ISSN: 2414-1895 DOI: 10.6919/ICJE.202604_12(4).0042

A Knowledge Distillation-Based Lightweight Task Offloading Algorithm for Mobile Edge Computing

Xinyu Wang[1, a], Yimai Wang[2, b], Can Ti[1, c], Bo Cui[1, d, *]

[1] College of Artificial Intelligence, North China University of Science and Technology, Tangshan, Hebei, China

[2] College of Sciences, North China University of Science and Technology, Tangshan, Hebei, China

Email: [a]13731493001@163.com, [b]13332080368@163.com, [c]ti_can_do@163.com,

[d, *]mikecui@ncst.edu.cn

Abstract

To address the issues of complex models, high inference latency, and difficulties in deploying traditional deep reinforcement learning offloading strategies on resource-constrained devices in mobile edge computing, this paper proposes a lightweight D3QN computation offloading method based on knowledge distillation. This method constructs a teacher-student network architecture, transferring the decision-making capabilities of the teacher model to a lightweight student network via offline knowledge distillation. Concurrently, an adaptive exploration strategy incorporating heuristic rules is designed to accelerate convergence, whilst LSTM temporal awareness and a dynamic energy consumption normalisation mechanism are introduced to optimise QoE. Experimental results demonstrate that, at a compression ratio of 0.4, the number of parameters in the student model is reduced by 77.61%, inference latency is reduced by 26.32%, and QoE reaches 829.07, approaching the performance of the teacher model, thereby providing an efficient and feasible lightweight solution for edge intelligence offloading.

Keywords

Mobile Edge Computing; Task Offloading; Deep Reinforcement Learning; Knowledge Distillation; Lightweight Models; D3QN.

1. Introduction

With the advancement of 5G/6G and artificial intelligence technologies, new applications such as autonomous driving, augmented/virtual reality, and smart manufacturing are continuously emerging. These tasks are typically computationally intensive, latency-sensitive, and highly dynamic, far exceeding the processing capabilities and battery capacity of a single mobile device. Mobile Edge Computing (MEC) provides an effective solution for task offloading by deploying computational resources on the wireless access network side [1]. Its core concept involves offloading complex tasks to edge servers for execution, thereby reducing latency and extending device battery life. In recent years, deep reinforcement learning has emerged as the mainstream method for MEC task offloading due to its sequential decision-making capabilities in dynamic environments [2–7], achieving significant progress in reducing latency and energy consumption.

However, existing DRL-based offloading strategies still have many shortcomings in practical deployment. On the one hand, most DRL models have a large number of parameters; running them

ISSN: 2414-1895 DOI: 10.6919/ICJE.202604_12(4).0042

directly on resource-constrained mobile devices results in significant inference latency and additional power consumption. On the other hand, during the early stages of training, the agent relies on random exploration, which makes it prone to making erroneous decisions-such as offloading large tasks to congested servers-leading to high packet loss rates and slow convergence. Furthermore, QoE optimization in existing work often employs a simple linear weighting of latency and energy consumption, neglecting the temporal evolution of edge node loads and the heterogeneous energy consumption preferences of end devices, making it difficult to maintain stable user experience quality in dynamic networks.

To address the above issues, this paper proposes a lightweight computation offloading method based on knowledge distillation. The main contributions of this paper are as follows:

(1) We propose a lightweight teacher-student knowledge distillation offloading framework for resource-constrained endpoints. By constructing a teacher-student architecture, we train a high-precision teacher network in the cloud and transfer decision features to a lightweight student network via knowledge distillation, thereby significantly reducing inference latency whilst maintaining high QoE.

(2) A context-aware adaptive exploration strategy incorporating heuristic rules has been designed. By embedding heuristic rules into the action selection phase and adapting the exploration probability based on task urgency, edge node congestion, and terminal power consumption patterns, the strategy accelerates convergence and reduces early packet loss rates.

(3) A QoE optimisation mechanism based on spatio-temporal awareness and dynamic energy consumption normalisation has been established. An LSTM is introduced to extract temporal load features of edge nodes, enabling congestion prediction; a dynamic-weighted QoE reward function is designed to uniformly normalise both delay and multi-level energy consumption, ensuring convergence stability in heterogeneous network environments.

2. Related Work

In recent years, the problem of task offloading in Mobile Edge Computing (MEC) has become a hot topic in academic research. Due to the inherent limitations of mobile devices in terms of computing power and battery capacity, the core challenge in MEC offloading research is how to efficiently offload tasks to edge servers in order to optimize key metrics such as system power consumption, task latency, task completion rate, and Quality of Experience (QoE). To address the single or joint optimization of these metrics, researchers have proposed various methods from different perspectives. Regarding the joint optimization of energy consumption and latency, Ale et al. [4] proposed an end-to-end DRL method that jointly optimizes edge server selection and computing frequency allocation, thereby improving task on-time completion rates while minimizing energy consumption. Lu et al. [2] designed a decentralized offloading framework based on a dual-critic DDPG for smart manufacturing scenarios, fitting energy consumption and data processing volume separately, which effectively accelerated model convergence. Li et al. [5] proposed a dual-experience-pool DDPG strategy that uses hybrid experience sampling to prevent training from getting stuck in local optima. Although the aforementioned methods have achieved significant results in reducing system energy consumption and latency, their DRL models generally employ multi-layer fully connected network architectures, which have a large number of parameters and high computational overhead, making them difficult to deploy directly on resource-constrained mobile terminals.

To address the issue of overly large DRL models, a small number of studies have explored model compression techniques. Hao et al. [15] were the first to introduce pruning techniques into MEC offloading scenarios, reducing model complexity by removing redundant connections. However, pruning methods have limited compression ratios, and performance often requires fine-tuning after compression to be restored. Knowledge distillation, as a more efficient model compression technique, can achieve higher compression ratios; however, its application in DRL-driven MEC offloading remains extremely rare. Furthermore, existing work generally assumes that both training and

ISSN: 2414-1895 DOI: 10.6919/ICJE.202604_12(4).0042

inference are performed on cloud or edge servers, overlooking the actual computational overhead of on-device inference-even if a model is trained in the cloud, the inference process still consumes the device's CPU/GPU resources and battery power, which severely limits the practical deployment of DRL methods on resource-constrained devices.

To further enhance user satisfaction, some studies have also adopted QoE as an optimization objective. Zhao et al. [7] investigated a multi-agent DRL framework in drone-assisted MEC systems, jointly optimizing drone trajectories, task allocation, and communication resource management to improve task completion rates. Chouikhi et al. [12] proposed an energy-efficient computational offloading method based on multi-agent DRL for the Industrial Internet of Things (IIoT), in which each device makes independent decisions to ensure individual QoE. However, most existing QoE modeling methods employ a fixed weighted sum of latency and energy consumption, lacking the ability to perceive the temporal evolution of edge node loads and ignoring the heterogeneous energy consumption preferences of end devices under different power modes. This results in insufficient stability and adaptability of the strategies in dynamically changing network environments.

In summary, existing DRL-based MEC task offloading methods still have shortcomings. The knowledge distillation-based lightweight offloading method proposed in this paper systematically addresses the aforementioned issues. It reduces model complexity and inference overhead through teacher-student distillation, accelerates convergence via heuristic adaptive exploration, and enhances stability in heterogeneous environments through spatio-temporal awareness and a dynamic QoE weighting mechanism, thereby achieving high-performance, lightweight edge offloading deployment.

3. System Model

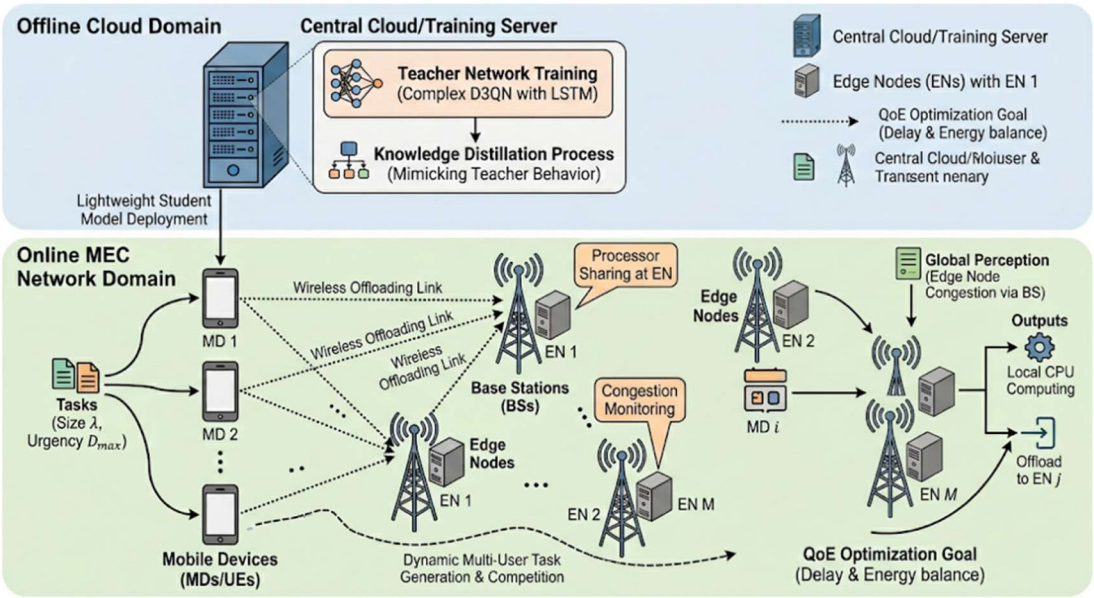

Figure 1. Dynamic MEC system

Consider a multi-time-slot-based Mobile Edge Computing network scenario comprising a set of mobile devices and a set of edge servers . Let the set of MDs be $N = \{1, 2, \ldots, N\}$, The set of ENs is $M = \{1, 2, \ldots, M\}$. The system time is divided into a series of discrete time intervals $t \in \Gamma$, The duration of each time interval is τ .

In each time slot t ,Mobile devices $i \in N$ a new computationally intensive task will be generated with a specific probability $z_i(t)$. The properties of this task are defined by a triplet $\langle \lambda_i(t), \rho_i(t), D_{\max} \rangle$.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202604_12(4).0042

MD_i Once the current system state has been observed, a decision $a_i(t) \in \{0,1,\ldots, M\}$ must be made regarding which tasks $z_i(t)$ to offload. When $a_i(t) = 0$, the task is executed locally; when $a_i(t) = \mathrm{j} > 0$, the task is offloaded to the j th edge server for execution.

3.1 Transmission Model

When MD_i decides to offload a task to an edge server, the task will undergo queueing and wireless transmission.task transmission delay: Once a task is generated, it is first placed in the local transmission queue.Let $\delta_i^T(t)$ denote the queuing delay for Task $z_i(t)$ whilst waiting in the transmission queue for the preceding task to finish.Task transmission delay: Let R_i be the uplink wireless transmission rate of MD_i. The actual physical transmission delay MD for task $z_i(t)$ being uploaded from $D_i^T(t)$ to j is:

$$D_i^T(t) = \frac{\lambda_i(t)}{R_i} \tag{1}$$

Task computation waiting delay: Once a task has been transmitted to the edge server, it enters the computation queue for target EN_j. Let $\delta_{i,j}^E(t)$ denote the queueing delay experienced by the task whilst waiting for computing power to be allocated for execution on the edge server.

Transmission energy consumption: Let the transmission power of MD_i be p_i^T. The transmission energy consumption $E_i^T(t)$ required to transmit the task in full to the edge server is:

$$E_i^T(t) = p_i^T \bullet D_i^T(t) \tag{2}$$

3.2 Computational Model

The computational cost of the task depends entirely on the direction of the unloading decision j.

Local computation

If a task is executed locally $(a_i(t) = 0)$, it is placed in the local computation queue, with a queueing delay set to $\delta_i^L(t)$. Local computation delay: Let f_i^L denote the local CPU processing capacity of MD_i. The pure local processing delay of the task is $D_i^{C,local}(t)$:

$$D_i^{C,local}(t) = \frac{\lambda_i(t) \cdot p_i(t)}{f_i^L} \tag{3}$$

Local computation energy consumption: Let p_i^C be the power consumed by the MD during local computation. The energy consumed by the task during local computation is $E_i^{local}(t)$:

$$E_i^{local}(t) = p_i^C \cdot D_i^{C,local}(t) \tag{4}$$

(2) Edge Computing

If a task is offloaded to edge server $j(a_i(t) = j)$, the server will employ a processor-sharing mechanism to dynamically allocate computing power based on the current load.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202604_12(4).0042

Edge computing latency: Let the total computing capacity of EN_j be f_j^E. Let $K_j(t)$ be the number of MDs currently active on server j; then the effective computing power allocated to task $z_i(t)$ is $\frac{f_j^E}{K_j(t)}$. The actual processing latency of the task at the edge is $D_{i,j}^{C,edge}(t)$:

$$D_{i,j}^{C,edge}(t) = \frac{\lambda_i(t) \cdot p_i(t)}{f_j^E / K_j(t)} \tag{5}$$

Standby power consumption during edge execution: Whilst a task is being executed at the edge, MD_i remains in standby mode awaiting the return of results. Let p_i^{idle} denote the standby power of MD; the idle standby power consumption generated by the terminal during this period is $E_i^{idle}(t)$:

$$E_i^{idle}(t) = p_i^{idle} \cdot (\delta_{i,j}^E(t) + D_{i,j}^{C,edge}(t)) \tag{6}$$

(3) Total delay and energy consumption for task completion

Based on the above derivation, the total delay $D_i^{C,local}(t)$ for task $z_i(t)$ from generation to completion can be expressed uniformly in terms of decision $a_i(t)$ as:

$$D_i^{total}(t) = \begin{cases} \delta_i^L(t) + D_i^{C,local}(t), ifa_i(t) = 0 \\ \delta_i^T(t) + D_i^T(t) + \delta_{i,j}^{C,edge}(t), ifa_i(t) = j > 0 \end{cases} \tag{7}$$

The total energy consumption $E_i^{total}(t)$ of mobile devices during Task $z_i(t)$ is expressed as follows:

$$E_i^{total}(t) = \begin{cases} E_i^{local}(t), ifa_i(t) = 0 \\ E_i^T(t) + E_i^{idle}(t), ifa_i(t) = j > 0 \end{cases} \tag{8}$$

4. Description of the Task Uninstallation Issue

In dynamic and highly uncertain multi-user mobile edge computing (MEC) networks, the offloading decisions of mobile devices (MDs) not only depend on the instantaneous system state but also have a profound impact on its historical evolution and future congestion trends. To minimise heterogeneous energy consumption whilst satisfying strict latency constraints, we rigorously model the task offloading decision-making process as a Markov decision process. This MDP framework is characterised by four core dimensions: the state space, the action space, the reward function, and the long-term optimisation objective.

4.1 State Space

For any time slot t, the environmental state $s_i(t) \in S$ faced by MD_i is defined as a composite vector comprising instantaneous physical observations and spatio-temporal historical information. The specific structure is defined as follows:

$$s_i(t) = \{o_i(t), h_i(t)\} \tag{9}$$

ISSN: 2414-1895 DOI: 10.6919/ICJE.202604_12(4).0042

(1) Observation of instantaneous characteristics $o_i(t)$: This reflects the real-time queuing load and physical constraints of the system at a local level, providing the fundamental data required for single-step decision-making.

$$o_i(t) = \{\lambda_i(t), \tau_i^C(t), \tau_i^T(t), B_i^E(t), \alpha_i(t)\} \tag{10}$$

$\lambda_i(t)$ is the size of the raw task data arriving in the current time slot. $\tau_i^C(t)$:MD is the current remaining processing time deviationof the local compute queue. $\tau_i^T(t)$:MD is the current remaining transmission time deviation of the local transmission queue. $B_i^E(t)$ is the current queue load status vector of the compute queue on the edge server cluster. $\alpha_i(t)$ is the current dynamic power consumption preference status parameter of the MD.

(2) Spatio-temporal historical state: To endow the reinforcement learning agent with the ability to perceive the evolution of the environment's non-stationary dynamics, the sequence of concurrent user counts on edge servers over historical time steps has been incorporated into the state input $h_i(t) = \{m(t-K), \ldots, m(t-1), m(t)\}$.

Here, $\mathrm{m}(t)$ denotes the total number of terminals in an active collaborative computing state on each edge server at time slot t. This sequence of state transitions is then fed into the LSTM to extract features describing the evolution of congestion.

4.2 Action Space

When the MD_i receives a newly arrived task in time slot t, it must perform a deterministic unloading action $a_i(t) \in A$ based on the composite state $s_i(t)$ it observes. The action space of a single agent in the system is defined as the discrete set $A = \{0, 1, 2, \ldots, M\}$.

$a_i(t) = 0$: Execute the local computation strategy. In this case, the agent places the generated computation task directly into the local computation queue for queuing and processing.

$a_i(t) = j (j \in \{1, 2, \ldots, M\})$: Execute the edge offloading strategy. The agent places the task into the local transmission queue and then sends it via a wireless channel link to the selected j th edge server for off-site collaborative computation.

4.3 Reward Function

The design of the reward function determines the ultimate direction of the agent's policy optimisation. To meet the personalised QoE requirements of heterogeneous mobile devices, the model incorporates a dynamic QoE reward mechanism that deeply integrates multi-level energy consumption preferences. By introducing the comprehensive penalty cost $\mathrm{Cos}\, t_i(t)$ for task execution, a dynamic coefficient is used to couple the total execution delay $D_i(t)$ with the normalised total system energy consumption $E_{\text{scaled}}(t)$ within the same dimension:

$$\mathrm{Cos}\, t_i(t) = 2 \times (\alpha_i(t) \cdot D_i(t) + (1 - \alpha_i(t)) \cdot E_{scaled}(t)) \tag{11}$$

To effectively avoid persistent congestion caused by fruitless exploration and to strictly guarantee the requirements of time-sensitive tasks, a hard deadline penalty threshold is introduced; the single-step decision reward $r_i(t)$ is ultimately defined as:

ISSN: 2414-1895 DOI: 10.6919/ICJE.202604_12(4).0042

$$r_i(t) = \begin{cases} R_{base} - \mathrm{Cos}\, t_i(t), success \\ -\mathrm{Cos}\, t_i(t), failure \end{cases} \tag{12}$$

R_{base} represents the base constant reward earned for successfully completing the task within the specified time limit. If the task execution time exceeds the maximum permissible delay $D_{\max}$, resulting in the task failing and being discarded, the agent not only forfeits the R_{base} reward but also incurs the full delay and energy consumption penalties associated with that task; this forces the model to learn to avoid extreme time-out scenarios.

4.4 Problem Formulation

For a given MDP quadruplet, the core task of MD_i-the unloading problem-aims to find an optimal policy π_i^*. This policy essentially establishes a probabilistic mapping from the state space of the environment to the discrete action space. The fundamental optimisation objective of the system is to maximise the expected cumulative discounted QoE reward over the course of long-term interactions:

$$\pi_i^* = \arg\max_{\pi_i} \mathrm{E}\left[\sum_{t=0}^{T} \gamma^t r_i(t) \mid \pi_i\right] \tag{13}$$

$\gamma \in (0,1)$ is a discount factor set by the system, used to precisely adjust the agent's weighting between immediate short-term gains and long-term future returns. Given the high dimensionality of the state space and the extreme dynamic uncertainty of the environment in edge systems, this non-linear stochastic optimisation problem is difficult to solve directly using traditional game theory and optimisation algorithms; this leads to the approximate solution algorithms based on deep reinforcement learning frameworks discussed in subsequent chapters.

5. Distillation-based DRL Unloading Algorithm

To enable efficient decision-making in dynamic MEC environments and address the computational bottlenecks associated with the deployment of traditional deep reinforcement learning models on resource-constrained edge devices, this paper proposes a lightweight D3QN task offloading algorithm based on knowledge distillation. This section will sequentially introduce the internal structure of its neural network, the reinforcement learning decision-making mechanism, and the distillation optimisation strategy tailored for edge devices.

5.1 Neural Network Strategies

The agent's neural network is designed to accurately learn the Q-value mapping from the complex state space of the environment to the discrete action space. To handle temporal congestion features, we have designed a hybrid neural network architecture incorporating LSTMs, which primarily consists of an input layer, an LSTM layer, a FC layer, an A&V layer, and an output layer.

(1) Input layer and feature branching: As the state vector $s_i(t)$ contains data of different dimensions, the network performs branching at the input stage. The instantaneous feature state $o_i(t)$ is fed directly into the fully connected layer, whilst the temporal congestion state $h_i(t)$ is fed into the LSTM layer for processing.

(2) LSTM layer: The LSTM module is specifically designed to extract the dynamic evolution patterns of edge server load. With a time step set to n_lstm_step, the temporal features output by the LSTM layer are concatenated with the instantaneous features $o_i(t)$ to form the complete input features for the downstream network.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202604_12(4).0042

(3) Fully connected layer: The concatenated high-dimensional feature vector is fed into a fully connected layer containing N_L1 neurons, where non-linear feature extraction is performed using the ReLU activation function.

(4) A&V layer and output layer: Following the fully connected layer, the network splits into two paths, one for fitting state values and the other for action values, which are ultimately aggregated in the output layer to produce the Q-values for each action in the current state.

5.2 DRL-based Unloading Method

This paper employs the D3QN algorithm for agent training to overcome the issues of Q-value overestimation and slow convergence that commonly arise in traditional DQN algorithms within complex action spaces.

(1) Dual Q-learning: The algorithm simultaneously maintains an evaluation network and a target network, which share the same network architecture but have independently updated parameters. When calculating the time-differenced target, the evaluation network is responsible for selecting the action with the current maximum Q-value, whilst the target network is responsible for calculating the actual target Q-value for that action:

$$Y_i(t) = r_i(t) + \gamma Q_{target}(s_i(t+1), \arg\max_{a'} Q_{eval}(s_i(t+1), a'; \theta_{eval}); \theta_{target}) \tag{14}$$

This mechanism for decoupling action selection from value estimation significantly reduces positive estimation bias.

(2) Competitive architecture: At the A&V layer, the calculation of the Q-value is explicitly decomposed into the state value function $V(s)$ and the action advantage function $A(s,a)$. The final Q-value synthesis formula at the output layer is:

$$Q(s,a;\theta) = V(s;\theta_V) + (A(s,a;\theta) - \frac{1}{|A|}\sum_{a'} A(s,a';\theta)) \tag{15}$$

The introduction of the mean subtraction operation not only enhances the stability of network training but also enables the agent to perceive the intrinsic value of the state more acutely, thereby accelerating convergence in multi-agent competitive scenarios.

5.3 Optimisation of DRL Models via Distillation

In real-world MEC deployment scenarios, the computational resources and battery capacity of mobile devices are extremely limited. Deploying a standard D3QN model with a large number of neurons directly on the mobile device would result in unacceptable inference latency and excessive energy consumption. To address this, this paper introduces a teacher-student knowledge distillation framework based on offline supervised learning to perform lightweight compression of the DRL model.

(1) Offline Training of the Teacher Model: Firstly, on a central node with sufficient computational power, a deep and extensive teacher network is trained using the complete online DRL interaction process. After thorough exploration across 1,000 episodes, the teacher model possesses offline decision-making knowledge that approximates the global optimum. Subsequently, a large volume of state observations and corresponding target Q-values are exported as an offline dataset.

(2) Proportional compression of the student model: For local deployment on MD, a student network is constructed with the same network topology but with the number of neurons in the hidden layers strictly reduced in proportion. The compression ratio is controlled by the parameter STUDENT_RATIO.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202604_12(4).0042

(3) Knowledge transfer and fitting: The student network no longer interacts with the environment through trial and error, but instead 'clones' the behaviour of the teacher network from the exported offline dataset via a purely supervised learning approach. The loss function for the distillation process is defined as the mean squared error between the student's output Q-values and the teacher's output Q-values:

$$L_{distall}(\theta_S) = \frac{1}{|D|}\sum_{k\in D}(Q_{student}(s_k, a_k; \theta_S) - Q_{teacher}(s_k, a_k; \theta_T))^2 \tag{16}$$

During this optimisation phase, we employ a hard-matching strategy to ensure that the student network fully inherits the multi-peaked value distribution of the teacher model in the high-dimensional state space. This distillation-based optimisation method effectively overcomes the computational constraints of DRL at the edge, achieving a dramatic reduction in both the number of parameters and forward inference latency with virtually no loss in decision-making accuracy.

6. Performance Evaluation

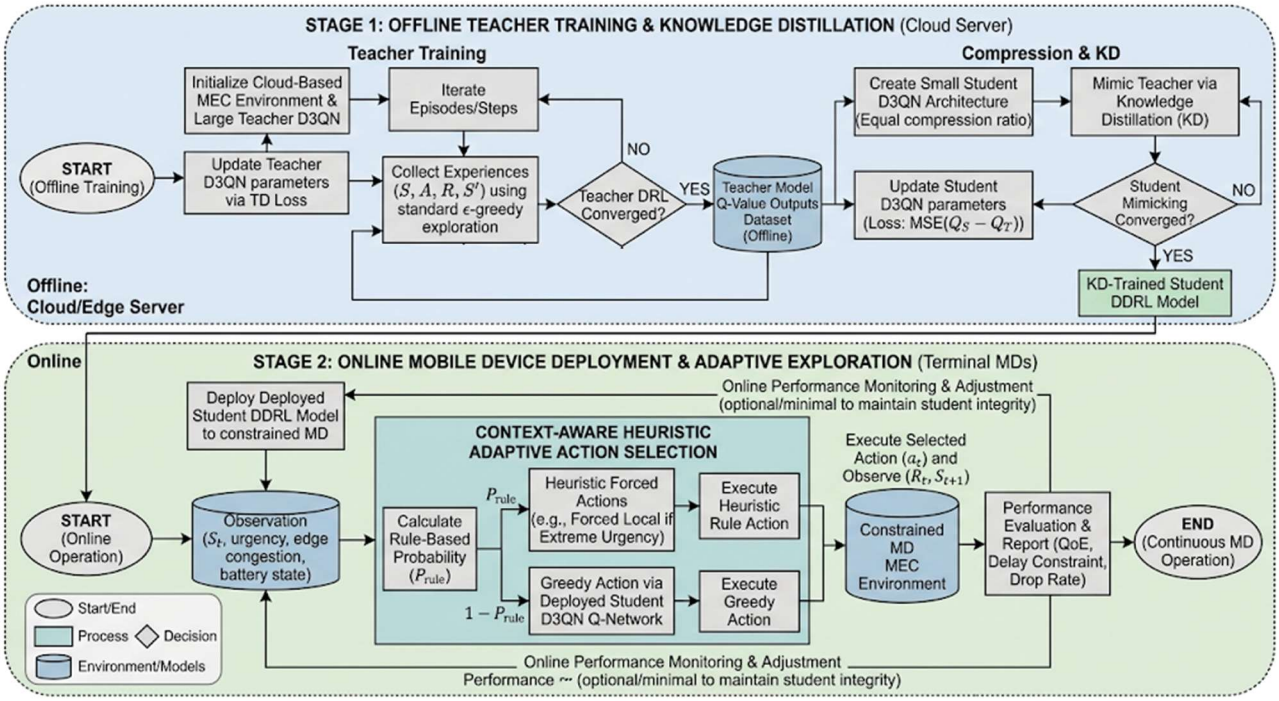

Figure 2. distillation-based Deep Reinforcement Learning Offloading Algorithm

This section will comprehensively evaluate the effectiveness of the knowledge distillation-based lightweight deep reinforcement learning offloading algorithm proposed in this paper through a series of simulation experiments. We will conduct a detailed performance evaluation across three dimensions: algorithm convergence, a comparison of teacher-student model performance, and a comparison with various benchmark algorithms under different system loads.

6.1 Simulation Design

To verify the actual performance of the algorithm in a multi-user dynamic MEC environment, this paper has developed a discrete-time-driven simulation system based on Python and the TensorFlow framework. The system configuration comprises 50 mobile devices (MDs) and 5 edge servers (ENs), with the simulation cycle divided into 100 time slots, each lasting 0.1 seconds. The task arrival probability is set at 0.3, with data volumes fluctuating dynamically between 1 and 7 Mbits, and the maximum tolerable task delay strictly constrained to 10 time slots. Regarding energy consumption

ISSN: 2414-1895 DOI: 10.6919/ICJE.202604_12(4).0042

modelling, dynamic terminal energy preference state parameters are precisely incorporated to simulate the heterogeneous characteristics of devices operating in ultra-power-saving, balanced, and performance modes. To highlight the superiority of the algorithm proposed in this paper, the following five representative benchmark strategies were selected for comprehensive comparison:

(1) All Local: All tasks are processed locally on the MD.

(2) All Offload: All tasks are forcibly offloaded via wireless links to edge servers for execution.

(3) Random Offload: The execution destination of tasks (locally or a randomly selected edge server) is determined by a fully randomised strategy.

(4) DCDRL: A mainstream distributed deep reinforcement learning offloading benchmark algorithm.

(5) QECO: An advanced deep reinforcement learning computation offloading algorithm optimised for QoE.

6.2 Performance Evaluation and Convergence

All evaluation metrics in this section are based on convergence results obtained from training over 1,000 episodes. To ensure a fair comparison, all algorithms were run under the same random seed conditions.

(1) Algorithm Convergence and Hyperparameter Analysis

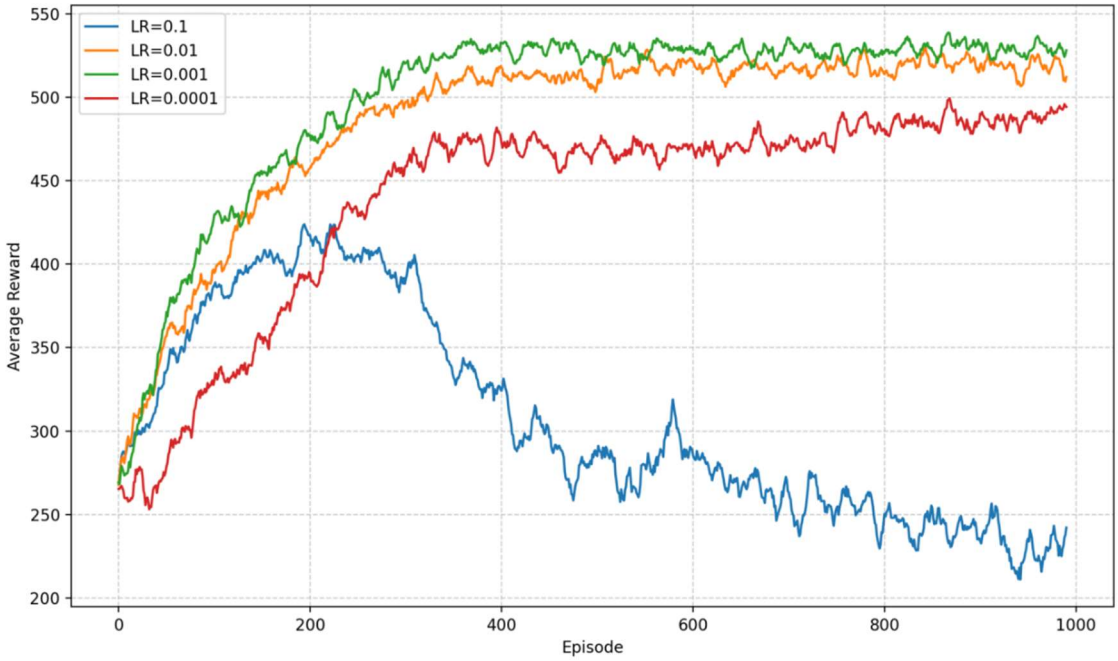

Figure 3. Hyperparameter ablation experiments for the teacher model versus the average QoE

Prior to performing teacher-student distillation, we first analyse the convergence performance of the teacher model (Teacher) and its sensitivity to hyperparameters. A reasonable design of hyperparameters is crucial for overcoming oscillations in DRL within complex, non-stationary MEC environments.

The experimental results shown in Figure 3 indicate that, as the number of iterations increases, the teacher model's QoE reward rises rapidly at first and then gradually stabilises through continuous interaction with the environment. This demonstrates that the D3QN algorithm designed in this paper, which incorporates an LSTM module, is capable of effectively learning the optimal offloading strategy.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202604_12(4).0042

Table 1. Comparison of student model performance across different training epochs

DISTILL_STEPS	Episodes	AvgDelay	AvgEnergy	AvgQoE	Drop	AvgInferTime	Rank
500	50	7.674 ± 0.123	393.159 ± 10.811	466.862 ± 27.194	359.600 ± 31.477	0.000328 ± 0.000123	5
1000	50	7.329 ± 0.111	404.754 ± 9.465	523.489 ± 21.006	300.300 ± 27.080	0.000503 ± 0.000214	6
2000	50	6.347 ± 0.111	397.576 ± 10.313	721.358 ± 18.528	161.380 ± 19.505	0.000591 ± 0.000211	4
5000	50	5.818 ± 0.101	397.030 ± 13.169	810.901 ± 17.735	90.740 ± 16.453	0.000321 ± 0.000121	3
10000	50	5.710 ± 0.112	394.251 ± 14.491	824.676 ± 14.327	74.280 ± 12.408	0.000322 ± 0.000116	1
20000	50	5.713 ± 0.098	397.940 ± 12.751	831.248 ± 15.248	73.740 ± 12.764	0.000327 ± 0.000123	2

As shown in Table 1, when the number of distillation steps was increased from 500 to 10,000, the performance metrics of the student model showed significant improvement. At 10,000 steps, the model achieved its best overall ranking; furthermore, continuing to increase the number of training steps resulted in performance reaching a plateau, with no significant gains observed. Consequently, the system has established 10,000 steps as the optimal knowledge fitting cycle.

Table 2. Comparison of the performance of student models at different compression rates

STUDENT_RATIO	Episodes	AvgDelay	AvgEnergy	AvgQoE	Drop	Params	AvgInferTime
0.1	50	7.618 ± 0.112	425.627 ± 9.950	472.657 ± 17.132	349.020 ± 27.789	230.0 ± 0.0	0.000323 ± 0.000115
0.2	50	6.679 ± 0.110	398.168 ± 11.020	642.356 ± 19.331	218.060 ± 21.528	542.0 ± 0.0	0.000317 ± 0.000118
0.4	50	5.818 ± 0.101	397.030 ± 13.169	810.901 ± 17.735	90.740 ± 16.453	1454.0 ± 0.0	0.000321 ± 0.000121
0.6	50	5.725 ± 0.114	397.842 ± 13.551	830.636 ± 18.809	75.640 ± 13.337	2750.0 ± 0.0	0.000319 ± 0.000119
0.8	50	5.709 ± 0.094	398.665 ± 15.233	831.584 ± 16.715	70.940 ± 12.273	4430.0 ± 0.0	0.000320 ± 0.000114
1	50	5.750 ± 0.103	402.844 ± 11.853	827.500 ± 19.294	74.000 ± 12.395	6494.0 ± 0.0	0.000342 ± 0.000114

Table 2 illustrates the trade-off between model size and decision accuracy. When the network compression ratio STUDENT_RATIO is set to 0.4, the Student network not only significantly reduces the number of parameters from 6,494 to 1,454, but also successfully maintains an average QoE of up to 810.9. Compared to models with compression ratios of 0.6 or 0.8, a compression ratio of 0.4 achieves the ultimate in lightweight design whilst avoiding a precipitous decline in performance, perfectly meeting the requirements for edge deployment.

(2) Performance Comparison of Lightweight Teacher and Student Models

Using the optimal parameters described above, having obtained a teacher model with high-quality decision-making capabilities, this paper transfers these capabilities to a significantly simplified student model (Student) via an offline knowledge distillation mechanism. We have conducted a comprehensive comparison of the models' performance before and after distillation.

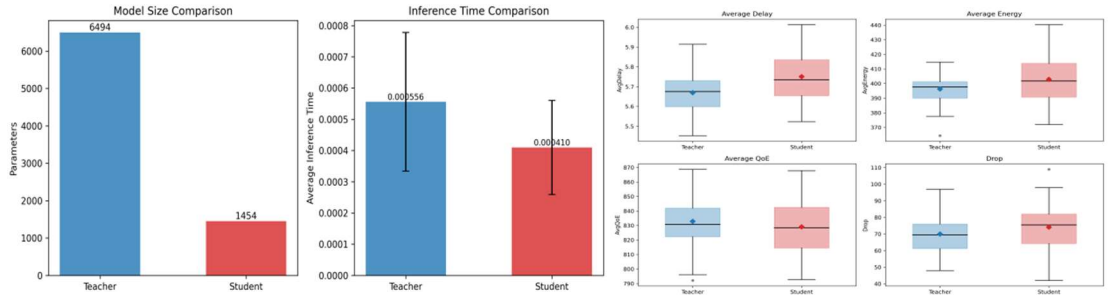

Figure 4. Performance comparison between the teacher model and the student model in terms of average latency, average power consumption and packet loss rate

ISSN: 2414-1895 DOI: 10.6919/ICJE.202604_12(4).0042

Table 3. Comparison of network complexity and inference latency between teacher and student models

Model	Episodes	AvgDelay	AvgEnergy	AvgQoE	Drop	Params	AvgInferTime	ParamReduction(%)	InferTimeChange(%)
Student	50	5.750 ± 0.106	402.872 ± 15.798	829.074 ± 17.678	74.100 ± 13.910	1454	0.000410 ± 0.000151	77.61	-26.32
Teacher	50	5.669 ± 0.102	396.299 ± 9.600	832.856 ± 17.795	69.960 ± 11.414	6494	0.000556 ± 0.000222	0	0

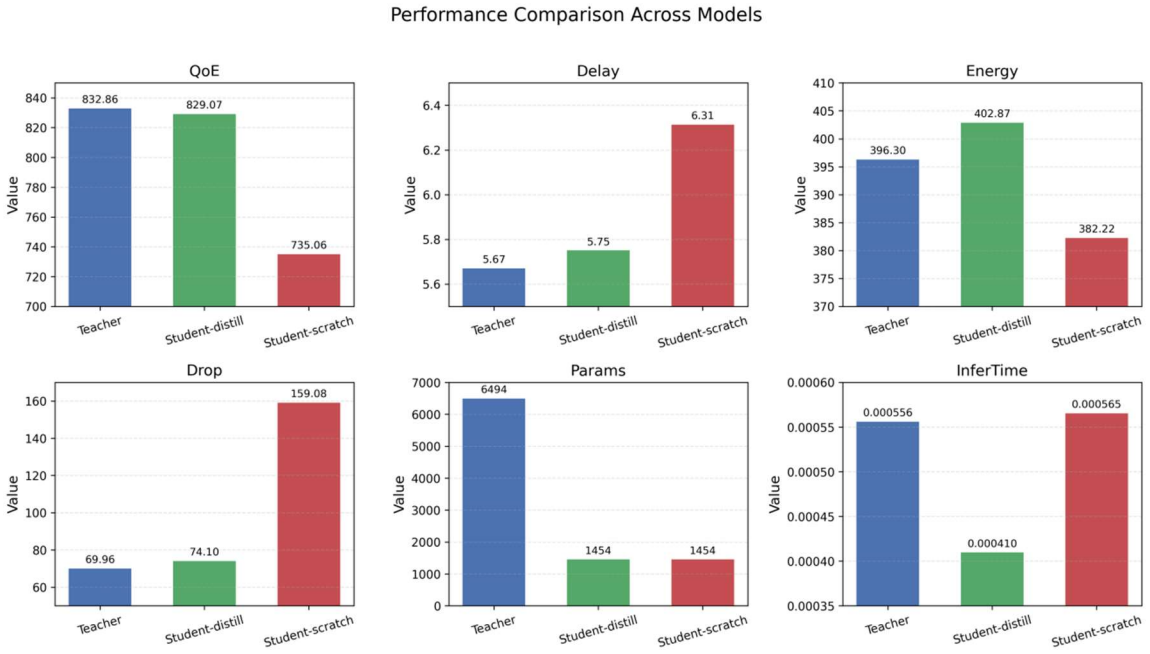

Figure 5. Performance comparison between the teacher model and student models with and without distillation

The comparative data in Table 3 ,Figure 4 and Figure 5 provide compelling evidence that the knowledge distillation framework demonstrates exceptional transfer performance. Decision accuracy remains intact: the distilled student model almost perfectly replicates the performance of the large teacher model in terms of average latency (5.75 vs 5.67) and system QoE (829.07 vs 832.86). Computational overhead plummets: The number of parameters in the student model was reduced by as much as 77.61%, resulting in a 26.32% decrease in the average forward inference time per action decision. Comparison with the disadvantages of training from scratch: It is worth noting that if a network of equivalent lightness is allowed to interact directly with a complex environment, it is highly prone to getting stuck in local optima, resulting in a task dropout rate as high as 159.08 and a significantly deteriorated QoE. This serves as reverse proof of the absolute necessity of the technical approach of 'intensive training in the cloud first, followed by lightweight distillation at the edge'.

To further evaluate the robustness of our algorithm, this section simulates system load conditions ranging from low to high by varying the number of tasks arriving in the system, and compares the results with those of five benchmark algorithms.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202604_12(4).0042

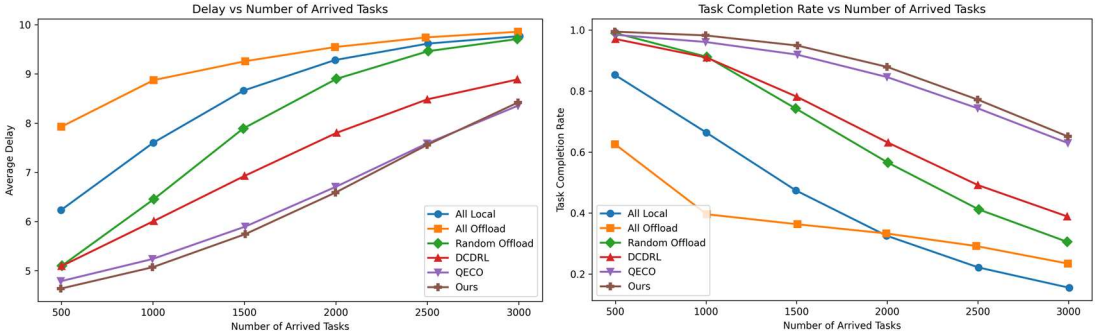

Figure 6. Comparison of average task latency and success rates for different algorithms under different workloads

Figure 6 shows that, as the task load surges, the system's communication and computational resources become subject to intense competition, resulting in a general spike in latency for the baseline algorithm and a sharp drop in task completion rates. However, the algorithm presented in this paper (Ours) consistently maintained the lowest average latency and the highest task completion rate across all load gradients. This is attributable to the LSTM module's ability to accurately predict temporal congestion within the state space, effectively avoiding the 'herd queuing effect' caused by multiple devices blindly competing for the same node.

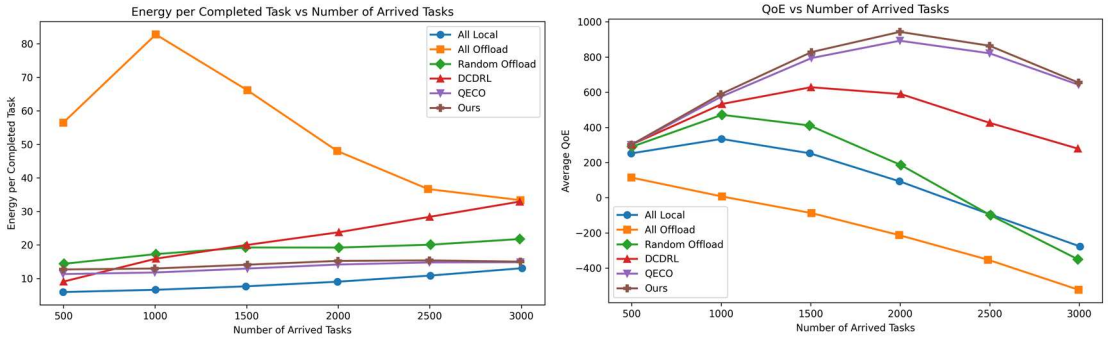

Figure 7. Comparison of average energy consumption per task and average QoE for different algorithms under different workloads

Figure 7(left) summarises the average energy consumption per unit for different algorithms when completing tasks. It can be seen that the All Local method, as it does not involve wireless transmission, demonstrates a certain advantage in terms of absolute energy consumption; however, due to its limitation by local computing power, it is often accompanied by an extremely high task timeout and discard rate, resulting in poor overall service quality. Compared to algorithms such as DCDRL and QECO, the method proposed in this paper (Ours) employs a refined dynamic energy reward weighting scheme to adaptively adjust strategies based on device power consumption patterns, ultimately achieving the optimal balance between latency sensitivity and system energy consumption.The QoE metric is a comprehensive, global final measure that combines task completion rate and penalty cost (weighted by latency and energy consumption). Figure 7(right) clearly demonstrates that, under various task load conditions-particularly under extreme high-congestion loads-the algorithm proposed in this paper achieves the highest or near-optimal system QoE. This demonstrates that a deep reinforcement learning architecture combining adaptive heuristic exploration with knowledge distillation can endow MEC offloading strategies with exceptional robustness and high practical deployment value.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202604_12(4).0042

7. Conclusion

Addressing the core challenges in mobile edge computing-namely the high complexity and significant inference latency of traditional deep reinforcement learning algorithms, which make them difficult to deploy directly on resource-constrained end devices-this paper proposes a lightweight D3QN computation offloading framework based on knowledge distillation and heuristic adaptive exploration. This framework not only introduces context-aware soft and hard constraint strategies during the action exploration phase to reduce packet loss rates in the early stages of training, but also unifies the QoE optimisation objectives across heterogeneous end devices through the design of dynamic energy consumption weights.

Through extensive simulation experiments, it has been verified that, at an optimal network compression ratio of 0.4, the offline knowledge distillation mechanism adopted in this paper achieves a dramatic reduction of up to 77.61% in the number of parameters of the student model, which directly contributes to a 26.32% reduction in the average forward inference latency per decision. The distilled lightweight student network almost perfectly replicates the decision-making capabilities of the massive teacher model. Experiments demonstrate that the system achieves an average QoE of 829.07, which is virtually indistinguishable from the performance of the large teacher model. Furthermore, compared to models of equivalent size that were trained from scratch without distillation, the task dropout rate is significantly reduced by approximately 53.4%, effectively avoiding the local optimum trap.

Although the method presented in this paper has achieved significant results in optimising offloading and model lightweighting within MEC systems, there remain areas for further exploration and development in future research. The current system model focuses on static or low-speed mobility scenarios; future work will consider highly dynamic scenarios such as vehicle-to-everything (V2X) networks, incorporating predictions of users' spatial mobility trajectories and time-varying fading channel models to further enhance the decision-making consistency of agents during frequent handover transitions. Current knowledge distillation relies on datasets collected offline. Future research could explore 'online knowledge distillation' or 'federated distillation' techniques, enabling student models to undergo incremental online fine-tuning and updates in response to real-time environmental changes, whilst safeguarding terminal privacy.

References

[1] M. Patel et al., "Mobile-edge computing introductory technical white paper," White Paper, Mobile-Edge Comput. (MEC), vol. 29, pp. 854-864, Sep. 2014.

[2] S. Lu, S. Liu, Y. Zhu, W. Liang, K. Li, and Y. Lu, "A DRL-based decentralized computation offloading method: An example of an intelligent manufacturing scenario," IEEE Trans. Cogn. Commun. Netw., vol. 9, no. 4, pp. 1042-1055, Aug. 2023.

[3] L. Ale, S. A. King, N. Zhang, A. R. Sattar, and J. Skandaramayan, "D3PG: Dirichlet DDPG for task partitioning and offloading with constrained hybrid action space in mobile-edge computing," IEEE Internet Things J., vol. 9, no. 19, pp. 19260-19272, Oct. 2022.

[4] L. Ale, N. Zhang, X. Fang, X. Chen, S. Wu, and L. Li, "Delay-aware and energy-efficient computation offloading in mobile-edge computing using deep reinforcement learning," IEEE Trans. Cogn. Commun. Netw., vol. 7, no. 3, pp. 881-892, Sep. 2021.

[5] J. Li, Q. Jiang, V. C. M. Leung, Z. Ma, and K. K. Abrokwa, "Deep-reinforcement-learning-based joint optimization of task migration and resource allocation for mobile-edge computing," IEEE Internet Things J., vol. 12, no. 13, pp. 24428-24441, 1 Jul. 2025.

[6] X. Qiu, W. Zhang, W. Chen, and Z. Zheng, "Distributed and collective deep reinforcement learning for computation offloading: A practical perspective," IEEE Trans. Parallel Distrib. Syst., vol. 32, no. 5, pp. 1085-1101, May 2021.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202604_12(4).0042

[7] N. Zhao, Z. Ye, Y. Pei, Y.-C. Liang, and D. Niyato, "Multi-agent deep reinforcement learning for task offloading in UAV-assisted mobile edge computing," IEEE Trans. Wireless Commun., vol. 21, no. 9, pp. 6949-6962, Sep. 2022.

[8] L. Ale, N. Zhang, X. Fang, X. Chen, S. Wu, and L. Li, "Delay-aware and energy-efficient computation offloading in mobile-edge computing using deep reinforcement learning," IEEE Trans. Cogn. Commun. Netw., vol. 7, no. 3, pp. 881-892, Sep. 2021.

[9] S. Lu, S. Liu, Y. Zhu, W. Liang, K. Li, and Y. Lu, "A DRL-based decentralized computation offloading method: An example of an intelligent manufacturing scenario," IEEE Trans. Cogn. Commun. Netw., vol. 9, no. 4, pp. 1042-1055, Aug. 2023.

[10] L. Ale, S. A. King, N. Zhang, A. R. Sattar, and J. Skandaramayan, "D3PG: Dirichlet DDPG for task partitioning and offloading with constrained hybrid action space in mobile-edge computing," IEEE Internet Things J., vol. 9, no. 19, pp. 19260-19272, Oct. 2022.

[11] J. Li, Q. Jiang, V. C. M. Leung, Z. Ma, and K. K. Abrokwa, "Deep-reinforcement-learning-based joint optimization of task migration and resource allocation for mobile-edge computing," IEEE Internet Things J., vol. 12, no. 13, pp. 24428-24441, 1 Jul. 2025.

[12] S. Chouikhi, M. Essgehir, and L. Merghem-Boulahia, "Energy-efficient computation offloading based on multiagent deep reinforcement learning for industrial Internet of Things systems," IEEE Internet Things J., vol. 11, no. 7, pp. 12228-12239, Apr. 2024.

[13] N. Zhao, Z. Ye, Y. Pei, Y.-C. Liang, and D. Niyato, "Multi-agent deep reinforcement learning for task offloading in UAV-assisted mobile edge computing," IEEE Trans. Wireless Commun., vol. 21, no. 9, pp. 6949-6962, Sep. 2022.

[14] X. Qiu, W. Zhang, W. Chen, and Z. Zheng, "Distributed and collective deep reinforcement learning for computation offloading: A practical perspective," IEEE Trans. Parallel Distrib. Syst., vol. 32, no. 5, pp. 1085-1101, May 2021.

[15] J. Hao, L. Wang, M. Odiathevar, W. K. G. Seah, G. Xu, B. Huang, and Y. Gao, "Prune-based deep reinforcement learning offloading algorithm for mobile edge computing," IEEE Trans. Cogn. Commun. Netw., vol. 12, pp. 2876-2889, 2026.