Article

Research on Decomposition and Offloading Strategies for Complex Divisible Computing Tasks in Computing Power Networks

Ping He 1,2, Jiayue Cang 1, Huaying Qi 1 and Hui Li 3,*

1 School of Management Sciences and Information Engineering, Hebei University of Economics and Business, Shijiazhuang 050061, China; pinghe@hueb.edu.cn (P.H.); cjyyjc2022@163.com (J.C.); huayingqi2023@163.com (H.Q.)
2 Hebei Cross-Border E-Commerce Technology Innovation Center, Shijiazhuang 050061, China
3 Baoding Technical College of Electric Power, Baoding 071000, China
* Correspondence: qazli_1006@126.com; Tel.: +86-173-6799-8262

Abstract: With the continuous emergence of intelligent network applications and complex tasks for mobile terminals, the traditional single computing model often fails to meet the greater requirements of computing and network technology, thus promoting the formation of a new computing power network architecture, of ‘cloud, edge and end’ three-level heterogeneous computing. For complex divisible computing tasks in the network, task decomposition and offloading help to realize a distributed execution of tasks, thus reducing the overall running time and improving the utilization of fragmented resources in the network. However, in the process of task decomposition and offloading, there are problems, such as there only being a single method of task decomposition; that too large or too small decomposition granularity will lead to an increase in transmission delay; and the pursuit of low-delay and low-energy offloading requirements. Based on this, a complex divisible computing task decomposition and offloading scheme is proposed. Firstly, the computational task is decomposed into multiple task elements based on code partitioning, and then a density-peak-clustering algorithm with an improved adaptive truncation distance and clustering center (ATDCC-DPC) is proposed to cluster the task elements into subtasks based on the task elements themselves and the dependencies between the task elements. Secondly, taking the subtasks as the offloading objects, the improved Double Deep Q-Network subtask offloading algorithm (ISO-DDQN) is proposed to find the optimal offloading scheme that minimizes the delay and energy consumption. Finally, the proposed algorithms are verified by simulation experiments, and the scheme in this paper can effectively reduce the task delay and energy consumption and improve the service experience.

Keywords: computing power networks; cloud–edge–end collaboration; task dependency; task decomposition; task offloading

1. Introduction

With the development of mobile communication and network technology, emerging network services such as augmented reality and autonomous driving have shown explosive growth. The task requirements of intensive computation and the real-time interaction of new applications necessitate a large amount of computing resources to fulfill the demand for ultra-reliable and low-delay communication. The resource limitations of mobile terminal devices themselves make it difficult to complete such applications alone. Cloud computing is an internet-based computing method with powerful computing power and large storage space, which can help to complete the processing and analysis of large amounts of data in a short period of time. Edge computing is a computing model that provides nearest-end services in the vicinity, offloading computing tasks to devices or nodes at the edge of the network for processing, thus reducing the delay of data transmission and making it suitable...
for computing tasks that require fast responses and processing. With the convergence of the internet and various industries, there is an explosive growth in computing demand, and the traditional cloud computing and edge computing architectures are no longer able to meet the huge computing demand for computing tasks, especially the limitations in real-time and data processing capabilities. As a result, the computing power networks arise.

As a new network architecture that combines the advantages of cloud computing and edge computing, the computing power network fully connects dynamically distributed computing and storage resources by integrating the computing power resources of the cloud, the edge and the terminals, and accomplishes unified coordination and scheduling through the network, breaks through the performance bottleneck of single-point computing power, and provides users with high-quality computing services. At the same time, task offloading technology is also an important means to reduce computing pressure; by effectively decomposing computing tasks and offloading them to different computing nodes, it helps to realize a distributed execution of tasks, thus reducing the overall running time of tasks and improving the utilization rate of fragmented computing resources in the network [1]. The computing power network provides a more efficient offloading service compared to the single offloading model. For computationally intensive tasks, if they are processed using cloud computing servers, although the computational resources are sufficient, due to the long transmission distance, they can easily lead to problems such as increased transmission delay and network congestion. If edge computing servers are used for processing, although the transmission distance can be effectively shortened and the processing delay reduced, there are problems, such as relatively limited computing resources and the low processing capacity of edge computing nodes. Therefore, through the computing power network, computing tasks can be decomposed and offloaded to appropriate cloud computing nodes or edge computing nodes, collaboratively utilizing the computing resources of the cloud, edge, and end to alleviate the computational pressure on the terminal equipment and reduce transmission delay, thereby improving the quality of user experience.

Current research on offloading strategies for divisible computing tasks can be categorized into data-oriented partitioning and code-oriented partitioning according to the types of tasks [2]. Data-oriented partitioning splits a computational task into multiple subtasks in a certain ratio or arbitrarily. The decomposition of tasks using these two models, although parallel, is characterized by the prevalence of dependencies between methods, components, or threads in most applications. Therefore, the assumption that tasks are arbitrarily divisible does not hold in most cases, and this approach is only applicable to the partitioning of a few data-based tasks and is not universal in reality. In fact, the execution of a task usually consists of multiple methods or threads, known as code partitioning [3]. Code partitioning exemplifies the application of symmetry principles to task structure analysis and helps to clearly identify the many repeating patterns and structural regularities of tasks. Code-oriented partitioning takes code divisibility as a precondition and takes the smallest software functional component that is code divisible as the basic unit of task decomposition, known as a task element. Typically, there are dependencies among task elements. If task elements are used as offloading objects and are offloaded to different computing nodes, this can easily result in increased task transmission delays and higher communication costs.

Aiming at the above problems encountered in the process of task decomposition and offloading, this paper proposes a new scheme for task decomposition and offloading, as shown in Figure 1. Firstly, the complex computing task is partitioned into multiple task elements with dependencies through code partitioning. The task elements are then reasonably clustered into subtasks, with the clustering itself being a symmetry operation. Finally, the subtasks are taken as the offloading objects, and the optimal offloading scheme is obtained with the objective of minimizing the system delay and energy consumption. The search process of the optimal offloading scheme is essentially a symmetry transformation search process, which tries to find the optimal solution that can balance the delay and energy consumption. In summary, the complex divisible computing task decomposition
and offloading scheme proposed in this paper is closely related to the symmetry principle. By applying the symmetry principle, it is possible to understand the structure and characteristics of the tasks more deeply, and thus perform task decomposition and offloading more effectively, which helps to improve the utilization of computing resources as well as reduce the overall delay and energy consumption.

Figure 1. Decomposition and offloading models for complex divisible computation tasks.

The main contributions of this paper are summarized as follows:

- This paper proposes a complex divisible computing task decomposition and offloading method. In order to solve the problem of divisible computing task decomposition with complex dependencies, we propose to improve the density-peak-clustering algorithm with an adaptive truncation distance and clustering center (ATDCC-DPC), which uses the Gini coefficient to adaptively determine the truncation distance as well as the elbow method to select the clustering-center points and determine the number of clusters. Using this algorithm to cluster the task elements into subtasks after the initial decomposition of the task, we take the subtask size as the task decomposition granularity standard, and take the subtasks as the offloading objects;

- For the divisible computing task offloading problem with complex dependencies, the local offloading model, edge offloading model, and cloud offloading model are constructed, respectively, and the subtask offloading problem is transformed into the problem of optimal strategy under the Markov decision process (MDP) with the objective of minimizing system delay and energy consumption. Then, an improved deep reinforcement learning (DRL)-based Double Deep Q-Network subtask offloading algorithm (ISO-DDQN) is proposed. This algorithm improves the DDQN by prioritizing the experience replay method based on importance sampling, which prioritizes the sampling of higher-value experience samples, and it is able to learn more effective offloading strategies in the experience to find the optimal offloading scheme that minimizes the delay and energy consumption.

The rest of the paper is organized as follows: Section 2 presents the current work on task decomposition, clustering algorithms, and task offloading. Section 3 presents the task decomposition method with complex dependencies and the proposed ATDCC-DPC task element clustering algorithm. Section 4 presents the ISO-DDQN subtask offloading algorithm for cloud–edge–end collaboration. Section 5 presents the experimental design
and validates the performance of the proposed algorithms. Finally, Section 6 concludes this paper.

2. Related Work

This section analyzes the latest research status of task decomposition, clustering algorithms, and task offloading, and points out the shortcomings in the current research.

2.1. Research Status of Task Decomposition

Fu et al. [4] studied the task decomposition problem in a cloud manufacturing environment with the objective of maximizing the internal coupling degree of subtasks and minimizing the correlation degree between subtasks, establish the fitness function, and solve the task decomposition model by using a simulated annealing algorithm, and obtain the final task decomposition results. Feng et al. [5] proposed an object–element model-based decomposition method for sequential tasks, considering cohesion and correlation for a cloud collaboration environment. Wei et al. [6] proposed a componentized service decomposition scheme based on a genetic algorithm to decompose tasks by controlling the number of subtasks. Wang et al. [7] componentized a decomposition of web applications and proposed a multi-granularity decomposition (MG-Decom) algorithm to produce multi-seeded task schemes with different granularities. The above studies mainly focus on cloud or edge computing environments, and as the complexity of applications grows, new constraints appear, such as computational and communication resources, as well as dependencies between task components and special requirements, which should be satisfied. All of them should be considered during task decomposition, otherwise an improper decomposition will greatly affect the task execution, resource overheads, and application performance.

2.2. Research Status of Clustering Algorithms

The research directions of clustering algorithms include division-based clustering, grid-based clustering, model-based clustering, and density-based clustering [8]. Division-based clustering algorithms, such as the classical K-Means algorithm [9] and K-modes algorithm [10], are simple and efficient, but prone to local optimization. They are less effective in clustering non-spherical clusters. Grid-based clustering algorithms [11] lack flexibility and adaptability, and for irregularly shaped clusters, grid divisions may not accurately capture the edges and shapes of the clusters. Model-based clustering methods [12] have stricter assumptions on data generation models and require higher-level a priori knowledge. Density-based clustering methods [13] can deal with clusters of any shape; the density peak cluster (DPC) [14] algorithm is simple in principle and more efficient. The algorithm’s core is based on local density and relative distance by drawing a decision diagram and artificially selected clustering center, and then completing the clustering. The algorithm can quickly select the center of the clustering, and the allocation of non-center points only needs to be carried out once, so as to realize the efficient clustering of data of arbitrary shapes.

Nevertheless, the algorithm still has obvious flaws. The first is that the local density calculation is overly dependent on the truncation distance, which is selected by human beings, and if it is made too large, the local density of each data point will be very large, resulting in poor discriminative properties. If it is made too small, it will be difficult to accurately identify the density attributes of each data point, leading to incorrect clustering results. Secondly, the manual selection of clustering centers is subjective and random, and it is very difficult to select clustering centers in decision diagrams for complex datasets. The unreasonable selection of clustering centers can also have an impact on the allocation results of non-clustering centers, which can easily lead to the error problem, and ultimately make the clustering accuracy lower. To solve these problems, scholars have proposed many improved algorithms.
In order to break the dependence of the DPC algorithm on fixed parameters, the calculation of local density is changed. Most of the current algorithmic literature \cite{15,16} uses the idea of k-nearest neighbors (KNNs) to replace the fixed truncation distance in the traditional DPC algorithm, which makes the datasets with different densities robust and improves the computational accuracy. However, the clustering effect is affected by the size of the $k$ value. If the $k$ value is too small, it results in samples with the same characteristics being divided into different clusters. If the $k$ value is too large, it results in samples with different characteristics being divided into the same clusters. Zhang et al. \cite{17} calculated the local density by a nonparametric kernel-density prediction method and adjusted the data point allocation using an adaptive reachable distance. Wang et al. \cite{18} proposed to use the whale optimization algorithm to iteratively find the truncation distance corresponding to maximizing the metrics. This algorithm reduces the dependence on fixed parameters, but the time complexity of the algorithm is greater than that of the traditional DPC algorithm. To address the multiple problems of manually selecting the clustering center, Flores and Villarreal \cite{19} proposed a method to automatically determine the clustering center by detecting the gap between the decision values between consecutive data points. Chen et al. \cite{20} proposed the domain adaptive density clustering algorithm, which sets a threshold for local density and relative clustering and specifies that points larger than the threshold are to be considered as clustering centers. The disadvantage is that the newly proposed parameters or thresholds of these improved algorithms need to be determined in advance.

2.3. Research Status of Task Offloading

Computing task offloading is the key research problem in computing network integration, but at present, the problem of task offloading is still less studied in the computing power networks. For the study of solving the task offloading problem in other networks, it is found that the existing research work widely uses particle swarm optimization (PSO), genetic algorithms (GAs) and other heuristic algorithms \cite{21–24}. Heuristic algorithms usually obtain a good approximation of the optimal solution, but they are online algorithms, and runtime decisions require a long solution time, which does not satisfy the requirements of delay-sensitive applications. Task offloading can be abstracted as the Markov decision process (MDP); reinforcement learning (RL) is mathematically based on MDP, and it is an offline algorithm with a short solution time for runtime decision making, but traditional RL is not adapted to high-dimensional state spaces and action spaces. DRL combines RL with deep neural networks (DNNs) for flexible and adaptive task offloading, which is a promising approach. The DRL algorithm is more adaptive and generalizable than traditional heuristic algorithms, and the decision time of the DRL algorithm is usually much shorter than that of traditional algorithms, which is an extremely important metric for scenarios that require instantaneous decision making. DRL learns effective strategies (i.e., mapping from environment states to actions) by interacting with the environment, thus maximizing numerical rewards. With the powerful representation capability of DNN, DRL can effectively solve complex decision-making problems with high-dimensional state and action spaces. Therefore, most of the latest research work uses DRL to solve the offloading problem.

Bali et al. \cite{25} conducted a systematic literature review of data offloading methods for IoT networks with edge and fog nodes in the form of a classical taxonomy. Heidari et al. \cite{26}, in order to facilitate the offloading of IoT devices, reduce delay, consume less energy, and extend battery life by utilizing edge architecture, proposed a Q-learning IoT-edge offloading technique, which adapts to the network dynamics to learn policies efficiently. Hao et al. \cite{27} studied the offloading problem based on time continuity in the multi-edge-node collaboration scenario, and proposed an offloading algorithm based on DRL, but this study assumed that the computing task was indivisible. Kuang and Chen \cite{28} studied the problem of task offload scheduling and server resource allocation in mobile-edge computing, and proposed a DRL-based task offload scheduling and resource
allocation algorithm with the aim of minimizing system delay and energy consumption. Wang et al. [29] used the Deep Q-Network (DQN) and Double Deep Q-Network (DDQN) methods to minimize delay and energy consumption for computational task and task allocation problems. Dai et al. [30] used the Deep Deterministic Policy Gradient (DDPG) algorithm to solve the joint optimization problem of computation offloading and resource allocation in order to minimize the energy consumption of the system. Tong et al. [31] proposed an adaptive task offloading and resource allocation algorithm in the mobile-edge-computing environment, using the DRL method to determine whether a task needs to be offloaded or not, which effectively reduces the average task response time and system energy consumption. Ren et al. [32] created a DRL model to deal with the dimensional catastrophe problem in the action space in fog computing, achieving a shorter delay and lower system energy consumption. Zou et al. [33] proposed a DRL-based task offloading algorithm for task offloading in edge scenarios to minimize the energy consumption problem, in order to balance the edge server load and reduce energy consumption and time overheads. From the above studies, it can be seen that the use of DRL to solve the task offloading problem has achieved satisfactory results. However, there are still some shortcomings in current research work. Firstly, when using the DRL method to solve the computational offloading problem, random sampling or simple sorting is used for the processing of experiences. This makes it difficult to distinguish valuable experiences and affects the convergence speed of the DRL model. Secondly, most offloading strategies use only one computational resource.

3. Task Decomposition

This section describes the proposed task decomposition method, which first decomposes a complex computational task into multiple task elements based on code partitioning. Then, the dependencies among task elements are analyzed, and the improved adaptive truncation distance and clustering-center density-peak-clustering algorithm (ATDCC-DPC) is proposed to cluster the task elements with dependencies into subtasks.

3.1. Initial Task Decomposition

In reality, most complex computing tasks tend to be composed of multiple components. Componentized software architectures provide a new dimension for deploying functional components of computing tasks at different computing nodes, thus improving the utilization of computing resources in the network. In addition, each functional component is designed for a single function and can be deployed, extended, and tested independently. With code partitioning, a computational task can be partitioned into multiple task elements with dependencies, and these dependencies of the task elements are modeled using directed acyclic graphs (DAGs) [34]. For example, the execution of an augmented-reality service typically involves steps such as raw video capture, camera calibration, alignment, tracking, rendering, and virtual information display. Each step can be regarded as a task element, as shown in the DAG task in Figure 2. Among them, the first task element, raw video capture, needs local devices to collect image information, and the last task element, the virtual information display, needs to present the final result to the user through local devices. Therefore, for tasks involving augmented reality and virtual display, the first task element and the last task element must be computed locally and cannot be offloaded to edge nodes or cloud centers for execution.

Due to dependencies between certain task elements, for example, the output data of the previous task element are often used as the input data for the subsequent task element. Therefore, if task elements are offloaded as a granularity, the data dependency between task elements deployed on different computing nodes introduces a new data transmission delay for the whole task execution process, which will lead to excessive communication overheads as well as an increased complexity of scheduling computational resources. To solve this problem, the improved adaptive truncated distance and clustering-center
density-peak-clustering algorithm (ATDCC-DPC) is proposed to cluster the task elements with dependencies into a series of subtasks, and then offload all task elements within the subtasks to the appropriate compute nodes as a whole task, which can minimize the introduction of additional delays while deploying the subtasks in a distributed manner.

![Task DAG](image)

**Figure 2.** Schematic diagram of task element clustering.

### 3.2. Task Element Clustering Constraints

The factors that affect the calculation of task correlation degree are the amount of data interacting between task elements, data dependency, communication bandwidth, communication delay, and special requirements. These five factors satisfy the following constraints when clustering task elements into the same subtask.

1. **The amount of data for inter-task-element interaction**: There is a certain degree of communication demand between task elements, and the information output of one task element serves as the input of another task element. Among them, the higher the amount of data interacting between two task elements, the closer the relationship between the two task elements, and the more likely they are to be in the same subtask. For example, in the face recognition business, there is a large amount of data to be compared between the face acquisition component and the face preprocessing component, so these two components should be executed together.

2. **Data dependency**: Different task elements may perform read/write operations on the same piece of data, and task elements that perform read/write operations on the same piece of data are placed in a subtask. Doing so reduces the repeated transmission of data and lowers the overall delay and bandwidth cost when the task is completed. For example, in a smart home scenario, the energy management unit is responsible for monitoring and controlling the use of home energy, such as power consumption and energy waste. The security monitoring unit, on the other hand, is responsible for monitoring the security status of the home, including intrusion detection, fire warning, and so on. If these two functional units are executed independently on different computing nodes, the same home environment data need to be transmitted to both nodes. However, if these two functional units are divided into the same subtask and executed on the same computing node, the repeated transmission of the same home environment data can be avoided, thus realizing data reusing and reducing network transmission delay.

3. **Communication bandwidth**: The bandwidth requirement for transmitting data is different for different task elements. For example, a task element needs to perform the next computation immediately after another task element signals to it, which requires a large bandwidth support to increase the transmission speed of the signal. To reduce the overall computational overhead, the task elements with large bandwidth requirements can be placed in the same subtask.

4. **Communication delay**: Assuming a significant communication delay between two task elements, if they are divided into different subtasks and offloaded to different computational locations, longer transmission delays will be incurred, increasing the task processing time and network load. Therefore, the higher the communication delay between two task elements, the more likely they are to be divided into the same subtask.
5. Special requirements: Generally, users do not want their sensitive data and critical information to be uploaded to the cloud data center; this is referred to as special requirements. In this case, consideration should be given to fulfilling the special requirements.

3.3. ATDCC-DPC Algorithm

Since the first task element and the last task element after the initial decomposition of the task need to be computed locally, the ATDCC-DPC algorithm performs the clustering operation on the remaining \( n - 2 \) task elements. Firstly, each task element is mapped into spatial particles based on the computation required by the task element itself and the dependency between the task elements. Secondly, the truncation distance is adaptively determined based on the Gini coefficient. Finally, the elbow method is used to realize the automatic selection of clustering-center points and the determination of the number of clusters.

3.3.1. Definition of Local Density and Relative Distance

Suppose a divisible computational task has \( n \) task elements and the set of task elements is denoted as \( T = \{ t_1, t_2, \ldots, t_n \} \). It is required that the first task element and the last task element must be computed locally, so a clustering operation needs to be performed on the remaining \( n - 2 \) task elements to obtain the corresponding subtasks. The correlations between task element \( t_i \) and task element \( t_j \) are denoted as \( TR_{ij}, i \neq j \) and \( i, j \in \{ 2, 3, \ldots, n - 1 \} \), as shown in Equation (1).

\[
TR_{ij} = \frac{P_1U_{ij} + P_2R_{ij} + P_3B_{ij} + P_4H_{ij}}{\lambda_{ij}},
\]

where \( U_{ij} \) is the amount of data interaction between task element \( t_i \) and task element \( t_j \), normalized between \([0,1]\), with 0 indicating no data interaction between task element \( t_i \) and task element \( t_j \), and 1 indicating data fully interacting between task element \( t_i \) and task element \( t_j \). \( R_{ij} \) is the data dependency between task element \( t_i \) and task element \( t_j \), normalized between \([0,1]\), with 0 indicating that the same block of data is not shared between task element \( t_i \) and task element \( t_j \), and 1 indicating that the task element \( t_i \) and task element \( t_j \) completely share the same data block. \( B_{ij} \) is the communication bandwidth requirement between task element \( t_i \) and task element \( t_j \), normalized between \([0,1]\), with a larger value indicating a high bandwidth requirement for both task element \( t_i \) and task element \( t_j \). \( H_{ij} \) is the communication delay between task element \( t_i \) and task element \( t_j \), normalized between \([0,1]\), with a larger value indicating a higher communication delay between task element \( t_i \) and task element \( t_j \). \( P_1, P_2, P_3, P_4 \) are the proportions of data volume, data dependency, communication bandwidth, and communication delay requirement for inter-task-element interactions, respectively, i.e., \( P_1 + P_2 + P_3 + P_4 = 1 \). \( \lambda_{ij} \) denotes the special demand between task element \( t_i \) and task element \( t_j \), normalized between \([0,1]\). The stronger the special demand, the smaller the value. When there is no dependency between two task elements, \( U_{ij}, R_{ij}, B_{ij}, H_{ij} \) are 0 and the value of \( \lambda_{ij} \) is 1.

The clustering weight coefficient \( W_i \) for task element \( t_i \) is expressed as shown in Equation (2).

\[
W_i = \frac{D_i}{\overline{D}},
\]

where \( D_i \) is the amount of computation required for task element \( t_i \), \( \overline{D} \) is the average amount of computation required for all task elements, and \( \overline{D} = \frac{\sum_{i=1}^{n-2} D_i}{n-2} \). The clustering weight coefficient \( W_i \) for task element \( t_i \) reflects the effect of the amount of computation required for the task element on task decomposition. The smaller the amount of computation required by the task element itself, the larger the value of \( W_i \) and the more likely it is to form a cluster with other task elements.

Factors affecting task decomposition include the amount of computation required for the task elements and the degree of correlation between the task elements. In order to
be able to reasonably cluster subtasks, the clustering weight coefficients and the degree of correlation between two task elements are mapped to the distance between two task elements, as shown in Equation (3).

\[ d_{ij} = \frac{W_i W_j}{T R_{ij}^2} \]  

Then, the local density \( \rho_i \) of task element \( t_i \) is calculated as shown in Equation (4).

\[ \rho_i = \sum_{j \neq i} \exp\left(-\frac{d_{ij}^2}{d_c^2}\right), \]  

where \( d_c \) denotes the truncation distance, which needs to be set manually, and the setting of \( d_c \) value will affect the clustering result. In this paper, we assume that the initial value of \( d_c \) is the minimum distance between all task elements, and \( d_c = \min(d_{ij}) \).

Based on Equation (3), the relative distance of task element \( t_i \) to the task element that has a greater local density than it and is the closest is denoted by \( \delta_i \), as shown in Equation (5).

\[ \delta_i = \min_{j(\rho_j > \rho_i)} (d_{ij}) \]  

Assuming that the task element \( t_i^* \) has the maximum local density, its relative distance is expressed as shown in Equation (6).

\[ \delta_i^* = \max_j (d_{i^*j}) \]  

Organizing Equations (5) and (6), the relative distance of all task elements can be expressed as shown in Equation (7).

\[ \delta_i = \begin{cases} \min_{j(\rho_j > \rho_i)} (d_{ij}), & i \neq i^* \\ \max_j (d_{ij}), & i = i^* \end{cases} \]  

### 3.3.2. Adaptive Selection of Truncation Distance

Traditional DPC algorithms usually determine the truncation distance \( d_c \) through experimental verification or empirically, with strong uncertainty. In clustering algorithms, the merit of a clustering result can be assessed by its purity. The Gini coefficient represents the impurity of the data. The smaller the Gini coefficient is, the smaller the impurity is, and the higher the purity is, the smaller the uncertainty is, and the more likely it is that the task elements will be clustered in reasonable subtasks. Therefore, when the Gini coefficient is the smallest, a better the clustering result is produced. The Gini coefficient is expressed as shown in Equations (8) and (9).

\[ p_i = \frac{\rho_i \times \delta_i}{\sum_{i=2}^{n-1} \rho_i \times \delta_i}, \]  

\[ G = 1 - \sum_{i=2}^{n-1} (p_i)^2, \]  

where \( p_i \) is denoted as the proportion of \( \rho_i \times \delta_i \) of a task element \( t_i \) to the \( \rho_i \times \delta_i \) of all task elements. During the truncation distance calculation, \( d_c \) is gradually increased to calculate the new Gini coefficient, and when the Gini coefficient value is the smallest, \( d_c \) at this point is defined as the final truncation distance.
3.3.3. Adaptive Selection of Clustering-Center Points and Determination of the Number of Clusters

In order to avoid the subjectivity and randomness brought about by manually selecting clustering-center points, which have an impact on the results of assigning non-clustering-center points and lead to low clustering accuracy, in this paper, the decision function for selecting the clustering-center points is defined, and the elbow method is used to select the clustering-center points, as well as to determine the number of clusters. Once the cluster center points are determined, the number of clusters is determined, and the sum of the number of cluster center points is the same as the number of clusters. The elbow method is often used to determine the optimal number of clusters, and the idea is that with the increase in the number of clusters, the intra-cluster variance gradually decreases, while the inter-cluster variance gradually increases, which leads to a gradual improvement in the clustering effect. However, too many clusters will lead to overfitting, and the clustering effect will be reduced or not have practical significance. Therefore, when increasing the number of clusters no longer significantly reduces the intra-cluster variance, the optimal number of clusters can be considered to have been determined, and this point is called the “elbow point”. The decision function $\gamma_i$ for selecting the clustering-center points is defined as shown in Equation (10).

$$\gamma_i = \rho_i \times \delta_i$$  \hspace{1cm} (10)

In order to avoid the decision function being affected by a single variable due to an uneven distribution of local densities or relative distances, the decision function $\gamma_i$ is normalized, as shown in Equation (11).

$$\gamma_i = \frac{\rho_i}{\rho_{\text{max}} - \rho_{\text{min}}} \times \frac{\delta_i}{\delta_{\text{max}} - \delta_{\text{min}}}$$  \hspace{1cm} (11)

The larger the $\gamma_i$ value of a task element, the more likely it is to be a clustering-center point. Therefore, the $\gamma_i$ value of each task element is calculated and sorted in descending order, and the sorted task elements are relabeled, assuming that they are labeled with 1, 2, ..., $n-2$, thus obtaining a decision map for the decision function $\gamma$.

In the decision diagram, a line is connected between the first task element and the last task element to obtain a straight line $y$. Since the cluster center points and the non-cluster center points have significantly different distances from the straight line $y$ in the decision diagram, which results in the sample point after the elbow point being the farthest away from the straight line, the sample point is defined as the non-cluster center point. Instead, the task element represented by the elbow point and all points before the elbow point serves as the clustering-center point, and the number of elbow points and all points before the elbow points is the corresponding optimal number of clustering-center points. Therefore, before clustering, it is necessary to first solve for the sample point with the farthest distance to the straight line $y$, and define the previous sample point of the farthest distance sample point as the elbow point of the decision diagram.

As shown in Figure 3, in the decision diagram, $A$ is the first task element, $B$ is the last task element, and the expression of the straight line $AB$ is $y = kx + b$. $k$ is the slope of the straight line $AB$, and $b$ denotes the corresponding intercept of the straight line $AB$. Solve for the sample point with the largest distance from the straight line $AB$, whose previous sample point labeling is indicated as shown in Equation (12), which is the elbow point.

$$F = \{\arg_i \max_{i} \frac{kx_i - y_i + b}{\sqrt{1 + k^2}} - 1\},$$  \hspace{1cm} (12)

where $x_i$ denotes the horizontal coordinates of sample point $i$ and $y_i$ denotes the vertical coordinates of sample point $i$. In Figure 3, $P$ is the elbow point and $M$ is the point after the elbow point with the maximum distance to the straight line $AB$. Point $P$ and the previous sample points are the cluster center points, and the sum of the number of cluster center points is the number of clusters, so the label $F$ of point $P$ is the final number of clusters. For
the remaining non-clustered center points, they are assigned one by one to the cluster where the nearest clustered center point is located to complete the clustering process. Therefore, the optimal decomposition granularity for the task is defined as follows:

**Definition 1.** Task optimal decomposition granularity. For the \( n - 2 \) task elements \( \{t_2, t_3, \ldots, t_{n-1}\} \) that need to be clustered, calculate the value of \( \gamma_i, i \in \{2, 3, \ldots, n - 1\} \), draw the \( \gamma_i \) descending discriminant graph, and, according to the elbow method, determine the elbow point \( P \). When \( \gamma_i \geq \gamma_P \), the task element represented by \( \gamma_i \) serves as the clustering center point, and all the task elements represented by the clustering center point are selected. For task elements that are not cluster center points, divide them into clusters where the nearest cluster center point is located, respectively. All task elements in the cluster, where each clustering center point is located, are treated as subtasks, with the subtask as the optimal decomposition granularity.

Because the first task element and the last task element are not involved in clustering, \( F + 2 \) is the number of subtasks after task decomposition.

### 3.3.4. Overall Flow of the Algorithm

The ATDCC-DPC algorithm process is shown in Algorithm 1.

**Algorithm 1. ATDCC-DPC Algorithm**

**Input:** Task elements set \( T = \{t_1, t_2, \ldots, t_n\} \); Task element calculation volume \( C_i \); The amount of interactive data between task elements \( U_{ij} \); Data dependency \( R_{ij} \); Communications bandwidth \( B_{ij} \); Communications delay \( H_{ij} \)

**Output:** Clustering subtasks set \( TS = \{TS_1, TS_2, \ldots, TS_{F+2}\} \)

1. Calculate the correlation \( TR_{ij} \) between task element \( t_i \) and task element \( t_j \) by Equation (1), \( i \in j \) and \( i,j \in \{2, 3, \ldots, n - 1\} \)
2. Calculate the clustering weight coefficient \( W_i \) for task element \( t_i \) by Equation (2)
3. The truncation distance \( d_e \) is set to the minimum distance, and the distance mapping \( d_{ij} \) between task element \( t_i \) and task element \( t_j \) is calculated by Equation (3)
4. Calculate the local density \( \rho_i \) of the task element \( t_i \) by Equation (4)
5. Calculate the relative distance \( d_{ij} \) of task element \( t_i \) by Equation (7)
6. The value of Gini coefficient \( G \) is calculated by Equations (8) and (9). Gradually increase the value of \( d_e \) and calculate \( G \); When the Gini coefficient \( G \) is the lowest, the final truncation distance \( d_e \) is obtained
7. Calculate the descending decision diagram that draws the decision function \( \gamma \) by Equations (10) and (11)
8. Using the elbow method to calculate the cluster center points and the number \( F \) by Equation (12)
9. Assignment of task elements to non-clustered center points completes the clustering process
10. Get the set of subtasks \( TS = \{TS_1, TS_2, \ldots, TS_{F+2}\} \)
4. Task Offloading

The terminal and edge nodes in the computing power network can meet the demand of the short delay of the task, but there is a limitation in the limited node computing-power resources. The cloud center is rich in computing power, but there is the problem of long transmission delays caused by the long distance from the terminal. Most of the offloading strategies in current research use only one computing resource. In order to successfully complete the task and minimize the cost, this paper requires cloud–edge–end computing power to work together to complete the whole task computation. When the task is decomposed into multiple subtasks and offloaded to different nodes for computation, the fragmented resources in the computing power network can be fully utilized to improve the task execution efficiency and reduce the cost overhead.

After the task decomposition, the subtasks are regarded as units of task offloading in the computing power network, and the computing power resource requirements and storage resource requirements of the subtasks need to be re-metered. Since the first subtask $TS_1$ and the last subtask $TS_{F+2}$ are the first task element and the last task element of this divisible computational task, which needs to be computed locally, they do not need to be re-metered. As for the other subtasks, the amount of computational data is the sum of the amount of computational data of all task elements that constitute the subtask. Assuming that subtasks $TS_i = \{t_2, t_3, t_w\}, 1 < w < n, w \neq 2, 3,$ and $1 < i < F + 2$ consist of three task elements, $t_2, t_3,$ and $t_w,$ the computational data volume $D_{TS_i}$ of subtask $TS_i$ is shown in Equation (13).

$$D_{TS_i} = D_2 + D_3 + D_w$$ (13)

The subtask computational resource requirement is the sum of the computational resource requirements of all task elements that make up the subtask, and the computational resource requirement $C_{TS_i}$ of subtask $TS_i$ is shown in Equation (14).

$$C_{TS_i} = C_2 + C_3 + C_w$$ (14)

The subtask storage resource requirement is the sum of the storage resource requirements of all task elements that make up the subtask, and the storage resource requirement $S_{TS_i}$ of subtask $TS_i$ is as seen in Equation (15).

$$S_{TS_i} = S_2 + S_3 + S_w,$$ (15)

where $D_w$ denotes the amount of computational data of the task element $t_w$, $C_w$ denotes the computational resource requirement of the task element $t_w$, and $S_w$ denotes the storage resource requirement of the task element $t_w$.

Assume that the cloud–edge–end collaborative computing power network contains $N$ edge nodes $ED = \{ed_1, ed_2, \ldots, ed_N\}$, a terminal device (UE), and a cloud center server (CC). Where each edge node $ed_i \in ED$ is configured with a server that can provide computing power, this is denoted as $f^{edge}_i$. The local device also has a certain amount of computing power, denoted as $f^{local}$. The cloud center has sufficient computing power, denoted as $f^{cloud}$.

4.1. System Model

Assuming that the subtasks are executed locally, the round-trip transmission delay and transmission energy consumption are 0. In addition to local computation, the terminal can offload the subtasks to be executed at a cloud center or an edge computing node. The communication models for offloading to different locations are described below, respectively.

4.1.1. Local Offloading Model

If subtask $TS_g \in TS$ is executed via the local device, the subtask transmission delay and transmission energy consumption are 0. Only the computation delay and computation
energy consumption of subtask $T_S^g$ need to be calculated. Then, the local computation delay is shown in Equation (16).

$$T^L_{S^g} = \frac{\mu \cdot D_{T_S^g}}{f_{\text{local}}} \quad (16)$$

Among them, $\mu$ represents the number of CPU cycles required for unit bit task execution, and the unit is cycles/bit. $D_{T_S^g}$ represents the value of the amount of data that subtask $T_S^g$ needs to calculate, then $\mu D_{T_S^g}$ represents the number of CPU cycles required to calculate subtask $T_S^g$. $f_{\text{local}}$ represents the computing power of the local device. The computing energy consumption of the device depends on the structure of the CPU chip, the computing power of the CPU, and the amount of calculation required for the task [35]. The energy consumption model of the local device in each CPU cycle is a superlinear function of the execution frequency, which is expressed as Equation (17).

$$\zeta = \kappa_L (f_{\text{local}})^2 \quad (17)$$

where $\kappa_L$ denotes the CPU structure energy constant factor of the local device. Therefore, the computational energy consumption of subtask $T_S^g$ when performing computations for the local device is shown in Equation (18).

$$E^L_{S^g} = \zeta \cdot \mu \cdot D_{T_S^g} = \kappa_L (f_{\text{local}})^2 \mu D_{T_S^g} \quad (18)$$

4.1.2. Edge Offloading Model

If the UE decides to offload the subtask to the edge server for execution, the UE needs to transmit the data of the subtask to the edge server over the wireless link. The transmission rate from the UE to the edge server can be defined by Shannon’s formula, as shown in Equation (19):

$$R_i = B \cdot \log_2(1 + \frac{p \cdot h_i}{\sigma^2}) \quad (19)$$

where $B$ is the bandwidth of the wireless channel, $p$ is the transmission power of the UE, $h_i$ is the channel gain from the UE to the edge server $ed_i \in ED$, and $h_i = 38.46 + 20 \log_{10}(d_i)$, where $d_i$ denotes the distance between the end device and the edge computing node [36]. $\sigma^2$ denotes the Gaussian white noise power. Then, the transmission delay of the subtask $T_S^g$ offloading to the edge server $ed_i \in ED$ is shown in Equation (20).

$$T_{\text{off}}^{g} = \frac{D_{T_S^g}}{R_i} \quad (20)$$

Assuming that the computational power of the edge server $ed_i \in ED$ is $f^\text{edge}_i$, the computational delay of the subtask $T_S^g$ on the edge server $ed_i \in ED$ is shown in Equation (21).

$$T_{\text{exe}}^{g} = \frac{\mu \cdot D_{T_S^g}}{f^\text{edge}_i} \quad (21)$$

Because the volume of transmission data of the subtask computation results is much smaller than that of the subtask itself, and the transmission rate of the network downlink is higher than that of the uplink, the backhaul delay of the computation results is usually not considered in the offloading process [30]. Therefore, the edge offloading delay $T^E_{S^g}$ is the sum of the transmission delay $T_{\text{off}}^{g}$ from the UE to the edge server $ed_i \in ED$ and the computation delay $T_{\text{exe}}^{g}$ in the edge server $ed_i \in ED$, as shown in Equation (22).

$$T^E_{S^g} = T_{\text{off}}^{g} + T_{\text{exe}}^{g} = \frac{D_{T_S^g}}{R_i} + \frac{\mu \cdot D_{T_S^g}}{f^\text{edge}_i} \quad (22)$$
The edge offload energy consumption of the subtask $T_{Sg}$ offloading to the edge server $ed_i \in ED$ is the sum of the transmission energy and computation energy, where the transmission energy is shown in Equation (23).

$$E_{g,i}^{\text{off}} = p \cdot T_{g,i}^{\text{off}} = \frac{p \cdot D_{T_{Sg}}}{R_i}$$  \hspace{1cm} (23)$$

And the computational energy consumption of the subtask $T_{Sg}$ on the edge server $ed_i \in ED$ is shown in Equation (24).

$$E_{g,i}^{\text{exe}} = \kappa_E \left( \frac{c_{\text{edge}}}{j_i} \right)^2 \mu D_{T_{Sg}}$$  \hspace{1cm} (24)$$

where $\kappa_E$ denotes the CPU structure energy constant factor of the edge server $ed_i \in ED$. Then, the edge offload energy consumption of the subtask $T_{Sg}$ is shown in Equation (25).

$$E_{g,i}^E = E_{g,i}^{\text{off}} + E_{g,i}^{\text{exe}} = \frac{p \cdot D_{T_{Sg}}}{R_i} + \kappa_E \left( \frac{c_{\text{edge}}}{j_i} \right)^2 \mu D_{T_{Sg}}$$  \hspace{1cm} (25)$$

4.1.3. Cloud Offloading Model

If the subtask $T_{Sg}$ is offloaded to the cloud computing center, the transmission delay is composed of two parts: the transmission delay of the subtask offloading from the UE to the edge server $ed_i \in ED$ and the transmission delay of the edge server $ed_i \in ED$ offloading the subtask to the cloud center through the backbone network, which can be expressed as shown in Equation (26).

$$T_{g,i}^{\text{off}} = \frac{D_{TSg}}{R_i} + \frac{D_{TSg}}{R_c}$$  \hspace{1cm} (26)$$

where the data transmission rate of the backbone network is denoted by a constant value $R_c$. The computational delay of the subtask $T_{Sg}$ is shown in Equation (27).

$$T_{g,i}^{\text{exe}} = \frac{\mu \cdot D_{T_{Sg}}}{f_{\text{cloud}}},$$  \hspace{1cm} (27)$$

where $f_{\text{cloud}}$ is the computing capacity of the cloud center. Then, the delay of the subtask $T_{Sg}$ offloading to the cloud center is the sum of the transmission delay and computation delay, as shown in Equation (28).

$$\tau_{Sg}^c = T_{g,i}^{\text{off}} + T_{g,i}^{\text{exe}} = \frac{D_{T_{Sg}}}{R_i} + \frac{D_{T_{Sg}}}{R_c} + \frac{\mu \cdot D_{T_{Sg}}}{f_{\text{cloud}}},$$  \hspace{1cm} (28)$$

The offloading energy consumption of the subtask $T_{Sg}$ offloading to the cloud center is the sum of the transmission energy and computation energy. The transmission energy consumption still consists of two parts: the transmission energy consumption of the subtask offloading from the UE to the edge server $ed_i \in ED$ and the transmission energy consumption of the edge server $ed_i \in ED$ offloading the subtask to the cloud center through the backbone network, as shown in Equation (29).

$$E_{g}^{\text{off}} = p \cdot \left( \frac{D_{T_{Sg}}}{R_i} + \frac{D_{T_{Sg}}}{R_c} \right)$$  \hspace{1cm} (29)$$

And the computational energy consumption of the subtask $T_{Sg}$ in the cloud center is shown in Equation (30).

$$E_{g}^{\text{exe}} = \kappa_c \left( \frac{c_{\text{cloud}}}{j_i} \right)^2 \mu D_{T_{Sg}}$$  \hspace{1cm} (30)$$

where $\kappa_c$ denotes the CPU structure energy constant factor of the cloud server. Therefore, the cloud center offload energy consumption of the subtask $T_{Sg}$ is shown in Equation (31).
\[
E_g^C = E_g^{off} + E_g^{exe} = p \cdot \left( \frac{D_{TS_g}}{R_i} + \frac{D_{TS_g}}{R_c} \right) + \kappa C \cdot \left( f^{cloud} \right)^2 \mu D_{TS_g} 
\] (31)

4.2. Problem Model

The computational tasks generated by the end devices are partitioned into subtasks, and their relationships are mapped into a DAG. The subtasks can be chosen to be executed via the local device or offloaded to the edge computing nodes and cloud centers for execution in order to realize the parallel execution of all the subtasks on multiple nodes, with the stipulation that the first subtask and the last subtask need to be executed via the local device. In this section, the subtask offloading problem is modeled with the optimization objective of minimizing the system delay and energy consumption as much as possible.

The binary vector \( a_i = [a_{i,1}, a_{i,2}, \ldots, a_{i,N+1}, a_{i,N+2}] \) represents the offloading decision of the subtask \( TS_i \). The offloading decision of all subtasks consists of two parts: whether the subtask needs to be offloaded or not and to which compute node. Suppose that all the computing nodes in the computing power network are represented by the set \( CN = \{cn_1, cn_2, cn_{N}, cn_{N+1}, cn_{N+2}\} \), where the first computing node \( cn_1 \) in the \( CN \) set is denoted as the local device UE, the last computing node \( cn_{N+2} \) is the cloud center CC, and \( cn_2, cn_{N}, cn_{N+1} \) in the middle is the \( N \) edge computing nodes. \( a_{i,j} \in \{0, 1\} \) indicates whether a compute node \( cn_j \) is a compute node for the subtask \( TS_i \), where \( a_{i,j} = 1 \) indicates that the subtask \( TS_i \) is offloaded to the compute node \( cn_j \) for execution, and vice versa. \( a_{i,1} \) indicates whether the subtask \( TS_i \) is offloaded to be executed via a local device, \( a_{i,N+2} \) indicates whether the subtask \( TS_i \) is offloaded to be executed via a cloud center, and \( a_{i,j} \) (where \( j \in \{2, 3, \ldots, N+1\} \)) indicates whether the subtask \( TS_i \) is offloaded to an edge compute node, as well as to be executed via that edge server. Each subtask can only be offloaded to one compute node for execution, so \( a_{i,j} \) satisfies the condition, as shown in Equation (32).

\[
\sum_{j=1}^{N+2} a_{i,j} = 1 
\] (32)

Since the first subtask \( TS_1 \) and the last subtask \( TS_{F+2} \) need to be executed locally, the offloading decision for the first subtask is \( a_1 = [1, 0, \ldots, 0, 0] \), and the offloading decision for the last subtask is \( a_{F+2} = [1, 0, \ldots, 0, 0] \). Denote the subtask offloading decision for a divisible computational task using a binary matrix \( (a_1, a_2, \ldots, a_{F+2}) \).

The divisible computing task issued by the terminal device consists of multiple subtasks, so the total execution completion delay of this task is the maximum of all subtasks’ offloading delays, as shown in Equation (33).

\[
T = \max_{i=1, \ldots, F+2} \left\{ T_i^L, T_i^E \left( j \in \mathbb{N} \right), T_i^C \right\}, 
\] (33)

where \( F + 2 \) is the number of subtasks. Then, the total energy consumption of the task is the sum of the energy consumption of each subtask, as shown in Equation (34).

\[
E = \sum_{i=1}^{F+2} \left( a_{i,1} \cdot E_i^L + \sum_{j=2}^{N+1} a_{i,j} \cdot E_{i,j-1}^E + a_{i,N+2} \cdot E_i^C \right) 
\] (34)

Thus, the total cost of performing the task is shown in Equation (35).

\[
\text{COST} = \alpha T + (1 - \alpha) E, 
\] (35)

where \( \alpha \in [0,1] \) denotes the delay weight of the computational task, and the value of \( \alpha \) can be adjusted according to the different requirements of delay and energy consumption to meet the needs of different application services. The optimization problem is formulated as a minimization cost problem, as shown in (36).
\[
\text{min } \text{COST} \\
\text{s.t. } c_1: a \in [0, 1] \\
c_2: a_1 = [1, 0, \ldots, 0, 0] \text{ and } a_{F+2} = [1, 0, \ldots, 0, 0] \\
c_3: \sum_{i=2}^{F+1} b_{ij} \cdot C_{STS} \leq C_j \quad j = 2, 3, \ldots, N + 1 \\
c_4: \sum_{i=2}^{F+1} b_{ij} \cdot \bar{S}_{TSi} \leq S_j \quad j = 2, 3, \ldots, N + 1 \\
c_5: T \leq T_D
\]

\[ (36) \]

\( c_1 \) constrains the delay and energy consumption weights, and \( c_2 \) ensures that the first subtask and the last subtask of a computational task are executed via the local device. \( c_3 \) indicates that the computational resource requirements of the subtasks offloaded to the edge server cannot exceed the computational resources available via the current edge server. \( c_4 \) indicates that the storage resource requirements of the subtasks offloaded to the edge server cannot exceed the storage resources available via the current edge server. \( c_5 \) indicates that the task’s actual completion delay cannot exceed the maximum tolerable delay \( T_D \) of the task.

4.3. ISO-DDQN Algorithm

In this section, a subtask offloading algorithm based on an improved DDQN is proposed. Firstly, the problem model is transformed into an MDP by defining the elements of the system state, action space, and reward function. In response to the existing studies that use the DRL method to solve the computational offloading problem, random sampling or simple sorting is used for the processing of regular and valuable experiences, which are difficult to distinguish between, and affects the convergence speed of the DRL model. In this section, the DDQN is improved with a prioritized experience replay method based on importance sampling, which prioritizes the sampling of higher-value experiences. Finally, the improved DDQN is used to solve the offloading policy with the objective of minimizing delay and energy consumption.

4.3.1. MDP of Subtask Offloading

In MDP, an intelligent body interacts with the environment cyclically by taking actions to change its state and obtaining rewards. The subtask offloading problem is transformed into the optimal policy problem under MDP.

The state space describes the information observed by the end device in time slot \( t \), the state \( s_t \in S \) at moment \( t \). All possible states form the state space \( S \), defined in Equation (37).

\[ s_t = \{ D(t), C(t), S(t) \} \]

Among them, \( D(t) = \{ D_1, D_2, \ldots, D_{F+2} \} \) represents the calculated data value of the subtasks. \( C(t) = \{ C_1(t), C_2(t), \ldots, C_{N+1}(t), C_{N+2}(t) \} \) represents the available computing resources of local devices, edge computing nodes, and the cloud center at time \( t \), \( C_1(t) \) represents the available computing resources of local devices, \( C_{i,j=2,3,\ldots,N+1}(t) \) represents the available computing resources of \( N \) edge computing nodes, and \( C_{N+2}(t) \) represents the available computing resources of the cloud center. \( S(t) = \{ S_1(t), S_2(t), \ldots, S_{N+1}(t), S_{N+2}(t) \} \) denotes the available storage resources of local devices, edge computing nodes, and the cloud center at time \( t \), \( S_1(t) \) denotes the available storage resources of local devices, \( S_{i,j=2,3,\ldots,N+1}(t) \) denotes the available storage resources of \( N \) edge computing nodes, and \( S_{N+2}(t) \) denotes the available computing resources of the cloud center.

The action means that the terminal device selects the best offloading node for the subtask according to the network state in time slot \( t \), and all possible offloading actions form the action space \( A \). Action \( a_t \in A \) is defined as Equation (38).

\[ a_t = \{ a_1(t), \ldots, a_{F+2}(t) \}, \]

where \( a_i(t) = [a_{i,1}(t), \ldots, a_{i,N+2}(t)] \) is a binary vector representing the offloading position of the subtask \( TS_i \).
Reward refers to the signal of environmental feedback after the execution of an action \( a_t \) according to the current state \( s_t \) at time \( t \), which is recorded as \( r(s_t, a_t) \) to evaluate the quality of the action, as shown in Equation (39).

\[
r(s_t, a_t) = \begin{cases} 
\frac{\text{COST}^{\text{full local}} - \text{COST}^{\text{min local}}}{\psi}, & \text{satisfying constraint} \\
-\psi, & \text{other}
\end{cases}
\]  

(39)

Among them, \( \text{COST}^{\text{full local}} \) is the delay and energy consumption required for the fully local computing of all subtasks, and the calculation formula is shown in Equation (40).

\[
\text{COST}^{\text{full local}} = \sum_{i=1}^{F+2} (T_i^L + E_i^L)
\]  

(40)

When the offloading strategy satisfies the constraint condition, the reward function is set to the ratio of the total cost under the current strategy to the total cost that can be saved when all subtasks are executed locally. The greater the reward, the smaller the total cost of the current decision. On the contrary, if the offloading strategy does not satisfy the constraint condition, a penalty value \(-\psi\) is accepted, and \( \psi > 0 \).

4.3.2. Improved DDQN

The evaluation network \( Q^{\text{eval}} \) and the target network \( Q^{\text{target}} \) are two neural networks with the same structure but different parameters, and the corresponding parameters are \( \theta \) and \( \theta^- \), respectively. \( Q^{\text{eval}} \) is a neural network used for training, and outputs the Q estimate of state \( s \), which is called Q prediction. \( Q^{\text{target}} \) is not involved in network training, and the output is the Q estimate of the next state \( s' \), which is called Q true. The DDQN separates action selection from action evaluation. When calculating the real value, the action selection is not obtained through the target network \( Q^{\text{target}} \), but based on the evaluation network \( Q^{\text{eval}} \) to select the action, and then obtain the Q value of the action on the target network \( Q^{\text{target}} \). That is, this is done by evaluating the network parameter \( \theta \) selection action, through the target network parameter \( \theta^- \) to evaluate the action, in order to solve the problem of the biased estimation of the value function.

In the DDQN, after calculating action \( a \), the environment state is changed from \( s \) to \( s' \), and the reward value \( r \) is returned, and the quadruple \( (s, a, r, s') \) is stored in the experience pool. When the number of tuples in the experience pool reaches a certain value, small batches of tuples are randomly selected to train the network. Firstly, \( s \) is input into the evaluation network \( Q^{\text{eval}} \) to calculate \( Q^{\text{eval}} = Q(s, a; \theta) \), and then the \( Q(s', a; \theta) \) value of \( s' \) under all actions is calculated using \( Q^{\text{eval}} \), and the action \( a^* = \arg\max_a Q(s', a; \theta) \) corresponding to the maximum Q is selected. Then, according to the selected action \( a^* \), the \( Q(s', a^*; \theta^-) \) value is calculated in the target network \( Q^{\text{target}} \), and the real value is obtained as shown in Equation (41).

\[
Q^{\text{target}} = r_t + \gamma \cdot Q(s', a^*; \theta^-)
\]  

(41)

Among them, \( r_t \) is the instant reward at moment \( t \), and \( \gamma \) is the discount factor.

Prioritized experience replay (PER) is used to sample from the experience pool according to the priority of all tuples, and valuable samples are preferentially extracted. The sample priority is defined as \( Q^{\text{target}} - Q^{\text{eval}} \), which represents the difference between the estimated value and the real value, namely, the TD-error. The larger the value, the more room for improvement in prediction accuracy, so the sample needs to be learned, and the higher the learning priority. TD-error is defined as Equation (42).

\[
\delta_t = r_t + \gamma \cdot \max_{a} Q(s', a^*; \theta^-) - Q(s, a; \theta)
\]  

(42)

In order to prevent network over-fitting, experience extraction is performed in a probabilistic manner. The probability of empirical sample \( (s_j, a_j, r_j, s_{j+1}) \) being selected is shown in Equation (43).
\[ P(j) = \frac{p_j^\chi}{\sum_m p_j^\chi}, \]  
(43)

where \( m \) is the experience pool capacity, \( \chi \in [0, 1] \) is used to regulate the degree of priority for extracting experience samples, and \( \chi = 0 \) is uniform sampling. \( p_j \) is the priority of the \( j \)-th experience sample, i.e., \( p_j = |\delta + \eta| \), \( \eta \) is a very small value that is not 0, ensuring that experiences with a TD-error of 0 are also likely to be selected. The use of prioritized sampling changes the distribution of experience and tends to cause experience samples with high TD-errors to be sampled multiple times, and network training is prone to overfitting. Therefore, an importance sampling method is used in the proposed method to introduce an importance sampling weight to reduce the bias error, as shown in Equation (44).

\[ W_j = \left( \frac{1}{M} \cdot \frac{1}{P(j)} \right)^\beta, \]  
(44)

where \( M \) is the number of samples and \( \beta \) is a hyperparameter indicating the degree of bias reduction, at which point the loss function is shown in Equation (45).

\[ L(\theta) = W_j \cdot E \left[ (Q_{\text{target}}(s, a) - Q_{\text{eval}}(s, a))^2 \right] \]  
(45)

Calculate the error according to Equation (45), train \( Q_{\text{eval}} \) and update the parameter, and update the evaluation network parameter \( \theta \) using the gradient descent method as shown in Equation (46).

\[ \theta \leftarrow \theta - \rho \cdot \delta_t \cdot \frac{\partial Q(s_t, a_t; \theta)}{\partial \theta}, \]  
(46)

where \( \rho \in [0, 1] \) is the learning rate and the evaluation network parameter, \( \theta \) is updated in real time, and after a certain number of times, the parameter of \( Q_{\text{eval}} \) is used instead of the parameter of \( Q_{\text{target}} \) to improve the accuracy of the algorithm.

4.3.3. Overall Flow of the Algorithm

The ISO-DDQN algorithm process is shown in Algorithm 2.

**Algorithm 2. ISO-DDQN Algorithm**

**Input:** Subtasks set \( TS = \{TS_1, TS_2, \ldots, TS_{F+2}\} \); Computing nodes set \( CN = \{cn_1, cn_2, cn_N, cn_{N+1}, cn_{N+2}\} \); Environmental information \( s \); Learning rate \( \rho \); Discount factor \( \gamma \)

**Output:** Subtask offloading decision

1. Randomly initialize the parameter \( \theta \) of the evaluation network \( Q_{\text{eval}} \)
2. Initialize parameter \( \theta^- = \theta \) of target network \( Q_{\text{target}} \)
3. Initialize the size of the experience pool as \( m \) and the update frequency of the target network \( Q_{\text{target}} \) as \( C \)
4. for episode = 1, E do  // each iteration of the training
5. Initialize state \( s_0 \)
6. for t = 1, T do
7. Choose action \( a_t \) based on strategy \( \varepsilon - \text{greedy} \)
8. return reward \( r_t \) and new state \( s_t \)
9. Calculate TD-error by Equation (42), calculate \( P(j) \) by Equation (43), Calculate \( W_j \) by Equation (44), and store \( (s_t, a_t, r_t, s_{t+1}) \) in the experience pool
10. Drawing empirical samples from the empirical pool based on sampling probabilities
11. Calculate the minimization loss function by Equation (45)
12. Updating the evaluation network \( Q_{\text{eval}} \) parameter by Equation (46)
13. The parameter of the evaluation network \( Q_{\text{eval}} \) is copied to the target network \( Q_{\text{target}} \) for parameter updating every \( Q_{\text{target}} \) steps, such that \( \theta^- = \theta \)
Algorithm 2. ISO-DDQN Algorithm

14. end for
15. end for
16. Optimal offloading decision \((a_1, a_2, \ldots, a_{F+2})\)

5. Simulation Experiments

This section evaluates the segmentable computational task decomposition and subtask offloading scheme with complex dependencies proposed in this paper. The experimental simulation parameters and evaluation metrics are set to compare the ATDCC-DPC algorithm and the ISO-DDQN algorithm, respectively, to verify the advantages of the proposed algorithm in task decomposition and task offloading.

5.1. Simulation Parameter Setting

In the experiments, use the Windows 10 operating system with an Intel Core i5-9500 processor and 12GB of RAM. The device cluster simulated in the simulation experiment is a computing power network containing one mobile terminal device, five edge servers, and one cloud computing center, and the mobile terminal device can only send one computing task request at a time.

\(T_D\) denotes the maximum tolerable delay of the task, \(U_{ij}\) is the amount of data interaction between the task element \(t_i\) and the task element \(t_j\), \(R_{ij}\) is the data dependency between the task element \(t_i\) and the task element \(t_j\), \(B_{ij}\) is the communication bandwidth requirement between the task element \(t_i\) and the task element \(t_j\), \(H_{ij}\) is the communication delay between the task element \(t_i\) and the task element \(t_j\), and \(\lambda_{ij}\) denotes the task element \(t_i\) and the task element \(t_j\) special requirements. \(U_{ij}, R_{ij}, B_{ij}, H_{ij}\) is normalized to \([0, 1]\), randomly generated within a predefined range. \(B\) is the bandwidth of the wireless channel, \(\sigma^2\) is the Gaussian white noise power, and \(d_i\) is the distance between the end device and edge server \(e_d_i \in ED\). \(f_{\text{local}}\) is the local computing power, \(f_{\text{edge}}\) is the edge server computing power, and \(f_{\text{cloud}}\) is the cloud center computing power. \(\kappa_L\) denotes the CPU structure energy constant coefficient of the local device, \(\kappa_{E_i}\) denotes the CPU structure energy constant coefficient of the edge server, \(\kappa_C\) denotes the CPU structure energy constant coefficient of the cloud server, and \(R_c\) is the data transmission rate of the backbone network. The task simulation parameters are set as shown in Table 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(T_D)</td>
<td>[0.5,1.5] s</td>
<td>(d_i)</td>
<td>[10,500] m</td>
</tr>
<tr>
<td>(U_{ij})</td>
<td>[0,1]</td>
<td>(f_{\text{local}})</td>
<td>2 GHz (cycle/s)</td>
</tr>
<tr>
<td>(R_{ij})</td>
<td>[0,1]</td>
<td>(f_{\text{edge}})</td>
<td>30–50 GHz (cycle/s)</td>
</tr>
<tr>
<td>(B_{ij})</td>
<td>[0,1]</td>
<td>(f_{\text{cloud}})</td>
<td>100 GHz (cycle/s)</td>
</tr>
<tr>
<td>(H_{ij})</td>
<td>[0,1]</td>
<td>(\kappa_L)</td>
<td>(2 \times 10^{-27})</td>
</tr>
<tr>
<td>(\lambda_{ij})</td>
<td>(0, 1]</td>
<td>(\kappa_{E_i})</td>
<td>(2 \times 10^{-29})</td>
</tr>
<tr>
<td>(B)</td>
<td>1 MHz</td>
<td>(\kappa_C)</td>
<td>(2 \times 10^{-30})</td>
</tr>
<tr>
<td>(\sigma^2)</td>
<td>–80 dBm</td>
<td>(R_c)</td>
<td>2 Mbps</td>
</tr>
</tbody>
</table>

ISO-DDQN is a fully connected neural network consisting of one input layer, two hidden layers, and one output layer. In this, the number of neurons in the hidden layers is 160, the parameter of the prediction network is copied to the target network after every 200 steps, the activation function is the ReLU function, and the number of samples per batch of training batch_size is set to 64. \(\gamma\) is the discount factor, \(\chi\) is the regulation priority level, \(\rho\) is the learning rate, and \(\beta\) is the hyperparameter. \(\eta\) is a small value that is not 0, and the experience that guarantees that the TD-error is 0 may also be selected. \(m\) is the empirical pool capacity. The ISO-DDQN configuration parameters are shown in Table 2.
Table 2. ISO-DDQN configuration parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\gamma$</td>
<td>0.9</td>
</tr>
<tr>
<td>$\chi$</td>
<td>0.6</td>
</tr>
<tr>
<td>$\rho$</td>
<td>0.001</td>
</tr>
<tr>
<td>$\beta$</td>
<td>0.4</td>
</tr>
<tr>
<td>$\eta$</td>
<td>0.01</td>
</tr>
<tr>
<td>$m$</td>
<td>1024</td>
</tr>
</tbody>
</table>

5.2. Evaluation Metrics

(1) Execution time

The algorithm takes the entire time from the beginning to the end. The shorter the execution time, the higher the efficiency of the algorithm.

(2) Degree of clustering subtask

The degree of clustering subtask is used to measure the performance of the algorithm in clustering subtasks; that is, the algorithm is able to cluster task elements with dependencies into the same subtask. The degree of clustering subtask degree is evaluated using the amount of data interacting between subtasks, data dependencies, bandwidth requirements, communication delay requirements, and the degree to which special requirements are fulfilled. The lesser amount of data, lesser data dependency, lesser bandwidth requirement, lesser communication delay requirement, and higher degree of fulfillment of the special requirements of inter-subtask interactions indicate a better degree of clustered subtasks and a better clustering effect.

(3) System delay

System delay is the total time required from the time the input data enters the system to the time the algorithm finishes processing and outputs the result. The shorter the system delay, the faster the algorithm responds.

(4) System energy consumption

System energy consumption is the amount of energy consumed by the entire system during the execution of the algorithm. The lower the energy consumption, the better the algorithm is in terms of energy efficiency.

5.3. Experimental Evaluation of the ATDCC-DPC Algorithm

In order to verify the advantages of the ATDCC-DPC algorithm proposed in this paper for task decomposition, experiments were conducted by writing the relevant code on Matlab R2022a to compare it with the Max-min algorithm, Full-Deco algorithm, and MG-Dcom algorithm. The Max-min algorithm is the classical task scheduling algorithm; this algorithm does not undergo any processing on the task and performs scheduling computation directly. The Full-Deco algorithm decomposes the task into task elements and performs the computation with the granularity of task elements. The MG-Dcom algorithm decomposes the task into multi-granularity subtasks and is able to select the appropriate granularity of the decomposed task for scheduling computation. The complex divisible computation tasks selected for simulation have task elements of 5, 10, 15, and 20. The parameters of these task elements, such as the amount of data to interact between the task elements, data dependency, communication bandwidth, communication delay, and special requirements, are randomly generated within the range given in Table 1 to satisfy the normal distribution as much as possible, which helps to simulate the real scenarios more accurately.

Figure 4 shows the comparison of the execution times of these four algorithms, from which it can be seen that the proposed ATDCC-DPC algorithm has the lowest execution time, which is 69.86%, 51.8% and 28.37% shorter on average compared to the Max-min
algorithm, Full-Deco algorithm, and MG-Dcom algorithm. Because the Max-min algorithm is undecomposed for tasks, its execution time is much higher than the other three algorithms. The Full-Deco algorithm, on the other hand, because there is usually some kind of dependency between task elements, calculates with task elements as the granularity, which is likely to lead to an increase in the transmission delay and an increase in the execution time. The MG-Dcom algorithm derives the results of task decomposition under different granularities and selects the appropriate decomposition granularity for calculation, so that the execution time is longer compared to that of the algorithm proposed in this paper.

Figure 4 shows the comparison of the execution times of these four algorithms, from which it can be seen that the proposed ATDCC-DPC algorithm has the lowest execution time, which is 69.86%, 51.8% and 28.37% shorter on average compared to the Max-min algorithm, Full-Deco algorithm, and MG-Dcom algorithm. Because the Max-min algorithm does not undergo any processing on the task and performs scheduling computation tasks selected for simulation have task elements of 5, 10, 15, and 20. The MG-Dcom algorithm derives the results of task decomposition for task processing. The comparison with the Full-Deco algorithm, in which each task element is regarded as a subtask, is taken as the baseline to analyze the subtasks after clustering. Compare the amount of interaction data, data dependency, bandwidth requirement, communication delay, and special requirements.

Figure 5 shows the performance comparison of the four algorithms for five indicators: the amount of interactive data between subtasks, data dependency, bandwidth requirement, communication delay, and special requirements. The Full-Deco algorithm considers each task element as a subtask; no clustering operation is carried out, and the dependency between subtasks is still the same as the dependency between task elements, which has not been changed, and the highest degree of dependency between subtasks is achieved. The ATDCC-DPC algorithm adaptively determines the truncation distance through the Gini coefficient; the elbow method selects the center point of clustering, determines the number of clusters, avoids experimentally verified or empirically determined truncation distance, has a high degree of uncertainty, and manually selects the center of clustering in a way that is subjective and random, which leads to inaccurate clustering results. Therefore, the ATDCC-DPC algorithm proposed in this paper has a better performance compared to the DPC algorithm and the MG-Dcom algorithm and will obtain a task metACLustering scheme with a lower amount of interactive data between subtasks, less data dependency, a lower bandwidth requirement, less communication delay, and be better able to meet the special requirements.

The comparison experiment using Max-min algorithm can well reflect the advantage of task decomposition for task processing. The comparison with the Full-Deco algorithm, MG-Dcom algorithm, and DPC algorithm can well reflect the advantages of the proposed algorithm.
algorithm in clustering subtasks with task elements after task decomposition. The above experiments reflect that after the task is decomposed into task elements, using subtasks as offloading objects can shorten the computation time and energy consumption of the task, and the ATDCC-DPC algorithm proposed in this paper has a better performance in terms of execution time and clustering subtasks.

![Radar chart comparing the performance of five indicators.](image)

**Figure 5.** Radar chart comparing the performance of five indicators.

### 5.4. Experimental Evaluation of the ISO-DDQN Algorithm

The experimental environment used PyCharm software (https://www.jetbrains.com/pycharm/) (accessed on 1 April 2024), a Python 3.8 development environment, and PyTorch version 1.10.2. $\alpha$ denotes the delay weight of the computational task, as shown in Equation (35). Figure 6 illustrates how the average delay and energy consumption change with the $\alpha$ value. As the value of $\alpha$ keeps increasing, the sensitivity to delay gradually increases, while the attention to energy consumption is relatively weakened, and this change is more suitable for computational tasks with strict delay requirements. In particular, when $\alpha = 1$, this means that the optimization objective at this point is to minimize the system delay without considering the effect of energy consumption at all. Since the weight of energy consumption is $1 - \alpha$, when $\alpha$ gradually decreases, the weight of energy consumption increases accordingly, and the need to control energy consumption gradually increases, which is more suitable for computationally intensive tasks. And when the weight of energy consumption is 1, i.e., $\alpha = 0$, it means that the system completely takes minimizing energy consumption as the optimization goal and no longer considers the delay factor. Therefore, the value of $\alpha$ can be adjusted to adapt to the needs of different types of applications, which in turn effectively reduces the overall operating cost.

To validate the proposed offloading strategy, the subtasks formed by task element clustering are used as the offloading objects and compared with the five offloading algorithms, and the value of $\alpha$ is set to 0.5. Where the local algorithm indicates that the task is computed locally only, the edge algorithm indicates that the task is offloaded to the edge computing nodes for processing. The cloud algorithm indicates that the task is offloaded to the cloud center for processing. The edge–cloud algorithm indicates that the task is offloaded to the edge computing nodes and the cloud center for execution. The DQN algorithm denotes a DRL method based on value $Q$. A neural network is used to generate $Q$ values, determine the next action based on the $Q$ values, and update the neural network parameters by back propagation. Here, three sets of experiments are designed to evaluate the performance of the algorithms from six perspectives: the impact of the task’s data volume on the system delay as well as energy consumption, the impact of the maximum tolerable task delay on
the system delay as well as energy consumption, and the impact of the number of edge servers on the system delay as well as energy consumption.

![Figure 6](image-url) Impact of different α-values on delay and energy consumption.

The impact of the amount of task data on the total system delay and energy consumption is evaluated as shown in Figure 7. In Figure 7a, the delay is kept as a stable value, as the local algorithm does not involve the offloading and transferring of tasks. As the amount of task data increases, the edge algorithm appears to compete for the computational resources of the edge nodes, resulting in a higher delay than the local and cloud algorithms, while the total system delay of the edge-cloud is less than edge algorithm and cloud algorithm. In Figure 7b, the system energy consumption of the local algorithm increases gradually and steadily as the task data volume increases. When the task data volume is small, the transmission energy consumption accounts for a small percentage of the system energy consumption, so the system energy consumption of the cloud algorithm is smaller than the edge algorithm. As the task data volume increases, the transmission energy consumption in the system energy consumption gradually increases, so the edge algorithm system energy consumption is less than the cloud algorithm. The edge-cloud algorithm reasonably utilizes the computational resources of the edge nodes and the cloud center and is close to the two offloading algorithms with lower energy consumptions under different task data volumes. The ISO-DDQN algorithm adopts the improved DDQN with a prioritized empirical replay method based on importance sampling, which prioritizes the samples with higher values, while the DQN is trained with random sampling. Further, the computation of the target Q value and the selection of actions in the ISO-DDQN algorithm are implemented by choosing different networks, respectively, which alleviates the problem of overestimating task offloading decisions in the DQN algorithm. In conclusion, the algorithm proposed in this paper always minimizes the total system delay and energy consumption compared to the other five schemes.

As shown in Figure 8, the experiment evaluates the impact of the maximum tolerable delay of tasks on the system delay and energy consumption. When the delay requirement is more relaxed, more tasks can run locally, while tasks with high computational resources are offloaded to the edge nodes for execution. As a result, the total system delay gradually decreases as the tolerable delay of tasks increases. The energy consumption shows a similar performance to the delay. In conclusion, the total delay and energy consumption of the proposed algorithm in this paper are the lowest compared to the other five algorithms.
Further, the computation of the target network evaluates the samples with higher values, while the DQN is trained with random sampling. With a prioritized empirical replay method based on importance sampling, which prioritizes samples with higher values, the ISO-DDQN algorithm decreases, where the ISO-DDQN algorithm has the lowest system delay and energy consumption compared to the other algorithms.

In conclusion, the algorithm proposed in this paper always minimizes the total system delay and energy consumption, which shows the superiority of this algorithm in cloud-edge-end cooperative processing.

Figure 7 illustrates the impact of the number of edge servers on the system delay and energy consumption when the task data size is 100 MB. Since the local algorithm tasks are only computed locally, and the cloud algorithm tasks are computed in the cloud center only, which does not involve the edge servers, the number of edge servers does not affect the delay and energy consumption of the processing of the tasks of these two algorithms. From the figure, it can be seen that as the number of edge servers increases, the available computational resources in the network increases as the system delay and energy consumption of the edge algorithm, edge-cloud algorithm, DQN algorithm, and ISO-DDQN algorithm decreases, where the ISO-DDQN algorithm has the lowest system delay and energy consumption, which shows the superiority of this algorithm in cloud-edge-end cooperative processing.

The ISO-DDQN algorithm is able to make offloading decisions intelligently by comprehensively considering the task characteristics, network conditions, and the real-time status of cloud-edge-side resources. By reasonably allocating tasks to be executed locally and in the cloud and edge, it can effectively reduce the energy consumption of the whole system,
maximize the use of cloud-edge resources, and avoid resource wastage and overloading, thus improving the overall resource utilization and optimizing the system performance.

Figure 9. Impact of the number of edge servers on system delay and energy consumption. (a) Impact of the number of edge servers on system delay. (b) Impact of the number of edge servers on system energy consumption.

6. Conclusions and Future Work

This paper proposes a decomposition and offloading scheme for complex divisible computing tasks, which effectively solves the problem of executing computationally intensive tasks with complex dependencies. Aiming at solving the problem that the task decomposition method is singular, and that the inappropriate decomposition granularity will affect the transmission delay, this paper first decomposes the whole task into task elements, analyzes the dependency relationship between task elements, clusters the task elements with a high degree of dependency into subtasks, and proposes the ATDCC-DPC algorithm to realize the process of clustering the task elements into subtasks. Aiming at solving the problem that most of the existing offloading strategies use only one kind of computing resource, this paper proposes an offloading strategy that comprehensively considers the synergy of local, edge, and cloud computing resources. Taking subtasks as offloading objects, the proposed ISO-DDQN algorithm is utilized to achieve intelligent offloading decisions based on the characteristics of tasks and the real-time state of resources. The proposed scheme in this paper is able to cope with complex computing challenges, ensure real-time requirements, reduce system delay and energy consumption, and provide strong support for cloud–edge–end collaborative computing to efficiently handle complex and divisible computing tasks.

We believe that the ATDCC-DPC algorithm can be applied in many fields, such as image segmentation, text clustering, social network analysis, etc., and that the ISO-DDQN algorithm has great potential to optimize the offloading decision for cloud–edge–end collaborative computing, which enhances the robustness of the system and ensures the service continuity. The proposed scheme in this paper provides guarantees for real-time applications, such as autonomous driving and telemedicine, to ensure that these applications receive timely responses, thus enhancing the user experience. However, there are still some issues that need to be continually considered, including security and privacy issues during task offloading, and how some of the computational tasks in the computing power network are redundant and repetitively executed. Next, we consider introducing computational reuse technology to optimize resource utilization and improve service experience while ensuring data security and privacy protection.
Author Contributions: Conceptualization, P.H. and J.C.; methodology, P.H.; software, J.C.; validation, H.Q. and J.C.; formal analysis, P.H.; investigation, J.C. and H.L.; resources, J.C.; data curation, J.C.; writing—original draft preparation, J.C. and H.Q.; writing—review and editing, P.H. and J.C.; visualization, J.C.; supervision, P.H.; project administration, P.H.; funding acquisition, P.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Scientific Research and Development Program Project of Hebei University of Economics and Business (2022ZD09), and the Science and Technology Project of Hebei Education Department (QN2022061).

Data Availability Statement: Data can be provided by the authors upon request.

Conflicts of Interest: The authors declare no conflicts of interest.

References

19. Flores, K.G.; Villarreal, S.E.G. Density peaks clustering with gap-based automatic center detection. Knowl.-Based Syst. 2020, 206, 106350. [CrossRef]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.