Deep Learning-Based Dynamic Computation Task Offloading for Mobile Edge Computing Networks

Shicheng Yang 1, Gongwei Lee 2 and Liang Huang 2,*

1 The College of Information Engineering, Zhejiang University of Technology, Hangzhou 310023, China; scyang@zjut.edu.cn
2 The College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310023, China; gwlee@zjut.edu.cn
* Correspondence: lianghuang@zjut.edu.cn

Abstract: This paper investigates the computation offloading problem in mobile edge computing (MEC) networks with dynamic weighted tasks. We aim to minimize the system utility of the MEC network by jointly optimizing the offloading decision and bandwidth allocation problems. The optimization of joint offloading decisions and bandwidth allocation is formulated as a mixed-integer programming (MIP) problem. In general, the problem can be efficiently generated by deep learning-based algorithms for offloading decisions and then solved by using traditional optimization methods. However, these methods are weakly adaptive to new environments and require a large number of training samples to retrain the deep learning model once the environment changes. To overcome this weakness, in this paper, we propose a deep supervised learning-based computational offloading (DSLO) algorithm for dynamic computational tasks in MEC networks. We further introduce batch normalization to speed up the model convergence process and improve the robustness of the model. Numerical results show that DSLO only requires a few training samples and can quickly adapt to new MEC scenarios. Specifically, it can achieve 99% normalized system utility by using only four training samples per MEC scenario. Therefore, DSLO enables the fast deployment of computation offloading algorithms in future MEC networks.

Keywords: mobile-edge computing; deep learning; computation offloading

1. Introduction

The rapid development of wireless communication technologies has driven the emergence of more and more latency-sensitive and resource-intensive applications and services, such as augmented reality, voice recognition, image recognition, and mobile health care. These mobile applications have dramatically increased the demand for computing and storage resources for wireless devices, which are typically provided by cloud servers. However, because cloud servers are usually deployed over long distances, high transmission latency results when tasks are offloaded to the cloud. Therefore, mobile edge computing (MEC) [1] is considered an effective solution to this problem. The fundamental principle of MEC is that by deploying hosts with cloud computing capabilities at the edge of the wireless network, wireless devices can offload tasks to edge servers, solve the problem of insufficient computational resources, reduce the delay of computing tasks [2,3], and save energy consumption [4,5].

As one of the key technologies of MEC, computation offloading solves the problem of resource limitation and low processing efficiency by offloading computation-intensive tasks to MEC hosts at the network edge. On the one hand, the task offloading process is affected by different factors, e.g., computational resources, wireless communication quality, etc. On the other hand, if computational tasks are aggressively offloaded to the edge servers, the bandwidth in the network system will be occupied. The uplink wireless channel will be severely congested, which significantly increases the transmission delay.
of computational tasks. Therefore, how to make the best decision is the critical issue of edge offloading. In general, computation offloading and resource allocation are jointly formulated as nonlinear programming problems with mixed-integer programming. Most of the existing solutions are based on heuristic or approximation algorithms [6,7]. However, these solutions rely on expert knowledge and exact mathematical models, which often require a fresh start to adjust the mathematical models once the MEC environment changes, resulting in inefficient offloading decisions. The computational complexity of such methods is relatively high. Therefore, it remains a challenge to design an MEC network with a low-complexity algorithm that can be applied to time-varying environments.

Recently, deep learning has been widely used to optimize offloading decisions [8]. Since the nature of a neural network is a black-box model that does not require precise expert knowledge and accurate mathematical models, deep learning provides a good idea for the above challenges. Deep reinforcement learning learns to solve complex problems, such as video games, chess activities, and intelligent robot control, through continuous trial and error. Currently, computational offloading based on deep Q-learning (DQL) [9] is most widely studied, which discretizes the state/action space and optimizes computational offloading decisions and system resource optimization through online learning. However, the discretization of continuous variables limits the performance of DQL and is not suitable for dealing with high-dimensional action spaces [10]. In addition, most of these methods are based on static interactive MEC environments with sufficient training samples to train neural networks. Once the MEC scene changes, it is difficult to collect enough training samples when converging to a new scene. However, considering the dynamic nature of the offloading task, how to converge to a new scene quickly with small samples is still a problem worth investigating.

There are some studies on dynamic MEC networks in the literature. Huang et al. [11] constructed dynamic MEC scenarios by time-varying wireless channel gains and weight factors of computational tasks and proposed a meta-learning-based computation offloading (MELO) algorithm to obtain a general model that can be quickly adapted to new task scenarios by a small number of training samples in MEC scenarios. Wang et al. [12] constructed dynamic task scenarios by different topologies and proposed a meta-reinforcement learning-based approach to improve the training efficiency and reduce the dependence on samples. Unfortunately, this research ignores the energy consumption of the system to achieve low latency. Moreover, the training process of the computational offloading algorithm based on meta-learning is tedious. A new model needs to be created additionally for computing the gradient at each training.

In this paper, we consider a multi-scenario MEC network with multiple WDs and a single edge server, where each MEC scenario is characterized by a specific task weighting factor. Each WD in the network makes real-time decisions on whether to offload its computing tasks to the edge server or execute it locally. By jointly optimizing WDs’ computing offload and bandwidth allocation, we propose a Deep-Supervised-Learning-based computation Offloading (DSLO) algorithm to learn to minimize the network utility. The main contributions of this paper are as follows:

- We model the system utility of the MEC network with prioritized computing tasks such as the weighted sum of energy consumption and delay cost. To minimize the system utility, we decompose the joint optimization problem into the offloading decision subproblem and the transmission bandwidth allocation subproblem, which are further solved via deep learning and optimization methods, respectively.

- We propose the DSLO framework to learn from a few training samples to optimize the offloading decision actions. We introduce the batch normalize (BN) layer in CNN/DNN network structure to accelerate the convergence process. It can efficiently learn the mapping from the workload and weight factors to computational offloading.

- Simulation results show that DSLO-CNN can generate near-optimal offloading decisions and outperforms DSLO-DNN under MEC scenarios training datasets of different sizes. Significantly, the normalized system utility of the DSLO-CNN algorithm
achieves a median value of 96% when only 10% of MEC scenarios are included in the training dataset with two training samples per MEC scenario. In new MEC scenarios, DSLO-CNN converges faster than MELO.

The remainder of this paper is organized as follows. The related work is introduced in Section 2. In Section 3, we present the system model and problem formulation. We propose the DSLO algorithm in Section 4. Numerical results are presented in Section 5, and a conclusion is provided in Section 6. Before leaving this section, those important notations and abbreviations used throughout this paper are respectively summarized in Abbreviations.

2. Related Work

So far, considerable research efforts have been devoted to the offloading scheme design for MEC networks. The computing offloading optimization problem can be modeled by three elements [13], i.e., state parameters, decision actions, and system utility. The current state of research on computational offloading can be divided into static MEC network computational offloading and dynamic MEC network computational offloading, depending on the number of state parameters.

Under static MEC network computational offloading [14–16], the offloading decision is optimized for a unique category of state parameters, such as wireless channel gain or task computation amount. You et al. [17] considered a single-user system with wireless channel gain as the state parameter and optimized the performance of local computing and offloading computing under the constraints of energy harvesting and processing latency. Tran et al. [18] proposed a low-complexity heuristic offloading framework with the wireless channel gain as the state parameter. Bi et al. [19] considered a cache-assisted MEC system, where the server can selectively cache the previously generated programs for future reuse. Resorting to the deep neural network (DNN), Huang et al. [20] proposed a deep reinforcement learning-based computational offloading framework (DROO), which maximizes the system computation rate by jointly optimizing computation offloading and resource allocation according to the time-varying channel gains. Considering the static interaction environment of MEC, we believe that there are sufficient training samples under the static MEC network. Most of the offloading algorithms based on deep learning are based on DNN.

Considering the dynamic properties of wireless applications, MEC networks make computational offloading decisions by evaluating multiple classes of state parameters jointly. Min et al. [21] considered a dynamic MEC network consisting of a time-varying radio link transmission rate and investigated the computational offloading of IoT devices with energy harvesting in a dynamic MEC network. They proposed an offloading scheme based on deep reinforcement learning, which uses a convolutional neural network (CNN) to compress the state space to speed up the convergence process of the algorithm. Huang et al. [11] considered a dynamic computing task scenario consisting of different weight priority coefficients and proposed a meta-learning-based computing offloading (MELO) framework. MELO can efficiently adapt to a new MEC scenario and minimize the total system delay. Qu et al. [22] considered a dynamic computing task scenario composed of different computing power and bandwidth and proposed a computing offloading framework based on meta-reinforcement learning. The algorithm can quickly adapt to complex and dynamic environments and can be used to improve the robustness of task offloading decisions in IoT environments. Wang et al. [12] constructed dynamic computing task scenarios based on different network topologies, modeled the computational tasks as directed acyclic graphs (DAGs), and designed a sequence-to-sequence (seq2seq) neural network model to generate offloading strategies that can quickly adapt to new environments. Chen et al. [23] considered an MEC system composed of a random task arrival model, where the state parameters include the size of the input data, the maximum tolerable delay, the number of CPU cycles, and the time slot of the task arrival. They designed a temporal feature extraction network composed of one-dimensional convolutional (Conv1D) residual
blocks and a long short-term memory (LSTM) network to solve the joint optimization problem of computational offloading and resource allocation. For dynamic MEC network computational offloading, the state parameters are too complicated to optimize the decision actions via classical optimizations, where computation offloading and resource allocation are jointly formulated as a mixed-integer nonlinear programming problem [24]. Considering the dynamics and complexity of the actual MEC environment, we assume that the modeled dynamic MEC scenario contains only a small number of training samples.

Recent works [12,13,20,23,25] resort to the deep learning methods, which utilize deep neural networks [26] to learn from data samples and generate the optimal mapping from state space to action space. However, they require a large number of data samples to train the neural network and cannot converge to the optimal offloading performance when lacking training samples. Achieving efficient deep learning-based computing offloading under a few training samples is challenging in dynamic MEC networks. To the best of our knowledge, there are relatively few studies investigating binary offloading designs based on dynamic MEC networks. Unlike the literature [11], the objective of the study in this paper also includes energy consumption and weight factors as input data together with the neural network and uses CNN networks for offloading prediction. Therefore, our work focuses on designing binary offloading strategies based on time-varying workloads and different computational tasks, aiming to minimize the weighted sum of energy consumption and time delay.

3. System Model

We consider an MEC network composed of one edge server, one wireless access point (AP), and $N$ WDs, denoted as $\mathcal{N} = \{1, 2, \ldots, N\}$, as shown in Figure 1. The AP and the edge server are connected by optical fiber, whose transmission delay can be ignored. Each WD has one task that needs to be processed locally or be offloaded to the edge server through the AP. Without loss of generality, each WD needs to execute one prioritized computing task with a specific task weight. Denote $w_n \in \mathcal{W}$ as the task weight of $WD_n$, whose value changes in the countable set $\mathcal{W}$ depending on the specific task category. An MEC task scenario is considered different from another one if at least one task’s weight is different.

Each WD can decide whether to offload its tasks to edge servers or compute locally by the device itself. Denote $a = \{a_n \in \{0, 1\} | n \in \mathcal{N}\}$ as the offloading decision. Specifically, $a_n = 1$ indicates that the $WD_n$ offloads its task to the edge server and $a_n = 0$ means its task is processed locally.

![Figure 1. System model of an MEC network with multiple WDs.](image)

3.1. Energy Consumption

When the offloading decision is present, we study the resource allocation optimization problem of the MEC network and model dynamic computation tasks via task weights. We
set a tuple \((d_n, \gamma_n)\) to represent \(WD_n\)’s task, for \(n \in \mathcal{N}\). Specifically, \(d_n\) is the workload of \(WD_n\), and \(\gamma_n\) is the required number of CPU cycles to complete the task. When \(WD_n\) offloads its total task to the edge server, the energy consumption includes data transmission and task computation, which can be expressed as

\[
E^c_n = E^t_n + \alpha d_n,
\]

where \(E^c_n\) is the transmission energy consumed by \(WD_n\) for uploading its workload to the edge server, which is a linear function of workload \(d_n\), \(\alpha d_n\) is the task computation energy consumption, and \(\alpha\) is the factor of the computation energy consumption at the edge server.

In the above formula, superscripts \(c\) and \(t\) stand for “edge computing” and “transmission”, respectively. When \(\alpha = 0\), we only consider the transmission energy consumption of WDs. Here, we ignore the energy consumption and delay of the edge server when transmitting the computation results back to \(WD_n\). Similar to the literature [27], we assume that the computed result size is much smaller than the input data size.

When \(WD_n\) executes its task locally, we define \(e^l_n\) as the local energy consumption per data bit of \(WD_n\), where the superscript indicates the offloading method. Here, the superscript is \(l\) denotes “local computing”. So, \(WD_n\)’s energy consumption for executing its total task locally can be given by:

\[
E^l_n = d_ne^l_n.
\]

By evaluating the energy consumption of both computation offloading and local execution under the offloading decision \(a_n\), we get the total energy consumption of \(WD_n\), as

\[
E_n = E^c_n a_n + E^l_n (1 - a_n).
\]

3.2. Time Delay

In addition to the total energy consumption, another major factor affecting the system’s overall efficiency is the time delay. It includes the transmission delay when WDs offload tasks to MEC servers and the processing delay of MEC servers, and WDs’ local execution latency.

Considering the delay in computation offloading, when \(WD_n\)’s task is offloaded to the edge server, we use \(c_n\) to denote the allocated bandwidth to \(WD_n\) for task transmission. Then, the time latency \(T^c_n\) can be expressed as

\[
T^c_n = \frac{d_n}{c_n} + \frac{\gamma_n}{f_n}.
\]

The transmission delay of \(WD_n\) is defined as \(d_n/c_n\), and the computational delay of \(WD_n\) is defined as \(\gamma_n/f_n\), where \(f_n\) is defined as the edge processing rate.

Meanwhile, the local execution delay for \(WD_n\) to execute its task is given by

\[
T^l_n = d_n t^l_n,
\]

where \(t^l_n\) is the local execution delay per data bit of \(WD_n\).

Therefore, the total time delay is a combination of delay under both computation offloading and local execution, which can be given as

\[
T_n = T^c_n a_n + T^l_n (1 - a_n).
\]

3.3. Problem Formulation

We formulate the joint computation offloading and bandwidth allocation for the MEC network as an mixed integer optimization problem. We introduce the system utility \(Q(d, a, w, c)\) to represent the sum cost of the MEC network as:

\[
Q(d, a, w, c) = \sum_{n=1}^{N} E_n w_n + \beta \max\{T_n w_n | \forall n \in \mathcal{N}\},
\]
where $d = \{d_n \mid n \in N\}$, $a = \{a_n \mid n \in N\}$, $c = \{c_n \mid n \in N\}$, and $\beta$ denotes the weight of energy consumption and task completion. Considering dynamic MEC scenarios with different computation task $d$ and weights $w$, we aim to optimize offloading decisions $a$ and bandwidth allocation $c$ to minimize the system utility $Q(d, a, w, c)$. The optimization problem can be defined as (P1):

$$(P1) : Q^*(d, w) = \min_{a,c} Q(d, a, w, c)$$

subject to

1. $c_n \geq 0, \forall n \in N$, (8)
2. $\sum_{n=1}^{N} c_n \leq C$, (9)
3. $w_n \in W$, (10)
4. $a_n \in \{0, 1\}$. (11)

Here, the constraints (8) and (9) mean that the bandwidth allocated to all WDs $c_n$ is non-negative, and $C$ limits the total bandwidth. Since the constraints on the offloading decision and bandwidth allocations are decoupled, we decompose problem (P1) into an offloading decision subproblem and a bandwidth allocation subproblem (P2), as illustrated in Figure 2.

![Figure 2. The two-level optimization structure for solving the problem (P1).](image)

Considering the continuously changing data size $d_n$ and the occasionally changing task weight $w_n$, the offloading decision subproblem aims to find an offloading strategy $\pi$ to effectively produce the optimal offloading decision $a^*$, as:

$$\pi : d_n \mapsto a_n^*.$$ (12)

The subproblem (P2) aims to optimize the bandwidth allocation $c$ and is expressed as

$$(P2) : Q^*(d, w) = \min_{c} Q(d, a, w, c)$$

subject to (9) and (8).

Given $a^*$, the subproblem (P2) is convex and can be efficiently solved by the standardized CVX tool package.

We assume that changes in workload $d$ are faster than changes in task weights $w$ due to the heterogeneity of mobile devices. The MEC scenario occasionally changes when the task weight changes. Some existing deep learning-based computational offloading methods are based on static network scenarios and require a large amount of training data samples. Once the MEC scenario changes, the deep learning model is difficult to adapt to a new scenario. In the next section, we propose an algorithm based on deep learning that uses few training samples and can quickly adapt to new MEC scenarios.
4. Deep Supervised Learning-Based Offloading Algorithm

In this section, we propose a DSLO algorithm to achieve the optimal binary offloading decision. The algorithm takes advantage of the translation invariance of the convolutional neural network to capture the local features of input data and accelerate the convergence of the offloading process. For dynamic MEC task scenarios, DSLO can quickly adapt to the new workload $d$ and task weights $w$. The overall pipeline of the DSLO algorithm is illustrated in Figure 3.

**Figure 3.** The pipeline of DSLO training.

We consider a dynamic MEC task scenario, where the task weight $w$ of the scenario is variable and the task workloads $d$ can be changed independently. The dataset contains $\mathcal{I} = \{1, 2, \ldots, L\}$ different MEC task scenarios $\{\Psi_i \mid i \in \mathcal{I}\}$, where each MEC task scenario $\Psi_i$ contains $K = \{1, 2, \ldots, K\}$ data samples, denoted as $\Psi_i = \{(d_k, w_k, a_k) \mid k \in K\}$. Here, each data sample $(d_k, w_k, a_k)$ is a combination of workload, weight factor, and optimal offloading decision. The dataset is further randomly split into $\mathcal{I}_{\text{train}}$ for the training phase and $\mathcal{I}_{\text{test}}$ for the testing phase. Correspondingly, we denote $K_{\text{train}}$ and $K_{\text{test}}$ as the index set of data samples for training and testing under each MEC scenario.

4.1. Neural Network Architecture

We implement DSLO based on two classical neural network architectures, CNN and DNN. As shown in Figure 4, the DSLO-CNN algorithm model utilizes three convolutional layers and three fully connected layers. Each layer is followed by the rectified linear (ReLU) activation function except for the output layer. The Sigmoid activation function is used to drop the output value near 0 or 1. The DSLO-CNN algorithm captures the associated information of $d_i$ and $w_i$ via block by block scanning and focuses on local contents.

**Figure 4.** The process of the DSLO-CNN algorithm.

The DSLO-DNN algorithm structure is simpler and only uses fully connected layers, as shown in Figure 5. It weights and processes all data through the full connection. Although the influence of all data on a single node is considered, the association information between workloads $d$ and task weights $w$ is not highlighted.
Table 1 is a parameter description of both DLSO algorithm structures. Each convolution kernel has a size of 2. BN denotes batch normalization whose size depends on the size of the corresponding layer. We set ReLU activation functions at each layer except the output layer in order to increase the generalization performance of the model and alleviate the problem of overfitting in a single scenario.

Table 1. The parameters of DLSO-CNN and DSLO-DNN algorithm structures.

| Layer   | Size | Activation | BN |
|---------|------|------------|----|
| Conv1d1| 16   | ReLU       | 16 |
| Conv1d2| 16   | ReLU       | 16 |
| Conv1d3| 3    | ReLU       | -  |
| fc1     | 21   | ReLU       | -  |
| fc2     | 64   | ReLU       | -  |
| fc3     | 10   | Sigmoid    | 10 |

| Layer   | Size | Activation | BN |
|---------|------|------------|----|
| fc1     | 20   | ReLU       | -  |
| fc2     | 120  | ReLU       | -  |
| fc3     | 80   | ReLU       | -  |
| fc4     | 10   | Sigmoid    | 10 |

We use BN to address issues related to internal covariance shifts in feature diagrams, so as to prevent model overfitting by smoothing the flow of gradients and improving network generalization. Its mathematical expression is as follows:

\[
\hat{y}_i = \gamma \frac{x_i - u}{\sqrt{\sigma^2 + \epsilon}} + \beta, \\
u = \frac{1}{S} \sum_{i=1}^{S} x_i, \\
\sigma^2 = \frac{1}{S} \sum_{i=1}^{S} (x_i - u)^2.
\]  

(13)

where \(S\) denotes the batch size, \(x_i\) denotes the input data, \(\hat{y}_i\) denotes the data after BN, \(u\) and \(\sigma\) are the means. Regarding the variance of input data, \(\gamma\) and \(\beta\) are the scale factor and the offset, respectively. To avoid a denominator of 0, \(\epsilon\) is usually set to \(1.0 \times 10^{-5}\) to increase numerical stability [28].

4.2. Train DSLO

To train DSLO, we randomly sample a batch of MEC scenarios \(\{\Psi_i \mid i \in I_{train}\}\), where each scenario \(\Psi_i\) contains \(K_{train}\) data samples where \(K_{train} \subset K\). We merge these train-
ing samples from different MEC scenarios together, denoted as \( M = \{(d_k, w_k, a_k)|k \subset K_{\text{train}}, i \subset I_{\text{train}}\} \). During each round of training, we randomly sample a set of training data samples \( M_b \) from \( M \), denoted as \( D = \{(d_m, w_m, a_m)|m \in M_b\} \). We use randomly sampling with replacement to prevent the model from being overly dependent on a particular batch of data during training and thus falling into a local optimum [29]. Then, we train the neural network model by minimizing its mean-squared error loss [30] as

\[
L(f_\theta) = \sum_{m \in M_b} \|f_\theta(d_m, w_m) - a_m\|^2_2, \quad (14)
\]

where \( f_\theta \) is a parameterized function of the model and represents the mapping relationship between computation task and offloading decision. Then, the model’s parameters \( \theta \) are updated by gradient descent [31], i.e.,

\[
\theta' = \theta - \eta \nabla_\theta L(f_\theta), \quad (15)
\]

where \( \nabla_\theta \) is calculated with respect to the gradient of \( \theta \), and \( \eta \) is a step length hyper-parameter. After \( G \) rounds of training iteration, the network model converges and is used as the offload decision prediction model for a dynamic MEC network composed of different workloads and weight factors.

4.3. Test DSLO

During the test phase, for each MEC scenario in the test dataset \( \{\Psi_i'|i' \in I_{\text{test}}\} \), we randomly sample a batch of data samples \( \mathcal{K}_{\text{test}} \subset \mathcal{K} \) from \( \Psi_{i'} \). For each test data sample, we input the workload \( d_{i'}^{k'} \) and weighting factors \( w_{i'}^{k'} \) into the DSLO and obtain the predicted offloading decision \( \hat{a} \), as

\[
\hat{a} = f_{\theta'}(d_{i'}^{k'}, w_{i'}^{k'}) , k' \subset \mathcal{K}_{\text{test}}, i' \in I_{\text{test}} \quad (16)
\]

Finally, based on \( \hat{a} \), we evaluate the network utility \( Q \) by solving the subproblem (P2).

Before leaving this section, we provide the pseudo-code of the DSLO algorithm in Algorithm 1.

**Algorithm 1:** Pseudo-code of the DSLO Algorithm.

| Input | Dataset \( I \) of different MEC scenarios and step-size hyper-parameter \( \eta \) |
|-------|-------------------------------------------------|
| Output| The trained neural network model |
|       | Randomly initialize \( \theta \) |
|       | Randomly split \( I \) into \( I_{\text{train}} \) and \( I_{\text{test}} \) |
|       | For each scenario, randomly split its data samples \( \mathcal{K} \) into \( \mathcal{K}_{\text{train}} \) and \( \mathcal{K}_{\text{test}} \) |
|       | Merge all training samples into a whole training set \( M \) |
|       | // Training procedure |
|       | for TimeFrame \( t \in \{1, 2, \ldots, G\} \) do |
|       | Sample data \( D \) from \( M \) |
|       | Calculate the losses \( L(f_\theta) \) based on data samples \( D \) according to (14) |
|       | Update the parameters of the network model according to (15) |
|       | end |
|       | // Testing procedure |
|       | Given a new sample set from \( \mathcal{K}_{\text{test}} \), generate its offloading decision \( \hat{a} \) |
|       | Evaluate the network utility \( Q \) by solving the subproblem (P2) |

5. Performance Evaluation

5.1. Parameter Settings

In this section, we study the numerical performance of the proposed DSLO algorithm under different MEC scenarios. In the simulation, we follow the settings in [27] and set the local computation time of the mobile device as \( t_n = 4.75 \times 10^{-7} \) s/bit and the processing energy consumption as \( e_n = 3.25 \times 10^{-7} \) J/bit. We assume that the input data size of all
tasks \( d_n \) is randomly distributed between 10 and 30 MB. In addition, we consider x264 video encoding application as the computing task of WDs where the number of computational cycles is relevant to the input size as \( \gamma_n = 1900 \text{ cycles/byte} \times d_n \) [32]. The transmission energy consumption of mobile devices is, respectively, \( e_n = 1.42 \times 10^{-7} \text{ J/bit} \). The CPU rate of the edge server is \( 10 \times 10^9 \text{ cycles/s} \). We further set \( \alpha = 3.5 \times 10^{-7} \text{ J/bit} \) and the uplink bandwidth limit as 100 Mbps. In Table 2, we summarize the simulation parameters for easy reference.

Table 2. Simulation parameters.

| Notation | Value          | Notation | Value          |
|----------|----------------|----------|----------------|
| \( C \)  | 100 Mbps       | \( d_n \) | 10–30 MB       |
| \( e_n \) | \( 3.25 \times 10^{-7} \text{ s/bit} \) | \( t_n \) | \( 4.75 \times 10^{-7} \text{ s/bit} \) |
| \( \alpha \) | \( 3.5 \times 10^{-7} \text{ J/bit} \) | \( \gamma_n \) | \( 1900 \text{ cycles/byte} \) |
| \( e_{tn} \) | \( 1.42 \times 10^{-7} \text{ J/bit} \) | CPU rate | \( 10 \times 10^9 \text{ cycles/s} \) |

To train and evaluate the DSLO algorithm, we pre-generate a dataset composed of various MEC task scenarios. For an MEC task scenario, given the workload \( d_t \) and the task weight \( w_t \) at time \( t \), we compute its best offloading action by enumerating all \( 2^N \) binary combinations, calculate all those corresponding optimization utilities \( Q_t \) by solving subproblem (P2), and choose the offloading decision \( a^*_t \) with the minimum utility \( Q^*_t \) as the optimal decision. Then, we add the state parameter \( (d_t, w_t) \) and the offloading decision along with its optimal \( (a^*_t, Q^*_t) \) into the dataset. Considering extending to generality, we set the task weights \( W = \{1, 1.5\} \). We evaluate an MEC network with \( N = 10 \) WDs and include all 1024 MEC task scenarios with different weights in the dataset, which are further used to train and test DSLO.

Regarding network utility performance, we compare our DSLO algorithm with three representative benchmarks:

1. **Random offloading decision**: All \( N \) WDs randomly generate 0–1 offloading decisions.
2. **Linear Relaxation (LR) algorithm** [33]: The binary offloading decision variable conditioned on (11) is relaxed to a real number between 0 and 1, as \( \hat{a}_n \in [0, 1] \). Then, the optimization problem (P1) with this relaxed constraint is convex with respect to \( \{\hat{a}_n\} \) and can be solved using the convex optimization toolbox. Once \( \hat{a}_n \) is obtained, the binary offloading decision \( a_n \) is determined as follows
   \[a_n = \begin{cases} 1, & \text{when } \hat{a}_n \geq 0.5 \\ 0, & \text{otherwise} \end{cases}\] (17)
3. **Greedy strategy**: For the greedy scheme, we enumerate all offloading decision combinations and then adopt the best one.

The following numerical results are an average of 200 realizations running on a laptop with Intel(R) Core(TM) i7-4710MQ CPU @ 2.50 GHz.

5.2. DSLO with Plenty Training Samples

We first evaluate the extreme performance of DSLO with a large amount of data samples. In Figure 6, we study the convergence performance of the DSLO algorithm for a specific MEC scenario, whose task weights are \( w' = [1, 1.5, 1, 1.5, 1, 1.5, 1, 1.5, 1, 1.5] \). In this scenario, we set \( |\mathcal{M}| = |\mathcal{K}_{\text{train}}| = 5000, |\mathcal{M}_{\text{b}}| = 128 \), and \( |\mathcal{K}_{\text{test}}| = 1000 \). There are enough training samples to train DSLO until convergence.
For the sake of a better illustration, we define a normalized system utility:

$$\hat{Q} = \frac{Q^*(h, w, a^*)}{Q(h, w, a)}$$

(18)

which is a ratio between the enumerated optimal offloading action and the one generated by the evaluated algorithm. Both DSLO-CNN and DSLO-DNN converge to a normalized system utility $\hat{Q} \geq 99\%$. With regard to the convergence speed, DSLO-CNN only takes 1000 time frames, while DSLO-DNN takes 5000 time frames. Therefore, the DSLO-CNN algorithm can quickly converge and achieve approximately optimal convergence $\hat{Q}$.

In Figure 7, we compare the system utility performance of different offloading algorithms under the varying numbers of WDs. The MEC task scenarios are different due to the different number of WDs. For fairness, we have considered two different MEC scenarios: $\Psi_1$ and $\Psi_2$, whose weight factors are evenly distributed. For each WD, we give weight factors data in Table 3. Compare $\Psi_1$ and $\Psi_2$ with the data at the same position. For example, when $N = 15$ and CNN is adopted in the algorithm, the minimize system utility $Q$ obtained, respectively, is 2230 and 2183, which is a small gap. To evaluate the extreme performance, both DSLO algorithms have been trained with 5000 independent workloads until convergence. Each value is an average of over 1000 independent test data samples in Figure 7. For different MEC scenarios, both DSLO-DNN and DSLO-CNN achieve approximately optimal performance as the Greedy algorithm and are significantly better than LR and Random algorithms. Under the condition of the same algorithm, only changing the task weight (priority relationship) between WDs will not produce a big difference in the results.

**Figure 6.** Convergence performance of DSLO with plenty of training samples.

**Figure 7.** Comparisons of system utility performance for different offloading algorithms. (a) $\Psi_1$; (b) $\Psi_2$. 
Table 3. Weight factors of different WDs.

| MEC Task Scenarios | Weight |
|--------------------|--------|
|                    | $N = 5$ | $N = 10$ | $N = 15$ |
| $\Psi_1$          | [1.0, 1.5, 1.0, 1.5, 1.0] | [1.0, 1.0, 1.5, 1.5, 1.0] | [1.0, 1.0, 1.5, 1.5, 1.5] |
|                    | 1.5, 1.5, 1.0, 1.0, 1.5] | 1.0, 1.5, 1.5, 1.5, 1.0] | 1.5, 1.5, 1.0, 1.0, 1.5] |
| $\Psi_2$          | [1.0, 1.5, 1.5, 1.5, 1.0] | [1.0, 1.5, 1.0, 1.5, 1.0] | [1.0, 1.5, 1.0, 1.5, 1.0] |
|                    | 1.5, 1.0, 1.5, 1.0, 1.5] | 1.5, 1.5, 1.5, 1.0, 1.0] | 1.0, 1.0, 1.5, 1.5, 1.0] |

In Table 4, we compare the CPU execution latency of DSLO algorithms based on different WD numbers. The CPU execution latency of DSLO is significantly less than the widely used heuristic LR algorithm. When $N = 5$, the CPU execution latency for LR to complete the given task is $4.14 \times 10^{-2}$ s. The latency increases linearly with the number of WDs to $3.14 \times 10^{-1}$ s for 15 WDs. In comparison, the test latency of DSLO is always below $10^{-3}$ s. While DSLO-CNN achieves better convergence performance due to its complex neural networks, it doubles the training latency of DSLO-DNN. Interestingly, the test latency of DSLO-CNN is almost the same as the one of DSLO-DNN, which generates an offloading within 1 millisecond.

Table 4. Comparisons of CPU execution latency.

| # of WDs | DSLO-CNN | DSLO-DNN | LR |
|----------|----------|----------|----|
|          | Train    | Test     | Train | Test |      |
| 5        | $7.5 \times 10^{-3}$ s | $1.4 \times 10^{-3}$ s | $3.5 \times 10^{-3}$ s | $1.4 \times 10^{-4}$ s | $4.1 \times 10^{-2}$ s |
| 10       | $8.3 \times 10^{-3}$ s | $2.5 \times 10^{-3}$ s | $3.8 \times 10^{-3}$ s | $2.3 \times 10^{-4}$ s | $1.4 \times 10^{-1}$ s |
| 15       | $9.2 \times 10^{-3}$ s | $3.4 \times 10^{-4}$ s | $3.9 \times 10^{-3}$ s | $3.4 \times 10^{-4}$ s | $3.1 \times 10^{-1}$ s |

In Figure 8, we evaluate the effect of BN in both DSLO algorithms. BN greatly improves the convergence performance of DSLO in metrics of both convergence speed and the optimal system utility. As shown in Figure 8a, with the BN structure, the training time of DSLO-CNN reduces from 4000 time frames to 1000 time frames. As shown in Figure 8b, models that do not contain BN layers have a high tendency to fall into local optima in the training phase.

Figure 8. Performance evaluation of BN layer. (a) DSLO-CNN; (b) DSLO-DNN.

5.3. DSLO with Few Training Samples

In Figure 9, we study the convergence performance of the DSLO algorithm for dynamic MEC scenarios with few training samples in each scenario. Specifically, we extract 70% of the entire scene from 1024 MEC task scenarios as the training set, i.e., $|I_{\text{train}}| = 714$ and set
\[ |M_b| = 128. \] For each MEC scenario, we only use \(|K_{\text{train}}| = 2 \) and \(|K_{\text{test}}| = 100. \) As shown in Figure 9, DSLO-DNN slowly converges to a normalized system utility \( \hat{Q} = 0.98 \) until the 9000 time frames. In comparison, it converges to \( \hat{Q} \geq 0.99 \) within 4000 time frames when there are plenty of training samples, as shown in Figure 6. On the other hand, the DSLO-CNN algorithm remains in a high convergence speed, resulting in a normalized system utility \( \hat{Q} \geq 99\% \) within 2000 time frames.

![Figure 9. DSLO with \(|K_{\text{train}}| = 2\) per MEC scenario.](image)

In Figure 10, we evaluate the convergence performance of DSLO with different scales of training scenarios. As shown in Figure 10a, when \(|I_{\text{train}}| = 10\%|I|\), we randomly choose 103 MEC scenarios out of 1024 scenarios to build the training dataset. Other system parameters are the same as the ones in Figure 9. It follows that there are only 206 training samples when \(|I_{\text{train}}| = 10\%|I|\). With such a few training samples, DSLO-CNN achieves a normalized system utility \( \hat{Q} = 98\% \) after convergence. As shown in Figure 10b, when \(|I_{\text{train}}| = 30\%|I|\), the DSLO-CNN algorithm converges within \( t = 1000 \) with \( \hat{Q} = 99\% \). In Figure 10c,d DSLO-CNN converges faster and achieves a more stable performance with the increase of training scenarios. In all these cases, DSLO-DNN converges much slower than DSLO-CNN. Hence, the proposed DSLO-CNN is suitable for dynamic MEC scenarios with few training samples.

![Figure 10. Cont.](image)
In Figure 11, we further statistically test the adaptability of DSLO to new MEC scenarios. We plot both the median and the confidence intervals of $\hat{Q}$ overall test scenarios. As shown in Figure 11a, we make the number of training scenarios 70% of the full scenario, as $|I_{train}| = 70%|I|$. We adjust the training set by changing the size of $|K_{train}|$ while keeping other parameters unchanged. When $|K_{train}| = 1$, it means that there is only one training sample per MEC scenario \( \Psi_i | i \in I_{train} \). The median of DSLO-CNN is greater than 98%, and the median of DSLO-DNN is no more than 96%. As the number of training samples increases, DSLO is more adaptable to new scenarios. In general, DSLO-CNN outperforms DSLO-DNN, especially with few training samples. As shown in Figure 11b, we evaluate DSLO that trained under different sizes of $|I_{train}|$. At the same time, the number of training samples is kept constant for each scenario, as $|K_{train}| = 2$. When $|I_{train}| = 10%|I|$, the median of the DSLO-CNN algorithm is greater than 96%. When $|I_{train}| = 70%|I|$, we see that the median of the DSLO-CNN algorithm can reach more than 99% and the confidence intervals are mostly above 98%. In comparison, the median of the DSLO-DNN algorithm is always less than 98%. Under different scales of training sets, the adaptability of DSLO-CNN to new MEC scenarios is always better than that of the DSLO-DNN algorithm.

5.4. Comparisons with MELO and ARM

To further evaluate the effectiveness and superiority of the proposed algorithm, we compare the proposed algorithm with other computational offloading algorithms. In addition to the LR [33] algorithm, two different benchmark algorithms, MELO [11] and ARM [34], have been evaluated. MELO is a meta-learning-based computing offloading
algorithm that learns a priori knowledge of historical MEC scenarios and quickly converges when encountering new MEC scenarios by fine-tuning training with only a few samples. ARM is an adaptive risk minimization framework, which extracts global features from different scenarios to update the network model and improves the model’s adaptability to unknown scenarios.

Before fine-tuning the training, the models were trained with a sufficient number of samples, where \(|I_{\text{train}}| = 70\%|I|\) as well as \(|K_{\text{train}}| = 4\). It is worth mentioning that the input of the MELO algorithm model does not contain weight information, and other parameters are set in the same way as the DSLO algorithm. To avoid chance, we use the average value of the fine-tuning test in 10 MEC scenarios as the actual test effect. As shown in Figure 12, the LR algorithm produces a constant normalized system utility value \(\hat{Q} = 0.934\). The MELO algorithm converges to the same performance as DSLO-DNN in 30 fine-tuning steps, resulting in a normalized system utility value \(\hat{Q} > 98.5\%\). The ARM algorithm is implemented based on CNN and generates a normalized system utility value \(\hat{Q} = 0.99\), which is greater than MELO and DSLO-DNN. However, the normalized system utility value \(\hat{Q}\) generated by DSLO-CNN is always greater than 99\%. Hence, DSLO-CNN makes full use of the weight information and can achieve better model initialization parameters than other computational offloading algorithms with a small number of training samples. The algorithm can be quickly adapted to new MEC scenarios.

![Figure 12. Test performance of different computational offloading algorithms.](image)

### 6. Conclusions

In this paper, we propose a deep supervised learning-based offloading algorithm, DSLO, which requires few training samples and can adapt to different MEC computing task scenarios. We take advantage of convolutional neural networks to efficiently handle multidimensional state parameters. The proposed algorithm can quickly adapt to dynamically changing MEC scenarios and achieve near-optimal offloading decisions. Numerical results have validated the accuracy of the proposed algorithm and the performance advantage compared with the existing MELO algorithm. Extensive numerical results are evaluated to study the performance of both DSLO-CNN and DSLO-DNN algorithms. DSLO outperforms the current benchmark algorithms in the metrics of system utility and CPU execution latency. In general, DSLO-CNN requires fewer training samples than DSLO-DNN. It can achieve 99\% normalized system utility \(\hat{Q}\) by using only four training samples per MEC scenario.

The proposed DSLO algorithm in this paper relies on a small size of labeled data, which limits its application in some MEC scenarios whose optimal labels are unavailable or difficult to be obtained. Moreover, DSLO requires all training data to be collected and trained in a centralized MEC server, which may raise some security and privacy issues.
related to personal data. For future work, we expect a DSLO algorithm implemented via reinforcement learning methods without manually labeled data. Furthermore, considering the data privacy in distributed MEC scenarios, implementing DSLO on a federated learning framework is interesting and necessary. Considering the heterogeneity and complexity of MEC networks, we expect that DSLO can be further extended to dynamically changing MEC scenarios with online learning. It will benefit future offloading algorithm deployment on large-scale MEC networks.

**Author Contributions:** Conceptualization, L.H.; Data curation, S.Y.; Formal analysis, S.Y. and G.L.; Funding acquisition, L.H.; Methodology, S.Y.; Project administration, L.H.; Software, S.Y.; Validation, G.L.; Visualization, G.L.; Writing—original draft, S.Y.; Writing—review & editing, L.H. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was funded by the National Natural Science Foundation of China under grant number 62072410.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Not applicable.

**Conflicts of Interest:** The authors declare no conflict of interest.

**Abbreviations**

- $N$: The number of WDs
- $w_n$: The weight assigned to the $n$-th WD
- $a_n$: Offloading decision of the $n$-th WD
- $d_n$: Workload of the $n$-th WD
- $\gamma_n$: Number of CPU cycles required by the $n$-th WD to complete tasks
- $E_n$: Energy consumption of the edge server to executing the $n$-th WD task
- $E_t$: Energy consumption to transfer offloading task of the $n$-th WD
- $c_n$: Bandwidth allocated to the $n$-th WD
- $C$: Total bandwidth
- $T_{cn}$: The edge processing delay of the $n$-th WD
- $f_n$: The processor’s computing speed of the $n$-th WD
- $e_n$: Local computing unit data energy consumption of the $n$-th WD
- $E_l$: Local computing energy consumption of the $n$-th WD
- $t_n$: Local unit data execution delay of the $n$-th WD
- $T_n$: Local execution delay of the $n$-th WD
- $Q(\cdot)$: The system utility function
- $\pi$: Offloading policy function
- $L(\cdot)$: The training loss function of the model
- $\Psi$: Scenario of MEC
- $\eta$: The training step
- $\theta$: The parameters of the model
- $\hat{a}$: Predictive offloading decisions
- $G$: Training iterations
- $\hat{Q}$: Normalized system utility
- $MEC$: Mobile Edge Computing
- $DSLO$: Deep Supervised Learning-based computational Offloading algorithm
- $MELO$: MEta-Learning-based computing Offloading algorithm
- $WD$: Wireless Device
- $DNN$: Deep Neural Network
- $CNN$: Convolutional Neural Network
- $BN$: Batch Normalize
- $AP$: Access Point
- $Conv1D$: One-dimensional Convolutional
- $ReLU$: Rectified Linear activation function
- $LR$: Linear Relaxation
31. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. Learning representations by back-propagating errors. *Nature* **1986**, *323*, 533–536. [CrossRef]

32. Miettinen, A.P.; Nurminen, J.K. Energy efficiency of mobile clients in cloud computing. In *Proceedings of the 2nd USENIX Conference HotCloud*, Boston, MA, USA, 22–25 June 2010.

33. Guo, S.T.; Xiao, B.; Yang, Y.Y.; Yang, Y. Energy-Efficient Dynamic Offloading and Resource Scheduling in Mobile Cloud Computing. In *Proceedings of the 35th Annual IEEE International Conference on Computer Communications (INFOCOM 2016)*, San Francisco, CA, USA, 10–14 April 2016.

34. Zhang, M.; Marklund, H.; Dhawan, N.; Gupta, A.; Levine, S.; Finn, C. Adaptive risk minimization: Learning to adapt to domain shift. In *Proceedings of the 35th Annual Conference on Neural Information Processing Systems (NeurIPS 2021)*, Virtual, 6–14 December 2021.