Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 2890473, 13 pages
https://doi.org/10.1155/2022/2890473

Research Article

Task Offloading and Scheduling Strategy for Intelligent Prosthesis in Mobile Edge Computing Environment

Ping Qi

Department of Mathematics and Computer Science, Tongling University, Tongling 244061, China

Correspondence should be addressed to Ping Qi; qiping929@tlu.edu.cn

Received 5 October 2021; Revised 19 October 2021; Accepted 29 November 2021; Published 7 January 2022

Academic Editor: Muhammad Shiraz

Traditional intent recognition algorithms of intelligent prosthesis often use deep learning technology. However, deep learning's high accuracy comes at the expense of high computational and energy consumption requirements. Mobile edge computing is a viable solution to meet the high computation and real-time execution requirements of deep learning algorithm on mobile device. In this paper, we consider the computation offloading problem of multiple heterogeneous edge servers in intelligent prosthesis scenario. Firstly, we present the problem definition and the detail design of MEC-based task offloading model for deep neural network. Then, considering the mobility of amputees, the mobility-aware energy consumption model and latency model are proposed. By deploying the deep learning-based motion intent recognition algorithm on intelligent prosthesis in a real-world MEC environment, the effectiveness of the task offloading and scheduling strategy is demonstrated. The experimental results show that the proposed algorithms can always find the optimal task offloading and scheduling decision.

1. Introduction

According to statistics, the number of disabled people in China has topped 85 million in 2020 [1]. Lower limb amputation is a major cause of disability; millions of transfemoral amputees are suffering from difficulty in moving, which accounts for approximately seventy of the total number of disabled persons [1]. With the progress of science and technology, the scientists concentrate on the research and the maintenance of rehabilitation equipment, appliances, and other aids for disabled people. The intelligent prosthesis is the only way to compensate or restore the motor function, which can enable transfemoral amputees to perform diverse daily activities.

However, even though the movement of the lower limbs is the most basic human movement, many disabled people are still different to accomplish some simple tasks through the prosthetic leg. Meanwhile, using a passive prosthesis may significantly impair the walking symmetry and metabolic energy efficiency of transfemoral amputees. Therefore, the premise of using intelligent prosthesis is that some appropriate sensors and intent recognition algorithms should be selected to obtain movement information. Then, the intelligent prosthesis can automatically calibrate the torque according to the analysis signals (such as biomechanical signal, surface electromyographic signal, and sEMG) perceived by sensors, which makes amputee's moving process more stable and natural [2].

As shown in Figure 1, the hierarchical control strategy of intelligent prosthesis is made up of three layers: perception layer, transformation layer, and execution layer. The perception layer recognizes amputee's motion intent by activity mode and context recognition. The transformation layer constantly adjusts the control strategy by comprehending human motion intent. Then, the above control strategy is used to actuate the intelligent prosthesis in execution layer. Obviously, as the motion intent recognition influences the effectiveness of the perception layer, the intent recognition algorithm with high performance and low latency is of the utmost importance.

Unfortunately, the traditional intent recognition methods often involve high complexity algorithms, such as template matching, convolutional neural network (CNN), and sensor fusion [3, 4]. Although there are many existing

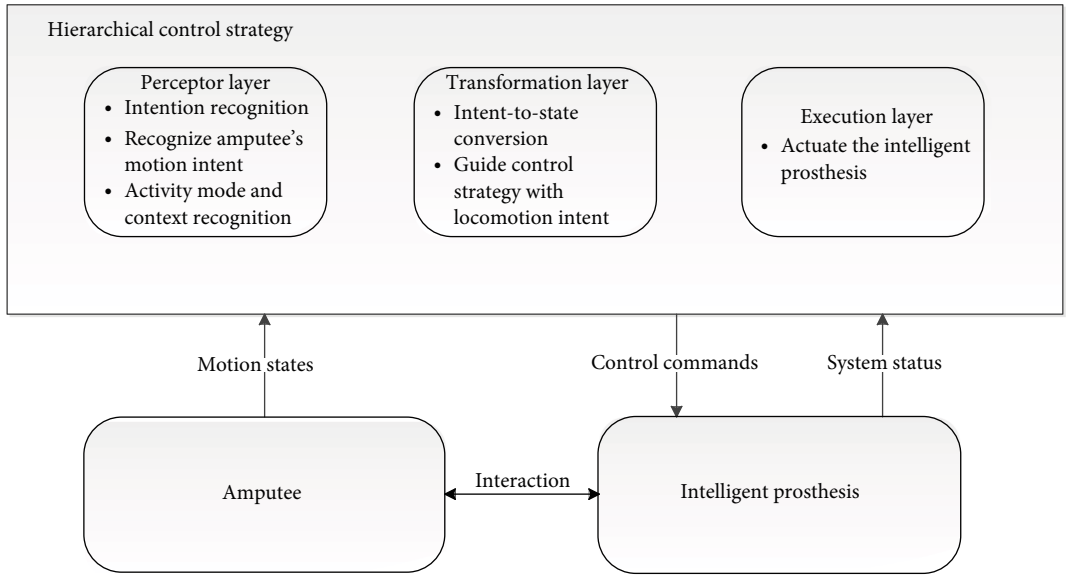

Figure 1: Hierarchical control strategy of intelligent prosthesis.

works focusing on reducing the computational complexity [5–9]. It is still difficult to achieve the purpose of real-time intent recognition. In the meantime, the computation and communication overheads of neural network model are affected by the model size, which bring a large amount of energy consumption and execution time. Therefore, the intelligent prosthesis, which is limited by their computing power and battery capacity, requires intensive computation and energy resources to provide services.

Recently, mobile edge computing (MEC) has become a promising solution which supports computation-intensive tasks. MEC is an efficient method to overcome the challenge by offloading some latency-sensitive tasks or computation-intensive to nearby edge servers through wireless communication [10, 11]. Then, the edge servers execute some of the received computation tasks and transmit the rest to resource-rich cloud infrastructures by low-latency connection. At last, the edge servers or the cloud server transmit the computation results to the mobile device. In this paper, the intelligent prosthesis can be represented as the mobile device in an MEC system.

To take full advantage of the mobile edge computing, an effective collaboration between the intelligent prosthesis, the edge servers, and the cloud is an essential problem. In this paper, we focus on solving latency and energy-constrained challenges, in which mobile processors share multiple heterogeneous edge servers. By deploying the deep learning-based motion intent recognition algorithm on the intelligent prosthesis in MEC environment, we demonstrate the effectiveness of the proposed task offloading strategy in reducing the latency and energy consumption.

The structure of this paper is as follows: Section 2 describes the related works on the intent recognition algorithm for intelligent prosthesis and some existing studies for applying AI technology in the MEC environment. MEC-based offloading model and problem definition are given in Section 3. The proposed algorithms are described in detail in Section 4. The experimental results are discussed in Section 5. Finally, the conclusion of this paper is provided in Section 6.

2. Related Work

The motion states, which are conducted by the lower limb, have an inherent regularity. One single gait cycle includes two phases: a stance phase and a swing phase. As shown in Figure 2, the swing phase consists of a steady state and a transitional state. The steady state begins with a toe-off and ends with a heel strike. The transitional state also begins with a toe-off. Then, the foot rises from the flat ground, and the transitional state ends when the heel touches the stair or the ramp [12]. Various pattern recognition and machine learning algorithms are used to analyze the regularity, locomotion modes, and transition modes within a single gait cycle.

Huang et al. [5] propose a motion intent recognition algorithm based on neuromuscular-mechanical fusion. Electromyographic (EMG) signals and ground reaction forces are used as the input data to a phase-dependent pattern classifier. In this study, six locomotion modes and five transitions can be recognized, and the recognition accuracy reaches 95.2%. Liu et al. [6] study the effectiveness of applying three different adaptive pattern classifiers based on surface electromyography and mechanical sensors. Under a variety of different terrains, the proposed algorithm predicts amputee's motion intent with a rate of 95.8%. During recent years, CNN has been widely used in different smart systems, such as smart health, smart logistics, and smart agriculture. Su et al. [7] put the inertial measurement units (IMUs) on the healthy leg of amputees. They design a CNN structure to automatically learn the features from the sensor signals without any prior knowledge. Idowu et al. [8] present an integrated deep learning model (deep neural networks, DNN) for motor intention recognition of multiclass signals.

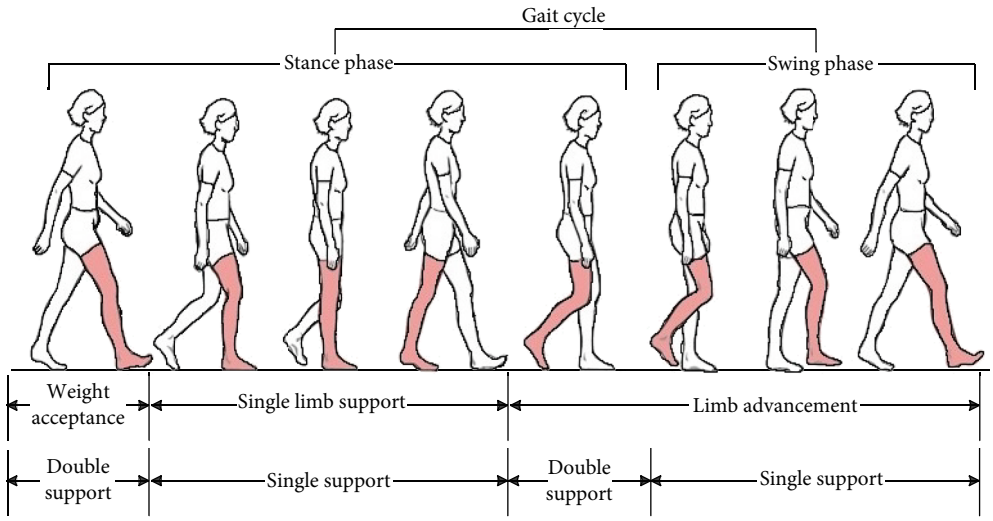

Figure 2: Illustration of stance phase and swing phase.

In this paper, the related features with different frequencies are introduced as input representation to the proposed deep learning model, and the recognition accuracy can reach a high level (average performance of 99.01%). Gautam et al. [9] propose a low-complex deep learning framework for sEMG-based movement recognition. The method of two-stage pipeline compression (input data compression and data-driven weight sharing) is designed to reduce the number of memories, energy consumption, and execution time.

Deep learning inference and training require substantial computation resources to run quickly; a common approach is to leverage cloud computing [13, 14]. However, sending data to the cloud for inference or training may incur additional queuing and propagation delays from the network and cannot satisfy strict end-to-end low-latency requirements. Cloud-based intent recognition algorithms need to be further modified to run on computation constrained devices, such as a Raspberry Pi or an Arduino.

In summary, the previous studies usually use mechanical, sEMG, or IMU sensors for data collection, which are embedded on the intelligent prosthesis or the healthy leg. Then, various machine learning algorithms, such as hidden Markov model, SVM, or deep learning, are selected as the classifier to recognize the motion intent of the amputees. In particular, the deep-learning-based intent recognition algorithms have the distinct advantage of executing the data-driven feature extraction without any prefeature extraction. However, the deep learning algorithm's high accuracy and real-time implementation for the intelligent prosthesis comes at the expense of high computational, memory, and energy consumption requirements for both the training and inference phases.

To meet the computational requirements of deep learning, edge computing is a viable solution to meet the latency and energy consumption challenges. There are many existing works focusing on applying deep learning technology in the MEC environment. Li et al. [15] present a deep learning model coinference framework. To reduce the execution time, they optimize the CNN partitioning and right-sizing based on the available bandwidth and the on-demand manner. Li et al. [16] propose a joint accuracy-and latency-aware execution framework named JALAD to minimize the edge computation latency, cloud computation latency, and transmission latency. Gao et al. [17] propose the Edge4Sys system to reduce the computation load of the edge server in a MEC-based UAV (Unmanned aerial vehicle) delivery scenario. Tariq et al. [18] present the FogNetSim++ environment, which covers the network aspects such as latency, transmission range, scheduling, mobility, and heterogeneous mobile devices. Asad et al. [19] propose a fog simulation framework named xFogSim to support latency-sensitive applications. It has a very efficient task distribution algorithm that can choose the best computation resource depending on the cost, availability, and latency. Syed et al. [20] present a fog computing framework to simulate the vehicle-assisted computing environment, which allow researchers to incorporate their own scheduling policies to simulate a realistic environment. Table 1 lists the comparison of these works.

However, it is not suitable for applying the above works directly to the intelligent prosthesis scenario. One major challenge is that the above works do not consider the mobility of the mobile device, which is an important property of the intelligent prosthesis. The second challenge is accommodating the high resource requirements of deep learning on less powerful intelligent prosthesis with mobile processors. Most offloading and scheduling strategy still transmit a large amount of data to the remote servers. The data processing time by deep learning model is large; the cooperation between the mobile device and the remote servers should be employed in a real-world MEC-based system to reduce the energy consumption and latency.

3. MEC-Based Task Offloading Model

The motion intent recognition algorithm need to be quickly processed to detect and return a response. However, the complexity of computing tasks brings a big burden to the mobile processors which are limited by their computation

Table 1: Comparison of existing work on edge computing.

Research	Proposed solutions	Key metrics	Edge devices	What is to be offloaded
Li et al. [15]	A deep learning model coinference framework	Latency, communication size	Devices with cameras	Computer vision algorithms
Li et al. [16]	A joint accuracy-and latency-aware execution framework	Latency, accuracy	Devices with cameras	Computer vision algorithms
Gao et al. [17]	Edge4Sys system to reduce the computation load of the edge server in a MEC-based UAV delivery scenario	Latency, energy, accuracy	UAV	DNN-based feature extraction and classification
Tariq et al. [18]	A fog simulator, covers the network, transmission rage, heterogeneous mobile devices, and mobility feature	Latency, transmission range, mobility	IoT devices	Typical IoT tasks
Asad et al. [19]	A fog simulation framework to support latency-sensitive applications	Latency, energy, accuracy	IoT devices	Typical IoT tasks
Syed et al. [20]	A fog computing framework to simulate the vehicle-assisted computing environment	Latency, energy consumption, communication size, memory	Vehicles	Compute-intensive tasks
The proposed algorithm	MEC-based system to reduce the latency and energy consumption	Latency, energy consumption	Intelligent prosthesis	DNN-based intent recognition tasks

resource and battery capacity. Meanwhile, sending data to the cloud for inference or training may incur propagation delays from the network. This method cannot satisfy real-time requirements of applications because of the unsteady character of the mobile network.

In the mobile edge computing environment, the edge devices provide computation abilities close to the mobile devices. Unfortunately, this potential solution of moving the computation and data from the mobile devices to the edge still has its limitations. While edge's computation resources are substantial, they are also limited when compared to the cloud. Therefore, the edge servers should coordinate with the mobile device, the cloud, and other edge servers to ensure a good performance. Each task will be decided to be processed locally or offloaded to the cloud or the edge.

As can be seen from Figure 3, we present the detail design of the MEC-based task offloading model for deep neural networks in the intelligent prosthesis scenario. The execution process is mainly divided by the following steps: (1) the sensor data are preprocessed by the mobile processors. We deploy the intent recognition algorithm [7] on the intelligent prosthesis which is designed specifically for embedded devices. (2) CNN model partitioning. As shown in Figure 4, the CNN is partitioned into layers, some layers are executed on the mobile processors, and some layers are offloaded to the edge or the cloud according to the task offloading and scheduling strategy; (3) when the amputees are moving, mobility-aware task offloading strategy (such as task migration and task deferred execution) is considered to guarantee the service continuity; (4) the computation results are transmitted to intelligent prosthesis.

3.1. System Model. In this section, we consider the computation offloading problem of multiple heterogeneous edge servers in MEC environment. The original data, which is preprocessed by the mobile processor, is continuously offloaded to the edge servers and the cloud based on the task offloading strategy. As can be seen in Figure 3, there are m edge servers, $S = \{S_1, S_2, \cdots, S_m\}$, which have been deployed in area A. The intelligent prosthesis (mobile device) in use by the amputee is walking through area A, the moving path can be represented as a set of continuous position coordinates, $\{c_1, c_2, \cdots, c_i, \cdots\}$, $c_i = (x_i, y_i)$, where c_i is the ith position coordinates of the moving path, x_i and y_i are defined as the x-coordinate and y-coordinate of c_i, respectively. We assume that the moving path is predetermined, and the speed is v (m/s).

The edge server S_i can be represented by a quadruple, $S_i = \{f_i, BW_i, Loc_i, Dis_i\}$, where f_i, Loc_i, BW_i, Dis_i are used to indicate the computational capability, position coordinate, transmission bandwidth, and maximum communication range of S_i, respectively. For each MEC server, when the distance between a mobile device and S_i is more than the maximum communication range Dis_i, they can not communicate with each other. Due to the amputees are walking round in area A, increasing distance between the mobile device and the edge server will decrease the communication rate. The instantaneous transmission rate between S_i and the mobile device can be calculated by

$$R_{\text{ins}} = BW_i \log_2(1 + f_{\text{SNR}}(d)), d \le Dis_i, \quad (1)$$

where R_{ins} is the instantaneous transmission rate, d is the distance between S_i and the mobile device, and $f_{\text{SNR}}(d)$ is the signal-to-noise ratio (SNR) [21]. The mobile device can be represented by a triple, $\text{MD} = \{f_m, P, Loc_m\}$, where f_m is the computational capability of the mobile processor, P is the energy consumption, and Loc_m is the position coordinate of the intelligent prosthesis. The cloud server is represented by a binary group, $CS = \{f_c, R_c\}$, where f_c is the

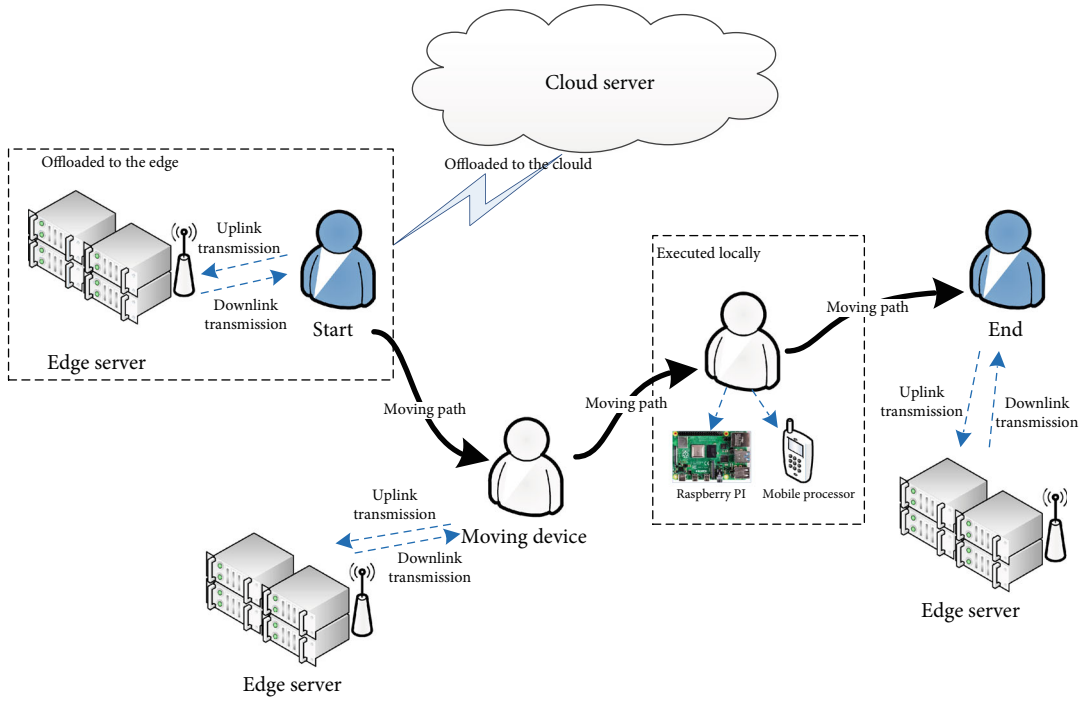

Figure 3: Overview of the MEC-based task offloading model.

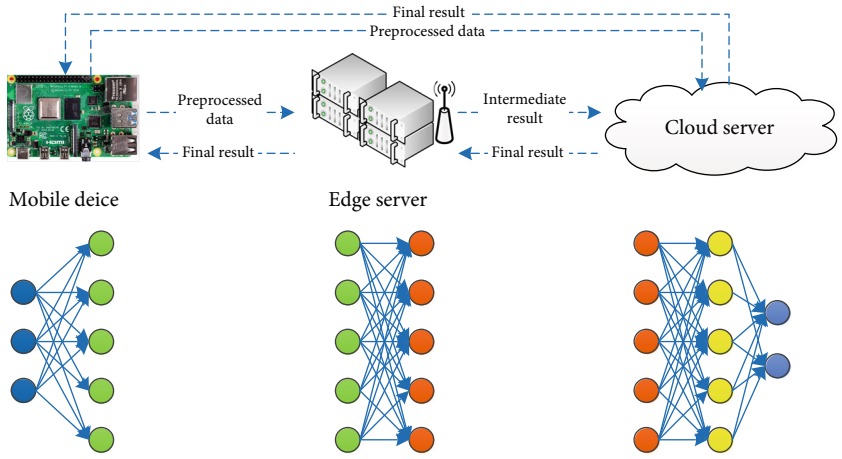

Figure 4: Deep learning inference with edge computing.

computational capability of the cloud server and R_c is the maximum data transfer rate.

In the computation offloading process, task execution spans the mobile device, the edge servers, and the cloud. The original data should be transmitted to the remote servers, and the execution results will be transmitted back to the mobile device. It is a tradeoff between the benefit of remote execution and the cost of data transmission. When the computation task is performed locally, the execution time and the energy consumption are only generated by the mobile device. Therefore, computation offloading happens only if the time and energy consumption of task offloading is better than the local execution. The relevant definitions are given below. Suppose that the uplink transmission power, downlink transmission power, idle power, and execution power of the mobile device be $p_{up}, p_{down}, p_{idle}$, and p_{exec}, respectively. Accordingly, the energy consumption can be represented by a quadruple, $P = \{p_{up}, p_{down}, p_{idle}, p_{exec}\}$.

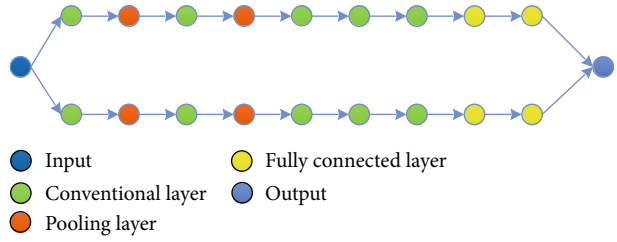

Figure 5: Workflow instance and abstract workflow DAG graph.

Table 2: Summary of key symbols in Section 3.

Symbol	Description
T	$T = \{t_1, \cdots, t_i\}$ is the set of tasks in mobile device.
IN_i, OUT_i	The input and output data size of task t_i.
l_i	The CPU cycles frequency that is required to process task t_i
f_c, f_m, f_i	The computational capability of the cloud server, mobile device, and edge server S_i
p_{tra}, p_{rec}, p_{idle}, p_{exec}	The transmitting, receiving, idle, and execution powers of mobile device
Loc_i, BW_i, Dis_i	The position coordinate, transmission bandwidth, and maximum communication range of edge server S_i
Loc_m	The position coordinate of the intelligent prosthesis
$f_{\mathrm{SNR}}(d)$	Signal-to-noise ratio (SNR)
R_{ins}	Instantaneous transmission rate between edge server and mobile device
T_{es}, T_c, T_m	The total execution time of the edge servers, cloud server, and mobile device.
E_{es}, E_c, E_m	The total energy consumption when the tasks are offloaded to the edge servers, cloud server, or computed locally.
R_{migr}	The instantaneous transmission rate between two edge servers.

The most important step toward computation offloading is partitioning, which divides the motion intent recognition algorithm into several parts that can be performed on different platforms. As shown in Figure 5, according to the structure of CNN [7], the neural network is partitioned into several layers. One layer, which can only be computed by one computation resource, is defined as the metatask. The metatask cannot be partitioned, and the set of metatasks can be represented by a directed acyclic graph (DAG), $\mathrm{DAG} = (T, E)$, where T is a set of nodes, $T = \{t_i \mid t_i = (\mathrm{IN}_i, \mathrm{OUT}_i, l_i)\}$. IN_i, OUT_i, and l_i are used to indicate the input data size, output data size, and CPU cycle frequency of task t_i, respectively. E is a set of edges, $E = \{(t_{\mathrm{pre}}, t_{\mathrm{succ}}) \mid t_{\mathrm{pre}}, t_{\mathrm{succ}} \in T\}$, where t_{pre} is the predecessor task and t_{succ} is the successor task. The topology relationships between different tasks can be described by E.

As can be seen in Figure 5, the convolutional layer and fully connected layer are the most commonly used neural network layers. Their computation load can be calculated by

$$l_C = (2 \times C_{\mathrm{in}} \times KH \times KW\,[-1]) \times H_{\mathrm{out}} \times W_{\mathrm{out}} \times C_{\mathrm{out}}, \quad (2)$$

$$l_{FC} = (2 \times I\,[-1]) \times O, \quad (3)$$

where H_{out} and W_{out} are defined as the output feature map size and C_{out} and C_{in} are the number of output channels and input channels. KH and KW are the size of the convolution kernel. I and O are the number of input and output neurons.

Summary of key symbols used in this section can be found in Table 2.

3.2. Energy Consumption and Latency Model

3.2.1. Local Execution Cost. When the task t_i is performed locally, the energy consumption is generated by the execution processes. We assume that n_m is the total number of tasks performed by the mobile device; the total execution time T_m and energy consumption E_m can be estimated by

$$\begin{cases} T_m = \sum_{i=1}^{n_m} \dfrac{l_i}{f_m}, \\ E_m = p_{\mathrm{exec}} \times \sum_{i=1}^{n_m} \dfrac{l_i}{f_m}. \end{cases} \quad (4)$$

3.2.2. Offloading Cost. When the task t_i is offloaded to the cloud server, the latency T_c is consisted by the transmission latency and the execution latency, and the energy consumption E_c is consisted by the transmission energy and the execution energy. Suppose that n_c is the total number of tasks offloaded to the cloud, T_c and E_c can be calculated by

$$\begin{cases} T_c = T_{\mathrm{up}} + T_{\mathrm{down}} + T_{\mathrm{exec}} = \sum_{i=1}^{n_c} \dfrac{\mathrm{IN}_i}{R_c} + \sum_{i=1}^{n_c} \dfrac{\mathrm{OUT}_i}{R_c} + \sum_{i=1}^{n_c} \dfrac{\mathrm{l}_i}{f_c}, \\ E_{\mathrm{c}} = E_{\mathrm{up}} + E_{\mathrm{down}} + E_{\mathrm{idle}} = p_{\mathrm{tra}} \times \sum_{i=1}^{n_c} \dfrac{\mathrm{IN}_k}{R_c} + p_{\mathrm{rec}} \times \sum_{i=1}^{n_c} \dfrac{\mathrm{OUT}_k}{R_c} + p_{\mathrm{idle}} \times \sum_{i=1}^{n_c} \dfrac{l_i}{f_c}, \end{cases} \quad (5)$$

where T_{up}, T_{down}, and T_{idle} are defined as the uplink transmission latency, downlink transmission latency, and execution latency, respectively. E_{up} and E_{down} are defined as the uplink transmission energy and the downlink transmission energy. E_{idle} is the idle energy consumption of the mobile device when the task is computed by the cloud server or edge server.

When the mobile device offloads computation to the edge, it is important to guarantee the service continuity. However, as motioned above, increasing distance between the mobile device and the edge server will decrease the communication rate. If the amputee is moving out of the communication range of the edge server, the task offloading will be failed. As shown in Figure 6, there exist four kinds of situations: normal offloading, offloading failure, task migration, and task deferred execution.

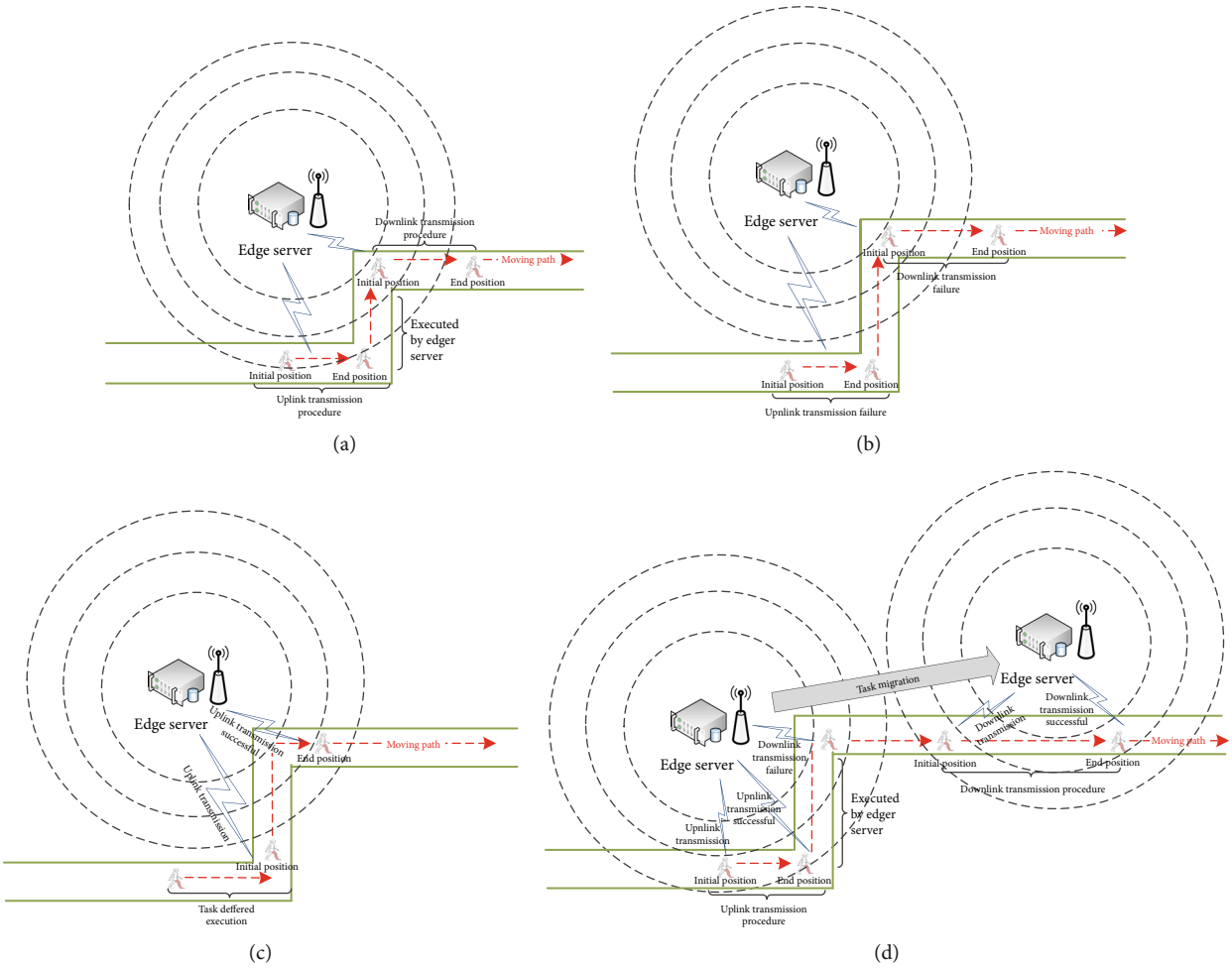

FIGURE 6: Four kinds of situations: normal offloading, offloading failure, task migration, and task deferred execution.

3.2.3. Normal Offloading. When the amputee is in the communication range of the edge server, the task can be normally offloaded. As shown in Figure 6(a), the amputee is moving from the initial position (denoted by c_{inti}) to the end position (denoted by c_{end}) in the uplink transmission procedure of task t_i. Suppose that $R(S_j, \text{MD})$ is the instantaneous transmission rate between the edge server S_j and the mobile device, $d(Loc_j, Loc_m)$ is the distance between them. We assume that the instantaneous transmission rate remains unchanged when the moving distance is no more than one meter. Therefore, the size of data transferred within one meter can be calculated by

$$\text{Data}_{\text{meter}} = R_{(S_j,MD)} \times \frac{1}{v}, \tag{6}$$

where $R(S_j, \text{MD})$ can be calculated by Formula (1).

When the summation of the transmitted data (denoted by Data_{tra}) is equal to the input data size of t_i, the uplink transmission procedure is finished. Then, the distance $d(c_{\text{inti}}, c_{\text{end}})$ between c_{inti} and c_{end} can be calculated by

$$\begin{cases} \text{Data}_{\text{tra}} = \sum_{k=1}^{d(c_{\text{inti}},c_{\text{end}})} \text{Data}_{\text{meter}}, \\ \arg \min_{d(c_{\text{inti}},c_{\text{end}})} |\text{Data}_{\text{tra}} - \text{IN}_i|, \text{Data}_{\text{tra}} \geq \text{IN}_i. \end{cases} \tag{7}$$

Accordingly, the end position of the mobile device can be estimated along the preset path according to the initial position. Similarly, in the downlink transmission procedure of task t_i, the transmission time and the end position can be calculated by the same way.

3.2.4. Offloading Failure. As shown in Figure 6(b), there is no edge server satisfying the communication conditions along the moving path. The task cannot be offloaded to the edge.

3.2.5. Task Deferred Execution. As shown in Figure 6(c), the task cannot be offloaded to the edge server in the uplink transmission procedure. However, there are several servers satisfying the communication conditions along the moving path. "Task deferred execution" is used to wait for the nearest edge server along the moving path. T_{delay} is the deferred time.

3.2.6. Task Migration. As shown in Figure 6(d), although the mobile device cannot connect to the edge in the downlink transmission procedure, there are several servers satisfying the communication conditions along the moving path. The computation results should be migrated from the current edge server to the target edge server to make sure the task execution result will be delivered successfully back to the mobile device. Suppose that the computation capability of the current edge server and the target edge server are f_{curr} and f_{tgt}, R_{migr} is the transmission rate between them. Then, the task t_i is performed by the current edge server, and the computation results OUT_i is migrated to the target edge server. The migration time is $T_{\text{migr}} = \text{OUT}_i / R_{\text{migr}}$.

In conclusion, when the mobile device offloads the task t_i to the edge, the latency T_{es} is consisted by the transmission latency (including uplink transmission time and downlink transmission time), execution latency, migration latency, and deferred execution latency. Suppose that n_{es} is the total number of tasks offloaded to the edge, T_{es} and E_{es} can be calculated by

$$\begin{cases} T_{\text{es}} = \sum_{i=1}^{n_{\text{es}}} T_{\text{up}} + \sum_{i=1}^{n_{\text{es}}} T_{\text{down}} + \sum_{i=1}^{n_{\text{es}}} T_{\text{delay}} + \sum_{i=1}^{n_{\text{es}}} T_{\text{migr}} + \sum_{i=1}^{n_{\text{es}}} T_{\text{exec}}, \\ E_{\text{es}} = p_{\text{up}} \times \sum_{i=1}^{n_{\text{es}}} T_{\text{up}} + p_{\text{down}} \times \sum_{i=1}^{n_{\text{es}}} T_{\text{down}} + p_{\text{idle}} \times \left(\sum_{i=1}^{n_{\text{es}}} T_{\text{delay}} + \sum_{i=1}^{n_{\text{es}}} T_{\text{migr}} + \sum_{i=1}^{n_{\text{es}}} T_{\text{exec}} \right), \end{cases} \tag{8}$$

where T_{up}, T_{down}, T_{exec}, T_{delay}, and T_{migr} are defined as the uplink transmission time, downlink transmission time, execution time, migration time, and deferred time (if the task can be normally offloaded, $T_{\text{delay}} = 0$, $T_{\text{migr}} = 0$). T_{up}, T_{down}, and T_{exec} can be calculated according to the initial location of the mobile device and the current edge server.

4. Proposed Algorithms

To reduce the latency and energy consumption, we propose algorithms to determine whether the methods to be executed remotely or locally with deadline constraint of each task. The proposed algorithms can be divided into three parts as follows:

(1) CNN-based intent recognition algorithm is partitioned into several layers, which can be expressed as a workflow task. According to the priority queue of the workflow, we construct a matrix of task execution as follows:

$$Q = \begin{bmatrix} q_1^1 & q_2^1 & q_3^1 \\ q_1^2 & q_2^2 & q_3^2 \\ \cdots & \cdots & \cdots \\ q_1^n & q_2^n & q_3^n \end{bmatrix}, \quad \begin{cases} q_1^i + q_2^i + q_3^i = 1, \ i \in \{1, 2, \cdots, n\} \\ q_1^i, q_2^i, q_3^i \in \{0, 1\} \end{cases}, \tag{9}$$

where n is the total number of tasks. q_j^i is an execution vector of task t_i, $j = 1,2,3$ denote that t_i is executed locally or offloaded to the edge or the cloud, respectively. Accordingly, the matrix of task execution is regarded as the task scheduling plan

(2) When the task t_i is scheduled to be offloaded to the edge according to the execution vector (q_2^i=1,q_1^i =0,q_3^i=0), the optimal edge server should be selected by selection optimization algorithm (SOA) based on the location and the moving path of the mobile device

(3) Based on the models presented in Section 3, a novel task offloading and scheduling strategy (TOSS) is proposed to optimize the whole scheduling process from a global viewpoint. After all offloading decisions are made, workflow scheduling is conducted for all types of resources allocated in the MEC environment

4.1. Selection Optimization Algorithm. In this subsection, we present a selection optimization algorithm. Once the task t_i is scheduled to be offloaded to the edge, the selection optimization algorithm (SOA algorithm) is used to screen out the optimal offloading edge server according to the energy consumption, location, moving path, and velocity of mobile device.

According to the location and velocity of intelligent prosthesis, SOA algorithm is used to screen out the optimal offloading edge server within the maximum communication range of edge server. The inner loop of SOA algorithm is based on the greedy algorithm. Let the number of elements in candidate edge server set be D; the computation complexity of SOA algorithm is $O(D)$.

4.2. Task Offloading and Scheduling Strategy. In this section, we propose a task offloading and scheduling strategy based on particle swarm optimization algorithm (PSO).

4.2.1. Fitness Value. The fitness value is designed to evaluate the impact of offloading decision, which can calculate the latency and energy consumption of the mobile device. The smaller fitness value is regarded as the lower energy consumption of the task offloading and scheduling strategy. According to the energy consumption and latency model proposed in Section 3, we construct the fitness function by

$$\begin{cases} T_{\text{sum}} = T_m + T_{\text{es}} + T_c, \\ E_{\text{sum}} = E_m + E_{\text{es}} + E_c, \end{cases} \tag{10}$$

$$\text{fitness} = (f_1 \times E_{\text{sum}}) + \left(f_2 \times 10 \times E_{\text{sum}} \times \frac{T_{\text{sum}}}{T_{\text{respond}}} \right), \tag{11}$$

where E_{sum} is the total energy consumption, T_{sum} is the latency of the workflow, and T_{respond} is the deadline constraint. According to the motor coordination study of

Input $t_i = (\mathrm{IN}_i, \mathrm{OUT}_i, l_i)$, $t_i \in T$; a candidate set of edge servers S; mobile device MD;
Output Optimal edge server(S_{opti});
1 For each $S_j \in S$
2 { If$(d(Loc_j, Loc_m) < Dis_j)$
3 $ESset \longleftarrow S_j$; //Initialize $ESset$
4 }//End For each $S_j \in S$
5 If $(ESset = \mathrm{NULL})$
6 { Calculate the deferred execution time;
7 Update the position coordinates of the mobile device by Formula (7);
8 **Return** S_{opti};
9 }//End if
10 For each $S_j \in ESset$
11 { Update the position coordinate of the mobile device according to the initial position coordinate and the moving path by Formula (6) and Formula (7)
12 **If** $(d(Loc_j, Loc_{end}) > Dis_j)$
13 { Calculate the migration time T_{migr} ; //$T_{\mathrm{migr}} = \mathrm{OUT}_i / R_{\mathrm{migr}}$
14 }//End if
15 Calculate T_{es} and E_{es} by Formula (8);
16 Update the minimum energy consumption;
17 }//End For each $S_j \in ESset$
18 Update the position coordinates of the mobile device;
19 Return S_{opti};

Algorithm 1: Selection Optimization Algorithm (SOA).

Input Workflow $T = \{t_1, t_2, \cdots, t_n\}$; matrix of task execution Q; a set of edge servers, S; mobile device, MD; cloud server, CS; moving path;
Output The optimal task scheduling plan
1 For each $i \in [1, k]$
2 { initial the matrix of task execution Q_i and search speed v_i randomly;
3 calculate the energy consumption, the latency and the fitness value according to the matrix of task execution and the moving path;
4 }//End For each $i \in [1, k]$
5 While ($i <$ Maximum iterations)//Set maximum number of iterations
6 { update the matrix of task execution according to the search speed v_i;
7 **For each** $j \in [1, k]$
8 { calculate the energy consumption, latency and fitness value according to the matrix of task execution Q_j and the moving path;
9 }//End For each $j \in [1, k]$
10 Select the matrix of task execution Q_j with the lowest fitness value as the optimal task scheduling plan;
11 Update the inertia weight;
12 Update the search speed v_j;
13 }//End While
14 Update the optimal task scheduling plan;

Algorithm 2: Task Offloading and Scheduling Strategy (TOSS).

human beings, the deadline constraint is set as 0.6 s [3]. As shown in Formula (11), the fitness value can be calculated by two parts. One part is the total energy consumption when $T_{\mathrm{respond}} \geq T_{\mathrm{sum}}$, $(f_1 = 1,\ f_2 = 0)$; the other part is the total energy consumption when $T_{\mathrm{respond}} < T_{\mathrm{sum}}$, $(f_1 = 0,\ f_2 = 1)$. Accordingly, the penalty for unsatisfying constraint condition is regarded as the fitness value $(f_1 = 0, f_2 = 1)$. The penalty coefficient is set as 10, which is the same as the previous work [22].

4.2.2. Algorithm Description. TOSS algorithm mainly includes three steps: (1) first step: initialization of the task scheduling plan; (2) second step: iterative process; (3) third step: update the best task scheduling plan. The outer loop updates the task scheduling plan, the inertia weight, and the search speed. The inner loop calculates the fitness value of each task scheduling plan. Let the number of initial task scheduling plans, iterations, and tasks are I, k, and T, respectively. The computation complexity of TOSS algorithm is $O(IkT)$.

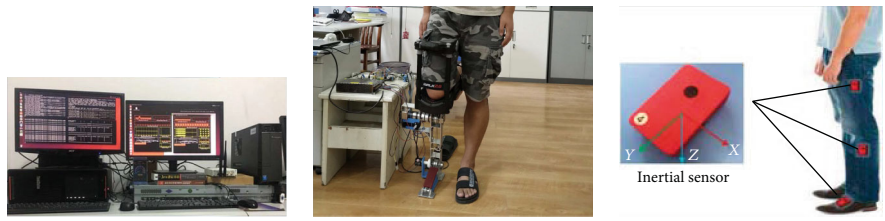

Figure 7: MEC environment and intelligent prosthesis used for experiments.

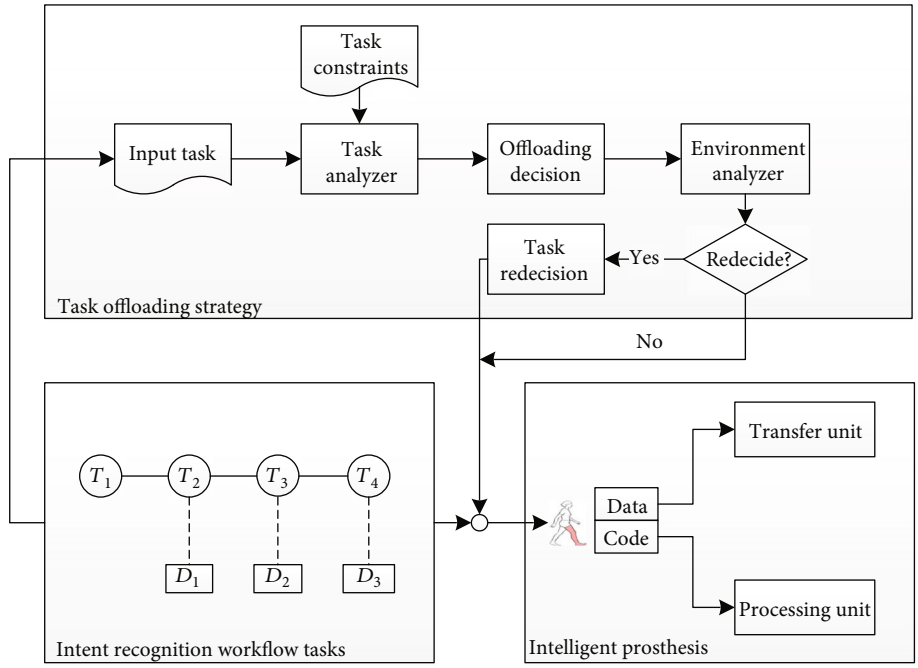

Figure 8: MEC-based computation management for intent recognition.

5. Experimental Results

In this section, we demonstrate the implementation of the proposed algorithm in MEC environment. As shown in Figure 7, the hardware environment is divided into two parts. The first part is consisted of computation resources. The cloud server is AlibabaCloud c6a (16 Core AMD EPYCTM Rome 7H12, 32 GB memory). The edge server is Dell PowerEdge XE2420 (8 Core Intel Xeon Bronze 3204, 16 GB memory), as shown in Figure 7(a). As can be seen in Figure 7(b), the second part is the intelligent prosthesis and the one we used for the experiments is iWALK 2.0(Virtex-7 Xilinx FPGA platform). The transmitting, receiving, idle, and executing powers of the mobile processor are 0.1 W, 0.05 W, 0.02 W, and 0.5 W.

As shown in Figure 7(c), three IMUs (Noitom Perception Legacy) are placed at the thigh, shank, and ankle of the healthy leg. Two edge servers have been deployed in a 200 m × 100 m area, and the moving path passes through this area.

We implement the proposed model by real-world application in intelligent prosthesis scenario which is intent recognition application. As can be seen in Figure 8, when the amputee is moving to the destination, the intelligent prosthesis begins to record the sensor data and send the intent recognition computation tasks to the edge servers or cloud server according to the task offloading strategy. The procedure is as follows: (1) the task offloading strategy first analyses the input data and tasks according to the characteristics and constraints of tasks. (2) The task strategy generates the offloading decision based on TOSS algorithm. (3) When the environment is changed (such as the moving path is changed), the offloading strategy decides the offloading decision again according to the environment analyser.

In the following experiments, we discuss the effect of the proposed algorithms, mainly focusing on the effect of fitness value, energy consumption, latency, and accuracy.

5.1. Fitness Value. In this experiment, we evaluate the performance of the proposed algorithm against CLOUD, EDGE, MOBILE, LoPRTC [23], and Edge4Sys [17] under varying number of frames. CLOUD, EDGE, and MOBILE indicate that the workflows are executed by cloud server, edge server, and mobile processor, respectively. Figure 8 shows the experiment results.

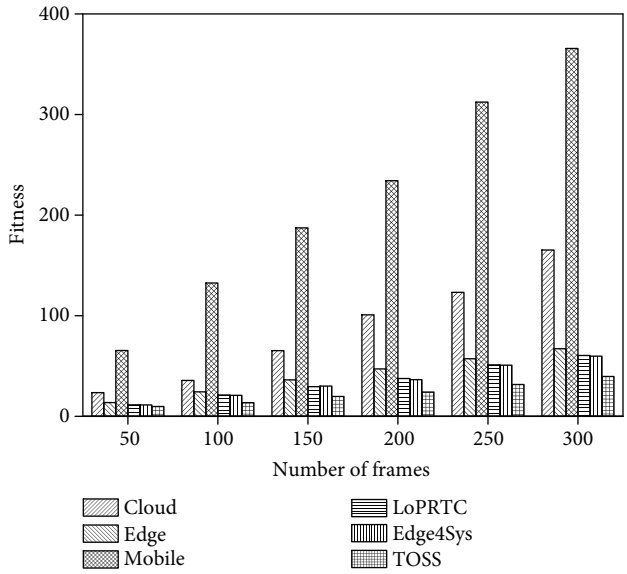

Figure 9: Comparison of fitness value.

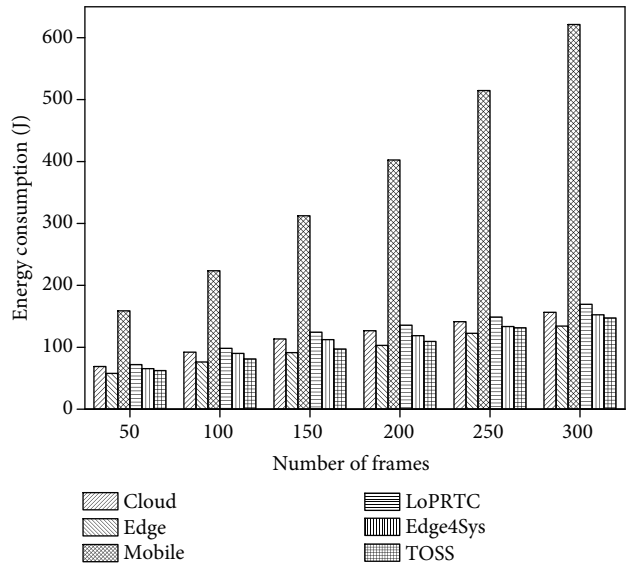

Figure 10: Comparison of energy consumption.

In Figure 9, as the number of frames increasing, the fitness value of all strategies increases. The fitness value of TOSS is apparently lower than the other strategies. Due to the fitness value which can reflect the effectiveness of task offloading and scheduling strategy, the experiment results show that TOSS can find the offloading and scheduling decision with lower energy consumption under the deadline constraint. In the meantime, it can be seen that the fitness value of MOBILE strategy is always much higher than the other strategies. The experimental results prove that the traditional mobile device-based intent recognition algorithm leads to the growth of the energy consumption.

5.2. Energy Consumption and Latency. In this experiment, we compare TOSS with CLOUD, EDGE, MOBILE, LoPRTC, and Edge4Sys in energy consumption and latency. Figures 10 and 11 show the results.

As can be seen in Figure 10, the energy consumption of MOBILE strategy is much higher than the other strategies, which can prove the experiment results shown in Figure 8 in another way. In the meantime, the energy consumption of EDGE strategy is always the lowest. However, the fitness value of EDGE strategy is higher than TOSS, LoPRTC, and Edge4Sys in Figure 9. The main reason is that EDGE strategy may miss the deadline constraint; the penalty for unsatisfying constraint condition leads to the growth of the fitness value. In addition, compared with Edge4Sys, LoPRTC, MOBILE, and CLOUD, the energy consumption returned by TOSS is 9.73%, 10.01%, 71.84%, and 15.98% less than these three strategies. The experimental results show that TOSS can effectively reduce the energy consumption under the deadline constraint.

The latency of five different strategies is shown in Figure 11. It can be seen that the latency of TOSS is always the lowest. When the number of frames is less than 100, MOBILE strategy is the second-lowest. However, when the number of frames exceeds 100, Edge4Sys strategy is the

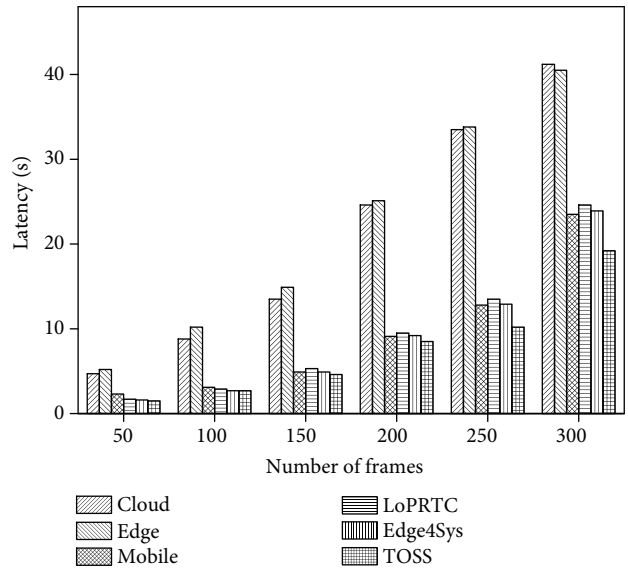

Figure 11: Comparison of latency.

second-lowest. It is easily to be noticed that, when the sizes of inputs and outputs are increasing, the latency generated by task transmission is less than that of computation locally.

Besides, although the latency of MOBILE strategy is less than CLOUD strategy and EDGE strategy, the energy consumption is much larger than the other strategies, as can be seen in Figure 10. The main reason is that task execution is the major energy consuming process; the size of data transmission has relatively less effect on the final results. In the meantime, the latency of CLOUD strategy and EDGE strategy is 70.44% and 77.73% higher than TOSS. It implies that data preprocessing can discard the useless information, which can save the communication time and reduce the computation load.

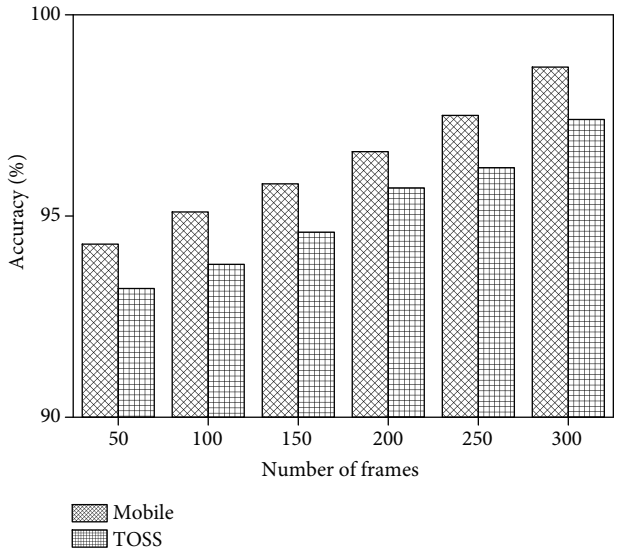

Figure 12: Comparison of accuracy.

In summary, the experiment results show that it is still difficult for cloud environment based intent recognition algorithm to achieve the purpose of real-time execution. The edge servers should coordinate with the other computation resources to ensure a good performance. For this reason, the proposed TOSS can always find the optimal task offloading and scheduling decision.

5.3. Testing Accuracy. In this experiment, we compare TOSS with MOBILE strategy in testing accuracy. Figure 12 shows the results.

In Figure 12, with the number of frames increased, the testing accuracy of the two strategies both increased as well. The accuracy of MOBILE strategy is a little higher. However, as can be seen in Figures 9 and 10, the energy consumption and latency of MOBILE strategy is much higher than TOSS. Compared with MOBILE strategy, TOSS can considerably reduce the energy consumption (71.84%) and latency (16.16%) at the expense of a relatively small decrease (1.36%) in accuracy, which is very practical in large-scale environment.

6. Conclusions and Future Work

In this paper, we consider the computation offloading problem of multiple heterogeneous edge servers in intelligent prosthesis scenario. The detail design of MEC-based task offloading model and mobility-aware task scheduling strategy are proposed to reduce the energy consumption and latency in a real-world MEC environment. The experimental results show that the proposed algorithms shows that the proposed algorithms can considerably reduce the energy consumption (71.84%) and latency (16.16%) at the expense of a relatively small decrease (1.36%) in accuracy.

In the future, we will develop more smart applications, such as path prediction and real-time scheduling. Furthermore, in an MEC system with multiple computation resources, reliability has a significant impact on task offloading and execution, and novel algorithms are needed to offloading the tasks on the trusty resource.

Data Availability

The experiment data supporting this experiment analysis are from previously reported studies, which have been cited, and are also included within the article.

Conflicts of Interest

The author declares that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work is supported by the Key Research and Development Program of Anhui Province, under grant no. 202004a05020010; the Key Program in the Youth Elite Support Plan in Universities of Anhui Province, under grant no. gxyqZD2020043; and Natural Science Foundation of Universities of Anhui Province, under grant no. KJ2020A0694.

References

[1] S. U. Ben-yue, N. I. Yu, S. H. E. N. G. Min, and Z. H. A. O. Li-li, "Intent recognition of power lower-limb prosthesis based on improved convolutional neural network," *Control and Decision*, 2020.

[2] B. H. Hu, E. J. Rouse, and L. J. Hargrove, "Using bilateral lower limb kinematic and myoelectric signals to predict locomotor activities: a pilot study," in *2017 8th International IEEE/EMBS Conference on Neural Engineering (NER)*, pp. 98–101, Shanghai, China, 2017.

[3] K. Nazarpour, A. R. Sharafat, and S. M. Firoozabadi, "Application of higher order statistics to surface electromyogram signal classification," *IEEE Transactions on Biomedical Engineering*, vol. 54, no. 10, pp. 1762–1769, 2007.

[4] A. Konstantin, T. Y. Yu, R. L. Carpentier, Y. Aoustin, and D. Farina, "Simulation of motor unit action potential recordings from intramuscular multi-channel scanning electrodes," *IEEE Transactions on Biomedical Engineering*, vol. 67, no. 7, pp. 2005–2014, 2020.

[5] He Huang, Fan Zhang, L. J. Hargrove, Zhi Dou, D. R. Rogers, and K. B. Englehart, "Continuous locomotion-mode identification for prosthetic legs based on neuromuscular-mechanical fusion," *IEEE Transactions on Biomedical Engineering*, vol. 58, no. 10, pp. 2867–2875, 2011.

[6] M. Liu, F. Zhang, and H. Huang, "An adaptive classification strategy for reliable locomotion mode recognition," *Sensors*, vol. 17, no. 9, article 2020, 2017.

[7] B. Y. Su, J. Wang, S. Q. Liu, M. Sheng, J. Jiang, and K. Xiang, "A CNN-based method for intent recognition using inertial measurement units and intelligent lower limb prosthesis," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 27, no. 5, pp. 1032–1042, 2019.

[8] O. P. Idowu, A. E. Ilesanmi, X. Li, O. W. Samuel, P. Fang, and G. Li, "An integrated deep learning model for motor intention recognition of multi-class EEG signals in upper limb amputees," *Computer Methods and Programs in Biomedicine*, vol. 206, no. 3, article 106121, 2021.

[9] A. Gautam, M. Panwar, A. Wankhede, S. P. Arjunan, and D. K. Kumar, "Locomo-net: a low-complex deep learning framework for sEMG-based hand movement recognition for prosthetic control," *IEEE Journal of Translational Engineering in Health and Medicine*, vol. 8, article 2100812, 2020.

[10] L. Lin, X. Liao, H. Jin, and P. Li, "Computation offloading toward edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1584–1607, 2019.

[11] J. Chen and X. Ran, "Deep learning with edge computing: a review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[12] S. K. Au, P. Dilworth, and H. Herr, "An ankle-foot emulation system for the study of human walking biomechanics," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006*, pp. 2939–2945, Orlando, FL, USA, 2006.

[13] X. Dongfang and Q. Wang, "Noninvasive human-prosthesis interfaces for locomotion intent recognition: a review," *Cyborg and Bionic Systems*, vol. 2021, article 9863761, pp. 1–14, 2021.

[14] D. Xu and Q. Wang, "On-board training strategy for IMU-based real-time locomotion recognition of transtibial amputees with robotic prostheses," *Frontiers in Neurorobotics*, vol. 14, no. 47, pp. 1–12, 2020.

[15] E. Li, Z. Zhou, and C. Xu, "Edge intelligence: on-demand deep learning modelco-inference with device-edge synergy," in *Proceedings of the Workshop on Mobile Edge Communications*, pp. 31–36, New York, NY, 2018.

[16] H. Li, H. Chenghao, J. Jiang, Z. Wang, Y. Wen, and W. Zhu, "JALAD: joint accuracy and latency-aware deep structure decoupling for edge-cloud execution," in *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 671–678, Singapore, 2018.

[17] H. Gao, X. Yi, X. Liu et al., "Edge4Sys a device-edge collaborative framework for MEC based smart systems," in *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Melbourne, VIC, Australia, 2020.

[18] A. Zomaya, A. Abbas, and S. Khan, "Modeling and simulation of distributed fog environment using fog net sim++(chapter 11)," in *Fog Computing: Theory and Practice*, John Wiley & Sons, 2020.

[19] A. Malik, T. Qayyum, and A. U. Rahman, "xFogSim: a distributed resource management framework for sustainable IoT services," *IEEE Transactions on Sustainable Computing*, vol. 23, no. 9, pp. 2005–2014, 2020.

[20] S. S. Shah, M. Ali, A. W. Malik, M. A. Khan, and S. D. Ravana, "VFog: a vehicle-assisted computing framework for delay-sensitive applications in smart cities," *IEEE Access*, vol. 7, pp. 34900–34909, 2019.

[21] J. Xu, X. Li, X. Liu et al., "Mobility-aware workflow offloading and scheduling strategy for mobile edge computing," in *Algorithms and Architectures for Parallel Processing. ICA3PP 2019*, S. Wen, A. Zomaya, and L. Yang, Eds., vol. 11945 of Lecture Notes in Computer Science, pp. 184–199, Springer, Cham, 2019.

[22] J. Hu, M. Jiang, Q. Zhang, Q. Li, and J. Qin, "Joint optimization of UAV position, time slot allocation, and computation task partition in multiuser aerial mobile-edge computing systems," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 7, pp. 7231–7235, 2019.

[23] T. Zhu, T. Shi, J. Li, Z. Cai, and X. Zhou, "Task scheduling in deadline-aware mobile edge computing systems," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4854–4866, 2019.