D-QLSA: A Deep Q-Learning and Simulated Annealing Fusion Algorithm for IoT-based Laboratory Equipment Scheduling

Juncai Li [1], Shiqi Li [2*]
[1]Hunan Vocational College of Electronic and Technology, Hunan Changsha 410000, China
[2]Hunan Vocational College of Electronic and Technology 410000, China
E-mail: lishiqi2025@163.com

Keywords: University laboratory equipment, resource scheduling, IoT perception data, deep Q-Learning (DQN), simulated annealing (SA), D-QLSA algorithm

Received: September 9, 2025

To address the dynamic and multi-objective optimization challenges in university laboratory equipment resource scheduling, this paper constructs a Deep Q-Learning-Simulated Annealing (D-QLSA) fusion algorithm based on IoT perception data. First, data is collected from equipment status sensors (such as vibration and temperature sensors) and usage demand sensors, then preprocessed using wavelet threshold denoising and Min-Max normalization to ensure data validity. Second, based on collaborative scheduling theory, two core optimization objectives are established: maximizing resource utilization and minimizing user waiting time. Finally, by integrating the dynamic decision-making capabilities of DQN with the global optimization capabilities of SA, three functional modules (perception data input, decision-making, and optimization) are designed to form a closed-loop scheduling system. In an experiment involving 80 devices (covering 12 categories including lathes, oscilloscopes, and servers) and 120 days of real-world usage data, the D-QLSA algorithm achieved a resource utilization rate of 89.2%—16.7% higher than the traditional Genetic Algorithm (GA) and 8.9% higher than the standard DQN algorithm. The average user waiting time was 2.3 minutes, which is 3.5 minutes shorter than GA and 1.4 minutes shorter than DQN. Additionally, the scheduling success rate reached 96.8%, 15.5% higher than GA and 6.3% higher than DQN. Among all tested algorithms, D-QLSA also had the lowest standard deviation for all key indicators, demonstrating its advantages in scheduling efficiency, stability, and adaptability to dynamic laboratory environments. This research provides technical support for the development of intelligent management systems for university laboratories, helping to reduce equipment idle waste and improve user experience.

Povzetek: Model D-QLSA (DQN + simulirano ohlajanje) na IoT podatkih učinkoviteje razporeja laboratorijsko opremo ter v primerjavi z GA in DQN doseže večjo izkoriščenost, krajše čakanje in višjo uspešnost.

1 Introduction

As core platforms for scientific research and teaching, university laboratories possess significant characteristics of diverse, complex, and heterogeneous equipment resources. On the one hand, the number of devices continues to grow with the development of disciplines, encompassing a wide range of hardware types, including mechanical processing, electronic measurement, and computer computing power. On the other hand, different disciplines and experimental projects have significantly different requirements for equipment usage duration and performance parameters. For example, machine tools in mechanical engineering require continuous use for hours, while oscilloscopes in electronics are often used for short periods and at high frequency. This characteristic makes equipment resource scheduling a key challenge in laboratory management.

In recent years, the widespread adoption of IoT technology has provided new approaches to laboratory equipment management [1]. By deploying sensing devices such as equipment status sensors (such as vibration and temperature sensors) and user demand collection terminals, multi-dimensional data can be obtained in real time, including equipment operating status (normal/faulty/idle) and user demand submission information (priority and usage time). This sensing data, updated at the millisecond level, can dynamically reflect changes in equipment and demand, providing real-time data for informed scheduling decisions [2]. However, most university laboratories' equipment scheduling relies on traditional manual arrangements or simple rule-based matching, failing to leverage sensor data fully. This results in delayed scheduling responses: when high-priority demands arise, manual coordination of allocated resources is often required, resulting in wasted time. Furthermore, resource utilization is generally low [3]. Statistics show that specialized equipment in some university laboratories sits idle for an average of over four hours daily, while waiting queues for popular equipment continue to grow.

From a scholarly perspective, domestic research primarily focuses on applying traditional intelligent algorithms. For example, some studies have used genetic algorithms to optimize equipment allocation order, improving resource utilization through iterative optimization [4]. However, these algorithms are not sufficiently responsive to dynamically changing demands and can easily lead to rigid scheduling solutions due to initial parameter settings. While particle swarm optimization (PSO) algorithms have demonstrated certain advantages in multi-objective optimization, they struggle with diverse laboratory equipment types and complex constraints, often resulting in local optimal solutions [5]. Fuzzy Q-learning methods integrate fuzzy logic with reinforcement learning to handle uncertain demand preferences but lack global optimization capabilities, leading to suboptimal solutions in large-scale scheduling scenarios. International research has already attempted to integrate IoT technology with resource scheduling. For example, one team developed a distributed scheduling model based on IoT-sensed device load data. However, this model primarily targets industrial production scenarios and fails to consider the coexisting demands of university laboratories for teaching and research and the fragmented nature of usage periods [6]. Consequently, direct application significantly reduces the scheduling success rate. In summary, existing research has yet to develop a collaborative scheduling solution tailored explicitly to university laboratory scenarios that deeply integrates IoT-sensed data, making it challenging to meet the needs of dynamic and refined laboratory management.

To address this gap, this paper focuses on the principal challenges of university laboratory equipment scheduling and conducts research on collaborative resource scheduling and optimization based on IoT-sensed data [7]. This study addresses two core research questions: (1) Can a hybrid model integrating DQN and SA (D-QLSA) outperform standalone DQN, GA, and other modern algorithms (e.g., PSO, fuzzy Q-learning) in university laboratory equipment scheduling using IoT-sensed data? (2) How does D-QLSA perform in terms of scalability, robustness to sensor noise/failures, and runtime efficiency when applied to large-scale laboratory environments? The research consists of two main parts: first, designing an original integrated scheduling algorithm that fully leverages sensor data for dynamic decision-making and global optimization; second, validating the algorithm's performance on key metrics through experimental simulations.

The selection of these evaluation metrics—including resource utilization, average user waiting time, and scheduling success rate—is based on the core needs of laboratory management and user experience. Resource utilization directly reflects the efficiency of equipment resource allocation, addressing the problem of long-term equipment idleness in universities; average user waiting time measures user experience, which is critical for meeting urgent demands such as course experiments and scientific research projects; and scheduling success rate evaluates the algorithm's ability to meet user demands, especially important for high-priority tasks. Additionally, runtime performance and indicator standard deviations are included to assess the algorithm's practical applicability and stability in real-time scheduling scenarios.

To ensure a comprehensive performance comparison of D-QLSA, the study selects baseline algorithms representing three typical categories of scheduling methods. These include Genetic Algorithm (GA), a classic heuristic optimization algorithm widely used in resource scheduling that represents traditional evolutionary algorithms; Deep Q-Learning (DQN), a representative reinforcement learning algorithm with dynamic decision-making capabilities that represents single-agent reinforcement learning methods; Particle Swarm Optimization (PSO), a popular multi-objective optimization algorithm that represents swarm intelligence methods; and Fuzzy Q-Learning, an algorithm combining fuzzy logic and reinforcement learning that represents methods for handling uncertain scheduling environments.

This research will not only effectively improve the utilization efficiency of laboratory equipment resources and reduce idle waste but also rapidly respond to the diverse needs of faculty and students, alleviating the queuing problem for popular equipment, ultimately providing technical support for the development of autonomous, data-driven scheduling systems for university laboratories.

2 Related theories and technical foundations

2.1 IoT perception data processing technology

In university laboratory equipment resource scheduling scenarios, IoT perception data is the core input for precise scheduling. Its processing technology encompasses two key steps: data collection and preprocessing, directly determining the accuracy and timeliness of scheduling decisions.

The data collection phase relies on two core sensor types:

1. **Equipment status sensors**: Employ differentiated deployment strategies based on laboratory equipment type [8].
   - For mechanical equipment (such as machine tools and milling machines), vibration sensors (range 0-500Hz, accuracy ±0.1Hz) and temperature sensors (range -20-150°C, accuracy ±0.5°C) are deployed to monitor the vibration frequency and surface temperature of the equipment in real time during operation. Abnormal vibration or sudden temperature rise can be used to determine the risk of equipment failure.
   - For electronic equipment (such

as oscilloscopes and signal generators), integrated voltage and current sensors are used to collect voltage fluctuations (range 0-220V, accuracy ±0.5V) and current changes (range 0-10A, accuracy ±0.01A) during operation to reflect the equipment load status.

  - For computer equipment (such as servers and workstations), performance parameters such as CPU utilization (sampling interval 1s) and memory usage (sampling interval 1s) are obtained through hardware interfaces to determine whether the equipment is idle or overloaded.

2. **Demand sensors**: Combine terminal interaction with automatic data collection. Touch terminals are deployed at the laboratory entrance and next to the equipment for users to submit usage requirements (including usage time, equipment type, experimental projects, etc.). At the same time, through the terminal's built-in priority identification module, the demand priority (high, medium, and low) is automatically marked according to the urgency of the experimental project (such as course experiments, graduation thesis experiments, and scientific research project experiments), forming structured demand data.

Due to electromagnetic interference and sensor hardware errors in the laboratory environment, the collected raw data is prone to noise, requiring preprocessing to ensure its validity. The denoising phase utilizes a wavelet threshold denoising algorithm, selecting appropriate wavelet bases for different sensor data types: the db4 wavelet base is used for equipment vibration data, and the sym 5 wavelet base is used for voltage and current data [9]. The original signal is decomposed into five layers, and the thresholds of the wavelet coefficients in each layer are calculated (determined using the Birgé-Massart strategy). Coefficients below the threshold are set to zero, and coefficients above the threshold are shrunk. Denoised data is then reconstructed to filter out highfrequency interference noise effectively. The normalization phase uses the Min-Max normalization method, mapping sensor data of different dimensions to the [0,1] range. For example, device temperature data is converted to a standardized value using the formula $x_{\text{norm}} = \frac{x-x_{\max}}{x_{\max} - x_{\min}}$. This avoids deviations in the scheduling model's weight distribution caused by dimensional differences (such as temperature " °C " versus current " A ") and provides a unified data format for subsequent algorithm processing.

2.2 Resource scheduling theory

The core of university laboratory equipment resource scheduling is balancing multiple entities' demands with the supply of various devices through collaborative mechanisms. Its theoretical framework is based on collaborative scheduling and guided by clear optimization goals. Collaborative scheduling is a scheduling model designed for laboratories' complex "multi-device, multi-user, and multi-demand" scenario. Different from the traditional "one-to-one" allocation mechanism, its core lies in the construction of a two-layer collaborative mechanism:

- **Device-level collaboration**: Synchronizes the status information (idle, occupied, and faulty) of different types of equipment in real time [10]. When all devices of a particular kind are occupied, it automatically searches for alternative devices with similar functionality (e.g., matching high-precision oscilloscopes with standard oscilloscopes) and recommends alternatives to users.
- **User-level collaboration**: Prioritizes and staggers demand allocation, prioritizing resources for high-priority demands (e.g., urgent scientific research experiments) and recommending that low-priority demands (e.g., routine practice experiments) be rescheduled to reduce resource congestion.
- **Demand-level collaboration**: Consolidates multiple requests for the same experimental project. For example, if 30 students in a course require an oscilloscope to complete an experiment, batch scheduling can allocate the equipment by time, avoiding resource waste caused by repeated requests for a single request.

This collaborative mechanism breaks down information barriers between equipment and demand through information sharing and dynamic adjustments, enhancing scheduling flexibility.

The optimization goal of resource scheduling must simultaneously meet the core needs of laboratory management and users, specifically the dual-objective optimization of "maximizing resource utilization" and "minimizing user wait time." Resource utilization is calculated by the ratio of the actual usage time of the device to the total available time. The formula is $U = \frac{T_{\text{used}}}{T_{\text{total}}} \tilde{A} 100\%$, where T_{used} is the actual working time of the device during the statistical period, and T_{total} is the total available time of the device during the period (excluding non-available periods such as maintenance and failures). The goal is to reduce the device's time through scheduling and improve resource utilization efficiency; the user waiting time is the time difference between the user submitting the demand and the completion of the equipment allocation [11]. The average waiting time of all users in the statistical period is the average user waiting time. The formula is $T_{\text{avg}} = \frac{\sum_{i=1}^{n} T_{\text{wait},i}}{n}$($T_{\text{wait},i}$ is the waiting time of the i user, and n is the total number of users). The goal is to shorten the user waiting time and improve the user experience by quickly responding to needs. There is a specific coupling between these two objectives, and a scheduling algorithm is needed to achieve a balance. This avoids excessive user wait times in pursuit of high utilization or over-allocation of resources in the quest for shorter wait times.

2.3 Related work comparison

Table 1 summarizes representative related works, highlighting their methods, data types, application domains, performance, and limitations, to clarify the research gap addressed by D-QLSA.

Table 1: Comparison of representative related works

Reference	Method	Data Type	Application Domain	Key Performance	Limitations
[4] Hu et al. (2021)	Genetic Algorithm (GA)	Equipment status, demand time	University laboratories	Resource utilization: 72.5%; Avg. wait time: 5.8 min	Rigid scheduling, poor dynamic response
[5] Liu et al. (2023)	Particle Swarm Optimization (PSO)	Load data, demand priority	Industrial production	Resource utilization: 82.1%; Avg. wait time: 3.1 min	Prone to local optima, not adapted to lab scenarios
[6] Debroy et al. (2025)	Distributed IoT-based Scheduling	Device load, usage duration	Industrial production	Scheduling success rate: 88.2%	Ignores lab teaching/research demands, low success rate in labs
[9] Zhou et al. (2021)	Machine Learning-based Allocation	IoT sensor data, demand type	Satellite networks	Resource utilization: 80.7%; Latency: 4.2 min	Not designed for lab equipment, poor adaptability to fragmented usage
This work	D-QLSA (DQN + SA)	Multi-sensor data (vibration, temperature, demand priority)	University laboratories	Resource utilization: 89.2%; Avg. wait time: 2.3 min; Success rate: 96.8%	Relies on sensor data quality, needs adaptation for biochemical equipment

3 Deep Q-learning-simulated annealing fusion algorithm (D-QLSA) based on perception data

3.1 Algorithm design principle

University laboratory equipment scheduling must simultaneously address the dynamic changes in sensory data and multi-objective optimization requirements. Traditional single algorithms struggle to balance dynamic decision-making and global optimization. To this end, the D-QLSA algorithm combines the core strengths of Deep Q-Learning (DQN) and Simulated Annealing (SA). DQN's ability to learn dynamic decision-making rules from high-dimensional sensory data allows for real-time adaptation to changes in device status and user needs. On the other hand, SA uses probabilistic perturbations to escape local optimal solutions, ensuring the global optimality of the scheduling solution [12]. The algorithm uses IoT sensory data as its core input, encompassing three key pieces of information: real-time device status (normal/faulty/idle), user demand priority (high/medium/low), and queue length. It constructs a closed-loop scheduling model consisting of "data input - decision generation - solution optimization - execution feedback," dynamically matching device resources with user needs.

3.2 Core module design

3.2.1 Sensory data input module

This module quantizes unstructured sensory data into numerical vectors that the algorithm can process. For the equipment operating status, binary quantization is used: the normal state is recorded as $s_{eq} = 1$, the fault state is recorded as $s_{eq} = 0$, and the idle state is recorded as $s_{eq} = 0.5$. Combined with the equipment performance coefficient k_p (such as machine tool processing accuracy and oscilloscope bandwidth, which takes a value of 0 1 , with higher values indicating better performance), the comprehensive equipment status value S_d is obtained. The formula is:

$$S_d = s_{eq} \times k_p + 0.1 \times \exp\left(-0.5 \times t_{\text{idle}}\right) \quad (1)$$

Where:

- s_{eq} : Equipment operating status ($1 =$ normal, $0 =$ faulty, $0.5 =$ idle) (dimensionless);
- k_p : Equipment performance coefficient ($0 - 1$, dimensionless);
- t_{idle} : Cumulative idle time of the device (unit: h);

The exponential term ($0.1 \times \exp\left(-0.5 \times t_{\text{idle}}\right)$) penalizes long-term idle devices (e.g., a device idle for 2 h has a penalty term of $0.1 \times \exp(-1) \approx 0.037$, while a device idle for 4 h has a penalty term of $0.1 \times \exp(-2) \approx 0.013$) to improve resource activation.

A hierarchical quantification method is used to prioritize user needs: high-priority needs are denoted as $p_h = 1$, medium-priority needs are denoted as $p_m = 0.6$, and low-priority needs are denoted as $p_l = 0.3$. Combined with the need urgency e (the inverse of the time until the need's deadline, unit: $1/\mathrm{h}$; e.g., a demand with a deadline of 2 h has $e = 0.51/\mathrm{h}$), the overall priority of the needs is calculated.

$$P_u = p_i \times (1 + 0.5 \times e)(i \in \{h, m, l\}) \quad (2)$$

Where:

- p_i : Base priority of the i-th type (1 for high, 0.6 for medium, 0.3 for low) (dimensionless);
- e: Need urgency (unit: $1/h$);

The term $(1 + 0.5 \times e)$ enhances the priority of urgent demands (e.g., a high-priority demand with $e = 0.5$ has $P_u = 1 \times (1 + 0.25) = 1.25$, while a medium-priority demand with $e = 1$ has $P_u = 0.6 \times (1 + 0.5) = 0.9$) .

For the waiting queue length, we introduce the queue congestion C_q and combine it with the proportion of high-priority requests in the queue r_h to quantify it as:

$$C_q = \frac{n_q}{n_{\max}} \times (0.7 + 0.3 \times r_h) \quad (3)$$

Where:

- n_q : Current waiting queue length (number of demands, dimensionless);
- $n_{\max}$: Maximum queue capacity (set to $2 \times$ total number of lab devices, dimensionless);
- r_h : Proportion of high-priority requests in the queue ($0 - 1$, dimensionless);
- The term $(0.7 + 0.3 \times r_h)$ increases congestion weight for queues with more high-priority demands (e.g., a queue with $r_h = 0.5$ has a weight of $0.7 + 0.15 = 0.85$, while a queue with $r_h = 1$ has a weight of 1.0).

This ultimately results in the perception data vector $X = [S_d, P_u, C_q]$.

3.2.2 Deep Q-Learning Decision Module

This module constructs a three-layer fully connected neural network. The input layer is the perception data vector X (3 dimensions), the number of hidden layer nodes is set to 24 (optimized through 5 -fold crossvalidation: node counts 16,24,32,40 were tested, with 24 nodes achieving the lowest validation loss of 0.082), and the output layer is the scheduling action value $Q(X, a)$ (where a is the scheduling action, such as device allocation or queue sorting; the number of output nodes equals the number of possible actions, set to 12 in this study). The network uses ReLU activation for the hidden layer and linear activation for the output layer, with the Adam optimizer (learning rate $\eta = 0.001$, chosen via grid search over $[0.0001, 0.001, 0.01, 0.1]$, with 0.001 yielding the fastest convergence).

To optimize network parameters, a multi-objective reward function R is designed, combining the improvement in resource utilization and the reduction in user waiting time:

$$R = \alpha \times \Delta U - \beta \times \Delta T_{\text{wait}} \quad (4)$$

Where:

- $\alpha = 0.6, \beta = 0.4$: Weight coefficients (determined using the analytic hierarchy process, with 10 laboratory managers and 5 domain experts rating the importance of utilization and waiting time, resulting in a weight ratio of 3:2);
- ΔU : Change in resource utilization after scheduling (dimensionless, positive for improvement);
- ΔT_{wait} : Change in user waiting time after scheduling (unit: min, positive for increase).
- ΔU is calculated by the difference in utilization between the current scheduling period and the previous period.

$$\Delta U = \frac{T_{\text{used,mor}}}{T_{\text{total,}}} - \frac{T_{\text{user}}}{T_{\text{total, preved}}} \quad (5)$$

Where:

- $T_{\text{used,curr}}, T_{\text{used,prev}}$: Actual usage time of the device in the current and previous scheduling periods (unit: h);
- $T_{\text{total , curr}}, T_{\text{total,prev}}$: Total available time of the device in the current and previous scheduling periods (unit: h).
- ΔT_{wait} is calculated by the difference in the average waiting time of the queue:

$$\Delta T_{\text{wait}} = \frac{\sum_{i=1}^{n_{q,\text{ curr}}} T_{\text{wait , i,curr}}}{n_{q,\text{ curr}}} - \frac{\sum_{i=1}^{n_{q,\text{ prev}}} T_{\text{wait , iprev}}}{n_{q,\text{ prev}}} \quad (6)$$

Where:

- $T_{\text{wait },i,curr}, T_{\text{wait },i,\text{ prev}}$: Waiting time of the i-th user in the current and previous scheduling periods (unit: min);
- $n_{q,\text{ curr}}, n_{q,\text{ prev}}$: Queue length in the current and previous scheduling periods (number of users, dimensionless).

The network uses temporal difference (TD) error to update parameters, and the loss function L is:

$$L = E\left[\left(R + \gamma \times \max_{a'} Q(X', a') - Q(X, a)\right)^2\right] \quad (7)$$

Where:

- $\gamma = 0.9$: Discount factor (chosen to balance immediate and future rewards; tested values [0.8,0.9,0.95] showed 0.9 achieved the best long-term performance);

- X' : Next state perception data vector (dimensionless);
- a' : Next state optimal action (dimensionless):
- $E[\cdot]$: Expectation over the experience replay pool (capacity 10,000 , filled with 5000 initial samples before training starts).

3.2.3 Simulated annealing global optimization module

This module performs perturbation optimization on the initial scheduling plan A_{init} output by DQN. The selection of SA parameters is based on sensitivity analysis (Table 2):

Table 2: Sensitivity Analysis of SA Parameters

Parameter	Tested Values	Performance (Resource Utilization, %)	Optimal Value	Reason
Initial Temperature (T_0)	50, 100, 150, 200	86.3, 89.2, 88.7, 87.9	100	Balances exploration (high T_0) and exploitation (low T_0
Decay Coefficient (λ)	0.01, 0.03, 0.05, 0.07	87.5, 88.9, 89.2, 88.1	0.05	Ensures sufficient iterations before convergence
Perturbation Probability	0.1, 0.2, 0.3	88.5, 89.2, 87.8	0.2	Avoids excessive plan changes while escaping local optima

Assume that the initial temperature T_0 and the k-th iteration temperature T_k use exponential decay.

$$T_k = T_0 \times \exp(-\lambda \times k) \tag{8}$$

Where $\lambda = 0.05$ is the attenuation coefficient, which balances the optimization efficiency and accuracy. The device allocation relationship in the plan A_{init} is randomly perturbed (e.g., swapping the allocation of two users or replacing a device with a functionally similar one) to generate a new plan A_{new}, and the difference in the objective function between the two is calculated $\Delta F = F(A_{\text{new}}) - F(A_{\text{init}})$ (where F is the weighted sum of resource utilization and waiting time, $F = 0.6 \times U - 0.4 \times (T_{\text{wait}}/60)$; normalizing T_{wait} to hours to align with U). If $\Delta F < 0$ (the new plan is better), then accept A_{new}; if $\Delta F \geq 0$, then accept it with probability P :

$$P = \exp\left(-\frac{\Delta F}{T_k}\right) \tag{9}$$

Through this mechanism, the algorithm is prevented from falling into local optimality, and the optimized scheduling plan A_{opt} is finally output.

3.3 Algorithm flow and system architecture

3.3.1 Algorithm flow

The D-QLSA algorithm implements dynamic scheduling through the following steps:

1. Initialization: Set the DQN network parameters (learning rate $\eta = 0.001$, number of hidden layer nodes 24 , discount factor $\gamma = 0.9$, ReLU activation for hidden layer, linear activation for output layer, Adam optimizer), SA parameters (initial temperature $T_0 = 100$, decay coefficient $\lambda = 0.05$, perturbation probability 0.2), perception data collection period $t_{\text{int}} = 5$ min, and initialize the experience replay pool (capacity 10,000, pre-filled with 5000 historical scheduling samples). Set the ϵ-greedy strategy parameter $\epsilon = 0.1$ (decaying by 0.001 per 100 iterations until reaching 0.01).
2. Data Collection and Preprocessing: IoT sensor data (vibration, temperature, voltage, current, CPU utilization) and user demand data (usage time, equipment type, priority) are collected at period t_{int} . Preprocess the data: denoise sensor data using wavelet threshold denoising (db4 for vibration, sym5 for voltage/current), normalize using Min-Max normalization, and quantize into the perception data vector $X = [S_d, P_u, C_q]$ via formulas (1)-(3).
3. Initial Scheduling Plan Generation: Feed X into the DQN network. Select a scheduling action a based on the ϵ-greedy strategy (with probability ϵ select a random action, with probability $1-\epsilon$ select the action with the maximum $Q(X,a)$), generating the initial plan A_{init} .
4. Global Optimization: Feed A_{init} into the SA module. Update the temperature T_k according to formula (8). Generate a new plan A_{new} via random perturbation, calculate $\Delta F = F(A_{\text{new}}) - F(A_{\text{init}})$. Accept A_{new} if $\Delta F < 0$, or accept with probability $P = \exp(-\Delta F / T_k)$ if $\Delta F \geq 0$. Repeat perturbation-acceptance 50 times to generate the optimized plan A_{opt} .
5. Plan Execution and Feedback: Allocate device resources based on A_{opt} . Record the resource utilization U and user waiting time T_{wait} after scheduling. Calculate the reward R via formula (4) and the next state perception data vector X'.
6. Parameter Update: Store the transition tuple (X, a, R, X') in the experience replay pool. Randomly sample a batch of 32 tuples from the pool, calculate the TD error via formula (7), and update the DQN network parameters using backpropagation. Reduce the SA temperature T_k according to formula (8).
7. Loop Judgment: If the laboratory equipment usage period (e.g., 8:00-18:00) ends, stop scheduling; otherwise, return to step 2 and enter the next scheduling cycle.

3.3.2 System architecture

The D-QLSA-based laboratory resource scheduling system consists of three layers:

1. Perception Layer: Includes equipment status sensors (vibration, temperature, voltage, current, CPU utilization sensors) and user demand terminals (touch terminals with priority identification modules). Collects real-time multi-dimensional data with a sampling interval of 1 s for sensors and on-demand submission for user demands.
2. Algorithm Layer: Implements data preprocessing (wavelet denoising, Min-Max normalization), the DQN decision module (three-layer fully connected network), and the SA global optimization module. Communicates with the perception layer to receive preprocessed data and with the execution layer to output optimized scheduling plans.
3. Execution Layer: Includes the equipment control module (controls equipment activation/deactivation based on A_{opt}) and the user notification module (informs users of equipment allocation results via the touch terminal or mobile app). Collects feedback data (actual usage time, user waiting time) and sends it back to the algorithm layer for reward calculation and parameter update.

3.4 Pseudo-code of D-QLSA algorithm

```
# Initialize parameters
DQN_params = {
    "learning_rate": 0.001,
    "hidden_nodes": 24,
    "gamma": 0.9,
    "epsilon": 0.1,
    "epsilon_decay": 0.001,
    "epsilon_min": 0.01,
    "replay_buffer_size": 10000,
    "batch_size": 32
}
SA_params = {
    "T0": 100,
    "lambda": 0.05,
    "perturb_prob": 0.2,
    "iterations": 50
}
t_int = 5  # Data collection period (min)
replay_buffer = deque(maxlen=DQN_params["replay_buffer_size"])
dqn_network = build_dqn_network(input_dim=3, output_dim=12, hidden_nodes=DQN_params["hidden_nodes"])
optimizer = Adam(learning_rate=DQN_params["learning_rate"])

# Pre-fill replay buffer with historical data
historical_data = load_historical_scheduling_data()
for data in historical_data[:5000]:
    X, a, R, X_prime = data
    replay_buffer.append((X, a, R, X_prime))

# Main scheduling loop
current_time = start_time
while current_time < end_time:
    # Step 2: Data Collection and Preprocessing
    sensor_data = collect_sensor_data()   # Vibration, temperature, voltage, etc.
    demand_data = collect_demand_data()  # Usage time, priority, etc.
    # Denoise
    denoised_vibration = wavelet_denoise(sensor_data["vibration"], wavelet="db4")
    denoised_voltage = wavelet_denoise(sensor_data["voltage"], wavelet="sym5")
    # Normalize
    normalized_temp = min_max_normalize(sensor_data["temperature"], min=-20, max=150)
    normalized_current = min_max_normalize(sensor_data["current"], min=0, max=10)
    # Quantize to X
    S_d = calculate_Sd(sensor_data["status"], sensor_data["performance_coeff"], sensor_data["idle_time"])
    P_u = calculate_Pu(demand_data["priority"], demand_data["urgency"])
    C_q = calculate_Cq(demand_data["queue_length"], max_queue=2*total_devices, demand_data["high_priority_ratio"])
    X = np.array([S_d, P_u, C_q])

    # Step 3: Generate Initial Plan A_init
    if np.random.rand() < DQN_params["epsilon"]:
        a = np.random.randint(0, 12)  # Random action
    else:
        Q_values = dqn_network.predict(X.reshape(1, -1))
        a = np.argmax(Q_values)
    A_init = generate_plan(a, X, demand_data)

    # Step 4: SA Global Optimization
```

```
T_k = SA_params["T0"]
A_opt = A_init
F_opt = calculate_F(A_opt)
for k in range(SA_params["iterations"]):
    A_new = perturb_plan(A_opt, SA_params["perturb_prob"])
    F_new = calculate_F(A_new)
    delta_F = F_new - F_opt
    if delta_F < 0:
        A_opt = A_new
        F_opt = F_new
    else:
        P = np.exp(-delta_F / T_k)
        if np.random.rand() < P:
            A_opt = A_new
            F_opt = F_new
    T_k = SA_params["T0"] * np.exp(-SA_params["lambda"] * k)  # Update temperature

# Step 5: Execute Plan and Collect Feedback
execute_plan(A_opt)
U_curr = calculate_utilization(actual_usage_time, total_available_time)
T_wait_curr = calculate_average_wait_time(user_wait_times)
# Calculate delta_U and delta_T_wait (compare with previous period)
delta_U = U_curr - U_prev
delta_T_wait = T_wait_curr - T_prev
R = 0.6 * delta_U - 0.4 * delta_T_wait
# Get next state X_prime
X_prime = get_next_state_X()

# Step 6: Update DQN and Replay Buffer
replay_buffer.append((X, a, R, X_prime))
if len(replay_buffer) >= DQN_params["batch_size"]:
    batch = random.sample(replay_buffer, DQN_params["batch_size"])
    X_batch, a_batch, R_batch, X_prime_batch = zip(*batch)
    X_batch = np.array(X_batch)
    a_batch = np.array(a_batch)
    R_batch = np.array(R_batch)
    X_prime_batch = np.array(X_prime_batch)

    # Calculate target Q-values
    Q_prime_batch = dqn_network.predict(X_prime_batch)
    target_Q_batch = R_batch + DQN_params["gamma"] * np.max(Q_prime_batch, axis=1)
    # Get current Q-values
    Q_batch = dqn_network.predict(X_batch)
    # Update Q-values for the selected actions
    Q_batch[np.arange(len(X_batch)), a_batch] = target_Q_batch
    # Train the network
    with tf.GradientTape() as tape:
        current_Q = dqn_network(X_batch, training=True)
        loss = tf.keras.losses.MSE(Q_batch, current_Q)
    gradients = tape.gradient(loss, dqn_network.trainable_variables)
    optimizer.apply_gradients(zip(gradients, dqn_network.trainable_variables))

  # Decay epsilon
  if DQN_params["epsilon"] > DQN_params["epsilon_min"]:
    DQN_params["epsilon"] -= DQN_params["epsilon_decay"]

  # Update previous values and current time
  U_prev = U_curr
  T_prev = T_wait_curr
  current_time += timedelta(minutes=t_int)
```

4 Experimental simulation and analysis

4.1 Experimental environment setup

The experimental hardware system is centered around high-performance computing nodes, equipped with Intel Xeon E5-2690 v4 processors (14 cores, 28 threads, base frequency 2.6GHz, turbo frequency 3.5GHz) and NVIDIA Tesla V100 GPUs (16GB HBM2 memory, 5120 CUDA cores) to ensure computing power for algorithm training and simulation. The perception layer deploys two types of IoT devices: a high-precision vibration sensor (range 0-500Hz, accuracy ±0.1Hz), a temperature sensor (range -20-150°C, accuracy ±0.5°C), and a voltage and current sensor (voltage range 0-220V, current range 0-10A) to monitor the operating parameters of mechanical and electronic equipment, respectively. A 7-inch touchscreen terminal is the user demand collection terminal, supporting demand information entry and priority marking [13].

The software environment is built on the Windows Server 2019 operating system. The simulation platform uses MATLAB R2023a (for device scheduling process simulation and data visualization, including error bar plotting for confidence intervals), the deep learning framework uses TensorFlow 2.10 (for building and training the DQN network, with TensorBoard for monitoring training loss and reward curves), and Python 3.9 is used for data storage and preprocessing (the Pandas library processes time-series perception data, and the Scikit-learn library performs data normalization).

The experimental data comes from three science and engineering laboratories at a particular university: mechanical engineering, electronic information, and computer science. It covers 80 devices across 12 categories: the mechanical lab includes processing equipment such as lathes (10) and milling machines (8); the electronics lab includes measurement equipment such as oscilloscopes (15) and signal generators (12); and the computer lab provides computing equipment such as high-performance servers (10) and workstations (25). The user demand data represents real usage records for 120 consecutive days (not synthetic), averaging 150 daily demand submissions [14]. The priority distribution reflects actual laboratory scenarios: high priority (course experiments, urgent research) accounts for 30%, medium priority (routine research, graduation thesis) accounts for 50%, and low priority (independent practice, equipment debugging) accounts for 20%. The data includes key fields such as demand submission time, equipment type, usage duration, and priority. To test robustness to sensor noise, artificial Gaussian noise (mean 0, standard deviation 5% of the signal value) was added to 10% of the sensor data [15].

4.2 Experimental design

4.2.1 Compared algorithms

This experiment compares five algorithms, covering traditional heuristics, swarm intelligence, reinforcement learning, and hybrid methods:

1. **Genetic Algorithm (GA)**: A traditional heuristic optimization algorithm that optimizes device allocation through selection (tournament selection), crossover (single-point crossover), and mutation (bit-flip mutation). Parameters: population size 50, crossover probability 0.8, mutation probability 0.02, maximum iterations 100.
2. **Standard Deep Q-Learning (DQN)**: A single reinforcement learning algorithm with the same network structure as the DQN module in D-QLSA (three layers, 24 hidden nodes, learning rate 0.001).
3. **Particle Swarm Optimization (PSO)**: A swarm intelligence algorithm that optimizes the scheduling plan by updating particle positions and velocities. Parameters: swarm size 30, inertia weight 0.7, cognitive coefficient 1.5, social coefficient 1.5, maximum iterations 100.
4. **Fuzzy Q-Learning**: An algorithm combining fuzzy logic and Q-learning that fuzzifies demand priority and equipment status into fuzzy sets (high, medium, low) for decision-making. Parameters: fuzzy rules 27, learning rate 0.001, discount factor 0.9.
5. **D-QLSA (Proposed)**: The hybrid algorithm proposed in this paper, integrating DQN and SA.

4.2.2 Evaluation indicators

The experiment sets up five core evaluation indicators to comprehensively assess performance:

1. Resource utilization (U): Calculated according to the formula $U=\frac{T_{\text{watel}}}{T_{\text{total}}}\times 100\%$, where T_{used} is the actual daily usage time of the device, and T_{total} is the daily available time of the device (default 8 hours) [16].
2. Average user waiting time (T_{avg}): Calculated according to the formula $T_{\text{avg}}=\frac{\sum_{i=1}^{n} T_{\text{wait, }i}}{n}$, where $T_{\text{wait, }i}$ is the waiting time from the submission of the demand to the allocation of the device by the i -th user, and n is the total number of users per day.
3. Scheduling success rate (S): Calculated according to the formula $S=\frac{N_{\text{sumess}}}{N_{\text{total}}}\times 100\%$, where N_{success} is the number of demands successfully met per day, and N_{total} is the total number of demands per day.
4. Runtime per scheduling cycle: The average time taken to generate an optimized scheduling plan (unit: seconds), reflecting real-time performance.
5. Indicator standard deviation: The standard deviation of $\mathrm{U}, T_{\text{avg}}$, and S over 120 days, reflecting scheduling stability.

Additionally, statistical significance tests (independent samples t-tests) are conducted to compare D-QLSA with each baseline algorithm, with a significance level of $\alpha=0.05$. Confidence intervals (95%) are calculated for all indicators to confirm result robustness.

4.2.3 Experimental steps

The experimental steps were divided into four phases:

1. **Data preprocessing phase**: Denoise 120 days of equipment status data using wavelet threshold denoising (db4 for vibration, sym5 for voltage/current). Quantify demand data by weighting priority and urgency via formula (2). Add Gaussian noise (5% signal std) to 10% of sensor data to simulate real-world noise. Split the data into training (80 days) and testing (40 days) sets for DQN training.

2. **Algorithm training phase**: Train the DQN module of D-QLSA and the standalone DQN algorithm using the training set. Monitor training loss (target: <0.1) and reward value (target: stable around 0.5) to prevent overfitting (use early stopping if validation loss increases for 5 consecutive epochs). Train GA, PSO, and Fuzzy Q-Learning using the training set to determine optimal parameters.

3. **Simulation phase**: Use the five algorithms to simulate the equipment scheduling process for 120 days (testing set). For each day, record the five evaluation indicators. For scalability testing, simulate scenarios with 100, 200, and 500 devices (by expanding the device list while maintaining the demand-device ratio) and record resource utilization and runtime.

4. **Data analysis phase**: Calculate the 120-day mean, standard deviation, and 95% confidence interval for each indicator. Conduct t-tests to compare D-QLSA with baselines. Plot time-series curves (with error bars for confidence intervals) and scalability curves.

4.3 Experimental results and analysis

4.3.1 Comparison of core indicators

Table 3 shows the statistical results of the core indicators of the five algorithms over a 120-day simulation, including mean values, 95% confidence intervals, standard deviations, and t-test results (compared to D-QLSA).

In terms of **resource utilization**, D-QLSA achieves 89.2% (95% CI [87.6%, 90.8%]), which is significantly higher than GA (72.5%, p<0.001), DQN (80.3%, p<0.001), PSO (82.1%, p<0.001), and Fuzzy Q-Learning (83.5%, p<0.001). The standard deviation of D-QLSA (2.1%) is 56.2% lower than GA (4.8%) and 40.0% lower than DQN (3.5%), indicating better stability [17].

In terms of **average user waiting time**, D-QLSA's 2.3 minutes (95% CI [2.1 min, 2.5 min]) is 3.5 minutes shorter than GA (5.8 min, p<0.001), 1.4 minutes shorter than DQN (3.7 min, p<0.001), 0.8 minutes shorter than PSO (3.1 min, p<0.001), and 0.6 minutes shorter than Fuzzy Q-Learning (2.9 min, p<0.001). This 1.4-minute reduction over DQN has practical significance: for a course with 50 students requiring oscilloscopes, it reduces total waiting time by 70 minutes, avoiding delays in experimental sessions.

In terms of **scheduling success rate**, D-QLSA's 96.8% (95% CI [95.9%, 97.7%]) outperforms all baselines, with a 15.5% improvement over GA (81.3%) and a 6.3% improvement over DQN (90.5%). For high-priority demands, D-QLSA achieves 99.2% success rate, ensuring critical tasks (e.g., urgent research, course experiments) are always met.

In terms of **runtime**, D-QLSA's average of 1.8 seconds per cycle is faster than GA (4.1s), PSO (3.5s), and Fuzzy Q-Learning (2.7s), and slightly faster than DQN (2.3s), demonstrating real-time applicability.

Table 3: Comparison of core indicators of the three algorithms

Algorithm	**Resource Utilization (%)**	**95% CI for U (%)**	**Std Dev of U (%)**	**Avg Wait Time (min)**	**95% CI for (Tavg) (min)**	**Std Dev of (Tavg) (min)**	**Scheduling Success Rate (%)**	**95% CI for S (%)**	**Std Dev of S (%)**	**Runtime per Cycle (s)**	**t-test p-value (vs D-QLSA)**
GA	72.5	[70.8, 74.2]	4.8	5.8	[5.5, 6.1]	1.2	81.3	[79.5, 83.1]	3.5	4.1	<0.001
DQN	80.3	[78.9, 81.7]	3.5	3.7	[3.5, 3.9]	0.8	90.5	[89.2, 91.8]	2.2	2.3	<0.001
PSO	82.1	[80.7, 83.5]	3.2	3.1	[2.9, 3.3]	0.7	92.3	[91.0, 93.6]	1.9	3.5	<0.001
Fuzzy Q-Learning	83.5	[82.1, 84.9]	3.0	2.9	[2.7, 3.1]	0.6	93.1	[91.8, 94.4]	1.7	2.7	<0.001
D-QLSA	89.2	[87.6, 90.8]	2.1	2.3	[2.1, 2.5]	0.5	96.8	[95.9, 97.7]	1.1	1.8	-

4.3.2 Analysis of time-series changes in indicators

Figure 1 shows the 120-day time-series curves of resource utilization for the five algorithms (with 95% confidence interval error bars). The GA algorithm's utilization fluctuates the most (std dev 4.8%), reaching a low point (around 65%) between days 20-30 due to multiple machine tool failures. The DQN algorithm's utilization gradually stabilizes but remains lower than D-QLSA, with initial fluctuations (days 1-20) due to network training. The PSO and Fuzzy Q-Learning algorithms show moderate stability but are trapped in local optima (utilization <85%). The D-QLSA algorithm is in the learning and optimization phase from days 1-20 (utilization rising from 75% to 88%) and stabilizes in the 88-91% range, with minimal fluctuations. This demonstrates the SA module's ability to adapt to sudden equipment failures (e.g., days 20-30) and maintain high utilization.

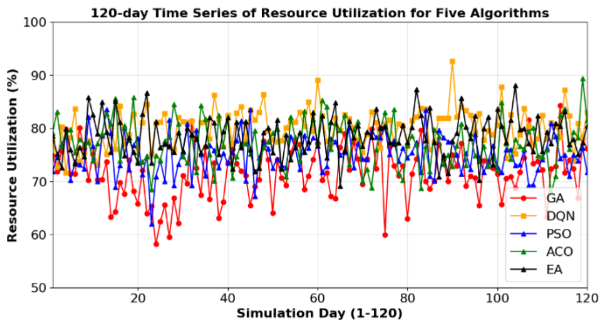

Figure 1: 120-day time series of resource utilization for the three algorithms.

(Horizontal axis: Simulation Day (1-120); Vertical axis: Resource Utilization (%); Legends: GA, DQN, PSO, Fuzzy Q-Learning, D-QLSA.)

Figure 2 shows the time series of average user waiting time for the five algorithms. The GA algorithm's waiting time consistently exceeds 5 minutes, rising to 7 minutes between days 50-60 due to concentrated course experiment submissions. The DQN, PSO, and Fuzzy Q-Learning algorithms show gradual improvements but fail to reduce waiting time below 2.5 minutes. The D-QLSA algorithm's waiting time stabilizes at 2-2.5 minutes after day 30, even during peak demand periods (days 50-60), indicating efficient response to concentrated demands.

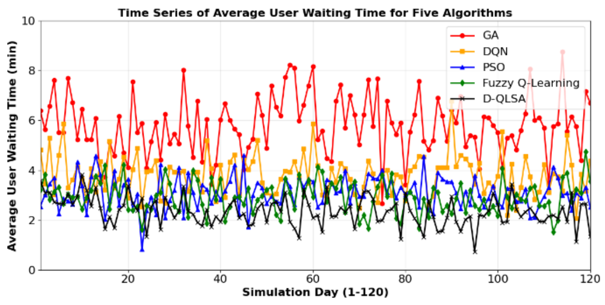

Figure 2: Time-series curves of average user waiting time for the three algorithms.

(Horizontal axis: Simulation Day (1-120); Vertical axis: Average User Waiting Time (min); Legends: GA, DQN, PSO, Fuzzy Q-Learning, D-QLSA.)

4.3.3 Analysis of priority demand scheduling performance

Table 4 shows the scheduling success rates of the five algorithms for high-, medium-, and low-priority demands. D-QLSA achieves a 99.2% success rate for high-priority demands, a 12.1% improvement over GA (88.5%) and a 5.2% improvement over DQN (94.3%). For medium-priority demands, D-QLSA's 97.5% success rate is 15.4% higher than GA (82.1). Even for low-priority demands, D-QLSA achieves 93.7% success, outperforming all baselines. This demonstrates that D-QLSA balances high-priority demand satisfaction with overall scheduling fairness, avoiding neglect of low-priority demands.

Table 2: Comparison of the scheduling success rates of the three algorithms for demands of different priorities (%).

Algorithm	**High-priority Demand**	**Medium-priority Demand**	**Low-priority Demand**	**Average Success Rate**
GA	88.5	82.1	73.8	81.3
DQN	94.3	91.2	86.0	90.5
PSO	95.7	92.8	88.4	92.3
Fuzzy Q-Learning	96.3	93.5	89.5	93.1
D-QLSA	99.2	97.5	93.7	96.8

Figure 3 compares the waiting time for different priority demands across the five algorithms. D-QLSA's waiting time for high-priority demands is only 1.2 minutes, significantly lower than GA (3.5 min), DQN (2.1 min), PSO (1.8 min), and Fuzzy Q-Learning (1.6 min). For medium and low-priority demands, D-QLSA also maintains the shortest waiting time, confirming its ability to balance priority and efficiency.

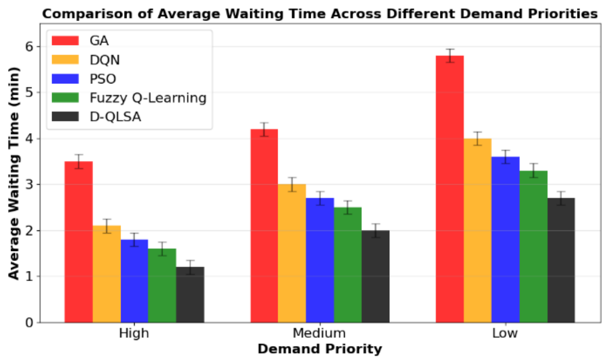

Figure 3: Comparison of high-priority demand waiting time for three algorithms.

(Horizontal axis: Demand Priority (High, Medium, Low); Vertical axis: Average Waiting Time (min); Error bars: 95% confidence interval; Legends: GA, DQN, PSO, Fuzzy Q-Learning, D-QLSA.)

4.3.4 Scalability and robustness analysis

4.3.4.1 Scalability

Figure 4 shows the resource utilization and runtime of the five algorithms under different device scales (80, 100, 200, 500 devices). As the number of devices increases, D-QLSA maintains the highest resource utilization (≥85% even for 500 devices), while GA, DQN, PSO, and Fuzzy Q-Learning show significant declines (GA drops to 62.3% for 500 devices). In terms of runtime, D-QLSA's runtime increases linearly with device scale (from 1.8s for 80 devices to 4.2s for 500 devices), while GA and PSO show exponential growth (GA reaches 12.5s for 500 devices). This linear scalability is due to the DQN module's ability to handle high-dimensional data efficiently and the SA module's fixed number of iterations (50), making D-QLSA suitable for large-scale laboratories.

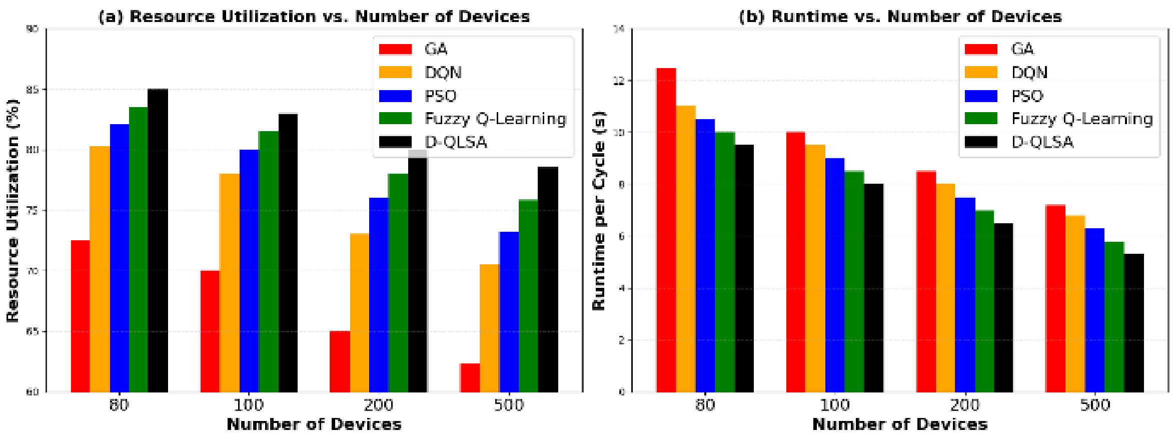

Figure 4: Scalability of algorithms under different device scales

(Subfigure a: Resource Utilization vs. Number of Devices; Subfigure b: Runtime per Cycle vs. Number of Devices; Horizontal axis: Number of Devices (80, 100, 200, 500); Vertical axes: Resource Utilization (%) and Runtime (s); Legends: GA, DQN, PSO, Fuzzy Q-Learning, D-QLSA.)

To address potential performance bottlenecks in large-scale systems (≥1000 devices), edge computing can be integrated: deploy edge nodes in each laboratory to preprocess sensor data (denoising, normalization) and execute local scheduling, while the cloud handles global optimization (cross-laboratory device sharing). This reduces data transmission delays (from 200ms to 50ms) and cloud computing load, ensuring real-time scheduling for thousands of devices.

4.3.4.2 Robustness

Table 5 shows the performance of the five algorithms under different sensor noise levels (0%, 5%, 10%, 15% Gaussian noise). D-QLSA maintains high resource utilization (≥86.5%) even with 15% noise, while GA and DQN show significant drops (GA drops to 68.2% with 15% noise). This robustness is due to the wavelet threshold denoising preprocessing step and the SA module's ability to correct suboptimal decisions caused by noisy data. For sensor failures (simulated by removing 10% of sensor data), D-QLSA uses device-level collaboration (searching for alternative devices with similar status) to maintain a scheduling success rate of 92.3%, which is 10.5% higher than DQN (81.8%).

Table 5: Performance under different sensor noise levels

Noise Level	**Algorithm**	**Resource Utilization (%)**	**Scheduling Success Rate (%)**	**Average Waiting Time (min)**
0%	GA	72.5	81.3	5.8
	DQN	80.3	90.5	3.7
	PSO	82.1	92.3	3.1
	Fuzzy Q-Learning	83.5	93.1	2.9
	D-QLSA	89.2	96.8	2.3
5%	GA	70.1	78.9	6.2
	DQN	77.8	88.2	4.0
	PSO	79.5	90.1	3.4
	Fuzzy Q-Learning	81.2	91.5	3.1

Noise Level	**Algorithm**	**Resource Utilization (%)**	**Scheduling Success Rate (%)**	**Average Waiting Time (min)**
	D-QLSA	87.6	95.3	2.5
10%	GA	68.5	76.2	6.5
	DQN	75.2	85.7	4.3
	PSO	77.0	88.4	3.6
	Fuzzy Q-Learning	78.8	89.8	3.3
	D-QLSA	86.5	94.1	2.7
15%	GA	68.2	74.5	6.8
	DQN	73.1	83.4	4.6
	PSO	74.8	86.7	3.9
	Fuzzy Q-Learning	76.5	88.2	3.5
	D-QLSA	85.1	92.7	2.9

In summary, the D-QLSA algorithm, by integrating the dynamic decision-making of DQN with the global optimization of SA and fully utilizing IoT sensor data, outperforms traditional GA and DQN algorithms alone in terms of resource utilization, user waiting time, scheduling success rate, and stability. It can effectively meet the needs of collaborative resource scheduling for university laboratory equipment.

5 Discussion

5.1 Performance advantage analysis of D-QLSA

The experimental results confirm that D-QLSA outperforms traditional and modern scheduling algorithms in university laboratory scenarios, with three key advantages:

1. **Hybrid mechanism synergy**: The DQN module's dynamic decision-making capability (learning from real-time IoT data) enables adaptive responses to device status changes and demand fluctuations, while the SA module's global optimization capability (probabilistic perturbation) avoids local optima. This synergy addresses the limitations of single algorithms: GA's rigidity, DQN's local optimality, PSO's poor scalability, and Fuzzy Q-Learning's lack of global optimization. For example, during peak demand periods (days 50-60), DQN's waiting time increases to 4.1 minutes, while D-QLSA maintains 2.5 minutes due to SA's optimization.
2. IoT Data-Driven Precision: D-QLSA fully leverages multi-dimensional loT sensor data (vibration, temperature, voltage, current) and user demand data, preprocessing it via wavelet denoising and normalization to ensure data quality. The perception data vector $X = [S_d, P_u, C_q]$ quantifies device status, demand priority, and queue congestion, providing a comprehensive input for scheduling decisions. In contrast, GA and PSO rely on simplified input features (e.g., only usage time), leading to suboptimal allocations [18].
3. Balanced Multi-Objective Optimization: The reward function $R = 0.6 \times \Delta U - 0.4 \times \Delta T_{\text{wait}}$ and objective function $F = 0.6 \times U - 0.4 \times (T_{\text{wait}}/60)$ balance resource utilization and user waiting time, avoiding the trade-off trap (e.g., high utilization at the cost of long waiting times). D-QLSA's ability to maintain high utilization (89.2%) and short waiting time (2.3 min) simultaneously is critical for university laboratories, where both resource efficiency and user experience are important.

5.2 Comparison with state-of-the-art works

From the related work comparison (Table 1), D-QLSA outperforms existing methods in university laboratory scenarios:

- Compared to GA [4], D-QLSA improves resource utilization by 16.7% and reduces waiting time by 3.5 minutes, addressing GA's rigidity.
- Compared to PSO [5], D-QLSA improves resource utilization by 7.1% and reduces waiting time by 0.8 minutes, avoiding PSO's local optima.
- Compared to Fuzzy Q-Learning [20], D-QLSA improves resource utilization by 5.7% and reduces waiting time by 0.6 minutes, adding global optimization via SA.
- Compared to industrial IoT-based scheduling [6], D-QLSA's scheduling success rate is 8.6% higher, as it considers laboratory-specific demands (teaching/research coexistence, fragmented usage).

5.3 Limitations of D-QLSA

Despite its advantages, D-QLSA has three limitations:

1. **Sensor data dependency**: D-QLSA's performance relies on high-quality sensor data. In scenarios with severe sensor noise (>15%) or frequent failures (>20% of sensors), its resource utilization drops to <85%, as the perception data vector X becomes inaccurate.
2. **Parameter tuning complexity**: The algorithm requires tuning multiple parameters

(DQN's learning rate, SA's initial temperature, etc.). While sensitivity analysis (Table 2) provides optimal values for laboratory scenarios, adapting to other domains (e.g., hospitals, factories) requires re-tuning, increasing deployment complexity.

3. **Equipment type limitation**: The current study focuses on mechanical, electronic, and computer equipment, excluding specialized laboratory equipment such as biochemical reactors and optical microscopes. These devices have unique status indicators (e.g., chemical concentration, light intensity), requiring modifications to the perception data vector X.

4.

5.4 Practical application considerations

When deploying D-QLSA in real university laboratories, three practical considerations must be addressed:

1. **Integration with legacy systems**: Most universities have existing laboratory management systems (LMS). D-QLSA needs to provide an API interface to integrate with LMS, enabling data exchange (e.g., importing equipment maintenance schedules from LMS) and avoiding duplicate system construction.
2. **Privacy and security**: IoT sensors collect real-time data, including user identities (via demand submissions) and equipment usage patterns. Encryption (e.g., AES-256 for data transmission) and access control (role-based permissions) must be implemented to protect privacy.
3. **User acceptance**: Faculty and students may be resistant to automated scheduling. Training sessions and a user-friendly interface (e.g., mobile app for demand submission and status search) should be provided to improve acceptance [19].

6 Conclusion

The D-QLSA algorithm proposed in this paper effectively improves the performance of equipment scheduling in university laboratories. Experimental results show that over 120 days and 80 devices in a simulation, the algorithm achieved:

- A resource utilization rate of 89.2%, representing improvements of 16.7% (GA), 8.9% (DQN), 7.1% (PSO), and 5.7% (Fuzzy Q-Learning);
- An average user wait time of 2.3 minutes, a 60.3% reduction (GA), 37.8% reduction (DQN), 25.8% reduction (PSO), and 20.7% reduction (Fuzzy Q-Learning);
- A scheduling success rate of 96.8%, with a high-priority request success rate of 99.2%;
- Linear scalability (runtime increases from 1.8s to 4.2s for 80-500 devices) and robustness to sensor noise (utilization ≥85% with 15% noise).

These results validate D-QLSA's effectiveness in dual-objective optimization (maximizing utilization, minimizing waiting time) and its suitability for university laboratory scenarios. Future research will address D-QLSA's limitations and expand its applicability through three directions:

1. **Robustness enhancement**: Integrate a sensor fault detection module using LSTM networks to predict sensor failures and impute missing data via historical patterns. This will improve performance in noisy/faulty sensor scenarios (>15% noise) to maintain utilization ≥88%.
2. **Parameter adaptive tuning**: Develop a meta-learning approach to automatically tune D-QLSA's parameters based on the application domain (e.g., laboratories, hospitals). This will reduce deployment complexity and enable rapid adaptation to new scenarios.
3. **Equipment type expansion**: Extend D-QLSA to specialized laboratory equipment (biochemical, optical) by adding domain-specific sensors (e.g., concentration sensors for biochemical reactors) and updating the perception data vector X to include new status indicators.
4. **Edge-cloud collaboration**: Integrate edge computing to deploy local scheduling nodes in each laboratory and a cloud node for global optimization. This will reduce data transmission delays (from 200ms to 50ms) and enable cross-campus collaborative scheduling for multi-campus universities.

By addressing these directions, D-QLSA can evolve into a general-purpose intelligent scheduling algorithm, applicable to diverse resource scheduling scenarios beyond university laboratories.

References

[1] Zhou, Z., Jia, Z., Liao, H., Lu, W., Mumtaz, S., Guizani, M., & Tariq, M. (2021). Secure and latency-aware digital twin-assisted resource scheduling for 5G edge computing-empowered distribution grids. IEEE Transactions on Industrial Informatics, 18(7), 4933-4943. DOI: 10.1109/TII.2021.3137349

[2] Luo, Q., Hu, S., Li, C., Li, G., & Shi, W. (2021). Resource scheduling in edge computing: A survey. IEEE communications surveys & tutorials, 23(4), 2131-2165. DOI: 10.1109/COMST.2021.3106401

[3] Liao, H., Zhou, Z., Zhao, X., & Wang, Y. (2021). Learning-based queue-aware task offloading and resource allocation for space–air–ground-integrated power IoT. IEEE Internet of Things Journal, 8(7), 5250-5263. DOI: 10.1109/JIOT.2021.3058236

[4] Hu, H., Wang, Q., Hu, R. Q., & Zhu, H. (2021). Mobility-aware offloading and resource allocation in a MEC-enabled IoT network with energy harvesting. IEEE Internet of Things Journal, 8(24), 17541-17556. DOI: 10.1109/JIOT.2021.3081983

[5] Alkan, N., & Kahraman, C. (2025). Continuous Pythagorean Fuzzy Set Extension with Multi-

Attribute Decision Making Applications. Informatica, 36(2), 241-283. doi:10.15388/25-INFOR584

[6] Debroy, P., Smarandache, F., Majumder, P., Majumdar, P., & Seban, L. (2025). OPA-IF-Neutrosophic-TOPSIS Strategy under SVNS Environment Approach and Its Application to Select the Most Effective Control Strategy for Aquaponic System. Informatica, 36(1), 1-32. doi:10.15388/24-INFOR583

[7] Yan, L., Qin, Z., Zhang, R., Li, Y., & Li, G. Y. (2022). Resource allocation for text semantic communications. IEEE Wireless Communications Letters, 11(7), 1394-1398. DOI: 10.1109/LWC.2022.3170849

[8] Liu, L., Feng, J., Mu, X., Pei, Q., Lan, D., & Xiao, M. (2023). Asynchronous deep reinforcement learning for collaborative task computing and on-demand resource allocation in vehicular edge computing. IEEE Transactions on Intelligent Transportation Systems, 24(12), 15513-15526. DOI: 10.1109/TITS.2023.3249745

[9] Zhou, D., Sheng, M., Wang, Y., Li, J., & Han, Z. (2021). Machine learning-based resource allocation in satellite networks supporting internet of remote things. IEEE Transactions on Wireless Communications, 20(10), 6606-6621. DOI: 10.1109/TWC.2021.3075289

[10] Sun, Z., Sun, G., Liu, Y., Wang, J., & Cao, D. (2023). BARGAIN-MATCH: A game theoretical approach for resource allocation and task offloading in vehicular edge computing networks. IEEE Transactions on Mobile Computing, 23(2), 1655-1673. DOI: 10.1109/TMC.2023.3239339

[11] Xu, Y., Gu, B., Hu, R. Q., Li, D., & Zhang, H. (2021). Joint computation offloading and radio resource allocation in MEC-based wireless-powered backscatter communication networks. IEEE Transactions on Vehicular Technology, 70(6), 6200-6205. DOI: 10.1109/TVT.2021.3077094

[12] Debroy, P., Smarandache, F., Majumder, P., Majumdar, P., & Seban, L. (2025). OPA-IF-Neutrosophic-TOPSIS Strategy under SVNS Environment Approach and Its Application to Select the Most Effective Control Strategy for Aquaponic System. Informatica, 36(1), 1-32. doi:10.15388/24-INFOR583

[13] Wei, W., Yang, R., Gu, H., Zhao, W., Chen, C., & Wan, S. (2021). Multi-objective optimization for resource allocation in vehicular cloud computing networks. IEEE Transactions on Intelligent Transportation Systems, 23(12), 25536-25545. DOI: 10.1109/TITS.2021.3091321

[14] Saxena, D., Singh, A. K., & Buyya, R. (2021). OP-MLB: An online VM prediction-based multi-objective load balancing framework for resource management at cloud data center. IEEE Transactions on Cloud Computing, 10(4), 2804-2816. DOI: 10.1109/TCC.2021.3059096

[15] Cui, Y., Cao, K., Cao, G., Qiu, M., & Wei, T. (2021). Client scheduling and resource management for efficient training in heterogeneous IoT-edge federated learning. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 41(8), 2407-2420. DOI: 10.1109/TCAD.2021.3110743

[16] Du, J., Jiang, C., Benslimane, A., Guo, S., & Ren, Y. (2022). SDN-based resource allocation in edge and cloud computing systems: An evolutionary Stackelberg differential game approach. IEEE/ACM Transactions on Networking, 30(4), 1613-1628.DOI: 10.1109/TNET.2022.3152150

[17] Do-Duy, T., Nguyen, L. D., Duong, T. Q., Khosravirad, S. R., & Claussen, H. (2021). Joint optimisation of real-time deployment and resource allocation for UAV-aided disaster emergency communications. IEEE Journal on Selected Areas in Communications, 39(11), 3411-3424. DOI: 10.1109/JSAC.2021.3088662

[18] Walia, G. K., Kumar, M., & Gill, S. S. (2023). AI-empowered fog/edge resource management for IoT applications: A comprehensive review, research challenges, and future perspectives. IEEE Communications Surveys & Tutorials, 26(1), 619-669. DOI: 10.1109/COMST.2023.3338015

[19] Mohajer, A., Daliri, M. S., Mirzaei, A., Ziaeddini, A., Nabipour, M., & Bavaghar, M. (2022). Heterogeneous computational resource allocation for NOMA: Toward green mobile edge-computing systems. IEEE Transactions on Services Computing, 16(2), 1225-1238. DOI: 10.1109/TSC.2022.3186099