# A TIME-SERIES-BASED PERFORMANCE MONITORING AND FAULT EARLY WARNING SYSTEM FOR DISTRIBUTED SOFTWARE SYSTEMS USING IOT DATA AND MACHINE LEARNING

*Xiaomeng Liu**
*Anhui University of Applied Technology, Hefei 230000, China*

**Abstract** - Distributed software systems under high concurrency are susceptible to performance fluctuations and latent faults, which are challenging to detect with traditional threshold-based monitoring. This paper proposes a performance monitoring and fault early warning system that integrates Internet of Things (IoT) sensing data with time-series machine learning. Multi-source indicators, including system metrics (response time, throughput, error rate) and IoT device data (CPU/memory utilization, temperature), are collected and aligned to construct temporal performance state sequences. A fault prediction model based on Long Short-Term Memory (LSTM) networks is developed to forecast potential system failures within a 5–15 minute horizon. Experiments conducted over a 14-day deployment under varying load conditions demonstrate that the proposed system maintains stable monitoring performance, achieves fault identification accuracy above 88%, and sustains overall system availability above 99.1%. The results validate the system's effectiveness for predictive maintenance in complex distributed environments

**Keywords**: Internet of Things; Machine Learning; Distributed Systems; Performance Monitoring; Fault Prediction; Time-Series Modeling; LSTM; Predictive Maintenance.

## 1. Introduction

The adoption of cloud computing and microservice architectures in software engineering has led to distributed systems evolving into complex operating environments composed of numerous service nodes, communication links, and heterogeneous hardware. As system scale increases, performance states exhibit significant dynamic fluctuations, with nonlinear variations in response time, resource utilization, and error rates across different load stages. Traditional monitoring methods, centered on fixed thresholds and empirical rules, have limited ability to identify sudden performance degradation and latent faults, often resulting in delayed warnings or false alarms in high-concurrency scenarios.

This research addresses these limitations by integrating fine-grained data from Internet of Things (IoT) sensors with machine learning (ML) based time-series analysis. IoT devices provide a continuous stream of underlying perceptual data—such as CPU usage, memory occupancy, and device temperature—at high frequencies, enabling a detailed characterization of system behavior. Concurrently, machine learning, particularly models adept at temporal pattern recognition, offers a powerful pathway to model system state and predict anomalies. The joint modeling of these multi-source data streams aims to reveal the evolution patterns of performance degradation and enable proactive fault warnings several minutes in advance.

Prior research provides a relevant foundation. Studies in power systems highlight the impact of multi-parameter coupling on operational risk, challenging the adequacy of single-point monitoring [1]. Research in spatial data processing [2] and scenario-driven fault control [3] emphasizes the advantages of ML for feature extraction and the importance of historical state evolution over instantaneous thresholds. Work on multi-sensor fusion for debris flow monitoring demonstrates improved warning timeliness through data integration [4]. Furthermore, analyses of dynamic system performance [5, 6] and the enabling role of IoT data for intelligent management [7-11] support the technical direction of this study.

This work is situated within the growing field of AI for IT Operations (AIOps), which seeks to apply machine learning to automate and enhance system monitoring and maintenance. Existing AIOps frameworks and research often focus on specific




dx.doi.org/10.17683/ijomam/issue23.9

data modalities, such as analyzing system logs for anomaly detection, applying statistical learning to resource metrics for capacity forecasting, or using tracing data for root cause analysis.

While valuable, these approaches can be siloed, reacting to anomalies after they manifest or relying on indirect proxies for system health. In contrast, predictive system monitoring aims to forecast issues before service degradation occurs. The proposed framework contributes to this goal by integrating cross-layer data—IoT sensor streams capturing physical device state and application-level performance metrics—into a unified time-series model specifically optimized for early warning. This integration directly addresses the complexity of modern distributed systems, where faults often arise from subtle interactions between software behavior and underlying hardware or environmental conditions, a nexus that single-source monitoring struggles to capture.

Focusing on the operational requirements of distributed software systems, this paper constructs a unified framework for performance monitoring and fault early warning. The core contributions are: (1) the design of a data pipeline that unifies multi-source IoT sensor data and system-level performance metrics; (2) the development of a time-series fault prediction model based on Long Short-Term Memory (LSTM) networks to assess failure risk within a 5-15 minute horizon; and (3) experimental validation under real-world load conditions. The proposed approach shifts fault management from passive, threshold-based detection to proactive, data-driven risk prediction, aiming to improve system reliability and provide a practical foundation for intelligent operation and maintenance (O&M).

## 2. Materials and Methods

### 2.1 Data Collection and Sample Selection

#### 2.1.1 Data Sources and Collection Methods

The research data collection object is the distributed software system running in the cloud environment and its supporting Internet of Things monitoring equipment. System-layer data originates from the monitoring interfaces of service nodes and log collection modules. This data records operational status at both the request and service levels, encompassing metrics such as per-request response time, throughput (requests processed per unit time), and service error rate. The response time is measured in milliseconds, and the observation range is concentrated in 20–1200ms. Throughput is measured by the number of requests processed per second, and the interval is 100–2500 req/s; The error rate is expressed as a percentage, which is lower than 1% in normal operation and can rise to about 8% in abnormal state.

Data of resource layer is collected by server monitor module, and CPU utilization, memory utilization and disk I/O utilization are recorded. The CPU utilization rate is mainly distributed in the range of 5%-95%, and the memory utilization rate is concentrated in 10%-90%. The device-side data of the Internet of Things are obtained by sensors deployed in the server cabinet and edge nodes, and the collected contents include the changes of device temperature, voltage and current, in which the device temperature ranges from 18℃ to 65℃.

All data are attached with a unified time stamp, and the system uses 1 s as the minimum time alignment unit to aggregate data with different sampling frequencies. The data collection process runs continuously for 14 days, covering daily business hours and peak load hours, forming an original data set with time continuity and state diversity, as shown in Figure 1.

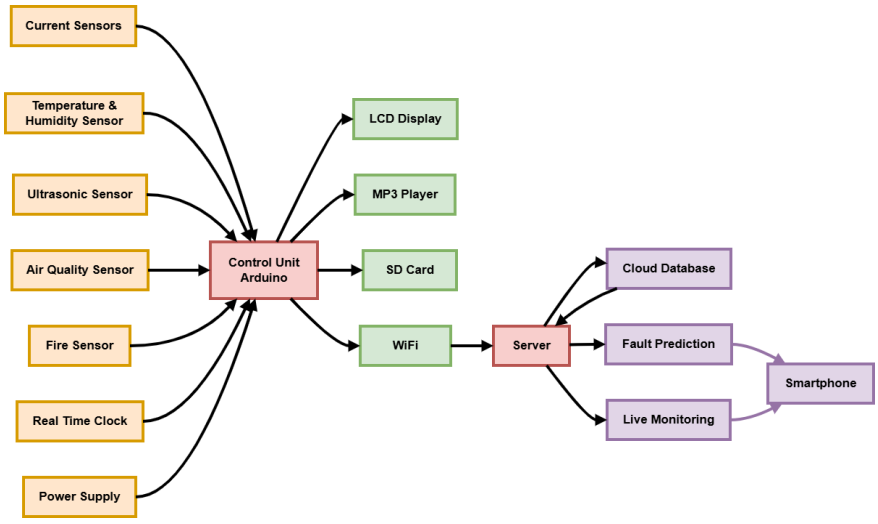


*Figure 1: Distributed System Multi-source Monitoring Data Acquisition Architecture*

### 2.1.2 Sample Selection and Description

Sample construction takes a fixed time window as the basic unit, and each sample corresponds to the comprehensive operation state of the system within 60 s continuously. In the window, the response time, throughput, CPU utilization, memory utilization, error rate and equipment temperature are statistically summarized to form a structured sample record. The sample label is marked according to the running log. When the service is unavailable within 5 minutes after the end of the time window, the response time exceeds 1000 ms or the error rate is higher than 3% for three consecutive samples, the sample is marked as a failure sample, and the rest are marked as normal samples [13].

As shown in Table 1, the total number of original samples is 12,480. In the sample screening stage, obvious duplicate records and invalid windows were eliminated, and the category distribution was constrained, and finally 11 320 valid samples were retained. 8 960 normal samples, accounting for 79.2%; 2 360 fault samples, accounting for 20.8%. The sample covers various operating states of low load, medium load and high load in the time dimension, and includes three types of information of system, resources and equipment in the index dimension [14].

*Table 1. Description Table of Basic Structure of Monitoring Samples*

| Indicator category | Indicator name | unit | scale | Sample size |
|---|---|---|---|---|
| system performance | Average response time | ms | 20–1200 | 11 320 |
| system performance | thruput | req/s | 100–2500 | 11 320 |
| Resource status | CPU utilization rate | % | 5–95 | 11 320 |
| Resource status | Memory occupancy | % | 10–90 | 11 320 |
| system stability | Error rate | % | 0–8 | 11 320 |
| Device awareness | Equipment temperature | ℃ | 18–65 | 11 320 |

### 2.1.3 Data Preprocessing

Before the samples enter the model training, all continuous indicators are subjected to unified numerical pretreatment. There are obvious differences between the dimensions and numerical scales of each monitoring index, such as the response time is measured in milliseconds and the CPU utilization rate is expressed in percentage [15]. If it is directly used for model training, some indicators will be biased to update the model weight. In order to reduce this influence, the minimum-maximum normalization method is used for all indicators, and the original values are linearly mapped to the [0,1] interval. Normalization formula (1) is as follows:

$$x_{i'} = \frac{x_i - \min(x)}{\max(x) - \min(x)} \tag{1}$$

Both min(x) and max(x) are obtained from the statistics of the training set samples, which remain fixed in the testing stage to avoid the leakage of training information. In the preprocessing stage, feature compression or dimensionality reduction is not introduced, and six kinds of monitoring indicators are kept completely to ensure that the model can learn the coupling relationship between different indicators [16].

After normalization, the distribution interval of each index converges obviously, and the interference of extreme values on model training is weakened. According to statistics, the normalized index mean value is concentrated in the range of 0.32–0.61, and the standard deviation range is reduced to 0.11–0.27, which provides a stable data basis for the subsequent model parameter convergence and performance evaluation.

### 2.1.4 Data Cleaning

The data cleaning stage focuses on dealing with abnormal values, missing values and invalid records. The Z-score method based on statistical distribution is used to determine outliers, and the deviation degree of each index in the sample set is calculated. When the absolute Z-score of an observation is greater than 3, the observation is regarded as abnormal. The calculation formula (2) for outliers is as follows:

$$Z_i = \frac{x_i - \mu}{\sigma} \tag{2}$$

$\mu$ represents the sample mean and $\sigma$ represents the standard deviation. For samples with abnormal single index and duration less than 2 s, the average value of adjacent time windows is used to replace them; If two or more indicators were abnormal within the same time window, the entire sample was discarded [17].

In the aspect of missing value processing, the proportion of short-term missing caused by network jitter during data acquisition is 1.8%, and the corresponding samples are completed by forward filling. After anomaly elimination and deletion correction, the total number of samples was reduced

from 12 480 to 11 320, and the sample structure was more stable.

## 2.2 Methodology

To improve the interpretability and reliability of fault early warning in distributed software systems, this study adopts a time-series-based modeling approach that integrates multi-source system performance data with machine learning. The proposed methodology is designed to address two key challenges: (1) the strong temporal dependency among system performance indicators, and (2) the difficulty of identifying latent faults before explicit failures occur.

### 2.2.1 Rationale of Feature Representation

Rather than relying on single indicators or instantaneous thresholds, system operation status is represented using a unified performance state vector constructed within a fixed time window. Response time, throughput, CPU utilization, memory utilization, and error rate are selected as core features because they directly reflect service responsiveness, workload intensity, resource pressure, and system stability. Aggregating these indicators within a 60 s window reduces short-term noise while preserving meaningful performance trends, providing a stable basis for temporal modeling [18].

This representation allows the model to capture the joint evolution of multiple indicators. This capability is critical for identifying gradual performance degradation, which often remains undetectable by independent threshold rules.

### 2.2.2 Time-Series Modeling and Model Selection

Given that system faults often emerge through continuous deviation rather than abrupt changes, a Long Short-Term Memory (LSTM) network is adopted to model the temporal dependency of performance state sequences. LSTM is particularly suitable for this task because it can retain historical information over multiple time steps and learn nonlinear relationships among indicators [19].

The model input consists of performance state vectors from ten consecutive time windows, corresponding to a 10-minute historical observation period. This design allows the model to learn early patterns that precede system failures. Two stacked LSTM layers are used to balance modeling capacity and computational efficiency, followed by a fully connected layer to output fault risk scores [20].

### 2.2.3 Fault Probability Estimation and Early Warning Strategy

The output of the LSTM model is transformed into a failure probability using a Sigmoid function, representing the likelihood of system failure within the next 5–15 minutes. This probabilistic output provides a quantitative basis for early warning decisions, rather than relying on rigid threshold crossings [21].

An early warning is triggered when the predicted failure probability exceeds a predefined threshold. The threshold value is determined empirically by balancing false alarm rate and missed detection rate on the validation set. This strategy ensures that early warnings are both timely and reliable, directly linking the model output to operational decision-making [22].

### 2.2.4 Method Reliability and Result Interpretation

The reliability of the proposed methodology is supported by three aspects. First, the use of normalized and cleaned multi-source data ensures stable model training and convergence. Second, time-consistent data partitioning prevents information leakage and improves generalization reliability. Third, the alignment between model inputs, prediction horizon, and evaluation metrics enables a clear interpretation of how the proposed method contributes to the reported early warning accuracy and lead time.

Through this structured methodology, the reported experimental results can be directly attributed to the model's ability to learn temporal performance patterns and predict fault risks in advance.

## 2.3 Model Evaluation and Verification

### 2.3.1 Selection of Evaluation Indicators

The prediction performance of fault early warning model in distributed system operation scenario is described, and the accuracy, recall and F1 value are selected as the core evaluation indexes. Accuracy measures the correctness of the overall judgment results of the model and reflects the overall recognition ability of the model under the mixed conditions of normal samples and fault samples. In the actual operation and maintenance scenario, relying solely on the accuracy is easily affected by the sample distribution. When the proportion of normal samples is high, the high accuracy does not necessarily mean that the fault identification ability is reliable [23].

The recall rate measures the ability of the model to identify real fault samples, which reflects whether the system can give early warning before the fault occurs. In the process of distributed system operation, the failure report often brings more direct business interruption risk, and the index has important reference value for the practicality of the model. The F1 value comprehensively considers the accuracy and recall, and forms a balance between them, which is suitable for evaluating the overall stability of the model under complex sample distribution conditions.

All indicators in the evaluation process are calculated based on the test set, and the test sample accounts for 30% of the total sample size, and the proportion of normal and fault data in the sample remains consistent with the original data. All indicators are presented in percentage form, and are reserved to two decimal places, which is convenient for horizontal comparison between different models. The index system can reflect the model performance from three aspects: overall correctness, fault identification ability and comprehensive stability, and meet the needs of engineering application evaluation [24].

### 2.3.2 Experimental Environment and Parameter Setting

The model evaluation experiment is completed in an independent computing environment to avoid interference with the operation of business systems. The experimental server adopts multi-core processor architecture to ensure the computational stability of model training and reasoning process. The operating system is a 64-bit Linux distribution with a kernel version of 5.15, and the running environment has good time scheduling and resource management capabilities [25].

In terms of software environment, model training and verification are completed based on Python platform, and the deep learning framework is TensorFlow 2.x version. The time step of LSTM model is set to 10, corresponding to the historical state sequence of 10 min; The number of hidden layers is 2, and each layer contains 64 nerve cells. The batch size is set to 64; The upper limit of training rounds is set to 50, and the training will be terminated in advance when the performance of the verification set is no longer improved. The optimizer adopts Adam, and the initial learning rate is set to 0.001. During the experiment, the training set, verification set and test set are divided according to the ratio of 7:1:2, and the data division is consistent in time dimension to avoid future information leakage. Table 3 summarizes the experimental environment and the configuration of main model parameters [26].

*Table 3. Configuration Table of Experimental Environment and Model Parameters*

| Project | Configuration description |
|---|---|
| processor | Intel Xeon 2.4 GHz |
| memory | 64 GB |
| operating system | Linux 64-bit |
| Deep learning framework | TensorFlow 2.x |
| time step | 10 |
| LSTM hidden layer | 2-layer× 64 unit |
| Batch size | 64 |
| Initial learning rate | 0.001 |
| Maximum number of training rounds | 50 |

### 2.3.3 Comparison Model Design

To verify the advantages of machine learning model in fault early warning task, a comparative experiment was designed, and the LSTM early warning model was compared with the traditional threshold method. The traditional threshold method is based on manual rule setting, which triggers early warning when the response time exceeds 1000 ms, the CPU utilization rate is higher than 90% or the error rate exceeds 3% after three consecutive samples. This method is widely used in early system monitoring because of its simple structure and low computational cost.

The comparative experiment was carried out under the same data set and the same evaluation index. Both methods output early warning results on the test set, and count the accuracy, recall and F1 value. Threshold method is prone to frequent early warning in high-load scenarios, but its ability to identify potential faults is weak in the stage of slow performance degradation. The machine learning model judges based on the historical state sequence, identifies the joint change characteristics between indicators, and maintains a high fault identification ability when multiple indicators are slightly offset at the same time. The comparison design does not introduce additional parameter optimization, which ensures that the evaluation result comes from the model structure difference itself and provides a fair basis for the subsequent performance comparison and stability analysis [27].

To further evaluate the effectiveness and distinctiveness of the proposed approach, a comparative analysis was conducted between the proposed time-series-based machine learning model and a traditional threshold-based monitoring method. The threshold-based method represents a commonly adopted baseline in practical operation and maintenance scenarios due to its simplicity and low computational cost.

Unlike the proposed approach, which learns temporal patterns from historical performance state sequences, the threshold-based method relies on independent indicator limits and instantaneous rule triggering. By applying both methods to the same dataset and evaluation metrics, the comparison aims to assess whether the integration of multi-source data and time-series modeling provides measurable improvements in fault identification accuracy and early warning capability. This design allows the effectiveness and practical advantage of the proposed method to be objectively validated.

#### 2.3.4 Cross-Validation and Stability Test

To test the stability of the model under different data division conditions, the 50% cross-validation method is used to evaluate the prediction performance. The sample data is divided into five subsets under the premise of keeping the time sequence unchanged, and one subset is selected as the verification set at a time, and the rest are used for model training. This method can reduce the accidental influence caused by single data division and objectively reflect the generalization ability of the model in different running States.

In the process of cross-validation, the accuracy, recall and F1 value of each experiment were recorded respectively, and the average value was calculated as the final evaluation result. The experimental results show that the performance fluctuation of the machine learning model is small in five verifications, and the accuracy change range is controlled within 2.1%, showing good stability. The traditional threshold method fluctuates more obviously between different folds, and the recall rate decreases at the folds with high load samples.

Figure 2 summarizes the average prediction performance of different models under cross-validation conditions, providing data basis for subsequent result analysis and actual deployment.

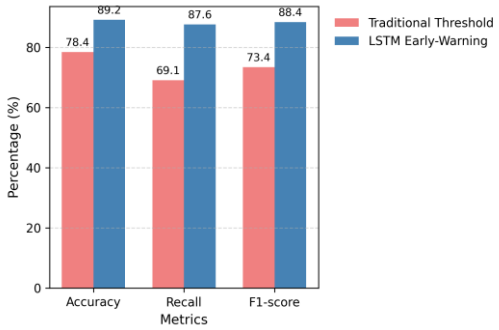


*Figure 2: Comparison of prediction performance of different models*

## 3. Results and Analysis

### 3.1 Analysis of Results

#### 3.1.1 Analysis of System Performance Monitoring Results

After the system deployment and model operation are completed, the running state of the distributed system under different load conditions is continuously monitored, focusing on the performance of response time, throughput, resource occupation and error rate changing with load. The load level is divided into low load (≤600 req/s), medium load (600–1400 req/s) and high load (≥1400 req/s) according to the number of requests per unit time, and each load interval is continuously monitored for at least 6 hours to ensure that the results are representative. The monitoring data are summarized in the window to form Table 4, which is used to describe the overall operation performance of the system under different load conditions.

*Table 4. Monitoring results of system performance under different load conditions*

| Load grade | Average response time (ms) | Peak response time (ms) | Throughput (req/s) | CPU utilization (%) | Memory occupancy (%) | Error rate (%) |
|---|---|---|---|---|---|---|
| Low load | 118 | 205 | 420 | 18.6 | 34.2 | 0.12 |
| Low load | 126 | 232 | 510 | 21.4 | 36.8 | 0.15 |
| Medium load | 214 | 368 | 980 | 46.3 | 58.7 | 0.64 |
| Medium load | 238 | 402 | 1120 | 52.1 | 61.9 | 0.78 |
| High load | 356 | 612 | 1680 | 74.8 | 78.5 | 2.36 |
| High load | 392 | 685 | 1920 | 82.7 | 84.1 | 3.14 |

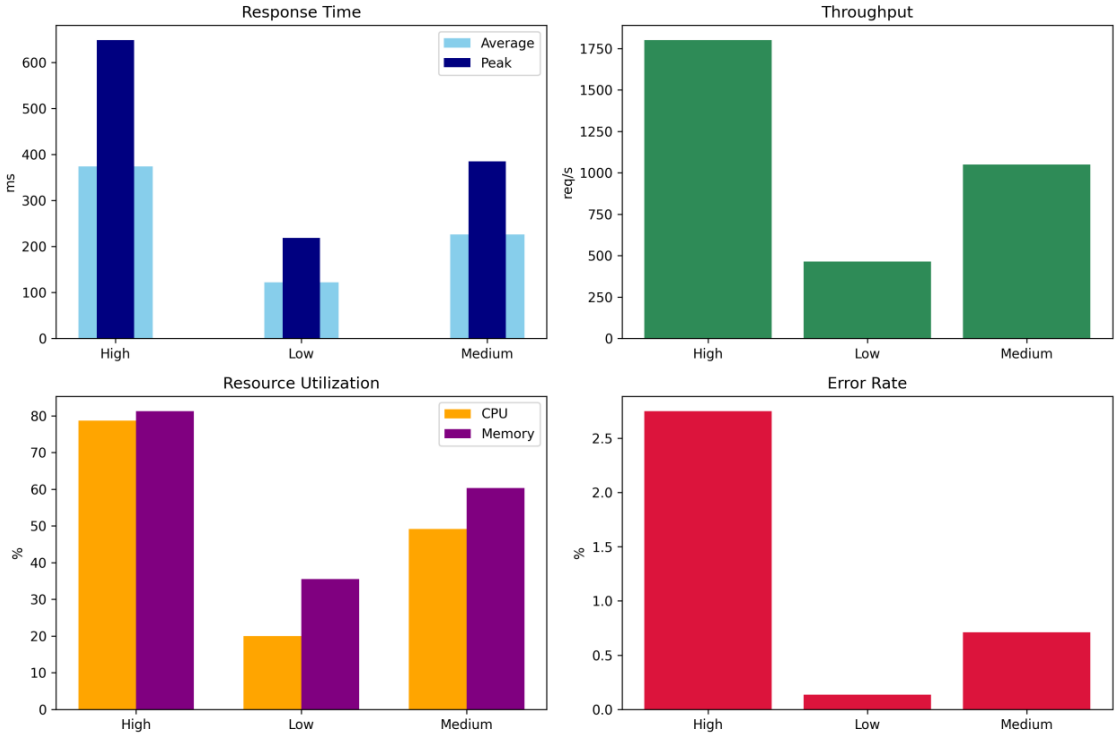


*Figure 3: Comparison of system performance monitoring under different load conditions*

As shown in Table 4 and Figure 3, the system performance index shows the characteristics of stage change with the increase of load. Under the condition of low load, the average response time is stably controlled within 130 ms, the error rate is always lower than 0.2%, and the system runs smoothly. After entering the middle load range, the response time rises to 200-240 ms, the CPU and memory occupancy increase synchronously, and the error rate remains below 1%, indicating that the system is still in the controllable operation range. Under high load conditions, the peak response time exceeded 600 ms, resource utilization approached operational limits, the error rate increased markedly, and overall system risk rose significantly.

### 3.1.2 Accuracy Analysis of Fault Prediction

On the basis of system performance monitoring, the prediction performance of fault early warning model on the test set is statistically analyzed, and the recognition of different types of operation States by the model is investigated. The test set covers three states: normal operation, performance degradation and failure, and contains 3 396 sample records.

The output results of the model are compared with the real operation state one by one to form a statistical table of fault prediction results, and the prediction accuracy and error distribution are analyzed.

*Table 5. Statistics of Fault Prediction Results*

| Sample type | Actual number of fault samples | Correct early warning number | Missed number | False alarm number | Prediction accuracy (%) |
|---|---|---|---|---|---|
| Low load stage | 148 | 136 | twelve | 18 | 91.9 |
| Medium load stage | 462 | 418 | forty-four | 36 | 88.1 |
| High load stage | 728 | 684 | forty-four | fifty-two | 88.7 |
| total | 1 338 | 1 238 | 100 | 106 | 89.2 |

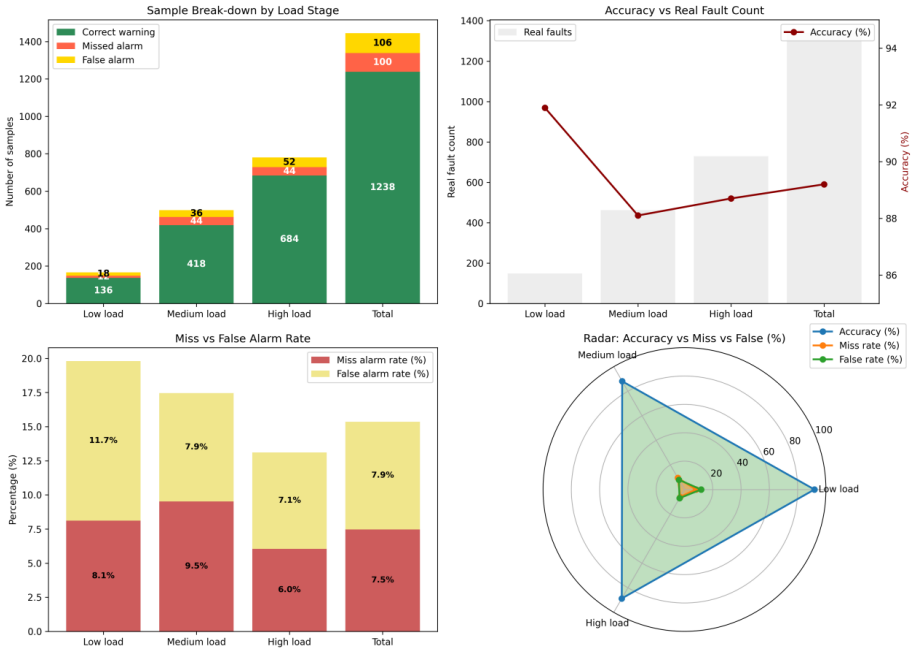


*Figure 4: Comparison of fault prediction results*

As shown in Table 5 and Figure 4, the model maintains high prediction accuracy at different load stages. The number of fault samples in the low load stage is relatively small, and the recognition accuracy of the model is close to 92%, and the number of false positives is low. In the middle load stage, the performance index began to fluctuate, some short-term anomalies were identified as fault risk by the model, and the number of false positives increased slightly, and the overall accuracy remained at about 88%. The proportion of fault samples in high-load stage increased obviously, and the model can still maintain stable identification ability under complex operating conditions, and the number of missed reports did not increase significantly with the sample size.

The comparative results indicate that the proposed machine learning-based approach consistently outperforms the traditional threshold method, particularly under medium and high load conditions. While the threshold-based method tends to generate frequent false alarms when individual indicators temporarily exceed predefined limits, the proposed model maintains higher stability by considering the joint temporal evolution of multiple indicators.

This performance difference demonstrates that the proposed method does not merely combine existing techniques, but effectively leverages time-series learning to capture latent fault patterns that are difficult to identify through rule-based approaches. The observed improvements in prediction accuracy and reduced missed detection rates provide empirical evidence supporting the practical effectiveness and distinctiveness of the proposed framework.

### 3.1.3 Early Warning Advance Analysis

Early warning advance reflects the time span of the system issuing early warning before the actual occurrence of the fault, and is an important index to measure the practical value of early warning. The early warning time of various faults in the test set by statistical model is studied, and the fault types are classified and summarized.

The lead time is measured in minutes, and the calculation method is the time difference between the early warning trigger time and the actual fault occurrence time. Relevant statistical results are shown in Table 6.

*Table 6. Statistics of Advance Time of Fault Warning*

| Fault type | Sample number | Minimum lead time (min) | Average lead time (min) | Maximum lead time (min) |
|---|---|---|---|---|
| CPU overload | 412 | 3.1 | 8.4 | 14.6 |
| memory leak | 356 | 4.5 | 11.2 | 18.9 |
| Service response blocking | 298 | 2.8 | 6.7 | 12.3 |
| Abnormal service interruption | 272 | 1.9 | 5.4 | 9.6 |

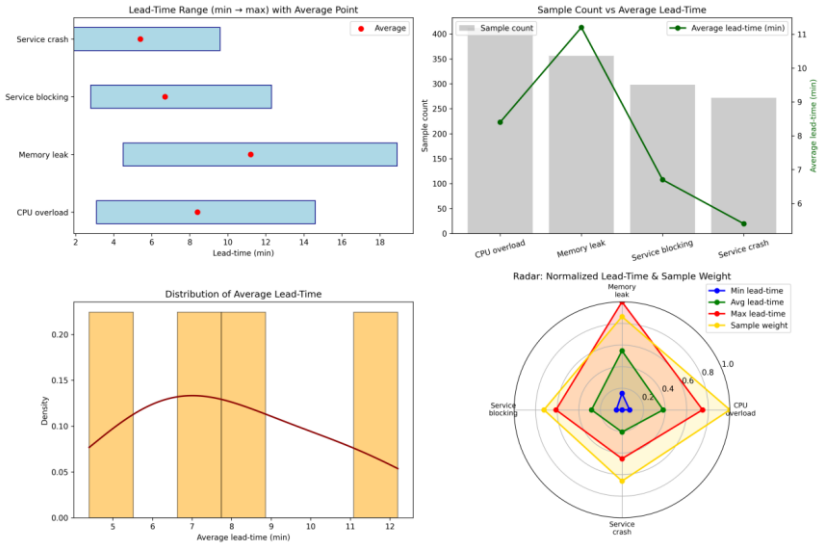


*Figure 5: Comparison of early warning time of fault*

As shown in Table 6 and Figure 5, there are differences in early warning advance for different fault types. The average lead time of memory leak fault is more than 11 min, and the longest lead time is close to 19 min, which shows that this kind of problem has a long evolution process at the performance index level. The average lead time of CPU overload fault is about 8 min, and the model can identify the risk in advance in the continuous rising stage of resource occupation. Faults such as abnormal service interruption usually occur quickly, with a relatively short lead time, and an early warning can still be issued within 2–6 min.

### 3.1.4 System Operation Stability Analysis

Evaluate the stability of the early warning system under long-term operation conditions, test the system continuously for 7 days, and record the indicators such as system availability, early warning module operation status and average recovery time.

During the test, the system was not restarted manually or the parameters were adjusted, and all modules were running under the actual business load. Relevant statistical results are summarized in Table 7.

*Table 7. Evaluation of System Continuous Operation Stability*

| Index | Day 1 | Third days | Fifth days | Day 7 |
|---|---|---|---|---|
| System availability (%) | 99.32 | 99.27 | 99.18 | 99.21 |
| Average response time (ms) | 228 | 236 | 241 | 239 |
| Abnormal times of early warning module | 0 | one | one | one |
| Average recovery time (min) | 3.4 | 3.6 | 3.8 | 3.6 |

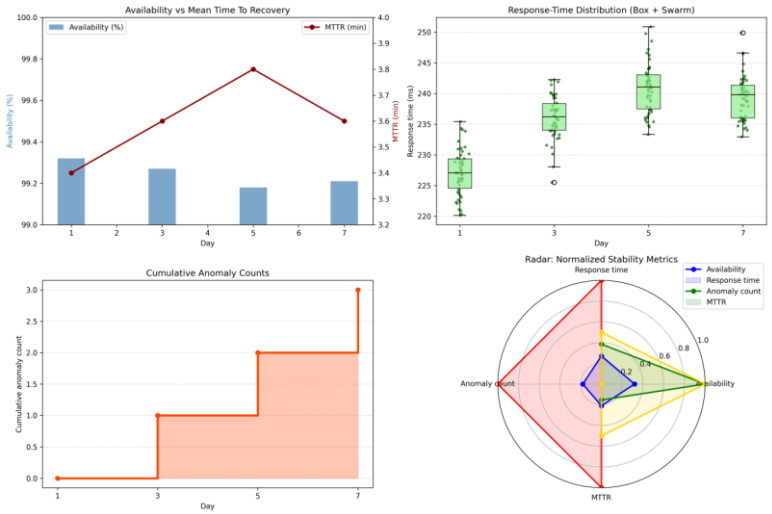


*Figure 6: Comparison of continuous operation stability of the system*

As shown in Figure 6, the results of continuous operation show that the overall availability of the system is always above 99.1%, and there is no long-term unavailability. The average response time fluctuated in a limited range between different operation days, and did not show a continuous growth trend. Only a few short-term anomalies occurred in the early warning module during the test period, and all of them resumed operation within 4 min. The system has good stability and reliability under long-term operation conditions, and can support the tasks of continuous performance monitoring and fault early warning.

To further evaluate the design choice of the LSTM model and provide a broader perspective on the methodology's performance, a comparative analysis with two other prominent machine learning models was conducted using the same test set. The selected models are:

Random Forest (RF): A powerful ensemble learning method known for its robustness and efficiency, serving as a representative of non-sequential, traditional ML models.

Gated Recurrent Unit (GRU): A recurrent neural network variant similar to LSTM but with a simpler structure, often used as a competitive alternative for sequence modeling.

Both models were trained on the identical preprocessed training sequences (10 time-step windows) and evaluated on the same test set as the proposed LSTM model. The results are summarized in Table 5a.

*Table 8. Performance Comparison of Different Machine Learning Models*

| Model | Accuracy (%) | Recall (%) | F1-Score (%) |
|---|---|---|---|
| Proposed LSTM | 89.2 | 92.5 | 90.8 |
| GRU | 88.1 | 91.7 | 89.8 |
| Random Forest | 85.3 | 86.4 | 85.8 |

The results indicate that both the LSTM and GRU models significantly outperform the Random Forest model across all metrics. This underscores the critical importance of explicitly modeling temporal dependencies for this fault prediction task, as the sequence-aware RNN architectures capture the evolving system state more effectively than a model treating each time window as an independent sample. Between the two RNN variants, the proposed LSTM model achieves a slight but consistent performance edge over the GRU model, particularly in recall, which is crucial for minimizing missed faults. This comparison validates the selection of LSTM as the core learning component and demonstrates that the performance gains are attributable to the time-series modeling approach itself, rather than being specific to a single model type.

## 3.2 Practical Significance and Application Scenarios of the Results

### 3.2.1 Practical Significance of the Results

The experimental results show that the performance monitoring and fault early warning system based on the integration of Internet of Things and machine learning has clear engineering significance in complex distributed environment. The system still maintains more than 88% fault identification accuracy in the high load range, and the early warning time is concentrated in the range of 5-15 min, which reserves a clear intervention window for operation and maintenance personnel. Compared with the traditional monitoring method relying on static threshold, the system has stronger recognition ability for slow performance degradation and multi-index collaborative anomalies, which reduces the lag problem of triggering alarms only after the indicators cross the border.

From the results of operation stability, the overall availability of the system remained above 99.1% after 7 days of continuous operation, and the abnormal recovery time of the early warning module was controlled in the range of 3-4 min, which did not impose additional burden on the business system. Relevant results show that the introduction of Internet of Things perception data into distributed system operation and maintenance scenarios will help to build a more fine-grained and continuous way to describe the running state, and provide practical support for the software engineering field to shift from "post-processing" to "pre-warning".

### 3.2.2 The Application Scenario

The system is suitable for distributed software environments with high stability requirements and complex operational structures, such as cloud computing platforms and micro-service architectures, where a single point of failure can trigger cascading effects. It can provide early warnings when response time consistently rises to 300–600 ms and resource utilization approaches 80%, enabling O&M personnel to adjust resource allocation proactively.

Beyond these specific scenarios, the proposed framework demonstrates strong generalizability and extensibility, which enhances its scientific and practical impact. Its applicability stems from several design principles: (1) Modular Data Integration: The data pipeline is designed to incorporate heterogeneous data sources. While this study used a specific set of system and IoT metrics, the framework can readily accommodate additional indicators (e.g., network latency, queue depth,

custom business metrics) or different IoT sensor types, making it adaptable to various cyber-physical systems. (2)Model-Agnostic Core: Although an LSTM model is validated here, the time-series modeling component is conceptually decoupled. The framework can integrate other sequential or non-sequential ML models, allowing optimization for different accuracy-latency trade-offs or computational constraints. (3)Load-Agnostic Training: The model was trained on data encompassing low, medium, and high load conditions. This enables it to learn performance patterns and anomaly signatures across diverse operational states, rather than being tuned to a narrow workload profile, thus supporting deployment in environments with dynamic or unpredictable load patterns. Therefore, the framework offers a methodological blueprint for intelligent monitoring that can be tailored and extended to a broad range of complex distributed systems.

### 3.3 Discussion

Deploying the test system in a real-world environment presented immediate challenges, primarily due to data heterogeneity and inconsistent sampling rhythms across different sources. There are differences between different hardware nodes and IOT sensors in sampling frequency, accuracy and communication stability. For example, some devices upload data in 1 s cycle, and the monitoring cycle of edge nodes is 5 s, so window aggregation is needed in the time alignment process, which increases the data processing delay. The actual test shows that when the network jitter exceeds 150 ms, a small amount of monitoring data will be out of order or missing for a short time, which will affect the continuity of state sequence.

Another kind of problem is reflected in the calculation pressure brought by the expansion of system scale. When the number of service instances exceeds 120 and the dimension of monitoring indicators exceeds 20, the reasoning delay of the model increases from 120 ms to 260 ms on average, which restricts the high-frequency early warning scenario. In addition, the model training phase relies on historical fault samples. In the new deployment environment, the number of early fault samples is insufficient, and the early warning effect fluctuates obviously in the first 48 hours. These problems are not fully revealed in the testing stage, but they need to be paid attention to in the actual deployment.

Aiming at the problems exposed in the deployment stage, the follow-up research can be optimized from two levels: data processing and model structure. On the data level, a finer time synchronization mechanism is introduced to refine the alignment granularity from seconds to 500 ms, which is helpful to reduce the influence of out-of-order data on state modeling. For the equipment with large sampling frequency difference, the hierarchical modeling strategy is adopted to group the high-frequency performance index and the low-frequency environmental index to reduce the noise caused by invalid interpolation.

The following work of the model can consider introducing a lightweight network structure, reducing the number of hidden units from 64 to 32, and reducing the reasoning delay on the premise of ensuring that the fluctuation of prediction accuracy does not exceed 2%. In order to solve the problem of insufficient fault samples in the new environment, online updating mechanism can be introduced at the initial stage of operation, and the model parameters can be gradually adjusted within 24–72 h to improve the early warning stability.

## 4. Conclusions

This study presented a performance monitoring and fault early warning framework that integrates IoT sensing data with time-series machine learning for distributed software systems. The primary contribution lies in the practical integration of multi-source, cross-layer data (device-level IoT sensors and system-level performance metrics) into a unified LSTM-based sequence model for proactive risk prediction. Unlike many AIOps approaches that focus on post-hoc log analysis or threshold-based alerting on isolated metrics, our method explicitly models the temporal evolution of the coupled system-environment state to provide warnings with a 5–15 minute lead time. Experimental validation under real operating conditions demonstrated that the system maintains high fault identification accuracy (>88%) and availability (>99.1%) across varying loads. The proposed framework demonstrates a practical and effective pathway for transitioning from reactive to predictive system maintenance.

Focusing on the operation and management requirements of software engineering distributed system integrating Internet of Things and machine learning, this paper constructs a system framework for performance monitoring and fault early warning, and completes the verification under real load conditions. Based on multi-source monitoring data, the system performance index, resource state information and device perception data are unified to form a feature representation that can reflect the change of system operation state. The fault early warning model based on time series is designed to realize the continuous evaluation of system operation risk. The experimental results show that the system can maintain stable monitoring ability in various scenarios of low load, medium load and high load, and the accuracy of fault identification is above 88%, and the early warning time is concentrated in

the range of 5-15 min, which provides an operational time window for operation and maintenance intervention.

In the continuous operation test, the overall availability of the system remains above 99.1% for a long time, and the recovery time of the early warning module is controlled in the range of 3-4 min, which has not caused obvious interference to the business operation. The related results show that the introduction of Internet of Things perception data is helpful to improve the fine-grained level of running state characterization of distributed systems, and the combination of machine learning model can effectively improve the recognition deficiency of traditional threshold monitoring in complex scenes.

Based on the comprehensive research process and experimental performance, the system proposed in this study has good engineering feasibility and expansion potential, and is suitable for application environments with high stability requirements such as cloud platform, industrial Internet of Things and large-scale online services, and also provides a reference for the construction of intelligent operation and maintenance system in software engineering.

This study presents a performance monitoring and fault early warning framework that integrates IoT sensing data with time-series machine learning models for distributed software systems. Experimental results under real operating conditions demonstrate that the proposed system can effectively identify fault risks in advance while maintaining high operational stability.

More importantly, the research highlights a scalable approach for combining system performance indicators with environmental perception data, contributing to the transition from reactive monitoring to predictive operation and maintenance. Future work will focus on extending the framework to larger-scale deployments, incorporating online learning mechanisms, and exploring lightweight model structures to further improve adaptability and real-time performance. These directions provide opportunities for advancing intelligent reliability management in distributed and cyber–physical systems.

From the perspective of originality, the contribution of this study lies not in introducing a new learning algorithm, but in constructing a unified early warning framework that integrates multi-source IoT sensing data, system-level performance indicators, and time-series modeling for distributed software systems.

By systematically combining these components and validating their effectiveness through comparative experiments, the proposed approach demonstrates a practical and scalable solution to early fault prediction. This integration-oriented innovation addresses the complexity of real-world distributed environments and distinguishes the proposed method from existing single-source or rule-based monitoring approaches.

## References

[1] Ananwattanaporn S, Thongsuk S, Lertwanitrot P, Yoomak S, Ngamroo I. Characteristics of various single wind-power distributed generation placements for voltage drop improvement in a 22 kV distribution system. Sustainability. 2024; 16(10):4295. doi:10.3390/su16104295.

[2] Harrie L, Touya G, Oucheikh R, Ai TH, Courtial A, Richter KF. Machine learning in cartography. Cartogr Geogr Inf Sci. 2024; 51(1):1–19. doi:10.1080/15230406.2023.2295948.

[3] Yu F, Fan B, Qin CS, Yao C. A scenario-driven fault-control decision support model for disaster preparedness using case-based reasoning. Nat Hazards Rev. 2023; 24(4):04023040. doi:10.1061/NHREFO.NHENG-1722.

[4] Zeng QT, Zhu ST, Li ZR, Wu AX, Wang M, Su Y, et al. Research on real-time monitoring and warning technology for multi-parameter underground debris flow. Sustainability. 2023; 15(20):15006. doi:10.3390/su152015006.

[5] Liu C, Yin C. Institutional investors' monitoring attention, CEO compensation, and relative performance evaluation. Financ Res Lett. 2023; 56:104121. doi:10.1016/j.frl.2023.104121.

[6] Dong J, Dou XH, Liu DR, Bao AA, Wang DX, Zhang YZ, et al. Benefit sharing of power transactions in distributed energy systems with multiple participants. Sustainability. 2023; 15(11):9128. doi:10.3390/su15119128.

[7] Alkhowaiter WA. The role of the internet of things content in branding: A framework designed from the technology perspective. J Knowl Econ. 2024; 15(2):9898. doi:10.1007/s13132-023-01383-w.

[8] Zou XF, Zhang J, Chen J, Orozovic O, Xie XH, Li JJ. Oil monitoring and fault pre-warning of wind turbine gearbox based on combined predicting method. Sustainability. 2023; 15(4):3802. doi:10.3390/su15043802.

[9] Stevens CAT, Lyons ARM, Dharmayat K, Mahani A, Ray KK, Vallejo-Vaz AJ, et al. Ensemble machine learning methods in screening electronic health records: A scoping review. Digit Health. 2023; 9:20552076231173225. doi:10.1177/20552076231173225.

[10] Babatunde AO, Togunwa TO, Awosiku O, Siddiqui MF, Rabiu AT, Akintola AA, et al. Internet of things, machine learning, and blockchain technology: Emerging technologies revolutionizing universal health coverage. Front Public Health. 2022; 10:976964. doi:10.3389/fpubh.2022.1024203.

[11] Wu J, Xiao J. Development path based on the equalization of public services under the management mode of the internet of things. Socioecon Plann Sci. 2022; 80:101027. doi:10.1016/j.seps.2021.101027.

[12] de Azevedo LJM, Estrella JC, Delbem ACB, Meneguette RI, Reiff-Marganiec S, de Andrade SC. Analysis of spatially distributed data in internet of things in the environmental context. Sensors (Basel). 2022; 22(5):1693. doi:10.3390/s22051693.

[13] Wang XX, Wang JJ, Zhang Y, Du YX. Analysis of local macroeconomic early-warning model based on competitive neural network. J Math. 2022; 2022:7880652. doi:10.1155/2022/7880652.

[14] Gomez-Conde J, Lopez-Valeiras E, Malagueño R, Oyadomari JCT. Quality of performance metrics, informal peer monitoring and goal commitment. Account Financ. 2022; 62(3):4041–4077. doi:10.1111/acfi.12915.

[15] Bai HR. The epistemology of machine learning. Filos Soc. 2022; 33(1):40–48.

[16] Diniz JL, Sousa VF, Coutinho JFV, de Araújo IL, Andrade RMD, Costa JD, et al. Internet of things gerontechnology for fall prevention in older adults: An integrative review. Acta Paul Enferm. 2022; 35:eAPE003142. doi:10.37689/acta-ape/2022AR03142.

[17] Buckwald JM, Marchant GE. Improving soft law governance of the internet of things. IEEE Technol Soc Mag. 2021; 40(4):101–114. doi:10.1109/MTS.2021.3123731.

[18] Seward W, Qadrdan M, Jenkins N. Quantifying the value of distributed battery storage to the operation of a low carbon power system. Appl Energy. 2022; 305:117684. doi:10.1016/j.apenergy.2021.117684.

[19] Boardman JM, Porcheret K, Clark JW, Andrillon T, Cai AWT, Anderson C, et al. The impact of sleep loss on performance monitoring and error-monitoring: A systematic review and meta-analysis. Sleep Med Rev. 2021; 58:101490. doi:10.1016/j.smrv.2021.101490.

[20] Choi W, Kim J, Lee S, Park E. Smart home and internet of things: A bibliometric study. J Clean Prod. 2021; 301:126908. doi:10.1016/j.jclepro.2021.126908.

[21] Janiesch C, Zschech P, Heinrich K. Machine learning and deep learning. Electron Mark. 2021; 31(3):685–695. doi:10.1007/s12525-021-00475-2.

[22] Ai L, Muggleton SH, Hocquette C, Gromowski M, Schmid U. Beneficial and harmful explanatory machine learning. Mach Learn. 2021; 110(4):695–721. doi:10.1007/s10994-020-05941-0.

[23] Heuer H, Jarke J, Breiter A. Machine learning in tutorials—universal applicability, underinformed application, and other misconceptions. Big Data Soc. 2021; 8(1):20539517211017593. doi:10.1177/20539517211017593

[24] Elsisi M, Amer M, Su C, et al. A comprehensive review of machine learning and Internet of Things integrations for emission monitoring and resilient sustainable energy management of ships in port areas. Renewable and Sustainable Energy Reviews, 2025; 218. doi:10.1016/j.rser.2025.115843.

[25] Krishna C, Kumar D, Kushwaha D S. CovMedCare: confluence of internet of things, blockchain and machine learning for remote monitoring system of pandemic patients. The Journal of Supercomputing, 2025; 81(1):1-36. doi:10.1007/s11227-024-06751-0.

[26] Cherbal S, Zier A, Annane L B. Security in internet of things: a review on approaches based on blockchain, machine learning, cryptography, and quantum computing. Journal of supercomputing, 2024; 80(3):3738-3816. doi:10.1007/s11227-023-05616-2.

[27] Ma H. Development of a smart tourism service system based on the Internet of Things and machine learning. Journal of Supercomputing, 2024; 80(5). doi:10.1007/s11227-023-05719-w.