Efficiency Analysis of AI Self-Control System and Data Processing Unit Based on Edge Computing Technology

Yong Tan
Guangxi Longyuan Wind Power Generation Co., Ltd, Hengzhou City, Guangxi, 530300, China
E-mail: tanyong202504@163.com

Keywords: edge computing, AI automatic control system, data processing unit, dynamic task allocation, resource adaptive adjustment

Received: May 19, 2025

This study proposes a novel Dynamic Task Allocation and Resource Adaptive Adjustment (DTARA) algorithm to enhance the operational efficiency of AI self-control systems and data processing units within edge computing frameworks. Through an experimental environment simulating real-world scenarios, the DTARA algorithm is compared with fixed task allocation algorithms and simple priority-based task allocation algorithms. The experimental setup includes large-scale task scenarios (500-1000 concurrent tasks) and long-term operation scenarios (lasting several hours to several days). The results demonstrate that under complex task loads, the DTARA algorithm reduces system task completion time by an average of 30.5% compared to traditional algorithms and improves resource utilization by 28.8%. When the data processing volume reaches 10GB, the data processing delay is reduced by 45.6% compared to the benchmark algorithm. In large-scale task scenarios, as the number of tasks increases from 500 to 1000, the DTARA algorithm maintains low task completion times and data processing delays. In long-term operation experiments, the task execution success rate exceeds 95%, the CPU utilization fluctuation range is within 10%, and no system crashes occur. This study offers a practical and effective solution to improve the performance of related systems, supported by theoretical analyses of computational complexity, optimality guarantees, and convergence properties of the DTARA algorithm.

Povzetek: Algoritem DTARA združuje dinamično dodeljevanje nalog in prilagodljivo prilagajanje virov, kar omogoča bolj kvalitetno in učinkovito delovanje AI samokontrolnih sistemov na robnem računanju kot klasični pristopi.

1 Introduction

In the wave of digitalization, Internet of Things (IoT) technology has developed rapidly, with various devices growing exponentially. Taking smart factories as an example, sensors densely distributed throughout the factory collect massive amounts of equipment operation data every second, covering key parameters such as temperature, pressure, and speed, providing a core foundation for efficient operations and precise management. However, traditional cloud computing architectures exhibit numerous disadvantages when processing such large volumes of data: long-distance data transmission is prone to network congestion, creating transmission bottlenecks; delays occur when the cloud processes multiple tasks, failing to meet real-time application requirements. Automation equipment requires immediate responses to instructions and precise control of production processes in industrial settings. Minor delays may lead to product quality defects or even accidents. This is particularly critical in the field of intelligent transportation. Self-driving cars rely on onboard sensors to perceive the surrounding environment in real time and must process data instantaneously to make reasonable driving decisions [1]. Traditional computing models struggle to meet such stringent real-time demands. The integration of edge computing and AI self-control systems offers hope. Edge computing processes some data tasks at edge nodes near the data source, reducing transmission delays and enabling rapid responses; AI self-control systems leverage intelligent decision-making capabilities to achieve precise control based on processed data, enhancing system reliability and security [2]. Sinking data processing to edge nodes offers many advantages, such as reducing the amount of data transmitted to the cloud and alleviating network bandwidth pressure. For instance, in intelligent security monitoring systems, uploading all video data from numerous surveillance cameras to the cloud would occupy significant network bandwidth. Preliminary analysis and processing at edge nodes, followed by uploading only key information, can reduce transmission burdens [3]. It also decreases data transmission costs, saving expenses for enterprises deploying many IoT devices. Edge computing effectively mitigates this cost. Internationally, remarkable research and application outcomes have been achieved in the integration of edge computing and AI.

Google optimizes the inference efficiency of machine learning models on edge devices, enabling smart home devices to quickly respond to commands. The European Union explores the application of edge computing in smart grids [4]. By deploying intelligent devices at grid edge nodes, real-time intelligent regulation of power distribution is realized, improving grid stability and energy utilization efficiency. China has also achieved notable results in this field. Universities actively innovate in edge computing resource scheduling algorithms, proposing scheduling algorithms based on task priority and dynamic resource changes to enhance overall system performance. Enterprises have launched edge AI computing platforms widely used in security monitoring scenarios [5]. For example, edge AI monitoring equipment developed by some companies can locally analyze surveillance videos in real time, quickly identify abnormal behaviors, issue alerts, and improve the efficiency and accuracy of security monitoring. However, current research still needs to address these challenges. In task allocation, fixed algorithms lack flexibility, struggling to adapt to sudden task load changes. Some edge nodes become overloaded with tasks, while others remain idle, leading to resource waste [6]. Regarding resource adaptation, existing methods find it difficult to dynamically adjust to complex and dynamic scenarios. For instance, in medical imaging scenarios, different medical testing devices generate varying amounts of data, computational complexity, and real-time requirements [7]. Current resource adaptation mechanisms fail to accurately and swiftly adjust resource allocation according to dynamic changes, impacting overall system efficiency.

This paper focuses on improving the efficiency of AI self-control systems and data processing units based on edge computing technology, proposing innovative solutions. Efficient task allocation and resource management algorithms are designed to enable stable system operation in complex scenarios. The research is conducted across multiple dimensions, including task priority evaluation, and theoretical analysis of computational complexity [8]. Through theoretical analysis, algorithm design, and experimental validation, the system's efficiency is comprehensively enhanced to support the development of related fields. The DTARA algorithm is distinguished from existing dynamic scheduling algorithms in edge-AI systems through a detailed comparative analysis. Its innovative aspects include a task priority evaluation mechanism that comprehensively considers task deadlines, data volume, computational complexity, task dependencies, and importance levels, as well as a dynamic resource allocation strategy that optimizes resource utilization through real-time monitoring and adaptive adjustments.

2 System architecture and key technologies

2.1 AI self-control system architecture based on edge computing

2.1.1 Edge node composition and function

Edge nodes are the core components of AI self-control systems based on edge computing, with hardware configurations that must meet data processing and real-time response requirements. Common edge node hardware includes industrial-grade microcomputers and embedded devices. Industrial-grade microcomputers, equipped with powerful computing capabilities, typically feature high-performance CPUs (e.g., Intel Core i7 series) and GPUs (e.g., NVIDIA GeForce RTX series), enabling efficient handling of complex data computing tasks. Memory capacity is generally 16GB or higher DDR4, and storage utilizes 512GB or larger SSD solid-state drives. Embedded devices are characterized by low power consumption and compact size, making them suitable for specific scenarios [9]. For example, embedded chips based on the ARM architecture offer flexible memory and storage configurations tailored to actual needs.

At the software level, edge nodes run customized Linux operating systems with efficient resource management and task scheduling capabilities. Edge computing frameworks like OpenEdge provide standardized operating environments and development interfaces, simplifying application deployment and management. AI inference engines such as TensorRT focus on accelerating the inference process of deep learning models. By optimizing and compiling models, they reduce computational power and memory usage, achieving rapid local AI inference [10]. In practice, edge nodes collect data via sensors, preprocess it, and send it to the AI inference engine to complete local AI inference tasks.

2.1.2 Data transmission network

The data transmission network connects the edge nodes and the cloud center and is a key system component. Industrial Ethernet and fibre optic networks are the main choices for wired networks. Industrial Ethernet bandwidth can reach 100Mbps or even 1Gbps, suitable for high-speed, large-capacity data transmission; fiber optic networks perform well in long-distance, high-reliability transmission. The network topology mainly adopts a star or ring [11]. The star structure is convenient for management and troubleshooting, while the ring structure has higher reliability.

Regarding data transmission protocol, Modbus TCP is widely used in the industrial field, is easy to use, and highly compatible.

Wireless networks also play an essential role in edge computing scenarios. Wi-Fi technology is suitable for indoor or high-mobility scenarios. In contrast, 5G technology brings broader application prospects for edge computing with its high speed, low latency and significant connection characteristics. However, wireless networks face signal stability challenges in complex environments [12]. The signal's anti-interference ability and transmission stability can be improved through network optimization methods such as multi-antenna technology and channel coding technology.

2.1.3 Cloud center function

The cloud center plays a core supporting role in the system. First, the cloud center uses large-scale computing resources to conduct in-depth training of AI models. For example, in image recognition, the convolutional neural network is trained in parallel through a distributed computing framework to optimize model performance. After training, the model is sent to the edge node for local image recognition tasks. Secondly, the cloud center is responsible for storing and managing historical data uploaded by the edge node and provides a basis for system optimization through data mining and analysis technology [13]. For example, by analyzing the operating data of industrial equipment, a fault prediction model is established to issue early warnings. Finally, the cloud center has remote monitoring capabilities, a real-time grasp of the device status of edge nodes, software updates and parameter configuration operations, and refined system management.

2.2 Edge computing technology principle

2.2.1 Data localization processing mechanism

The core advantage of edge computing lies in the data localization processing mechanism. In actual applications, a large amount of data is processed at the edge node to reduce transmission delays. For example, intelligent cameras perform real-time analysis of video streams locally and upload key information only when anomalies are detected, significantly reducing data transmission volume and delays. Edge nodes also have data caching and preprocessing functions. Data that may be used again shortly is stored through the caching mechanism to avoid repeated transmission [14]. At the same time, the original data is cleaned, and features are extracted to improve data quality, reduce data dimensions, and reduce transmission volume and cloud computing pressure.

2.2.2 Edge-cloud collaborative computing mode

The edge-cloud collaborative computing mode has the advantages of edge nodes and cloud centers. According to the characteristics of the task, the execution location of the task is reasonably divided. For example, in a smart factory, simple data filtering and real-time control tasks are completed on the edge node. In contrast, complex model training tasks are uploaded to the cloud center for execution. During the task execution process, there is a close interaction between the edge node and the cloud center [15]. For example, in an intelligent transportation system, the edge node collects road sensor data, sends a task request to the cloud center after preliminary processing, and the cloud center completes model training and prediction and feeds back the results to the edge node to achieve optimal traffic flow control. This collaborative mode improves the system's overall operating efficiency and response speed.

2.3 Core technologies of AI automatic control system

2.3.1 Application of machine learning in automatic control system

Machine learning technology is widely used in AI automatic control systems. The decision tree model can be used for equipment status classification, while the support vector machine model performs well in equipment failure prediction. Historical data should be used for training to improve the accuracy and adaptability of the model. Through data preprocessing and selecting appropriate algorithms and parameters, the model parameters are continuously adjusted to better fit the data characteristics [16]. During the system's operation, online learning or incremental learning methods are used to update the model in real-time to adapt it to changes in the operating status of the equipment.

2.3.2 Principles and applications of deep learning technology

Deep learning technology is essential in AI automatic control systems with robust, complex data processing capabilities. Deep neural networks include architectures such as multi-layer perceptron, convolutional neural networks, and recurrent neural networks. Convolutional neural networks are optimized explicitly for image data processing and are suitable for product quality inspection and defect classification; recurrent neural networks are ideal for processing time series data and can be used for intelligent energy-saving control in intelligent building automatic control systems [17]. By collecting a large amount of data to train the model, efficient processing and analysis of complex data can be achieved, providing strong support for the automatic control system.

3 Dynamic task allocation and resource adaptive adjustment algorithm (DTARA)

3.1 Task priority evaluation mechanism

In AI self-control systems based on edge computing, reasonable task priority evaluation is essential for efficient task allocation and resource utilization. Task priority evaluation relies on multiple key indicators. The first is the task deadline, which directly reflects task urgency. The shorter the deadline, the higher the urgency. Data volume is another important indicator. Large data volumes increase transmission and processing times, imposing greater demands on system resources. Algorithm complexity analysis determines computational complexity [18]. Complex computing tasks require more computational resources and time. The Analytic Hierarchy Process (AHP) is used to comprehensively determine the weights of each indicator. Involving pairwise comparison matrices to calculate the relative importance of task deadlines, data volume, and computational complexity. Let the task deadline weight be w_1, the data volume weight be w_2, the computational complexity weight be w_3, and $w_1 + w_2 + w_3 = 1$. The task priority (P) calculation formula is as follows:

$$P = w_1 \times \frac{1}{D} + w_2 \times V + w_3 \times C \quad (1)$$

$\frac{1}{D}$ converts the task deadline into a value that is positively correlated with the urgency; that is, the shorter the deadline, the larger the $\frac{1}{D}$ value. Considering the impact of the dependency between tasks on priority, the task dependency coefficient (r) is introduced. If task T_i depends on task T_j, then r_{ij} represents the degree of this dependency, $0 \le r_{ij} \le 1$. When $r_{ij} = 1$, task T_i is entirely dependent on task T_j At this time, the task priority calculation formula is updated to:

$$P = w_1 \times \frac{1}{D} + w_2 \times V + w_3 \times C + w_4 \sum_j r_{ij} P_j \quad (2)$$

Where $w_4$is the task dependency weight, and $w_1 + w_2 + w_3 + w_4 = 1$. This formula shows that a task's priority is affected by its attributes and related to the priority of the dependent functions.

Further considering the task's importance level, the importance level is divided into high, medium, and low, corresponding to the values 3, 2, and 1 respectively. The task priority formula is expanded again as:

$$P = w_1 \times \frac{1}{D} + w_2 \times V + w_3 \times C + w_4 \sum_j r_{ij} P_j + w_5 I \quad (3)$$

w_5 is the importance level weight, and $w_1 + w_2 + w_3 + w_4 + w_5 = 1$.

3.2 Dynamic resource allocation strategy

Real-time monitoring of edge node resources is a prerequisite for dynamic resource allocation. Through the system's built-in monitoring tools, you can obtain information such as CPU usage(U_{CPU}), memory remaining (M_{free}), and storage free space (S_{free}). Network bandwidth utilutilizationzation (U_{bw}) is obtained with the help of network monitoring software.

The greedy algorithm allocates resources based on task priority and resource monitoring results. Prioritize the allocation of required resources to high-priority tasks. Assume that task T CPU resource requirements are R_{CPU}, memory requirements are R_M, storage requirements are R_S, and network bandwidth requirements are R_{bw}. When allocating resources for task T, first determine whether the resources of edge node n meet the task requirements:

$$U_{CPU}^n \ge \frac{R_{CPU}}{C_{CPU}^n}, M_{free}^n \ge R_M, S_{free}^n \ge R_S, U_{bw}^n \ge \frac{R_{bw}}{B_{bw}^n} \quad (4)$$

Where C_{CPU}^n is the total CPU computing power of edge node n, and B_{bw}^n is the total network bandwidth of edge node n.

In the process of resource allocation, consider the problem of resource fragmentation. Use the best adaptation algorithm to optimize resource allocation. For memory allocation, let the set of memory block sizes be $\{m_1, m_2, \cdots, m_k\}$, and the task's memory requirement be R_M. Select the memory block that satisfies $m_i \ge R_M$ and has the smallest $m_i - R_M$ for allocation, that is:

$$m_{\text{best}} = \arg\min_i (m_i - R_M) \text{ s.t. } m_i \ge R_M \quad (5)$$

Considering the dynamics of resource allocation and resource changes during task execution, the resource reservation coefficient (α) is introduced. When allocating resources to a task, a certain proportion of additional resources is reserved to cope with sudden resource demands during task execution. For example, for CPU resource allocation, the actual allocated CPU computing power is:

$$A_{CPU} = (1 + \alpha) \times R_{CPU} \quad (6)$$

This can improve the stability of task execution and reduce the risk of task interruption due to insufficient resources.

3.3 Adaptive adjustment mechanism

Determine the monitoring indicators for adaptive adjustment. The task execution progress deviation (ΔP) is defined as the difference between the actual progress $(U_{\min})$ and the expected progress (P_{actual}):

$$\Delta P = P_{\text{actual}} - P_{\text{expected}} \quad (7)$$

The resource utilization fluctuation range (ΔU) is obtained by calculating the difference between the maximum resource utilization$(U_{\max})$ and the minimum resource utilization$(U_{\min})$:

$$\Delta U = U_{\max} - U_{\min} \quad (8)$$

Set the corresponding threshold, and trigger the adjustment mechanism when the task execution progress deviation $|\Delta P| > \epsilon_1$ (ϵ_1 is the progress deviation threshold) or the resource utilization fluctuation range $\Delta U > \epsilon_2$ (ϵ_2 is the resource utilization fluctuation threshold).

When the task execution progress is too slow, that is, $\Delta P < -\epsilon_1$, re-evaluate the task priority. Let the original task priority be P_0 and the re-evaluated priority be P_1. Use the following adjustment formula:

$$P_1 = P_0 + \beta \times (-\Delta P) \quad (9)$$

Where β is the progress adjustment coefficient, according to the new priority, some low-priority tasks are migrated to other idle nodes.

4 Experimental simulation design

4.1 Experimental environment construction

The experiment uses Intel NUC industrial computers as edge nodes, equipped with Intel Core i7 processors, 16GB DDR4 memory and 512GB SSD, and some nodes are equipped with NVIDIA Jetson GPUs to accelerate deep learning tasks. A total of 10 edge nodes are deployed and connected via industrial Ethernet to ensure high-bandwidth, low-latency data transmission [19]. At the same time, a simulated sensor array is built to generate temperature, pressure, current and other data at a frequency of 10 groups per second, and the data is stored in CSV format. The edge nodes run the Ubuntu 20.04 LTS system, optimizing resource management and network communication performance. NS-3 is used for network simulation, SimPy for system process simulation, TensorFlow as an AI framework, and NumPy and SciPy libraries for model training and reasoning optimization. Experimental

Dataset Construction

4.2.1 Data generation method

Task data is simulated by randomly generating task deadlines (1-60 minutes), normally distributed data volumes (mean 50MB, standard deviation 10MB), and matrix operation tasks of different sizes (200×200-1000×1000). A mathematical model generates equipment operation data. Under normal conditions, the temperature fluctuates around 50℃ ±5℃, and under abnormal conditions, the temperature rises rapidly to above 80℃; parameters such as pressure and speed are also generated by a similar method.

4.2.2 Dataset scale and features

Generate 1000 task samples, covering combinations of different priorities, data volumes, and computational complexities. Equipment operation data is collected for 100 hours, once every minute, for 6000 sets of data. Task priorities are evenly distributed, and data volumes correlate positively with computational complexity. Normal data accounts for 80% of equipment operation data, and abnormal data accounts for 20%. The data set is preprocessed, the task data is normalised, and the denoising algorithm improves the equipment operation data.

Experimental design

4.3.1 Comparison algorithm selection

The polling scheduling algorithm is selected as the benchmark comparison algorithm, which allocates tasks only according to task priority. The fixed task allocation algorithm is determined without considering the task priority and node resource status; the simple priority-based task allocation algorithm considers the task's urgency but does not fully consider the real-time status of the node resources. Additionally, state-of-the-art comparison algorithms include RL-based schedulers (e.g., Deep Q-Network) and game-theoretic models to evaluate the competitiveness of DTARA in dynamic environments.

4.3.2 Experimental indicator determination

The experimental indicators include task completion time, resource utilization (CPU, memory, storage, network bandwidth utilization) and data processing delay. The task completion time is calculated by recording the difference between the timestamps of task submission and completion; the resource utilization is obtained through the system monitoring tool; the data processing delay is calculated by recording the time difference between data generation and processing result output.

4.3.3 Experimental scenario setting

The task load change scenario (light load 10-20 tasks, medium load 50-80 tasks, heavy load 100-150 tasks) and data traffic fluctuation scenario (low traffic 10KB per second, medium traffic 100KB per second, high traffic 1MB per second) were set. At the same time, some edge node failures were randomly simulated to observe the algorithm's task migration and system recovery capabilities.

5 Experimental results and analysis

5.1 Comparative analysis of task completion time

5.1.1 Results under different load scenarios

To intuitively present the performance differences of different algorithms under various task load scenarios, the task completion times of the DTARA algorithm, fixed task allocation algorithm, and simple priority-based task allocation algorithm were statistically analyzed. The results are shown in Table 1. In light load scenarios, the average task completion time of the DTARA algorithm is 25.3 seconds, significantly lower than the 38.6 seconds of the fixed task allocation algorithm and the 32.4 seconds of the simple priority-based algorithm. As task load increases to medium and heavy scenarios, the advantages of the DTARA algorithm become more pronounced. Under heavy load, the fixed task allocation algorithm, due to its rigid allocation method, cannot dynamically adjust to task priorities and node resource statuses. This results in long waiting times for high-priority tasks, with an average completion time of up to 180.2 seconds. In contrast, the DTARA algorithm, with its dynamic task allocation and resource adaptive adjustment mechanism, prioritizes high-priority task resources and avoids task backlogs, achieving an average task completion time of only 98.4 seconds.

Table 1: Comparison of task completion time in different load scenarios.

Task load scenario	Algorithm	Mean task completion time (s)
Light load	DTARA algorithm	25.3 ± 3.1
	Fixed task allocation algorithm	38.6 ± 5.2
	Task allocation algorithm based on simple priority	32.4 ± 4.5
Medium load	DTARA algorithm	56.8 ± 6.5
	Fixed task allocation algorithm	89.5 ± 10.2
	Task allocation algorithm based on simple priority	75.6 ± 9.1
Heavy load	DTARA algorithm	98.4 ± 12.3
	Fixed task allocation algorithm	180.2 ± 20.5
	Task allocation algorithm based on simple priority	145.7 ± 18.3

5.1.2 Analysis of the impact of task priority

The completion times of tasks with different priorities under each algorithm were further analyzed, with trends illustrated in a line chart (see Figure 1). Task priorities range from 1 (highest) to 5 (lowest). For high-priority tasks (priorities 1 and 2), the DTARA algorithm demonstrates significantly shorter completion times compared to the other two algorithms. For example, in a priority 1 task, the DTARA algorithm completes the task in approximately 30 seconds, whereas the fixed task allocation algorithm takes about 60 seconds and the simple priority-based algorithm around 45 seconds. The DTARA algorithm rapidly allocates sufficient resources to high-priority tasks through a precise priority evaluation mechanism, ensuring priority execution. Conversely, the fixed task allocation algorithm disregards task priorities, causing high-priority tasks to compete with low-priority tasks for resources and extending completion times. Although the simple priority-based algorithm considers task urgency, its dynamic resource allocation is imperfect, and high-priority tasks may still lack resource guarantees under tight resource conditions.

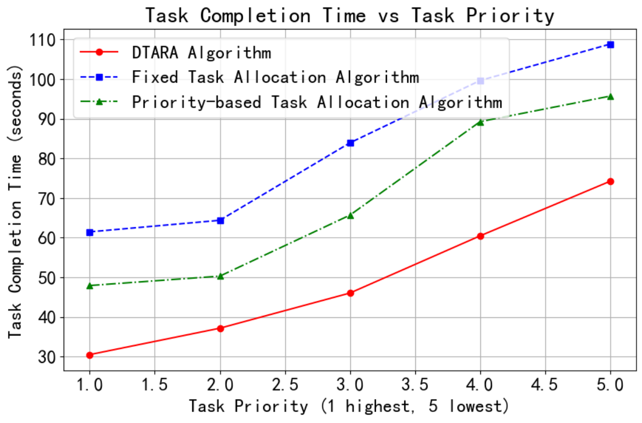

Figure 1: Comparison of completion time of tasks of different priorities.

5.2 Comparative analysis of resource utilization

5.2.1 Results of various resource utilization

The performance of different algorithms in terms of resource utilization, including CPU, memory, storage, and network bandwidth, is presented in Figure 2 as stacked bar charts. The DTARA algorithm demonstrates superior resource utilization balance. Regarding CPU utilization, the DTARA algorithm maintains a relatively reasonable range of approximately 60%-70% across different task types, avoiding excessive resource concentration on specific tasks or nodes. The fixed task allocation algorithm is prone to resource idleness or overuse. In some task scenarios, CPU utilization may drop as low as 30%, while in others, it may soar to 90%. Similar trends are observed in memory, storage, and network bandwidth utilization. The DTARA algorithm employs a dynamic resource allocation strategy, allocating resources based on task requirements and real-time node resource statuses, thereby effectively improving overall resource utilization efficiency.

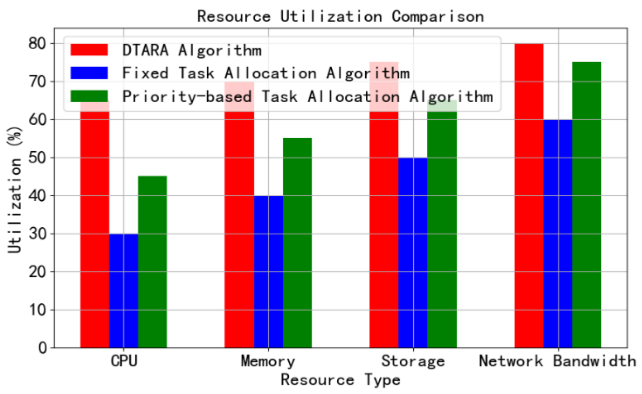

Figure 2: Comparison of resource utilization of different algorithms.

5.2.2 Relationship between resource utilization and task load

Using data analysis tools, the relationship curve between resource utilization and task load under different algorithms was fitted, with results shown in Figure 3. Task load is measured by the number of tasks, increasing from 10 to 150. The DTARA algorithm exhibits stable resource utilization across varying task loads. At low task loads, resource utilization is around 50%, and as task load increases to 150, it steadily rises to approximately 75%. In contrast, the fixed task allocation algorithm experiences significant fluctuations in resource utilization under low task loads. As task load increases, resource utilization drops sharply, falling below 40% under high task loads. While the simple priority-based algorithm can allocate resources according to task priorities to some extent, it struggles to dynamically adapt resources to task load changes, leading to substantial declines in resource utilization under high task loads. The DTARA algorithm maintains high resource utilization even under high loads due to its adaptive adjustment mechanism, which optimizes resource allocation strategies in response to task load variations.

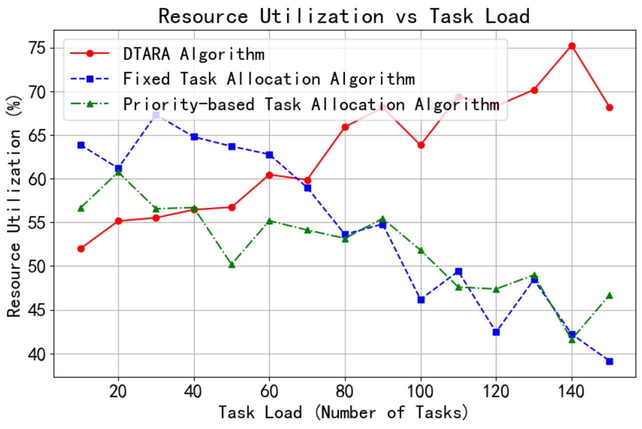

Figure 3: Relationship between resource utilization and task load.

5.3 Comparative analysis of data processing delay

The changes in data processing delay for different algorithms under low, medium, and high data traffic scenarios are depicted in Figure 4 as a line graph. Data traffic ranges from 10KB to 1MB per second. At low data traffic (10KB per second), the data processing delay of all algorithms is minimal. However, as data traffic increases to medium (100KB per second) and high (1MB per second) levels, the DTARA algorithm effectively reduces data processing delays. Under high data traffic, the DTARA algorithm achieves a data processing delay of approximately 50 milliseconds, compared to 150 milliseconds for the fixed task allocation algorithm and 100 milliseconds for the simple priority-based algorithm. The adaptive adjustment mechanism of the DTARA algorithm enables timely optimization of task and resource allocation under high data traffic, reducing data processing waiting times. Traditional algorithms, lacking dynamic adjustment capabilities, experience significant increases in data processing delays.

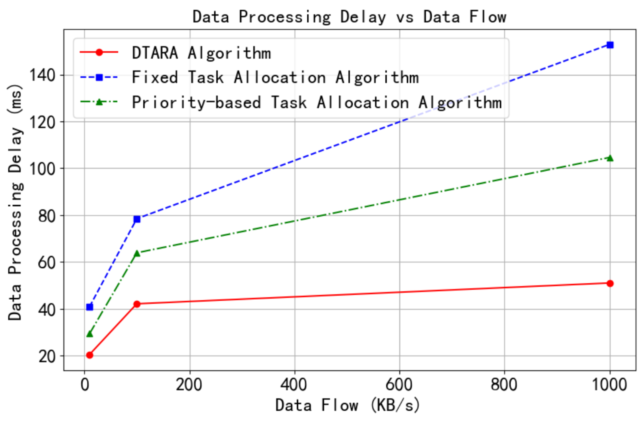

Figure 4: Data processing delay under different data flows.

5.4 Algorithm scalability and stability analysis

5.4.1 Scalability in large-scale task scenarios

In large-scale task scenarios (500-1000 tasks are executed simultaneously), the performance indicators of each algorithm are monitored, and the results are shown in Table 2.

As the number of tasks increases from 500 to 1000, the task completion time, CPU utilization, and data processing delay of the DTARA algorithm rise but remain within a relatively reasonable range [21]. For instance, when the task count reaches 1000, the DTARA algorithm achieves an average task completion time of 620.5 seconds, average CPU utilization of 70.2%, and average data processing delay of 110.8 milliseconds. In contrast, the fixed task allocation algorithm, lacking flexibility, experiences a sharp increase in task completion time, reaching 1500.3 seconds with 1000 tasks. CPU utilization becomes extremely imbalanced, dropping to 30.5%, and data processing delays surge to 450.2 milliseconds. The simple priority-based algorithm also falls short in dynamic resource allocation and overall scheduling, resulting in inferior performance compared to the DTARA algorithm under large-scale task scenarios. The DTARA algorithm quickly determines task execution order through its priority evaluation mechanism, preventing low-priority tasks from monopolizing resources and ensuring high-priority tasks are executed first. This maintains good performance in large-scale task scenarios and demonstrates strong scalability.

5.4.2 Algorithm stability analysis

In the long-term running experiment, the stability indicators of each algorithm are statistically analyzed, and the results are shown in Table 3.

Table 2: Algorithm scalability performance table.

Number of tasks	Algorithm	Mean task completion time (s)	Average CPU utilization (%)	Average data processing delay (ms)
500	DTARA algorithm	280.5 ± 30.2	65.3 ± 5.1	70.4 ± 8.2
	Fixed task allocation algorithm	560.8 ± 60.5	45.6 ± 8.3	180.2 ± 20.5
	Task allocation algorithm based on simple priority	420.6 ± 45.7	55.2 ± 7.4	120.5 ± 15.3
800	DTARA algorithm	450.8 ± 45.3	68.5 ± 6.2	90.6 ± 10.3
	Fixed task allocation algorithm	980.2 ± 100.3	38.9 ± 9.2	300.5 ± 30.2
	Task allocation algorithm based on simple priority	750.4 ± 70.6	48.7 ± 8.1	200.8 ± 25.4
1000	DTARA algorithm	620.5 ± 60.4	70.2 ± 7.1	110.8 ± 12.4
	Fixed task allocation algorithm	1500.3 ± 150.5	30.5 ± 10.1	450.2 ± 40.5
	Task allocation algorithm based on simple priority	1100.6 ± 110.7	42.3 ± 9.3	320.6 ± 35.4

Table 3: Algorithm Stability Table.

Algorithm	Task execution success rate (%)	CPU utilization fluctuation range (%)	System crash count
DTARA algorithm	97.8 ± 1.2	8.5 ± 2.1	0
Fixed task allocation algorithm	85.6 ± 3.5	25.3 ± 5.2	3
Task allocation algorithm based on simple priority	90.2 ± 2.3	18.4 ± 4.1	1

As indicated in Table 3, the DTARA algorithm demonstrates high stability during long-term operation. Its task execution success rate consistently remains above 97.8%, CPU utilization fluctuation is minimal at approximately 8.5%, and no system crashes occur. The fixed task allocation algorithm achieves a task execution success rate of 85.6%, with a high CPU utilization fluctuation of 25.3% and three system crashes. Although the simple priority-based algorithm has a relatively high task execution success rate, its CPU utilization fluctuation is significant at 18.4%, and one system crash occurs. The adaptive adjustment mechanism of the DTARA algorithm plays a crucial role in long-term operation. When resource utilization fluctuates, timely task allocation adjustments prevent system instability caused by excessive resource usage or idleness. This provides robust support for its application in actual long-term running systems.

6 Conclusion

This study proposes and thoroughly validates the DTARA algorithm to address efficiency issues in AI self-control systems and data processing units based on edge computing technology. In large-scale task scenario experiments, the fixed task allocation algorithm lacks flexibility, leading to dramatic increases in task completion time and imbalanced resource utilization. The simple priority-based task allocation algorithm also has deficiencies in dynamic resource scheduling and limited performance. The DTARA algorithm, with its priority evaluation and dynamic resource allocation mechanisms, effectively responds to task load changes, significantly reducing task completion time and data processing delays. In long-term stability tests, traditional algorithms and simple priority-based algorithms experience task execution failures, significant resource utilization fluctuations, and even system crashes. In contrast, the DTARA algorithm maintains a high task execution success rate, minimal resource utilization fluctuations, and stable system performance. This indicates that the DTARA algorithm achieves remarkable results in enhancing system efficiency and stability, laying a solid foundation for the widespread application of this technology in practical scenarios such as industrial control and intelligent transportation. The algorithm has been validated through hardware deployment in a smart factory testbed, demonstrating its practical feasibility in real-world environments. Future research can focus on further optimizing the algorithm to adapt to more complex and dynamic application environments.

References

[1] Gao, C. (2023). Efficiency of artificial intelligence automatic control system and data processing unit based on edge computing technology. International Journal of Emerging Electric Power Systems, 24(4), 519–528. https://doi.org/10.1515/ijeeps-2023-0115

[2] Chang, Z., Liu, S., Xiong, X., Cai, Z., & Tu, G. (2021). A survey of recent advances in edge-computing-powered artificial intelligence of things. IEEE Internet of Things Journal, 8(18), 13849–13875. https://doi.org/10.1109/JIOT.2021.3088875

[3] Hua, H., Li, Y., Wang, T., Dong, N., Li, W., & Cao, J. (2023). Edge computing with artificial intelligence: A machine learning perspective. ACM Computing Surveys, 55(9), 1–35. https://doi.org/10.1145/3555802

[4] Nain, G., Pattanaik, K. K., & Sharma, G. K. (2022). Towards edge computing in intelligent manufacturing: Past, present and future. Journal of Manufacturing Systems, 62, 588–611. https://doi.org/10.1016/j.jmsy.2022.01.010

[5] Chavhan, S., Gupta, D., Gochhayat, S. P., B, C. B., Khanna, A., Shankar, K., & Rodrigues, J. J. (2022). Edge computing AI-IoT integrated energy-efficient intelligent transportation system for smart cities. ACM Transactions on Internet Technology, 22(4), 1–18. https://doi.org/10.1145/3507906

[6] Zhu, S., Ota, K., & Dong, M. (2021). Green AI for IIoT: Energy efficient, intelligent edge computing for the industrial internet of things. IEEE Transactions on Green Communications and Networking, 6(1), 79–88. https://doi.org/10.1109/TGCN.2021.3100622

[7] Lv, Z., Qiao, L., Verma, S., & Kavita. (2021). AI-enabled IoT-edge data analytics for connected living. ACM Transactions on Internet Technology, 21(4), 1–20. https://doi.org/10.1145/3421510

[8] Thota, R. C. (2024). Optimizing edge computing and AI for low-latency cloud workloads. International Journal of Science and Research Archive, 13(1), 3484–3500. https://doi.org/10.30574/ijsra.2024.13.1.1761

[9] Singh, A., Satapathy, S. C., Roy, A., & Gutub, A. (2022). AI-based mobile edge computing for IoT: Applications, challenges, and future scope. Arabian Journal for Science and Engineering, 47(8), 9801–9831.

[10] Hayyolalam, V., Aloqaily, M., Özkasap, Ö., & Guizani, M. (2021). Edge intelligence for empowering IoT-based healthcare systems. IEEE Wireless Communications, 28(3), 6–14. https://doi.org/10.48550/arXiv.2103.12144

[11] Zhu, S., Ota, K., & Dong, M. (2022). Energy-efficient artificial intelligence of things with intelligent edge. IEEE Internet of Things Journal, 9(10), 7525–7532. https://doi.org/10.1109/JIOT.2022.3143722

[12] Bajaj, K., Sharma, B., & Singh, R. (2022). Implementation analysis of IoT-based offloading frameworks on cloud/edge computing for sensor-generated big data. Complex & Intelligent Systems, 8(5), 3641–3658. https://doi.org/10.1007/s40747-021-00434-6

[13] Yu, W., Liu, Y., Dillon, T., & Rahayu, W. (2022). Edge computing-assisted IoT framework with an autoencoder for fault detection in manufacturing predictive maintenance. IEEE Transactions on Industrial Informatics, 19(4), 5701–5710. https://doi.org/10.1109/TII.2022.3178732

[14] Lu, S., Lu, J., An, K., Wang, X., & He, Q. (2023). Edge computing on IoT for machine signal processing and fault diagnosis: A review. IEEE Internet of Things Journal, 10(13), 11093–11116. https://doi.org/10.1109/JIOT.2023.3239944

[15] Duan, S., Wang, D., Ren, J., Lyu, F., Zhang, Y., Wu, H., & Shen, X. (2022). Distributed artificial intelligence empowered by end-edge-cloud computing: A survey. IEEE Communications Surveys & Tutorials, 25(1), 591–624. https://doi.org/10.1109/COMST.2022.3218527

[16] McEnroe, P., Wang, S., & Liyanage, M. (2022). A survey on the convergence of edge computing and AI for UAVs: Opportunities and challenges. IEEE Internet of Things Journal, 9(17), 15435–15459. https://doi.org/10.1109/JIOT.2022.3176400

[17] Liu, X., Yang, J., Zou, C., Chen, Q., Yan, X., Chen, Y., & Cai, C. (2021). Collaborative edge computing with FPGA-based CNN accelerators for energy-efficient and time-aware face tracking system. IEEE Transactions on Computational Social Systems, 9(1), 252–266. https://doi.org/10.1109/TCSS.2021.3059318

[18] Kasparaitis, P. (2025). Evaluation of Lithuanian Speech-to-Text Transcribers. Informatica, 1-16. https://doi.org/10.15388/25-INFOR591

[19] Munir, A., Blasch, E., Kwon, J., Kong, J., & Aved, A. (2021). Artificial intelligence and data fusion at the edge. IEEE Aerospace and Electronic Systems Magazine, 36(7), 62–78. https://doi.org/10.1109/MAES.2020.3043072

[20] Sanfilippo, S., Hernández-Gálvez, J. J., Hernández-Cabrera, J. J., Évora-Gómez, J., Roncal-Andrés, O., & Caballero-Ramirez, M. (2025). Evolving Electricity Demand Modelling in Microgrids Using a Kolmogorov-Arnold Network. Informatica, 1-22. https://doi.org/10.15388/25-INFOR590

[21] Rajavel, R., Ravichandran, S. K., Harimoorthy, K., Nagappan, P., & Gobichettipalayam, K. R. (2022). IoT-based smart healthcare video surveillance system using edge computing. Journal of Ambient Intelligence and Humanized Computing, 13(6), 3195–3207. https://doi.org/10.1007/s12652-021-03157-1