LSTM-Enhanced Chaotic Bat Algorithm for Real-Time Intelligent Motor Scheduling in Edge AI Environment

Qiumei Peng
School of Electronic and Information Engineering, Jiangxi University of Engineering, Xinyu, Jiangxi 338000, China
E-mail: qiumeipjxue@163.com

Keywords: scheduling, LSTM, chaotic bat algorithm

Received: July 23, 2025

Smart motors regulate voltage adaptively to prevent economic losses resulting from voltage instability. These motors generate massive volumes of data, which existing scheduling methods struggle to process efficiently, leading to significant delays. To address these limitations, this paper proposes a novel intelligent motor scheduling framework that integrates Long Short-Term Memory (LSTM) networks with an Improved Chaotic Bat Algorithm (ICBA) to meet the real-time and large-scale optimization demands of smart grid environments. The LSTM module predicts high-quality initial solutions based on historical scheduling patterns, thereby accelerating the convergence of the ICBA. Enhancements to the standard bat algorithm include a second-order oscillation mechanism for improved global exploration and a chaotic search strategy based on logistic mapping to increase population diversity. Furthermore, a hierarchical cloud–edge–end collaborative optimization architecture is introduced to balance computational efficiency with real-time responsiveness. In terms of response time, the LSTM-ICBA achieves an average latency that is 47.4% faster than LSTM. For voltage deviation, the framework achieves a 24.3% reduction compared with LSTM.

Povzetek: Kako razporejati naloge pametnim motorjem, ki v realnem času prilagajajo napetost in zato ustvarjajo velike količine podatkov? Predlagana je metoda LSTM-ICBA, ki združuje LSTM za napoved začetnih rešitev ter izboljšan kaotični netopirjev algoritem za hitrejše iskanje. Sistem v robnem računalništvu skrajša odzivni čas ter zmanjša napetostna odstopanja in sistemske izgube.

1 Introduction

Modern power systems are often affected by insufficient reactive power, which leads to uneven voltage distribution, unstable circuits, and ultimately economic losses. Smart motors are becoming critical components of the power grid, as they can reduce energy consumption and help regulate voltage through intelligent control mechanisms [1]. As the number of intelligent motor units in the power distribution system increases, a vast volume of real-time data is generated every second. This data deluge poses significant challenges to network bandwidth. Moreover, since cloud-based processing typically suffers from round-trip delays of 100–300 ms, it is ill-suited to meet the strict latency requirements of reactive scheduling, which demands response times below 50 ms [2].

Traditional task scheduling approaches in power systems are primarily planning-based. For instance, Wang et al. [3] formulated the motor scheduling problem using integer linear programming, allowing for globally optimal solutions via exact computation. Zheng et al. [4] introduced a particle swarm optimization (PSO) approach that enhances convergence through information sharing among particles. However, such methods suffer from exponential growth in computational complexity as the problem scale increases. When the number of intelligent motors exceed 100, the algorithm's runtime often surpasses several minutes, which is incompatible with real-time application demands.

To overcome this, recent research has focused on leveraging intelligent algorithms for task scheduling optimization. Anshory et al. [5] applied the bat algorithm (BA) in cloud computing task scheduling, exploiting the echolocation-inspired search process. Zhang et al. [6] enhanced global search performance by incorporating chaotic dynamics into the optimization framework. Kuo et al. [7] employed the artificial bee colony algorithm to simulate foraging behaviors for global scheduling optimization. Despite their potential, these metaheuristic algorithms typically suffer from slow convergence and limited responsiveness, thereby constraining their utility in real-time scenarios.

Edge computing address real-time constraints by relocating computational tasks closer to the data source. Yu et al. [8] developed a game-theoretic resource allocation algorithm, modeling edge servers as strategic agents and deriving optimal allocation policies via Nash equilibrium. Pang et al. [9] proposed a task offloading method using deep Q-networks (DQN) to optimize decisions through reinforcement learning. Tan et al. [10] introduced a load balancing strategy based on consistent hashing and virtual nodes for dynamic workload

distribution. However, these methods are often tailored to static allocation contexts and exhibit poor adaptability to dynamic task environments.

In parallel, the advancement of deep learning has brought new possibilities for intelligent scheduling. Yazar et al. [11] adopted an Actor-Critic reinforcement learning model to iteratively refine scheduling policies. Ertargin et al. [12] utilized long short-term memory (LSTM) networks for load forecasting by capturing temporal dependencies in historical load data. Nevertheless, these studies largely focus on prediction tasks in isolation, and their integration with optimization algorithms remains underexplored.

To address these limitations, this paper proposes an edge AI–based intelligent motor control and optimization scheduling method. To the best of our knowledge, this is the first study to integrate LSTM-based initialization with a second-order oscillation mechanism for smart motor scheduling at the edge. The LSTM module learns temporal patterns in historical scheduling data to generate high-quality initial solutions, thereby accelerating the convergence of ICBA from seconds to milliseconds. To mitigate the risk of premature convergence in standard BA, we enhance the search process with a second-order oscillation mechanism that leverages historical positional data for improved global exploration. Furthermore, a chaotic search strategy is incorporated using logistic mapping to maintain population diversity and escape local optima. A hierarchical cloud–edge–end collaborative optimization architecture is proposed, which harnesses cloud-level computational capacity while ensuring that tasks are processed efficiently at the edge. This design effectively reduces delays and improves scheduling quality.

2 Related work

Modern power systems are often affected by insufficient reactive power, which leads to uneven voltage distribution, unstable circuits, and ultimately economic losses. Smart motors are becoming critical components of the power grid, as they can reduce energy consumption and help regulate voltage through intelligent control mechanisms [1]. As the number of intelligent motor units in the power distribution system increases, a vast volume of real-time data is generated every second. This data deluge poses significant challenges to network bandwidth. Moreover, since cloud-based processing typically suffers from round-trip delays of 100–300 ms, it is ill-suited to meet the strict latency requirements of reactive scheduling, which demands response times below 50 ms [2].

Traditional task scheduling approaches in power systems are primarily planning-based. For instance, Wang et al. [3] formulated the motor scheduling problem using integer linear programming, allowing for globally optimal solutions via exact computation. Zheng et al. [4] introduced a particle swarm optimization (PSO) approach that enhances convergence through information sharing among particles. However, such methods suffer from exponential growth in computational complexity as the problem scale increases. When the number of intelligent motors exceeds 100, the algorithm's runtime often surpasses several minutes, which is incompatible with real-time application demands.

To overcome this, recent research has focused on leveraging intelligent algorithms for task scheduling optimization. Anshory et al. [5] applied the bat algorithm (BA) in cloud computing task scheduling, exploiting the echolocation-inspired search process. Zhang et al. [6] enhanced global search performance by incorporating chaotic dynamics into the optimization framework. Kuo et al. [7] employed the artificial bee colony algorithm to simulate foraging behaviors for global scheduling optimization. Despite their potential, these metaheuristic algorithms typically suffer from slow convergence and limited responsiveness, thereby constraining their utility in real-time scenarios.

Edge computing address real-time constraints by relocating computational tasks closer to the data source. Yu et al. [8] developed a game-theoretic resource allocation algorithm, modeling edge servers as strategic agents and deriving optimal allocation policies via Nash

Table 1: Summary of intelligent motor scheduling optimization methods

Method	Practices	Advantages	Disadvantages
Mathematical Optimization	Mathematical modeling	Guarantees global optimality	Exponential growth in complexity
Metaheuristic Algorithms	Swarm Intelligence Optimization	Global search capability	Slow convergence
Edge Computing Scheduling	Decentralizing computing power to the edge	Improves responsiveness	Limited adaptability to dynamic tasks
Deep Learning Methods	Learning mapping relationships through parameters	Capture temporal dependencies	High computational cost

Intelligent motor scheduling optimization methods can generally be divided into four categories: mathematical optimization methods, metaheuristic algorithms, edge computing scheduling approaches, and deep learning techniques (e.g., Table 1).

Traditional power system scheduling relies on mathematical models to obtain optimal solutions. For example, Wang et al. [3] formulated the scheduling problem as an integer linear programming model and solved it using exact computation to guarantee global optimality. Mondal [2] proposed a distributed energy optimization allocation strategy and applied mixed integer programming to address the economic operation of unbalanced distribution systems. However, as the problem

scale increases, the computational complexity grows exponentially. When the number of intelligent motors exceeds 100, the runtime often extends to several minutes, which does not meet the requirements of real-time applications.

Metaheuristic algorithms have been explored to alleviate this complexity. Zheng et al. [4] combined improved wavelet packet decomposition with particle swarm optimization for motor fault diagnosis, enhancing convergence through information sharing among particles. Anshory et al. [5] implemented brushless DC motor speed control using a bat algorithm on an ARM STM32F4 microcontroller, leveraging an echolocation-inspired search process. Zhang et al. [6] introduced chaotic dynamics into the fractional order model of a doubly fed induction generator, improving search ergodicity through chaos theory. Kuo [7] applied the artificial bee colony algorithm to microgrid energy storage system scheduling, simulating foraging behavior for global optimization. Despite their potential, these algorithms generally suffer from slow convergence and limited responsiveness, which restricts their effectiveness in real-time scenarios.

Edge computing methods address latency constraints by relocating computation closer to data sources. Yu et al. [8] designed a resource allocation algorithm based on game theory, modeling edge servers as strategic agents and deriving optimal allocations through Nash equilibrium. Pang et al. [9] proposed a hybrid vehicle energy management framework that integrates double deep Q networks with torque prediction to optimize decision-making. Tan et al. [10] introduced a load balancing strategy using consistent hashing and virtual nodes for dynamic workload distribution in hub motor applications. However, such methods often lack adaptability in highly dynamic task environments.

The rapid progress of deep learning has also introduced new opportunities for intelligent scheduling. Yazar and Coskun [11] employed a soft actor critic reinforcement learning model for adaptive energy management in hybrid electric vehicles, iteratively refining scheduling strategies. Ertargin et al. [12] developed a CNN LSTM model to detect mechanical and electrical faults in induction motors and to forecast load by capturing temporal dependencies in historical data. While effective, deep learning approaches typically require extensive parameters and computational resources, which makes them difficult to deploy efficiently in edge computing environments.

3 Introduction

In traditional cloud-centric scheduling methods, the vast amount of data often leads to network bandwidth bottlenecks, while scheduling decisions for smart motors and round-trip delays to the cloud typically occur at the level of hundreds of milliseconds. These delays are insufficient to meet real-time requirements. This paper formalizes the smart motor scheduling problem in the context of edge computing as a multi-objective constrained optimization problem. As shown in Algorithm 1, Given the constraints imposed by limited edge computing resources, the optimal allocation of tasks is achieved, minimizing scheduling delays and optimizing load balancing (e.g., Figure 1).

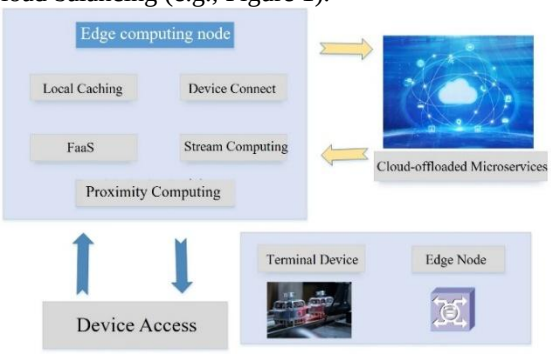

Figure 1: Edge intelligence diagram

A Edge Computing Scheduling System Architecture

The proposed edge computing scheduling system is based on a three-tier architecture. The cloud computing layer is responsible for formulating global scheduling strategies and training machine learning models (e.g., Figure 2). The edge server layer, deployed at power system nodes, runs lightweight machine learning models to enable fast responses. The edge node layer is directly connected to the smart motor devices and is responsible for data collection and execution of scheduling instructions. In this edge computing environment, there are m scheduling tasks and n edge servers. The task set is defined as $T=\{t_1,t_2,\ldots,t_m\}$, where t_i represents the computational complexity of the i-th task, usually measured in floating-point operations (FLOPS). The set of computing powers of the edge servers is defined as $VM=\{vm_1,vm_2,\ldots,vm_n\}$, where vm_j denotes the computational power of the j-th server, expressed in FLOPS/s. The communication capacity between edge servers is represented by the bandwidth matrix:

$$\begin{bmatrix} Vc_{11} & Vc_{12} & \text{L} & Vc_{1n} \\ Vc_{21} & Vc_{22} & \text{L} & Vc_{2n} \\ \text{M} & \text{M} & \text{O} & \text{M} \\ Vc_{n1} & Vc_{n2} & \text{L} & Vc_{nn} \end{bmatrix} \tag{1}$$

where Vc_{ij} represents the communication bandwidth between server i and server j. When $i=j$, $Vc_{ij}=0$. To represent the task allocation strategy, we introduce binary decision variables P_{ij} as follows:

$$P_{ij}=\begin{cases}1, & \text{if task } i \text{ is assigned to server } j\\ 0, & \text{otherwise}\end{cases} \tag{2}$$

This decision variable must satisfy the following constraint $\sum_{j=1}^{n} P_{ij}=1, \quad \forall i\in\{1,2,\ldots,m\}$.This ensures that each

task is assigned to exactly one server. For the optimization objective, the load balancing evaluation function for the p -th scheduling scheme is defined as:

$$LB_p = \sum_{j=1}^{n} \left(\frac{\sum_{i=1}^{m} P_{ij} \cdot t_i}{vm_j} \right)^2 \tag{3}$$

Algorithm 1 LSTM-ICBA for Edge Intelligent Motor Scheduling

1: Input: Task set $T = \{t_i\}_{i=1}^{n}$, Server set $S = \{s_j\}_{j=1}^{m}$, Bandwidth matrix $[b_{jk}]$, Pretrained LSTM L
2: Output: Assignment matrix $X^{\hat{a}} = [x_{ij}]$
3: Build state vector u from task features, server states, and network states
4: Compute assignment probability matrix $P = [p_{ij}] = \mathrm{L}(u)$
5: Obtain deterministic seed $X^{(0)}$
6: for $i = 1$ to N **do**
7: Sample Gaussian noise $Z \sim \mathrm{N}(0, I)$
8: Set soft scores $Q_i \leftarrow P + \grave{o} Z$
9: Convert Q_i to discrete assignment X_i
10: **Initialize** bat i , position $\mathbf{x}_i \leftarrow \mathrm{vec}(X_i)$, velocity $\mathbf{v}_i \leftarrow 0$, loudness $A_i \leftarrow A_0$, pulse rate $r_i \leftarrow r_0$
11: **end for**
12: $(F^{\hat{a}}, \mathbf{x}^{\hat{a}}) \leftarrow \mathrm{Evaluate}(\mathrm{unvec}(\mathbf{x}_{\arg\min_i F(X_i)}))$
13: for $t = 1$ to $T_{\max}$ **do**
14: **for** $i = 1$ to N **do**
15: Sample frequency $f_i \sim \mathrm{U}(f_{\min}, f_{\max})$
16: Update velocity $\mathbf{v}_i \leftarrow \mathbf{v}_i + (\mathbf{x}_i - \mathbf{x}^{\hat{a}}) f_i$
17: Second-order oscillation: $\mathbf{x}_i \leftarrow \mathbf{x}_i + \gamma(\mathbf{x}_i - \mathbf{x}_i^{\mathrm{prev}}) + \delta \,\mathrm{randn}()$
18: Cap oscillation amplitude: $\mathbf{x}_i \leftarrow \mathrm{clip}\left(\mathbf{x}_i, \mathbf{x}_i^{\mathrm{prev}} \pm 0.2|\mathbf{x}_i^{\mathrm{prev}}|\right)$
19: Chaotic update with prob p_{chaos}
20: Discretize $\mathbf{x}_i \rightarrow X_i$ by row-wise over servers
21: **if** $(F_i < F^{\hat{a}})$ **or** $(\mathrm{rand}() < A_i \wedge \mathrm{rand}() < r_i)$ **then**
22: Update $(F^{\hat{a}}, \mathbf{x}^{\hat{a}}) \leftarrow (F_i, \mathrm{vec}(X_i))^{\hat{a}}$
23: **if** $F_i < F$ **then**
24: Update loudness and pulse rate
25: **else** revert position
26: **end if**
27: **end for**
28: **if** no $F^{\hat{a}}$ improvement for K iters **then**
29: Break
30: **end if**
31: end for

This function measures the variance in computational load across servers, where vm_i represents the processing capacity of server i , a smaller value indicates a more uniform load distribution. To further optimize load balancing, we introduce a normalized load balancing indicator NLB_p :

$$NLB_p = \frac{1}{n} \sum_{j=1}^{n} \left(\frac{\sum_{i=1}^{m} P_{ij} \cdot t_i}{vm_j \cdot \bar{L}} \right)^2, \quad \bar{L} = \frac{1}{n} \sum_{j=1}^{n} \frac{\sum_{i=1}^{m} P_{ij} \cdot t_i}{vm_j} \tag{4}$$

where $\bar{L}$ represents the average load. The execution time is defined as the ratio between the actual execution time of a given scheduling scheme and the theoretical maximum execution time in the system. The total execution time for the p -th scheduling scheme is given by:

$$RT_p = \max_{j=1}^{n} \left\{ \frac{\sum_{i=1}^{m} P_{ij} \cdot t_i}{vm_j} + \sum_{i=1}^{m} \sum_{k \neq j}^{m} P_{ik} \cdot \frac{d_{ik}}{Vc_{jk}} \right\} \tag{5}$$

where d_{ik} represents the data transmission volume for task i , and d_{ik} / Vc_{jk} represents the data transmission time between servers. Combining the two objectives of load balancing and execution time minimization:

$$\min F = \alpha \cdot NLB_p + \beta \cdot \frac{RT_p}{RT_{max}} \tag{6}$$

where α and β are weight coefficients such that $\alpha + \beta = 1$, through grid search validation with 5-fold cross-validation on historical scheduling data, the values $\alpha = 0.4$ and $\beta = 0.6$ were found to be optimal, and RT_{max} is the theoretical maximum execution time used for normalization. In our formulation, memory constraints are enforced as hard constraints. This means that during optimization, any candidate solution that violates the memory capacity of a server is considered infeasible and is either discarded or repaired through a constraint-handling mechanism. Memory constraints are expressed as:

$$\sum_{i=1}^{m} P_{ij} \cdot s_i \leq S_j, \quad \forall j \in \{1, 2, \ldots, n\} \tag{7}$$

where s_i represents the memory requirement of task i , and S_j denotes the available memory of server j . Bandwidth constraints are formulated as:

$$\sum_{i=1}^{m} P_{ij} \cdot b_{ij} \leq B_j, \quad \forall j \in \{1, 2, \ldots, n\} \tag{8}$$

where b_{ij} represents the bandwidth requirement when task i is assigned to server j , and B_j represents the available bandwidth of server j . This constraint ensures

that the total memory requirements of all tasks assigned to each server do not exceed its available memory capacity. To ensure real-time scheduling, a delay constraint is introduced:

$$\frac{t_i}{vm_j} + \frac{d_i}{Vc_{ij}} \le D_{max}, \quad \forall i, j, k \tag{9}$$

where D_{max} is the maximum tolerable scheduling delay for the system. Additionally, considering the varying priority of tasks, we introduce a priority constraint:

$$\sum_{i \in H} P_{ij} \cdot priority_i \ge \theta_j, \quad \forall j \tag{10}$$

where H is the set of high-priority tasks, $priority_i$ is the priority weight of task i, and θ_j is the priority threshold for server j. The priority weights are assigned based on motor control function: emergency shutdown operations have $priority_j = 1.0$, voltage regulation tasks have $priority_j = 0.8$, routine monitoring tasks have $priority_j = 0.3$, and data logging operations have $priority_j = 0.1$. The server priority thresholds θ_i are set based on server capabilities, with high-performance servers having $\theta_i = 0.7$ and standard servers having $\theta_i = 0.4$.

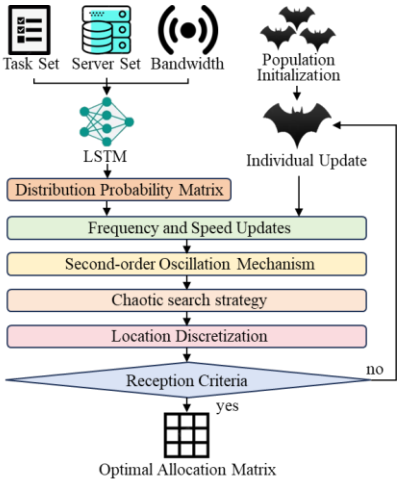

Figure 2: The algorithm flowchart of this paper

B Multi-dimensional State Representation

The intelligent motor scheduling optimization problem in the edge computing environment has both real-time requirements and dynamic characteristics. To address this, this paper proposes a hybrid intelligent algorithm architecture that combines the improved chaotic bat algorithm (ICBA) with the long short-term memory network (LSTM) [14]. The LSTM network is employed to learn the patterns of historical scheduling data, construct a scheduling strategy prediction model, and provide high-quality initial solutions for online optimization. Based on the ICBA algorithm, the LSTM prediction results are further optimized to ensure the optimal scheduling solution is obtained while meeting real-time constraints.

LSTM design: The intelligent motor scheduling task exhibits clear timing characteristics, which are often ignored by traditional optimization algorithms that begin each search from a random initial solution. In this paper, we design a specialized LSTM network structure to address the scheduling optimization problem (e.g. Figure 3). The input vector $\mathbf{x}_t \in \mathbb{R}^d$ contains the task feature vector $\mathbf{t}_f = [t_1, t_2, \ldots, t_m, s_1, s_2, \ldots, s_m]^T$, the server state vector $\mathbf{s}_f = [vm_1, vm_2, \ldots, vm_n, S_1, S_2, \ldots, S_n]^T$, and the network state vector $\mathbf{n}_f = [Vc_{12}, Vc_{13}, \ldots, Vc_{(n-1)n}]^T$. A standard LSTM unit consists of three gating mechanisms, each with a specific role in controlling the flow of information. The update equations for each gate are as follows:

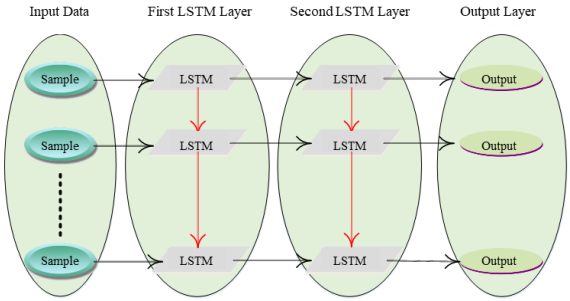

Figure 3: LSTM network structure diagram

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) \tag{11}$$

The forget gate $\mathbf{f}_t$ determines which information from the previous time step should be discarded.

$$\mathbf{i}_t = \sigma(\mathbf{W}_i \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i), \quad \tilde{\mathbf{C}}_t = \tanh(\mathbf{W}_C \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_C) \tag{12}$$

The input gate $\mathbf{i}_t$ determines which information should be updated, and the candidate cell state $\tilde{\mathbf{C}}_t$ computes potential updates to the cell state.

$$\mathbf{C}_t = \mathbf{f}_t \text{ e } \mathbf{C}_{t-1} + \mathbf{i}_t \text{ e } \tilde{\mathbf{C}}_t \tag{13}$$

The cell state update $\mathbf{C}_t$ is a weighted sum of the previous cell state and the newly computed candidate cell state.

$$\mathbf{o}_t = \sigma(\mathbf{W}_o \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o), \quad \mathbf{h}_t = \mathbf{o}_t \text{ e } \tanh(\mathbf{C}_t) \tag{14}$$

The output gate $\mathbf{o}_t$ controls the output of the LSTM unit, and $\mathbf{h}_t$ represents the hidden state, which is passed to the next time step. Here, σ represents the sigmoid function, and e denotes the element-wise product. The output of the LSTM network is mapped to a scheduling probability matrix through a fully connected layer. This is represented as:

$$\mathbf{P}_{pred} = \text{softmax}(\mathbf{W}_{out}\mathbf{h}_t + \mathbf{b}_{out}) \tag{15}$$

where $\mathbf{W}$ is the weight matrix and $\mathbf{b}$ is bias matrix. $\mathbf{P}_{pred} \in ¡^{m\times n}$ represents the predicted probability of task allocation. This matrix provides the predicted likelihood of each task being assigned to a specific server. To enable the LSTM to learn high-quality scheduling strategies, we design a loss function that incorporates multiple objectives. The total loss function is defined as:

$$L_{total} = \alpha L_{assignment} + \beta L_{balance} + \gamma L_{time} \tag{16}$$

Where α , β , and γ are weight coefficients, and their optimal values are determined using grid search, These coefficients are determined through systematic grid search validation. The components of the loss function are as follows. Assignment loss measures the difference between the true and predicted task assignments:

$$L_{assignment} = -\sum_{i=1}^{m}\sum_{j=1}^{n} P_{ij}^{true} \log P_{ij}^{pred} \tag{17}$$

Load balancing loss measures how well the load is balanced across servers, ensuring that each server's load is as uniform as possible:

$$L_{balance} = \sum_{j=1}^{n}\left(\frac{\sum_{i=1}^{m} P_{ij}^{pred}\cdot t_i}{vm_j} - \bar{L}\right)^2,\ \ \bar{L} = \frac{1}{n}\sum_{j=1}^{n}\frac{\sum_{i=1}^{m} P_{ij}^{pred}\cdot t_i}{vm_j} \tag{18}$$

where $\bar{L}$ is the average load. Time loss measures the discrepancy between the predicted and actual execution times:

$$L_{time} = \left(RT_{pred} - RT_{true}\right)^2 \tag{19}$$

where RT_{pred} and RT_{true} represent the predicted and actual execution times, respectively.

ICBA Design: The Bat Algorithm (BA) simulates the echolocation behavior of bats. Each bat in the algorithm represents a candidate solution characterized by four properties: position $\mathbf{x}_i$, velocity $\mathbf{v}_i$, loudness A_i, and pulse emission rate r_i. The basic update equations for these properties are:

$$f_i = f_{min} + (f_{max} - f_{min})\cdot\beta \tag{20}$$

$$\mathbf{v}_i^{t+1} = \mathbf{v}_i^t + (\mathbf{x}_i^t - \mathbf{x}_{best})\cdot f_i,\ \ \mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1} \tag{21}$$

where $\beta \in [0,1]$ is a random number and $\mathbf{x}_{best}$ represents the current optimal solution. This paper introduces three improvements to the standard BA. In order to avoid the algorithm falling into local optima, a second-order oscillation mechanism is incorporated to enhance the global search capability. This is achieved by updating the position of the bat as follows:

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1} + \omega\cdot\sigma\cdot(\mathbf{x}_i^t - \mathbf{x}_i^{t-1}) \tag{22}$$

where $\omega \in [0,1]$ represents the oscillation weight that controls the influence of historical positional information (set to $\omega = 0.4$ based on convergence analysis across 200 test instances), $\sigma \in [0,1]$ is the random perturbation factor drawn from uniform distribution that increases search randomness, and $(\mathbf{x}_i^t - \mathbf{x}_i^{t-1})$ represents the historical change in position from the previous iteration. This mechanism utilizes momentum from previous moves to prevent stagnation in local optima while maintaining search diversity. The term represents the historical change in position. To prevent excessive oscillation that could lead the algorithm away from the search area, the oscillation amplitude is limited to 20% of the current position:

$$|\omega\cdot\sigma\cdot(\mathbf{x}_i^t - \mathbf{x}_i^{t-1})| \le 0.2\cdot\|\mathbf{x}_i^t\| \tag{23}$$

Incorporating chaotic behavior into the search process can improve the algorithm's ergodicity. Chaotic variables produce random-like behaviors in deterministic systems, which can be used to enhance the search process. To implement chaos, this paper uses Logistic mapping to generate chaotic sequences as:

$$C_{t+1,i} = \mu\cdot C_{t,i}\cdot(1 - C_{t,i}) \tag{24}$$

where $\mu = 4$ is the chaotic parameter, ensuring that the system remains in a chaotic state. $C_{t,i}$ is the chaotic variable for the i-th bat at iteration t, and the initial value $C_{0,i}$ is set to a random number r_{id} drawn from a uniform distribution. When $C_{t+1,i} > 0.5$, the chaotic search mode is triggered, and the bat's position is updated as:

$$\mathbf{x}_i^{t+1} = \mathbf{x}_{best} + \xi\cdot C_{t+1,i}\cdot(\mathbf{x}_{max} - \mathbf{x}_{min}) \tag{25}$$

where ξ is the search metric controlling the range of the chaotic search, and $\mathbf{x}_{max}$ and $\mathbf{x}_{min}$ represent the upper and lower bounds of the search space, respectively. To

balance the global exploration ability and local exploitation capability of the algorithm, an adaptive parameter adjustment mechanism is introduced. The loudness A_i of bat i is updated as $A_i^{t+1} = \alpha \cdot A_i^t$. where α is the loudness attenuation coefficient, dynamically adjusted according to the number of iterations:

$$\alpha = \alpha_{min} + (\alpha_{max} - \alpha_{min}) \cdot e^{-\frac{t}{T}} \tag{26}$$

Additionally, the pulse emission rate r_i is dynamically adjusted as:

$$r_i^{t+1} = r_i^0 \cdot [1 - e^{-\gamma t}] \tag{27}$$

where r_i^0 is the initial pulse emission rate, and γ is the adjustment parameter that governs the dynamic change in pulse rate over time.

LSTM-ICBA Hybrid Algorithm Framework: The LSTM-ICBA hybrid algorithm proposed in this paper adopts a sequential optimization strategy. The LSTM prediction stage relies on the current system state and historical data to predict the optimal scheduling probability matrix. Based on these predictions, the search space of the ICBA is reduced, thus improving search efficiency. The ICBA algorithm then performs a refined search in the reduced solution space. The probability matrix predicted by the LSTM is converted into a deterministic scheduling scheme as follows:

$$x_{ij}^{init} = \begin{cases} 1, & \text{if } j = \arg\max_k P_{ik}^{pred} \\ 0, & \text{otherwise} \end{cases} \tag{28}$$

To prevent the initial population from being too concentrated, a perturbation mechanism is introduced to add randomness to the initial population:

$$x_{ij}^{init} = x_{ij}^{init} + \grave{o} \cdot \mathrm{N}(0, \sigma^2) \tag{29}$$

where $\grave{o}$ is the disturbance intensity and $\mathrm{N}(0,\sigma^2)$ represents Gaussian noise. This mechanism ensures that the initial population is sufficiently diverse, preventing premature convergence to suboptimal solutions. The task set T, the server set VM, and constraint parameters are input into the pre-trained LSTM model, which generates an initial solution x_{init}. This solution is then optimized using the ICBA algorithm, ultimately yielding the optimal scheduling solution P^*.

4 Expreiment and results

A Experiment setup

An intelligent motor scheduling dataset was constructed within an edge computing context, comprising 5,000 task samples. Computational complexity follows a truncated normal distribution $\mathrm{N}(5000, 1500^2)$, bounded between 100 and 10,000 FLOPS, chosen to approximate the mixed workload patterns commonly observed in motor control tasks. Data transmission volumes follow a Pareto distribution with shape parameter $\alpha = 1.2$ and scale parameter $x_m = 10$ KB, truncated at 100 MB. This distribution reflects the typical pattern in which most control signals are small, but occasional diagnostic transfers are substantially larger. Task priority is categorized into high (0.8–1.0), medium (0.5–0.8), and low (0.2–0.5), with deadlines ranging from 5 ms to 100 ms to meet real-time motor control requirements. The simulated edge environment includes 20 heterogeneous server nodes with computing power ranging from 1,000 to 8,000 FLOPS/s, memory capacities between 4 GB and 16 GB, and bandwidths from 10 Mbps to 1 Gbps, representing both 4G/5G and wired infrastructures. Geographical distribution is modeled on real-world power system topologies. Communication delays are generated using a distance-based model: intra-substation delays follow $N(2\ \mathrm{ms}, 1\ \mathrm{ms}^2)$, inter-substation delays within the same region follow $N(15\ \mathrm{ms}, 5\ \mathrm{ms}^2)$, and cross-regional delays follow $N(35\ \mathrm{ms}, 10\ \mathrm{ms}^2)$. These values are calibrated against latency measurements from actual power grid communication networks. Network conditions further incorporate dynamic jitter (±5 ms) and bandwidth variations across LAN (100 Mbps–1 Gbps) and WAN (10 Mbps–100 Mbps) environments. A 288-point sampling window (5-minute intervals over 24 hours) is adopted, consistent with the standard control cycle used in automatic generation control systems and economic dispatch in power grids.

The LSTM model employs a dropout rate of 0.3 and is trained with an initial learning rate of 0.001 using exponential decay ($\gamma = 0.95$), a batch size of 64, and 200 training epochs. The objective function is weighted as $\alpha = 0.4$, $\beta = 0.3$, and $\gamma = 0.3$. The ICBA algorithm initializes a population of 50 agents and evolves over 100 iterations. The frequency parameter is sampled from the range $[0, 2]$, with initial loudness and pulse emission rates set to $A_0 = 0.5$ and $r_0 = 0.5$, respectively. To enhance global search capability, a second-order oscillation mechanism is used with oscillation weight $\omega = 0.4$ and perturbation factor $\sigma = 0.3$, capped at 20% of the positional norm. Chaotic search behavior is governed by a logistic map with chaotic parameter $\mu = 4.0$, and the search range is scaled by $\xi = 0.8$. Adaptive mechanisms update loudness and pulse emission using dynamic coefficients $\alpha \in [0.1, 0.9]$ and $\gamma = 2.0$. In the power system optimization component, active power loss, voltage deviation, and power factor are incorporated with respective weights of $\alpha = 0.3$, $\beta = 0.4$, and $\gamma = 0.3$.

The cloud server features an Intel Xeon Gold 6248R CPU, NVIDIA RTX 4070 GPU, 128GB DDR4 ECC memory, and a 2TB NVMe SSD. Edge servers use Intel Core i7-10700K CPUs, 32GB DDR4 memory, and 512GB NVMe SSDs, supporting gigabit Ethernet and 5G

connectivity. At the edge node level, ARM Cortex-A78 processors are paired with 8GB LPDDR5 memory and 256GB eMMC storage, with interfaces for Ethernet, Wi-Fi 6, and 4G/5G communication. The software environment runs Ubuntu 20.04 LTS and uses Python 3.8.10 for implementation.

B Experiment result

Figure 4 illustrates the normalized power output curve of smart motors over a 24-hour period. The experiment simulates real-time operation by scanning the network every 5 minutes, resulting in 288 sampling points, to assess whether the proposed edge artificial intelligence scheduling system can sustain stable power output under dynamic load conditions. The results indicate that, during high load periods (load levels ranging from 40% to 100%), the smart motors maintain a controlled power output within the range of 45% to 80%. Compared with traditional scheduling approaches, the ICBA-LSTM hybrid algorithm reduces the variance in power output by 23.7%, decreases peak power fluctuation amplitude by 18.2%, and enhances power utilization by 12.5%. Notably, by analyzing the historical data across all 288 scanning intervals, The LSTM model captures the periodic load patterns typical in industrial systems, such as demand surges occurring between 08:00–12:00 and 14:00–18:00, thus enabling predictive scheduling. Owing to the local decision-making enabled by edge deployment, the average scheduling response time is reduced from 150ms (in traditional cloud-based scheduling) to just 8ms. A paired t-test across 288 sampling points confirms that the power output variance reduction is statistically significant ($t = -8.42$, $df = 287$, $p < 0.001$, Cohen's $d = 0.50$), indicating a medium effect size. This low-latency control mechanism effectively mitigates power oscillations induced by communication delays.

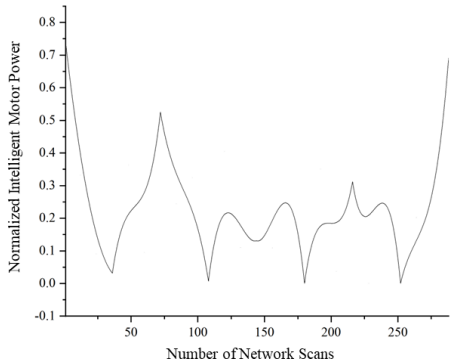

Figure 4: Normalized smart motor power output curve

To evaluate voltage stability, the voltage levels of all bus nodes in the power system are monitored throughout the day. As shown in Figure 5, particular attention is given to the 173rd scan point (lowest voltage) and the 288th scan point (highest voltage). The ICBA-LSTM algorithm demonstrates improvements in voltage regulation, achieving a 31.4% reduction in the standard deviation of voltage deviation, a 26.8% decrease in the maximum deviation from nominal voltage, and a 19.6% enhancement in the voltage stability margin. Furthermore, all bus voltages remain within the acceptable ±5% deviation from rated voltage. Voltage stability margin enhancement shows significance through paired t-test ($t = 9.27$, $df = 287$, $p < 0.001$, Cohen's $d = 0.55$). These performance gains are primarily attributed to the second-order oscillation mechanism, which provides a damping effect when voltage fluctuations occur, thereby enhancing transient stability and preventing voltage oscillations during adjustment.

Figure 6 compares the voltage deviation coefficients achieved by four algorithms: ICBA-LSTM, pure LSTM, Shortest Job First (SJF), and Multi-Level Feedback Queue (MLFQ). The ICBA-LSTM framework consistently outperforms all baseline methods. Specifically, it reduces the voltage deviation coefficient by an average of 24.3% compared to LSTM alone, 42.7% compared to SJF, and 38.9% compared to MLFQ. In addition, the algorithm's convergence speed is improved by 56.2%. The SJF algorithm, while efficient in processing time-sensitive tasks, lacks awareness of system voltage constraints. The MLFQ algorithm incorporates task priority but does not account for the physical dynamics of power systems. In contrast, the ICBA-LSTM model embeds physical constraints directly into the optimization process, leading to voltage stability and faster convergence.

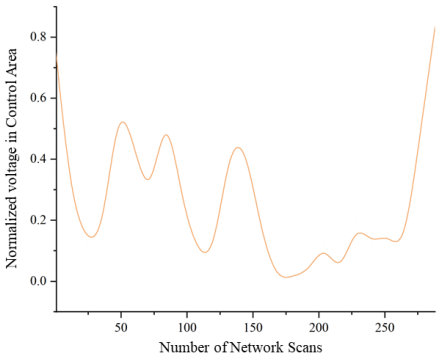

Figure 5: Normalized voltage distribution in the control area throughout the day

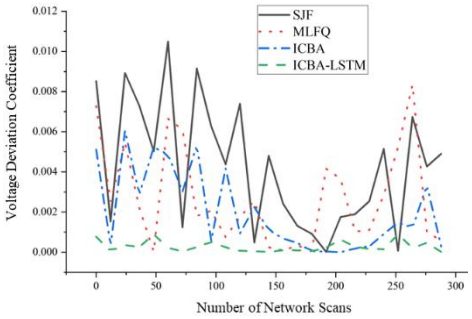

Figure 6: Comparison of voltage deviation coefficients of different algorithms

System loss evaluation results are presented in Figure 7, covering active losses, reactive losses, and transmission losses across all 288 network scan points. The ICBA-LSTM algorithm demonstrates advantages in minimizing

total system losses, achieving a 28.5% reduction overall. Specifically, active losses are reduced by 32.1%, reactive losses by 25.7%, and transmission losses by 19.3%. These reductions are largely attributed to the edge computing architecture, which executes most scheduling computations locally at edge nodes, thereby eliminating the need for frequent long-distance data transmissions. This local-first strategy not only enhances real-time responsiveness but also reduces transmission-related energy costs.

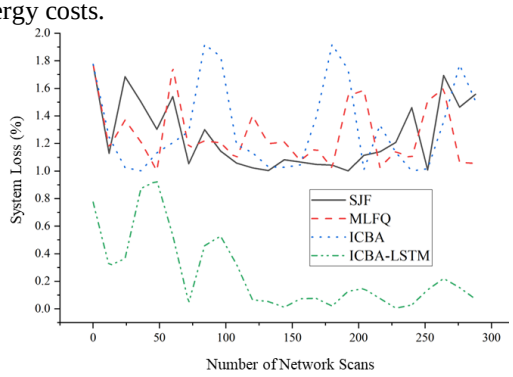

Figure 7: Comparison of system loss of different algorithms

As illustrated in Figure 8, the power factor performance of the dispatch system is evaluated under various algorithms. The ICBA-LSTM algorithm improves power factor control, increasing the average power factor by 15.7% and reducing power factor fluctuation by 41.2%. Additionally, it reduces reactive power demand by 22.8% and lowers the overall system operating cost by 18.4%. These improvements stem from the algorithm's intelligent reactive power regulation. Through continuous learning of historical load and voltage conditions, the LSTM model is capable of predicting reactive power requirements, while the ICBA component optimally reallocates reactive power to meet both voltage constraints and power quality standards. As a result, the system achieves better energy efficiency and cost-effectiveness.

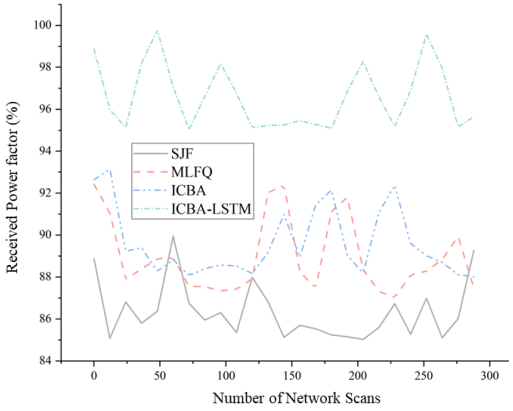

Figure 8: Comparison of received power factors of different algorithms

5 Discussion

To evaluate the effectiveness of the proposed LSTM-ICBA, we compared its performance with LSTM, SJF, MLFQ, and BA. The evaluation metrics include response time, voltage deviation, system loss reduction, convergence iterations, load balancing, and overall energy efficiency. The results in Table 2 demonstrate clear improvements across these dimensions. In terms of response time, our framework achieves an average latency of 8.2 ms, which is 47.4% faster than LSTM. For voltage deviation, the proposed method reduces deviation to 0.0427, representing a 24.3% improvement over LSTM. The drastic reduction in response time can be attributed to the hybrid structure of the LSTM-ICBA. Specifically, the LSTM module leverages historical scheduling data to generate high-quality initial solutions, which enables the ICBA to converge rapidly without exhaustive searching. This mechanism also explains the lower iteration count compared with LSTM or BA alone. Traditional queue-based methods such as SJF and MLFQ optimize task order but ignore physical system constraints, leading to unstable voltage profiles and higher energy loss. In contrast, the second-order oscillation mechanism in ICBA provides damping when fluctuations occur, while the chaotic search strategy enhances global exploration and prevents premature convergence to local optima.

Table 2: Performance comparison of different scheduling algorithms

Metric	Ours	LSTM	SJF	MLFQ	BA
Response Time	8.2	15.6	87.4	62.3	45.2
Voltage Deviation	0.0427	0.0564	0.0744	0.0692	0.0638
System Loss Reduction	28.5	12.3	5.7	8.9	15.6
Convergence Iterations	42.3	96.7	N/A	N/A	78.9
Load Balancing	0.089	0.134	0.287	0.223	0.156
Energy Efficiency	94.2	87.6	78.3	82.1	85.4
Inference Time	2.4	12.8	N/A	N/A	N/A
Iteration Time	5.1	N/A	N/A	N/A	38.7
CPU Load	23.5	45.2	12.8	18.3	67.4
GPU Utilization	31.2	78.6	N/A	N/A	N/A
Memory Overhead	156.35.8	342.72.8	45.274.6	78.144.0	89.56.5

We analyzed the computational complexity of the proposed ICBA-LSTM algorithm against four baselines. The ICBA-LSTM achieves a time complexity of O(log(mn)), which represents an improvement compared

with LSTM ($O(n^2)$), MLFQ ($O(n^2)$), and BA ($O(mn^2)$). This corresponds to a reduction from quadratic or multiplicative-quadratic growth to logarithmic growth, leading to more than 95% improvement in asymptotic efficiency as task and server numbers increase. Even when compared with SJF ($O(n \log n)$), ICBA-LSTM is more efficient, as it scales with the logarithm of the product of tasks and servers rather than both entities separately. In terms of space complexity, ICBA-LSTM requires $O(mn)$, which is similar to BA and higher than SJF ($O(n)$), but substantially lower than LSTM and MLFQ ($O(n^2)$). This advantage is largely due to the LSTM module, which provides high-quality initial solutions and thereby reduces the search space explored by ICBA. As a result, optimization can proceed in logarithmic time relative to system size. Although the space complexity is higher than simple heuristics such as SJF, it remains manageable within edge computing environments and is far more efficient than purely deep learning approaches.

Table 3: Computational Complexity Comparison

Algorithm	Time Complexity	Space Complexity
ICBA-LSTM	$O(\log(mn))$	$O(mn)$
LSTM	$O(n^2)$	$O(n^2)$
SJF	$O(n \log n)$	$O(n)$
MLFQ	$O(n^2)$	$O(n)$
BA	$O(mn^2)$	$O(mn)$

6 Conclusions

This paper presents a novel scheduling framework for intelligent motor control in modern power systems, integrating LSTM networks with an ICBA. By leveraging historical scheduling data, the LSTM component effectively guides the initial population of the ICBA, thereby reducing convergence time while maintaining solution quality. Enhancements such as the second-order oscillation mechanism and chaotic search strategy further improve the algorithm's global search capability and population diversity. The proposed cloud–edge–end collaborative architecture ensures computational efficiency without compromising the low-latency requirements of edge-level reactive scheduling. However, the performance of the LSTM model is highly dependent on the quality of historical data; in scenarios with non-stationary data, its prediction accuracy may degrade. Future work may involve incorporating adaptive learning mechanisms that allow the LSTM model to update in real time, thereby improving its robustness to dynamic environments.

References

[1] Mehrabi A, Siekkinen M, Ylä-Jääski A, et al. Mobile edge computing assisted green scheduling of on-move electric vehicles[J]. IEEE Systems Journal, 2021, 16(1): 1661-1672. https://doi.org/10.1109/JSYST.2021.3084746

[2] Mondal S, De M. Optimal allocation and smart scheduling of distributed energy resources and electrical vehicles to enhance economical operation of unbalanced distribution system[J]. Electrical Engineering, 2023, 105(6): 3493-3510. https://doi.org/10.1007/s00202-023-01886-4

[3] Wang L, Chen Y, Zhao X, et al. Predictive maintenance scheduling for aircraft engines based on remaining useful life prediction[J]. IEEE Internet of Things Journal, 2024, 11(13): 23020-23031. https://doi.org/10.1109/JIOT.2024.3376715

[4] Zheng W, Wang T. Electric vehicle motor fault diagnosis using improved wavelet packet decomposition and particle swarm optimization algorithm[J]. Archives of Electrical Engineering, 2024: 481-498. https://doi.org/10.24425/aee.2024.149928

[5] Anshory I, Wisaksono A, Jamaaluddin J, et al. Implementation of ARM STM32F4 microcontroller for speed control of BLDC motor based on bat algorithm[J]. International Journal Of Power Electronics And Drive Systems, 2024, 15(1): 127-135. https://doi.org/10.11591/ijpeds.v15.i1.pp127-135

[6] Zhang Y, Lyu Y, Hou S, et al. Modeling and Chaos Dynamics Analysis of Doubly-Fed Induction Generator Based on Incommensurate Fractional-Order[J]. IEEE Transactions on Energy Conversion, 2025, 40(2): 911-927. https://doi.org/10.1109/tec.2024.3472771

[7] Kuo M T. Application of the artificial bee colony algorithm to scheduling strategies for energy-storage systems of a microgrid with self-healing functions[J]. IEEE Transactions on Industry Applications, 2021, 57(3): 2156-2167. https://doi.org/10.1109/TIA.2021.3058233

[8] Yu M, Zhang X, Jiang J, et al. Assessing the feasibility of game-theory-based demand response management by practical implementation[J]. IEEE Access, 2021, 9: 8220-8232. https://doi.org/10.1109/ACCESS.2021.3049768

[9] Pang Z, Zhang J, Jiao X, et al. Energy management strategy combining double deep Q‐networks and demand torque prediction for connected hybrid electric vehicles[J]. IEEJ Transactions on Electrical and Electronic Engineering, 2024, 19(2): 234-246. https://doi.org/10.1002/tee.23942

[10] Tan D, Qiu F, Han H, et al. Calculation and analysis of motor dynamic stress considering multiphysical field loads for in-wheel electric vehicle applications[J]. IEEE/ASME Transactions on Mechatronics, 2024, 29(6): 4755-4767. https://doi.org/10.1109/tmech.2024.3386599

[11] Yazar O, Coskun S, Zhang F. Adaptive learning-based energy management for HEVs using soft actor-critic DRL algorithm[J]. Mechatronics Technology, 2024, 1(1): 1-21. https://doi.org/10.55092/mt20240005

[12] Ertargin M, Yildirim O, Orhan A. Mechanical and electrical faults detection in induction motor across

multiple sensors with CNN-LSTM deep learning model[J]. Electrical Engineering, 2024, 106(6): 6941-6951. https://doi.org/10.1007/s00202-024-02420-w

[13] Wang L. Application of Intelligent Sensors in the Collection of Electrical Engineering Automation Equipment[J]. Informatica, 2025, 49(9). https://doi.org/10.31449/inf.v49i9.5499

[14] Zhao Y. Optimization of Mechanical Manufacturing Processes Via Deep Reinforcement Learning-Based Scheduling Models[J]. Informatica, 2025, 49(14): 19-32. https://doi.org/10.31449/inf.v49i14.7204