiency factor, which is determined by the NN implementation and CPU architecture. Obviously, the inference delay of NN can be adjusted by changing the binary value η.

In this paper, not only is the depth of the NN changeable, but the width can also be adjusted as needed. Unlike the simple serially connected structure, we use a modular architecture to design the NN [27], the main part of which can be viewed as a parallel connection of many simple modules. Therefore, the width of the NN can be adjusted by changing the number of modules connected in parallel, and the total inference delay is

$$T^{\text{infer}} = \sum_j a_i T_j^{\text{MLP}}, \tag{10}$$

where a_j is the binary value that controls whether the j-th module is used or not, T_j^{MLP} represents the inference delay of the j-th MLP.

D. Problem Formulation

We formulate an optimization problem that determines whether computing locally or at the server, and manage the bandwidth and computing resources as $\mathcal{F}(\cdot|\boldsymbol{\theta})$ with trainable parameters $\boldsymbol{\theta}$. The objective is to jointly minimize the service delay of tasks, the deadline miss ratio, and the task processing energy consumption. The problem is formulated as follows

$$\begin{aligned}
\min_{\mathbf{x},\mathbf{b},\mathbf{f},\boldsymbol{\eta},\boldsymbol{a}} \sum_{i=1}^{N} &\alpha \left(T^{\text{infer}} + x_i T_i^{\text{edge}} + (1-x_i)T_i^{\text{loc}}\right) \\
&+ \beta \left(x_i E_i^{\text{edge}} + (1-x_i)E_i^{\text{loc}}\right) \\
&+ \delta \mathbb{I}\left(T^{\text{infer}} + x_i T_i^{\text{edge}} + (1-x_i)T_i^{\text{loc}} > d_i\right),
\end{aligned} \tag{11a}$$

$$\begin{aligned}
s.t. \quad & x_i \in \{0,1\}, \forall i \in \{1,2,\cdots,N\} && \text{(11b)}\\
& \eta_i \in \{0,1\}, \forall i \in \{1,2,\cdots,L\} && \text{(11c)}\\
& 1 \le \sum_{i=1}^{L} \eta_i \le L, && \text{(11d)}\\
& a_i \in \{0,1\}, \forall i \in \{1,2,\cdots,K\} && \text{(11e)}\\
& 1 \le \sum_{i=1}^{K} a_i \le K, && \text{(11f)}\\
& \mathbf{x}, \mathbf{b}, \mathbf{f} = \mathcal{F}(\cdot|\boldsymbol{\eta}, \boldsymbol{\theta}), && \text{(11g)}\\
& b_i \ge 0, \forall i \in \{1,2,\cdots,N\}, && \text{(11h)}\\
& \sum_{i=1}^{N} b_i \le b_{\max}, && \text{(11i)}\\
& f_i \ge 0, \forall i \in \{1,2,\cdots,N\} && \text{(11j)}\\
& \sum_{i=1}^{N} f_i \le f^{\text{ser}}, && \text{(11k)}
\end{aligned}$$

where α, β, and δ are weighting constants for delay, deadline miss ratio, and energy, respectively. The purpose of combining delay and energy weighting into one equation is to get an optimized solution for overall performance [28]. L is the maximum depth, K is the maximum width, and $\mathbb{I}(\cdot)$ is an indicator function when the input is true it equals 1 otherwise 0. The first part in Equation (11a) represents the processing delay of tasks consisting of the inference delay caused by the algorithm and the task computing delay. Due to the Constraint (11b), the x_i is either 0 or 1 that the task can only be processed either locally or at the server. The second part in Equation (11a) is the energy consumption when users process the task locally, and the third part aims to minimize the deadline miss ratio for all tasks. Constraints (11c) and (11d) are used to adjust the depth of the NN, and when η_i is equal to 0, the i-th layer is skipped during the inference process. Similarly, constraints (11e) and (11f) are used to adjust the depth of the NN. Obviously, both the width and depth of NN must be greater than 0, otherwise, it will not work, so constraints (11d) and (11f) are used. Since the architecture of the NN has not yet been described in detail, only an overview of the inference delay is given here, and we will analyze the inference delay further in the next section. Since the NN is used to determine whether to offload and to manage the bandwidth and computing resources, the Constraint (11g) is used to couple the value $\boldsymbol{\eta}, \boldsymbol{\theta}$ and $\mathbf{x}, \mathbf{b}, \mathbf{f}$. Constraints (11h)-(11k) are used to guarantee the resources allocated to each user performing the task at the server are larger than 0, and the total allocated resources do not exceed the resource budget.

IV. DyNN-based Approach for MEC

The representational power of deep neural network models is limited by the number of parameters. High-performance models are computationally expensive, which is unacceptable in delay-sensitive communication systems. Conditional computation, activating specific model parts based on sample characteristics, improves model capacity. Mixture of experts (MoE)-based DyNNs enhance performance by dynamically determining expert combinations using gating networks [29]. However, MoE lacks the ability to adjust computational complexity as needed. To address this issue, we propose an MoE-based model that allocates bandwidth and computing resources to users, and employs reinforcement learning (RL) for task scheduling. Advanced hyperparameter selection techniques are combined to balance model performance and inference delay. Additionally, we employ an efficient model training method to reduce deployment costs. In general, the scheme we designed consists of three parts, which are resource allocation model, scheduling model and policy model. In the actual deployment, these three models need to be connected in series to exercise the functions. The scheduling model outputs an offloading decision $\mathbf{x}$, which the policy model uses to determine $\boldsymbol{\eta}$ and $\boldsymbol{a}$ for adjusting the allocation model. Ultimately, the allocation model generates $\mathbf{b}$ and $\mathbf{f}$, completing the MEC system's resource management.

It should be noted that although the dynamic nature of DyNN allows adaptive adjustment and enhanced flexibility, it may also increase training complexity and potential instability, thereby affecting the performance and generalization of the model. Although these potential problems are not reflected in the issues addressed in this paper, they should not be ignored.

A. MoE-based Resource Allocation Model

Model architecture. With sparse gates [29], we design a DyNN for efficient resource allocation and obtain multiple resource allocation schemes through a single model. The architecture of the allocation model is shown in Fig 2(a). The main body of the model is an MoE layer consisting of multiple experts and a gating network. Each expert is represented by a neural network with the same structure but separate parameters. The trainable gating network that controls which experts are executed based on the input. This allows for a dynamic allocation of resources that adapts to the input data, making computations more efficient and improving overall performance. Suppose we have user information that needs to be processed, and as this information is fed into the model, the gating network analyzes the input and determines which experts should be activated. These selected experts then perform computations on the input, generating individual outputs that are later combined to produce the final output of the MoE layer. The key advantage of this approach is that it allows for a flexible and adaptable resource allocation strategy that changes depending on the input data. By leveraging sparse gates, our model can focus computational resources only on the necessary experts, reducing unnecessary computations and improving efficiency. Since the experts have the same structure, they have same input and output dimensions. When the input information x is given, we denote the output of the i-th expert as $E_i(x)$. When the number of experts is M, the output of the gating network is a sparse M-dimensional vector $G(x)$, so the output y of the MoE layer can be expressed as

$$y = \sum_{i=1}^{M} G(x)_i E_i(x), \tag{12}$$

where $G(x)_i$ is the weights for expert i. After getting $G(x)$, only the experts with non-zero weights need to be computed, and thus the computational efficiency is improved. To implement the sparsity computation, the output of the MoE layer is given as

$$G(x) = \mathcal{F}_{\text{Smax}}(TopK(H(x), k)), \tag{13}$$

where

$$\mathcal{F}_{\text{Smax}}(z_i) = \frac{z_i}{\sum_{c=1}^{C} e^{z_c}} \tag{14}$$

denotes the normalized exponential function, C is the output dimension,

$$TopK(n,k)_i = \begin{cases} n_i & \text{if } n_i \text{ is in the top } k \text{ elements of } n. \\ -\infty & \text{otherwise.} \end{cases} \tag{15}$$

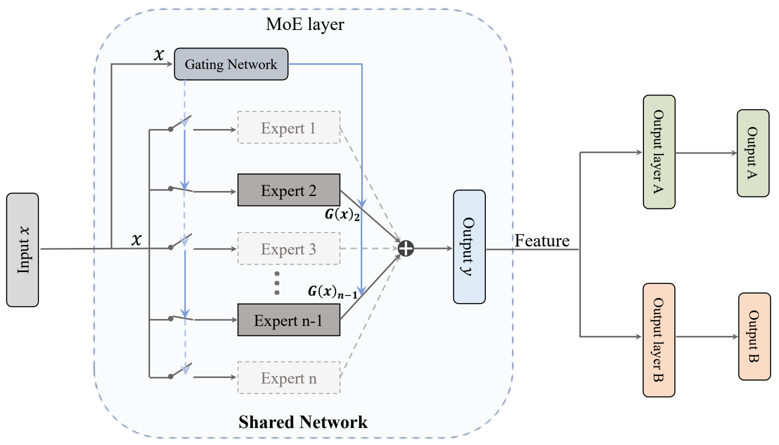
is used to extract the largest k elements of the vector, which keeps only the largest k gating values and sets the rest n-k gating values to negative infinity, so that these weights are zero after the operation of $\mathcal{F}_{\text{Smax}}$. $H(x)$ is the user feature function. In order to balance the weights of the experts to make each expert fully trained, a noise term is added to the input features, and the noise size is controlled by the trainable noise matrix W_{noise}.

$$H(x)_i = (x \cdot W_g)_i + \Phi \cdot \mathcal{F}_{\text{Splus}}\left((x \cdot W_{\text{noise}})_i\right), \tag{16}$$

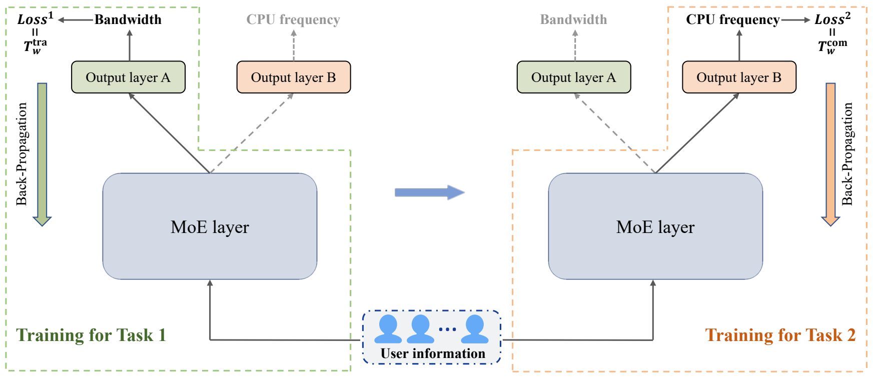
where W_g is the trainable weight matrix, Φ is the standard normal distribution,

$$\mathcal{F}_{\text{Splus}}(x) = \ln\left(1 + e^x\right). \tag{17}$$

Training method. Fig 2(b) demonstrates the training process of the proposed dynamic neural model. Since the energy



(a) Model Architecture



(b) Training Process

Fig. 2. Overview of the proposed allocation model.

consumption of task processing at the user side is independent of the resource allocation scheme, the transmission energy consumption is proportional to the transmission delay, we optimize the parameters of the allocation model using only delay-related metrics. In order to make the model accomplish the allocation of transmission bandwidth and CPU frequency simultaneously, we use the same input data for two parameter updates during training. Specifically, the user information is first propagated forward according to the path on the left, computing in the order from the MoE layer to the output layer A to obtain the bandwidth allocation vector $\mathbf{b}$, expressed as

$$\mathbf{b} = \mathcal{F}_{\text{Smax}}(\mathcal{F}_{\text{Line}}^{\text{A}}(y)), \tag{18}$$

where $\mathcal{F}_{\text{Line}}^{\text{A}}(\cdot)$ is the linear function of the output layer A and y is the output of the MoE layer. After obtaining $\mathbf{b}$, the task transmission delay for user i can be calculated according to the Equation (1). In order to meet the deadline while optimizing the delay and energy, we define an importance weight in training

$$w_i = \gamma \frac{c_i^{\nu} s_i^{\nu}}{d_i}, \tag{19}$$

where γ is the importance proportionality constant, c_i^{ν} and s_i^{ν} denote the normalized workload and task size, respectively. Equation (19) indicates that tasks with more stringent delay requirements are significantly important. In addition, the processing tasks with higher workloads and larger data volume are challenging to meet the deadline, so we assume that their importance increases accordingly. Then, we weight the delay by importance to obtain the average weighted transmission delay $T_w^{\text{tra}} = \sum_{i=1}^{N} w_i T_i^{\text{tra}}$. Since b_i are all continuous variables, T_w^{tra} calculated from them can be directly backpropagated [30]. Therefore, we use T_w^{tra} as the loss function to train the model. After completing the update of the model parameters for bandwidth allocation, we train the model in the same way for the computing resource allocation. Similarly, the frequency allocation vector $\mathbf{f}$ is denoted by

$$\mathbf{f} = \mathcal{F}_{\text{Smax}}(\mathcal{F}_{\text{Line}}^{\text{B}}(y)), \tag{20}$$

where $\mathcal{F}_{\text{Line}}^{\text{B}}(\cdot)$ is the linear function of the output layer B. Based on $\mathbf{f}$, the average weighted processing delay T_w^{com} can be calculated to complete the second parameter update. The model is trained according to the above process until convergence. The inference efficiency is significantly improved because the output layers account for a lower proportion of the computation.

B. Progressive Shrinking Training for Allocation Model

In the practical deployment scenario, due to the limitation of hardware efficiency, the computation cost of the algorithm is an important factor affecting the quality of service. Therefore, to adapt to different hardware platforms and user demands, the computational complexity of model should be adjusted as needed to balance inference performance and computational overhead. For the proposed allocation model, the main computational complexity depends on the width of MoE k (number of activated experts) and the depth of expert modules l (number of layers in experts). Therefore, we dynamically adjust them in the inference process. However, it is usually not feasible to directly change the model size during inference operations, due to significant performance loss. Since when expanding the model size, the added network structure has not been trained, it does not have inference ability; when the model size is reduced, the abandoned network structure changes the model mapping function, which will also lead to great damage to performance. Therefore, large and sub-models need to be trained to support the dynamic inference.

To obtain trained sub-networks, the traditional approach is to optimize all the models from scratch [31], in which the gradients of all the models need to be computed to update the parameters. Obviously, the training overhead of this approach is huge and linearly related to the number of sub-networks. To reduce the training overhead, we use an advanced training method, namely progressive shrinking [32]. Specifically, we first train the largest neural network, and then sequentially shrink the width and depth to further train the sub-networks. The small network is initialized with the trained parameters of the large network during the progressive shrinking process, rather than being trained from scratch. Since the small network inherits the weights learned by the large network, the training efficiency can be significantly improved.

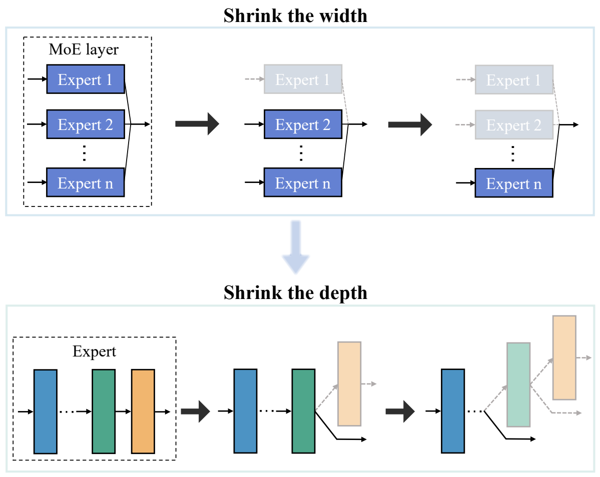


Fig. 3. Progressive Shrinking.

C. Joint Optimization of Task Scheduling and Model Adjustment

After training the allocation model, it becomes necessary to select an appropriate size for the NN used in resource allocation based on the requirements of the given scenario. The computational complexity and inference performance of the model are determined jointly by the hyperparameters k and l. Drawing inspiration from [32], we propose the construction of a policy network aimed at predicting the optimal model size. However, it is not feasible to independently train the policy network solely based on the performance of allocation models with varying sizes. This is due to the fact that the system's overall performance is also influenced by scheduling decisions.

Specifically, in the context of resource allocation problems, once task scheduling decisions have been made, we can determine the optimal size for the allocation model by considering information about the tasks that have been offloaded to the server. As locally executed tasks do not require transmission and computing resources, they do not affect the selection of the model size. Similarly, when it comes to task scheduling problems, the optimal scheduling scheme differs depending on the size of the allocation model being used. Under identical conditions, using a resource allocation algorithm that exhibits superior performance leads to the scheduling model offloading more tasks. Consequently, optimizing the offloading decision and the model size must be done jointly. However, both of these problems involve discrete and non-microscopic decision processes, rendering them challenging to solve using conventional optimization methods. Furthermore, the offloading decision is contingent upon highly dynamic task characteristics, making it difficult to obtain training labels. To address these difficulties, we employ a reinforcement learning approach to train the scheduling model while simultaneously optimizing the policy model through supervised learning.

Specifically, we train the scheduling model using a policy gradient approach, where the state information $\boldsymbol{s}$ is the user feature vector,

$$\boldsymbol{s} = \{u_1, u_2, \ldots, u_N\}, \tag{21}$$

the action is the offloading decision, which is denoted as

$$\boldsymbol{a} = \{p_1, p_2, \ldots, p_N\}, \tag{22}$$

where p_1 indicates the probability of offloading the task o_1. Typically, the training of reinforcement learning requires a lot of exploration to find the best strategy. For the task scheduling problem, since each task has two decision behaviors, the action space is 2^N, and the training overhead is intolerable in the case of a large number of tasks. To accelerate the convergence of the model, for the task o_i, we define the reward as

$$R_i = \begin{cases} (\mathcal{C}_i^{\text{loc}} - \mathcal{C}_i^{\text{edge}})p_i, & \text{if offloading;} \\ (\mathcal{C}_i^{\text{edge}} - \mathcal{C}_i^{\text{loc},})p_i, & \text{otherwise,} \end{cases} \tag{23}$$

where $\mathcal{C}_i^{\text{loc}}$ denotes the cost of executing task i locally, $\mathcal{C}_i^{\text{edge}}$ denotes the execution cost of offloading task i to the edge server. Equation (23) shows that the reward value is positive when the offloading decision made is better than the opposite decision, thus increasing the probability of the scheduling network outputting the action. Conversely, when the decision is inferior to the opposite decision, the reward value is negative, thus decreasing the probability of the action. The execution cost includes the weighted total delay and energy consumption, which can be expressed as

$$\mathcal{C}_i^{\text{loc}} = \alpha(w_i\left(T_l^{\text{infer}} + T_i^{\text{loc}}\right)) + \beta E^{\text{loc}}, \tag{24}$$

$$\mathcal{C}_i^{\text{edge}} = \alpha(w_i\left(T_e^{\text{infer}} + T_i^{\text{edge}}\right)) + \beta E_i^{\text{edge}}, \tag{25}$$

where T_l^{infer} denotes the inference delay for local execution, which is generated by the scheduling model, and T_e^{infer} denotes the inference delay in the offloading case, which is the sum of the inference delays of the scheduling, policy, and allocation models. To obtain the best system performance, the average reward $R(\mathcal{A}) = \frac{1}{N}\sum_{i=1}^{N} R_i$ for all tasks is used as the overall reward to update the parameters of the scheduling model. Then, the gradient of the scheduling network can be expressed as

$$\nabla_{\theta^\mu} J = \frac{1}{\mathcal{N}} \sum_{i=1}^{\mathcal{N}} R(\boldsymbol{a}) \nabla_{\theta^\mu} \mu\left(\boldsymbol{s} \mid \theta^\mu\right)\big|_{\boldsymbol{s}_i}, \tag{26}$$

where $\mathcal{N}$ is the number of training samples. Each interaction of the scheduling model with the MEC system generates a set of training samples. $\mu\left(s \mid \theta^\mu\right)$ denotes the strategy of the scheduling model, which is the task offloading probability of each user, i.e., $\mu\left(\boldsymbol{s} \mid \theta^\mu\right) = \{p_1, p_2, \ldots, p_N\}$. θ^μ is the trainable parameter. Then, the model parameters can be updated according to the following formula,

$$\theta^{\mu\prime} = \theta^\mu + l_r \nabla_{\theta^\mu} J, \tag{27}$$

where $\theta^{\mu\prime}$ is the updated parameters and l_r is the learning rate.

The policy network determines the number of active experts k in the specified model and the number of layers executed in each expert l. If l is less than the total number of layers in the expert, only the first l layers need to be computed. Note that resource allocation is not required for tasks executed locally, so only tasks offloaded to the server need to be considered. The optimization goal of the policy network is to balance the computational complexity of the model and the inference performance to obtain the lowest total service delay. Therefore, we define the training label of the policy network as

$$\mathcal{P}_{\text{Label}} = \underset{k,l}{\arg\min}\ \mathcal{C}^{\text{edge}}, \tag{28}$$

where $\mathcal{C}^{\text{edge}} = \frac{1}{N}\sum_{i=1}^{N} \mathcal{C}_i^{\text{edge}}$ denotes the average cost of the offloaded tasks. Then, the loss function of the policy network can be represented as

$$\mathcal{L}_{\text{Policy}} = \left\|\mathcal{P}_{\text{Label}} - \mathcal{P}_{\text{out}}\right\|^2, \tag{29}$$

where $\mathcal{P}_{\text{out}}$ denotes the output of the policy network.

In equation (25), the inference delay T_e^{infer} depends on the prediction result of the policy network, and the input of the policy network depends on the offloading decision given by the scheduling model. The parameter optimization process of the above two models is coupled. In the initial stage, neither the scheduling model nor the policy model can give valid decisions, which will make the model training difficult. This is because the parameter updates of the policy model cause changes in the environment information, which makes it difficult for the model to converge. For the policy model, the change of the scheduling strategy produces different sample labels, which makes the optimization direction of the model oscillate and thus hinders the training. To solve this problem, we first pre-train the policy model with all tasks offloaded, then optimize the scheduling model based on the pre-trained policy model, and further train the policy model. The specific training process is shown in Algorithm 1.

Algorithm 1 Joint Training of Policy Model and Scheduling Model

```
1: Train the allocation model and fix the parameter
2: Pre-train the policy model
3: Randomly initialize policy model with parameter $\boldsymbol{\omega}$
4: Generate user information samples and get the policy labels by
   Eq. (28)
5: Update $\boldsymbol{\omega}$ to minimize $\mathcal{L}_{\text{Policy}}$
6: Obtain the pre-trained parameter $\boldsymbol{\omega}^\circ \leftarrow \boldsymbol{\omega}$
7: RL-based joint training
8: Randomly initialize scheduling model with parameter $\boldsymbol{\theta^\mu}$
9: Initialize policy model with parameter $\boldsymbol{\omega}^\circ$
10: for episode = 1,. . ., $E_{\max}$ do
11:     Generate user information samples as state $s$
12:     Obtain offloading decision $\mathcal{A}$ by Scheduling Model and get
        training samples accordingly
13:     Get $k$, $l$ by policy model and allocate resources by allocation
        model
14:     Get reward according to Eq. (23)
15:     Update $\omega$ to minimize $\mathcal{L}_{\text{Policy}}$
16:     Update $\theta^\mu$ to maximize $R$
17: end for
18: return  the trained parameters of policy model and scheduling
    model $\{\boldsymbol{\omega}^* \longleftarrow \boldsymbol{\omega}\}$, $\{\boldsymbol{\theta^{\mu *}} \longleftarrow \boldsymbol{\theta^\mu}\}$
```

D. Computational Complexity Analysis of NNs with Different Sizes

Forward inference in a NN is the process of deriving an output from the input by performing a finite number of matrix operations, which depends on the network structure. The total number of operations in any NN model is the sum of the operations in each subpart. The proposed scheme aims to dynamically adjust the computational load of the NN by modifying the network structure involved in inference. Specifically, we enable elastic width and depth by adjusting the number of activated expert modules in the MoE and the number of layers in each expert. Thus, the amount of operations of all expert modules in the inference process can be expressed as

$$\mathcal{M}_{\text{MoE}} = \sum_{i=1}^{K} a_i \mathcal{M}_i^{\text{ept}}, \tag{30}$$

where $\mathcal{M}_{\text{MoE}}$ denotes the total MACs of the experts during the inference process, $\mathcal{M}_i^{\text{ept}}$ is the MACs of the i-th expert. Since the structure of each expert in MoE is the same, they have the same MACs, which is denoted as $\mathcal{M}^{\text{ept}}$. Therefore, Equation (30) can be converted to

$$\mathcal{M}_{\text{MoE}} = k\mathcal{M}^{\text{ept}}, \tag{31}$$

where k denotes the number of experts activated. For each expert, since we set it as a serial network structure, its computation is the sum of the computations of the used layers. After obtaining the number of network layers l used by the policy model, only the first l layers of the expert need to be executed, i.e., the binary value η can be expressed as

$$\eta_i = \begin{cases} 1, & i \le l; \\ 0, & i > l. \end{cases} \tag{32}$$

Therefore, the computation of an expert can be expressed as

$$\mathcal{M}^{\text{ept}} = \sum_{i=1}^{L} \mathcal{M}_i^{\text{layer}}, \tag{33}$$

where $\mathcal{M}_i^{\text{layer}}$ represents the MACs of the i-th layer. From the above analysis, it can be seen that the operation amount of MoE is roughly linear with its width and depth.

In addition to the above structurally adjustable NNs, the proposed scheme includes a portion of fixed-architecture NNs with a constant computational complexity. Although these NNs cannot be adjusted on-demand, they have less impact on the system performance because of their small computation. Considering the inference delay generated by these NNs, we give the expressions for T_l^{infer} and T_e^{infer}

$$T_l^{\text{infer}} = \lambda \frac{\mathcal{M}_{\text{Sced}}}{f}, \tag{34}$$

$$T_e^{\text{infer}} = \lambda \frac{\mathcal{M}_{\text{Sced}} + \mathcal{M}_{\text{Poli}} + \mathcal{M}_{\text{Alloc}}}{f}, \tag{35}$$

where $\mathcal{M}_{\text{Sced}}$ is the MACs of scheduling model, $\mathcal{M}_{\text{Poli}}$ is the MACs of policy model, $\mathcal{M}_{\text{Alloc}}$ is the MACs of the allocation model, including both $\mathcal{M}_{\text{MoE}}$ and the output layer computation amount $\mathcal{M}_{\text{out}}$.

V. Simulation Results

In this section, to demonstrate the performance of the proposed algorithm and its ability to adapt to different situations, we evaluate it in two cases. We first evaluate the performance of the proposed DyNN-based resource allocation method in the case of users without computing power. All tasks in this case need to be transmitted to the server for processing and thus do not require the scheduling model for offloading decisions. In addition, the users in this case only consume transmission energy, which is much smaller than the computational energy of the tasks, so we only focus on the delay-related metrics. Then, we comprehensively evaluate the proposed joint resource allocation and task scheduling algorithm in the case where the user has computing power.

For the DyNN, we set each expert as an MLP with a maximum depth of L and the number of neurons per layer is 64. Both the scheduling model and the policy model are MLPs with one hidden layer, which contains 32 neurons. In addition, each layer in the above networks is activated by the LeakyReLU function. The dynamic and fixed amount of computations and parameters of NN in the proposed algorithm are listed in Table I.

TABLE I
Amount of Computations and Parameters

Dynamic MACs	Fix MACs	Dynamic Param	Fix Param
5.126M	0.302M	87.04K	7.291K

In simulations, the user information is assumed to be uniformly distributed. Specifically, the data size $s_i \sim \text{Unif}\,[0, 2s_{\text{avg}}]$ and task complexity $c_i \sim \text{Unif}\,[0, 2c_{\text{avg}}]$, where $s_{\text{avg}} = 1$ kbits [23], $c_{\text{avg}} = 797.5$ cycles/bit [33], and the task deadline d_i is chosen randomly from the interval of $[0, 160]$ms. In addition, the user computing power $f_{loc,i} \sim \text{Unif}\,[0.01, 0.15]$GHz in the second case. Since we are concerned with a delay-sensitive system, α is set to 1 and β is set to 0.1. The values of other simulation parameters are shown in Table II.

TABLE II
Simulation Parameters

Parameters	Values
Number of users N	30
Bandwidth of wireless channel $b_{\max}$	0.5MHz
Computation capacity of sever f_{ser}	2.5GHz
Computational efficiency factor λ	1
Distance between user i and the server l_i	[5, 500]m
Transmission Power p	0.1w
Noise Power σ_0	-174dBm/Hz
Path-loss constant g_0	-40dB
Path-loss exponent h	2.8
Energy efficiency constant	10^{-20}
Importance proportionality constant γ	60
Maximum width of MoE K	10
Maximum depth of expert L	3
Learning rate of allocation model	0.001
Learning rate of policy model	0.0001
Learning rate of scheduling model	0.002

A. Performance Evaluation of Resource Allocation

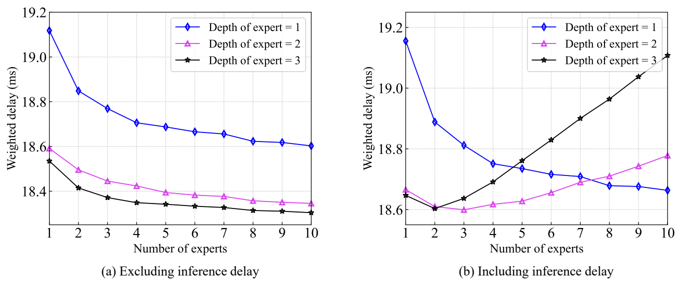


Fig. 4. Comparison of allocation performance for models of different sizes. (a): The performance of the model itself, i.e., the weighted delay without considering the inference process. (b): The deployment performance of the model, i.e., the total weighted delay considering the inference process.

To illustrate the necessity of choosing the appropriate model size, Fig. 4 shows the performance of different models. Fig. 4(a) shows the inference performance versus model size. We can observe that, as the width and depth increase, the weighted delay decreases, i.e., the allocation performance of the model improves. However, the performance improvement comes with the expense of increased inference cost. Fig. 4(b) shows the total weighted delay considering the inference cost, and the largest model does not achieve optimal performance because of its higher inference cost, thus it is necessary to rationalize the model size.

In the case of users without computing power, we compare the proposed resource allocation algorithm $PolicyMoE$ with the following baseline algorithms:

This article has been accepted for publication in IEEE Transactions on Cognitive Communications and Networking. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCCN.2023.3346824

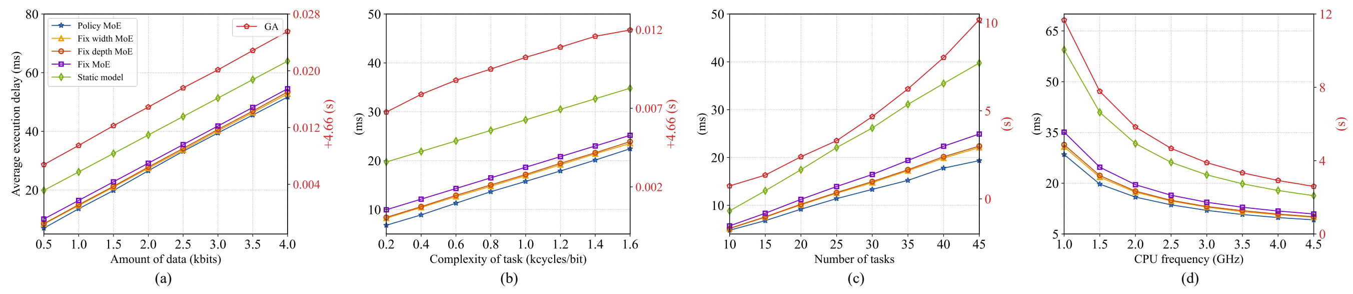


Fig. 5. Average execution delay of tasks in the case of users without computing power

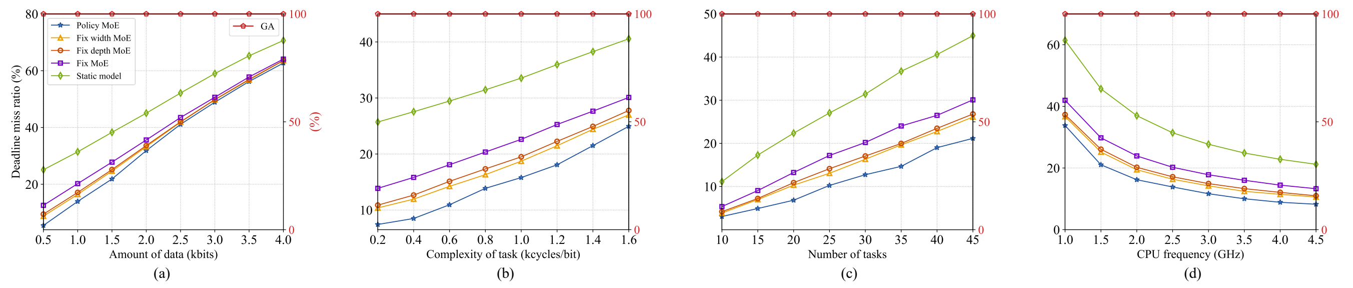


Fig. 6. Deadline miss ratio in the case of users without computing power

- $Fix\ width\ MoE$: The allocation model uses a fixed width MoE for inference, i.e., the policy model only adjusts the depth.
- $Fix\ depth\ MoE$: The allocation model uses a fixed depth MoE for inference, i.e., the policy model only adjusts the width.
- $Fix\ MoE$: The allocation model uses a fixed width and depth MoE for inference, i.e., it does not use the policy model to adjust the size.
- $Static\ model$: A conventional static neural network is used as the allocation model with the same structure as the deepest expert module and the number of neurons in the hidden layer is 384.
- GA: Genetic algorithm (GA) is used for transmission and computing resource allocation. GA generates a large population, and each person in the population includes the amount of resources allocated to each user. The fitness of an individual is set as the objective function defined in III.D. In each iteration, individuals with higher fitness are retained and other individuals with lower fitness are discarded. The crossover operator and mutation operator are used to generate new individuals, and uniform crossover is used as the crossover operator. The crossover rate of the GA is set to 0.2, which decays to one-half of the original value every 25 iterations, and the mutation rate is set to 0.1, which decays to one-quarter of the original value every 25 iterations. The population size is set to 6 times the number of tasks. The rotation method is used for individual selection so that individuals with higher fitness have a greater chance of being retained. Since the GA and the NN-based algorithm have huge differences in delay-related indicators, for the convenience of presentation, we use the additional coordinates on the right side in the result figures of delay and miss ratio to represent the values of the GA.

Note that among the comparison schemes, both Fix MoE and Static model cannot adjust their computations, while Fix width MoE and Fix depth MoE have the ability to adjust.

Fig. 5 shows the average execution delay of different conditions. It can be observed that the policy MoE is consistently superior to the comparison scheme under different scenarios. The various MoE-based allocation algorithms outperform the static model because the sparse gate structure and multi-task learning of MoE enhance the computational efficiency. Since the fixed MoE cannot adjust the computation on demand, there is still a large computational redundancy and the performance is not as good as that of the remaining MoE schemes. The policy MoE has the largest tuning range and therefore achieves optimal performance. Compared with NN-based schemes, the performance of GA changes in the same trend with different parameters. However, because the iteration process of GA is very time-consuming, its total delay reaches the second level, which is dozens or even hundreds of times that of other schemes.

Fig. 5(a) illustrates that the delay increases with the amount of data, which is due to the fact that the transmission delay and computation delay are proportional to the amount of data. Since the computation delay is proportional to the task complexity, Fig. 5(b) demonstrates a similar result. Furthermore, it can be observed that the performance advantage of the policy MoE in Fig. 5(a) and 5(b) decreases slightly with increasing delay. This is because the policy MoE improves performance by flexibly adjusting the computational volume, i.e., choosing a proper size so as to reduce the inference delay with less loss of accuracy. The increase in the amount of data and task complexity leads to a decrease in the proportion of inference delay and makes the policy model tend to execute more modules to guarantee the inference performance, thus reducing the gap with the fixed model.

Fig. 5(c) shows that the delay increases with the number of tasks and the performance advantage of the policy MoE increases as well. For a given amount of communication and computational resources in the network, more tasks mean fewer resources available for each task, and thus the service delay increases. Moreover, the increase in the number of tasks increases the dimensions of the model, which increases its computational volume and hence the inference delay. Therefore, the computational redundancy of various comparison algorithms increases, which makes the performance advantage of policy MoE more prominent.

Fig. 5(d) shows that the delay decreases with the increase in server power and the performance advantage of policy MoE decreases. With the same task characteristics, the increase in server computing power reduces the computational delay of the task and hence the service delay. In addition, since the model is deployed on the server for execution, the increase in server computing power decreases the inference delay, thus decreasing its proportion of the total delay, and thus the performance advantage of MoE. If the server has enough arithmetic power that the inference delay can be ignored, the delay will depend only on the inference performance of the model.

Fig. 6 shows the deadline miss ratio under different conditions. Similarly, it can be seen that the policy MoE outperforms the various comparison schemes. Since the deadline miss ratio is positively correlated with the delay, the trend in Fig. 6 is similar to that of Fig. 5. Moreover, it can be seen that the performance advantage of policy MoE in Fig. 6 is more pronounced corresponding to the low delay case in Fig. 5. This is because with low delay, most tasks can be completed within the deadline and the overrun delay for failed tasks is small, so a small reduction in delay can significantly decrease the deadline miss ratio. Since the large delay of GA far exceeds the tolerance of the delay-sensitive tasks of concern (the upper limit of the deadline is no more than 1 s), its miss ratio has reached 100%.

B. Performance Evaluation of Joint Resource Allocation and Task Scheduling

In the case of users having computing power, we compare the proposed joint resource allocation task scheduling algorithm $Policy\ MoE(RL)$ (PMRL) with the baseline algorithm using different scheduling approaches as follows:

- RL: Joint optimal resource allocation and RL-based task scheduling.
- $Rand$: Each user randomly chooses offloading or local execution with equal probability.
- $Local\ execution$: Each user executes the task with the full local computing power, thus eliminating the need for task scheduling and resource allocation.
- GA: GA is used for joint task scheduling and resource allocation. As the complexity of the problem increases, we change the decay interval of the crossover rate and mutation rate to 40 to allow the algorithm to explore more fully in the solution space. Other settings remain unchanged.

Fig. 7 shows the average execution delay of different conditions. It can be seen that PMRL consistently outperforms the comparison schemes under different conditions. In the case of using the same offloading strategy, the performance comparison results for the various algorithms are the same as the conclusions obtained in the previous subsection, because the performance difference at this point depends only on the allocation algorithm. For the same allocation algorithm, the performance using the RL-based scheduling scheme is significantly better than the random scheduling method. This is because RL is able to optimize strategies based on environmental information, thus significantly improving the utilization of computing resources. Similar to the results shown in V.A, the total delay of GA reaches the second level due to the excessive computation, and the convergence slows down due to the increase in problem complexity, causing the delay of GA to further increase compared to the case of V.A.

Fig. 7(a) and Fig. 7(b) show that the delay increases with the data size and task complexity, which is the same as A. In the case of low delay, the solution with a random offloading strategy does not perform as well as the local execution. This is because these schemes cannot maximize the utilization of the computing power of the system according to the task features. The inference delay generated by them in allocating resources accounts for a larger proportion of the total delay, and thus the performance is poor. The local execution scheme, on the other hand, can not utilize the server's computing power, but does not generate inference delay, so the performance is better. The RL-based offloading strategy can maximize the utilization of the system's computing power, and thus outperforms the local execution scheme.

Fig. 7(c) shows that the delay increases with the number of tasks for all schemes except the local execution scheme, and the performance advantage of PMRL expands accordingly, which is consistent with the findings obtained by Fig. 5(c). The local execution scheme does not require the use of channel and server resources, so the delay does not increase with the number of tasks. Similarly, the performance of the local execution scheme in Fig. 7(d) is independent of the server computing power, and the rest of the schemes follow the same trend as Fig. 5(d).

Fig. 8 evaluates the deadline miss ratio for each scheme under different conditions. The performance comparison results are consistent with the delay for all schemes except local execution. The performance of the local execution scheme improves, outperforming the random offloading scheme in most cases. This is because with the same average delay, each user has to bear the inference delay generated by resource allocation when using the random offloading scheme, and the inference delay is the same for different users, which leads to a significant increase in the miss ratio for tasks with tighter deadlines. Therefore, local execution is a proper solution if most of the tasks themselves require very small processing delays. Likewise, GA has a miss rate of 100% because its delay is far beyond the tolerance of the tasks.

Fig. 9 shows the user-side energy cost under different conditions. The local execution scheme generates the largest energy cost because it requires all users to process the tasks with their own computing power. Unlike other metrics, the energy cost in Fig. 9(c) is positively correlated with the number of users since we are concerned with the total energy cost on the user

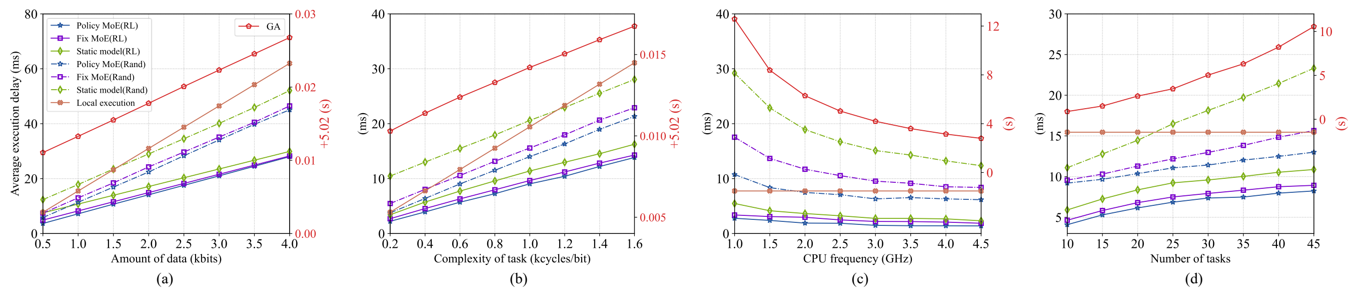


Fig. 7. Average execution delay of tasks in the case of users having computing power

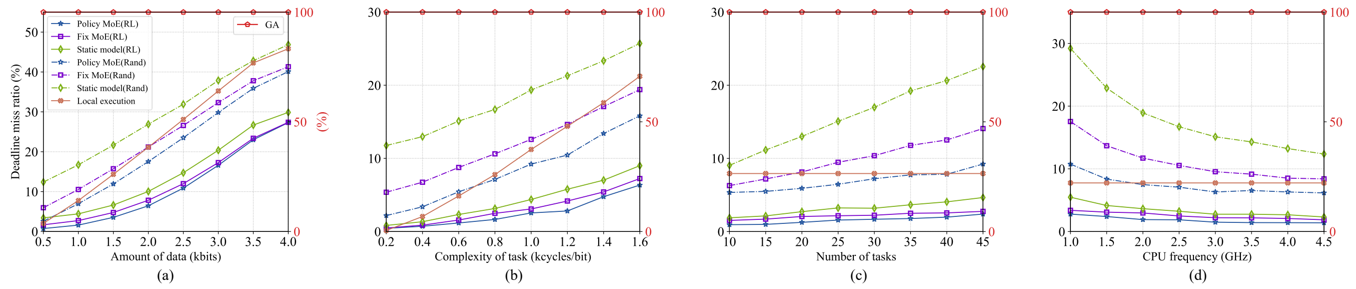


Fig. 8. Deadline miss ratio in the case of users having computing power

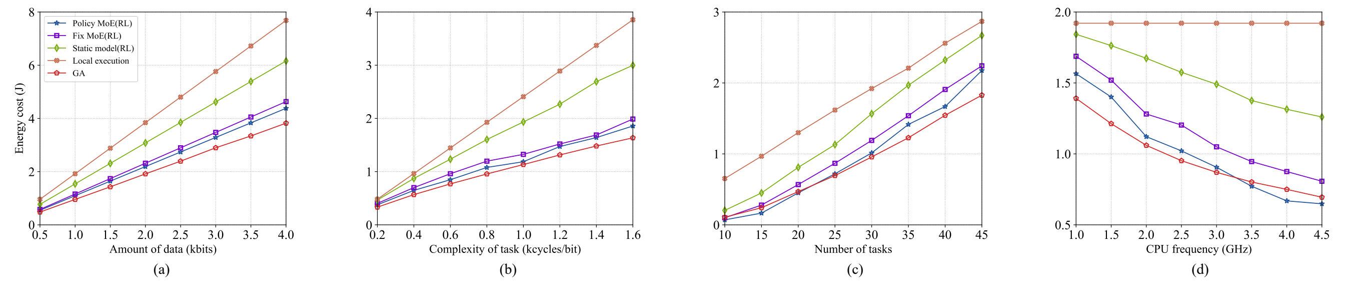


Fig. 9. Energy cost in the case of users having computing power

side. In addition, the difference between the other schemes is mainly determined by the number of offloaded tasks, which is due to the fact that the transmission energy consumption of the tasks is much greater than the computing energy consumption. For the schemes using RL-based scheduling strategy, using a resource allocation algorithm with better performance is equivalent to improving the computing power of the server, thus making the scheduling strategy tend to offload more tasks. Therefore, the energy cost comparison results for each scheme are the same as the delay, i.e., the scheme with lower delay can also achieve lower energy cost. It is worth noting that for energy consumption metrics, GA outperforms neural networks because energy consumption has nothing to do with algorithm execution overhead. This shows that although the traditional algorithm has shortcomings in computational overhead and flexibility, the quality of the given solution can be guaranteed, and it is an effective solution without considering the algorithm overhead.

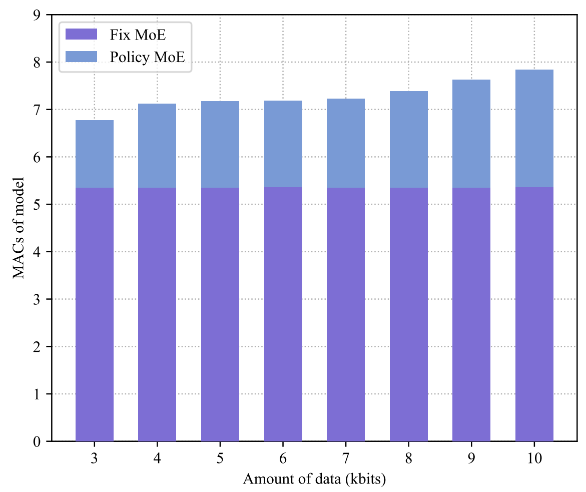


Fig. 10. MACs of model versus data amount

To better show the behavior of the policy model, we further evaluate the choice of model size. Fig. 10 shows the average model MACs for different data amounts, and it can be seen that the MACs of the policy MoE are positively correlated with the data amount and significantly smaller than the fixed MoE. This is because the delay and energy consumption increase with the data amount, increasing the performance gap between different models, and thus the performance inferiority of small models widens. So a larger model needs to be selected to achieve the best performance. In contrast, the fixed MoE always uses large-scale for inference, so there is a huge amount of computational redundancy. Further, Fig. 11 shows the percentage of inference delay for different data volume cases, and it can be seen that the percentage of inference delay

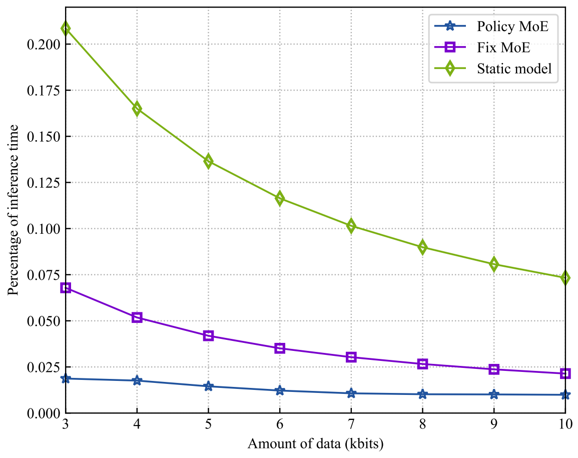


Fig. 11. Percentage of inference delay versus data amount

for the static model and fixed MoE is inversely proportional to the data volume, while the policy MoE is more stable and always stays at a low level.

C. Training Cost Analysis

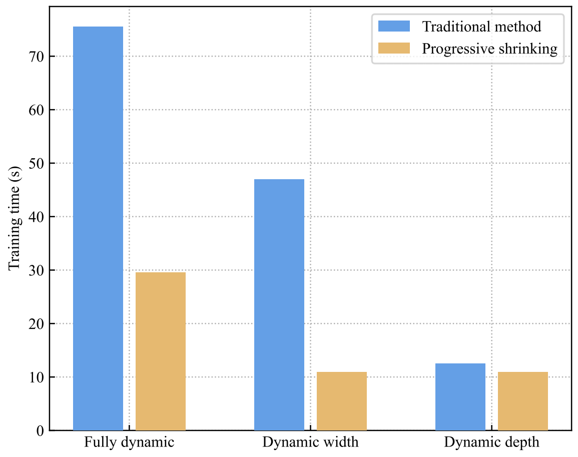


Fig. 12. Training delay of different methods

Fig. 12 shows the delay of training different MoE models with progressive shrinkage and traditional methods. Training full dynamic models (width and depth) and width dynamic models using progressive shrinkage can significantly reduce the delay, while the difference in delay for training depth dynamics is relatively small. This is because the MoE model has a smaller width shrinkage (one-tenth of the maximum width) and a larger depth shrinkage (one-third of the maximum depth), and the initial performance of the small network is poorer when the shrinkage is larger, thus requiring a longer training delay.

VI. Conclusion

In this paper, the algorithm inference delay, computing delay, and transmission delay have been jointly optimized by the proposed DyNN-based resource management method. In addition, a new shrinking training method has been proposed to reduce the training cost of DyNN. Simulation results show that the proposed DyNN-based method outperforms the traditional optimization algorithm and the static-NN-based method. By applying the proposed method in the network, the controller can balance algorithm complexity and performance, on demand of features of tasks. In future work, we will investigate how to use DyNN to solve the timing optimization problem.

References

[1] L. Ma, N. Cheng, X. Wang, R. Sun, and N. Lu, "On-Demand Resource Management for 6G Wireless Networks Using Knowledge-Assisted Dynamic Neural Networks," in *Proc. IEEE ICC*, 2022, pp. 1–6.

[2] N. Cheng, W. Xu, W. Shi, Y. Zhou, N. Lu, H. Zhou, and X. Shen, "Air-ground integrated mobile edge networks: Architecture, challenges, and opportunities," *IEEE Commun. Mag.*, vol. 56, no. 8, pp. 26–32, 2018.

[3] 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, 2021.

[4] T. Ma, H. Zhou, B. Qian, N. Cheng, X. Shen, X. Chen, and B. Bai, "Uav-leo integrated backbone: A ubiquitous data collection approach for b5g internet of remote things networks," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 11, pp. 3491–3505, 2021.

[5] Y. Chen, N. Zhang, Y. Zhang, X. Chen, W. Wu, and X. Shen, "Energy efficient dynamic offloading in mobile edge computing for internet of things," *IEEE Trans. Cloud Comput.*, vol. 9, no. 3, pp. 1050–1060, 2021.

[6] C. Zhou, W. Wu, H. He, P. Yang, F. Lyu, N. Cheng, and X. Shen, "Deep reinforcement learning for delay-oriented iot task scheduling in sagin," *IEEE Trans. Wireless Commun.*, vol. 20, no. 2, pp. 911–925, 2021.

[7] N. Cheng, H. Jingchao, Y. Zhisheng, Z. Conghao, W. Huaqing, L. Feng, Z. Haibo, and S. Xuemin, "6g service-oriented space-air-ground integrated network: A survey," *CJA*, vol. 35, no. 9, pp. 1–18, 2022.

[8] Q. He, G. Cui, X. Zhang, F. Chen, S. Deng, H. Jin, Y. Li, and Y. Yang, "A game-theoretical approach for user allocation in edge computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 3, pp. 515–529, 2020.

[9] M. R. Garey and D. S. Johnson, "Computers and intractability: A guide to the theory of np-completeness," W. H. Freeman & Co.: New York, NY, USA, 1979.

[10] A. Yazar, S. Dogan-Tusha, and H. Arslan, "6g vision: An ultra-flexible perspective," *ITU J. Future Evol. Technol.*, vol. 1, no. 1, pp. 121–140, 2020.

[11] N. Cheng, F. Lyu, W. Quan, C. Zhou, H. He, W. Shi, and X. Shen, "Space/Aerial-Assisted Computing Offloading for IoT Applications: A Learning-Based Approach," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 1117–1129, 2019.

[12] X. Wang, L. Fu, N. Cheng, R. Sun, T. Luan, W. Quan, and K. Aldubaikhy, "Joint Flying Relay Location and Routing Optimization for 6G UAV–IoT Networks: A Graph Neural Network-Based Approach," *Remote Sensing*, vol. 14, no. 17, p. 4377, 2022.

[13] Y. Liu, H. Yu, S. Xie, and Y. Zhang, "Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 11, pp. 11 158–11 168, 2019.

[14] J. Xu, L. Chen, and S. Ren, "Online learning for offloading and autoscaling in energy harvesting mobile edge computing," *IEEE Trans. Cogn. Netw.*, vol. 3, no. 3, pp. 361–373, 2017.

[15] X. You, Y. Huang, S. Liu, D. Wang, J. Ma, W. Xu, C. Zhang, H. Zhan, C. Zhang, J. Zhang *et al.*, "Toward 6G TKμ Extreme Connectivity: Architecture, Key Technologies and Experiments," *arXiv preprint arXiv:2208.01190*, 2022.

[16] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang, "Dynamic neural networks: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 11, pp. 7436–7456, 2022.

[17] R. Dong, C. She, W. Hardjawana, Y. Li, and B. Vucetic, "Deep learning for hybrid 5g services in mobile edge computing systems: Learn from a digital twin," *IEEE Trans. Wireless Commun.*, vol. 18, no. 10, pp. 4692–4707, 2019.

[18] Y. Li, X. Wang, X. Gan, H. Jin, L. Fu, and X. Wang, "Learning-aided computation offloading for trusted collaborative mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 19, no. 12, pp. 2833–2849, 2020.

[19] X. Wang, Z. Ning, and S. Guo, "Multi-agent imitation learning for pervasive edge computing: A decentralized computation offloading algorithm," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 2, pp. 411–425, 2021.

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

[21] J. Chen, S. Chen, Q. Wang, B. Cao, G. Feng, and J. Hu, "iraf: A deep reinforcement learning approach for collaborative mobile edge computing iot networks," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 7011–7024, 2019.

[22] Y. Yang, F. Gao, M. Wang, J. Xue, and Z. Xu, "Dynamic Neural Network for MIMO Detection," *IEEE J. Sel. Areas Commun.*, vol. 40, no. 8, pp. 2254–2266, 2022.

[23] Y. Mao, J. Zhang, and K. B. Letaief, "Joint task offloading scheduling and transmit power allocation for mobile-edge computing systems," in *Proc. IEEE WCNC*, 2017, pp. 1–6.

[24] W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, and D. O. Wu, "Energy-optimal mobile cloud computing under stochastic wireless channel," *IEEE Trans. Wireless Commun.*, vol. 12, no. 9, pp. 4569–4581, 2013.

[25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[26] M. A. Nahmias, T. F. de Lima, A. N. Tait, H.-T. Peng, B. J. Shastri, and P. R. Prucnal, "Photonic multiply-accumulate operations for neural networks," *IEEE J. Sel. Topics Quantum Electron.*, vol. 26, no. 1, pp. 1–18, 2020.

[27] S. Mittal, Y. Bengio, and G. Lajoie, "Is a modular architecture enough?" *arXiv preprint arXiv:2206.02713*, 2022.

[28] J. Song, Q. Song, Y. Wang, and P. Lin, "Energy–delay tradeoff in adaptive cooperative caching for energy-harvesting ultradense networks," *IEEE Trans. Comput. Soc. Syst*, vol. 9, no. 1, pp. 218–229, 2022.

[29] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," *arXiv preprint arXiv:1701.06538*, 2017.

[30] Y. Shen, Y. Shi, J. Zhang, and K. B. Letaief, "Graph neural networks for scalable radio resource management: Architecture design and theoretical analysis," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 1, pp. 101–115, 2021.

[31] J. Yu and T. S. Huang, "Universally slimmable networks and improved training techniques," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2019, pp. 1803–1811.

[32] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," *arXiv preprint arXiv:1908.09791*, 2019.

[33] A. P. Miettinen and J. K. Nurminen, "Energy efficiency of mobile clients in cloud computing," in *HotCloud*, 2010.