International Journal of Emerging Technologies and Advanced Applications

Real-time Fault Detection and Stability Enhancement Mechanism Based on Large Models

Chuanyong Zhao[1], Yuan Xi[2]
[1] Beijing Didichuxing Technology Development Co., Ltd., Beijing, China
[2] Beijing Dongchezhu Technology Co., Ltd., Beijing, China
xiyuan@vip.163.com

***Abstract*—This paper proposes a framework for real-time fault detection using large models, which rapidly identifies potential faults through system log and operational data analysis, triggering stability enhancement mechanisms. The research designs a self-supervised learning algorithm that enables large models to continuously improve detection accuracy in dynamic environments. The method adopts a transformer architecture and attention mechanism to capture temporal dependencies and complex patterns in system behavior. Through comparison with traditional methods, we validate the advantages of the proposed method in terms of accuracy and real-time performance. Experiments conducted across various fault scenarios demonstrate that the method significantly reduces fault response time in high-concurrency systems, decreasing average detection latency by 47.3% and shortening system recovery time by 35.8%, thereby improving overall stability. Additionally, the self-supervised nature of the method enables continuous adaptation to new fault patterns, providing an innovative solution for reliability assurance in distributed systems.**

***Index Terms*—Large models, Fault detection, Real-time monitoring, Stability enhancement, Self-supervised Learning**

I. Introduction

A. Research Background

With the widespread application of distributed systems and cloud computing, the complexity of systems in high-concurrency environments has grown exponentially, presenting unprecedented challenges for system fault detection and handling. Currently, large-scale distributed systems typically comprise hundreds or thousands of microservices with complex interdependencies, where the failure of any single component can trigger a chain reaction leading to system performance degradation or even complete collapse. Statistics show that approximately 68% of service interruption incidents in large internet companies stem from difficult-to-predict complex fault cascade phenomena [1]. This complexity makes traditional fault detection methods based on thresholds or simple rules difficult to adapt, as they cannot capture deep interaction patterns and temporal dependencies between system components. Traditional methods also often rely on manually defined rules, which perform poorly when faced with emerging fault types, resulting in high false positive and false negative rates that seriously affect system reliability and user experience [2]. Furthermore, as business scale expands, the cost of manually defining and maintaining these rules grows exponentially, posing severe challenges to the scalability of traditional methods. In this context, large model technology, with its powerful pattern recognition capabilities and adaptability, shows potential for solving complex system fault detection problems. Large models based on transformer architectures can process and understand long sequence data, capturing complex dependencies in time and space, which is perfectly suited for processing multi-dimensional monitoring data in distributed systems [3]. The breakthrough progress of large models in fields such as natural language processing and computer vision in recent years provides reference for their application in system reliability. In particular, the self-supervised learning capability of large models enables them to learn from massive unlabeled data, which is especially important for fault detection where labeled fault data is typically scarce in real environments [4].

B. Research Significance

In today's highly digitally dependent era, system stability has become a key factor affecting user experience and corporate reputation. Research shows that service interruptions in enterprise applications can cause economic losses averaging hundreds of thousands to millions of dollars per hour, not including the long-term impact of lost user trust [5]. Therefore, innovation in real-time fault detection and stability enhancement technology has significant practical importance. The main innovation of this research lies in the deep integration of large model technology with fault detection, designing an end-to-end real-time monitoring and self-healing framework. Unlike traditional methods, our proposed approach can automatically learn normal patterns from system historical behavior without requiring numerous manually labeled fault samples, greatly reducing deployment costs. Meanwhile, through the self-supervised learning mechanism, the method can continuously adapt to system evolution and new fault pat-

Copyright: © 2025 The Author(s); CC BY-NC 4.0.
Exclusive licensee IJETAA.

ISSN:3006-2985(Print)
ISSN:3006-9297(Online)

terns, solving the problem of difficult maintenance and updates that traditional methods face. In terms of application value, the framework proposed in this research is not only applicable to infrastructure monitoring for cloud service providers but can also be extended to critical fields with high system reliability requirements such as finance, healthcare, and manufacturing. The method significantly improves system availability and reliability by reducing manual intervention and achieving early fault detection and automatic mitigation. Additionally, the stability enhancement mechanism in the framework can automatically trigger corresponding repair strategies based on the detected fault type, forming closed-loop control, which has important value for autonomous operation and maintenance of large-scale distributed systems. With the acceleration of enterprise digital transformation and the popularization of cloud-native architecture, the solution proposed in this research will help build more resilient computing infrastructure, providing strong support for continuous business operations.

II. Related Work

A. Overview of Fault Detection Technologies

Fault detection technology has evolved from simple rules to complex intelligent algorithms. Rule-based methods were the earliest applied fault detection technology, judging system states through preset thresholds and logical rules. Zhao et al. [6] proposed a rule-based network traffic anomaly detection framework that identifies DDoS attacks by defining traffic pattern rules. Although simple and intuitive to implement, these methods are highly sensitive to threshold settings and difficult to handle complex fault scenarios. With the development of data-driven methods, traditional machine learning techniques began to emerge in the fault detection field. Liu et al. [7] used support vector machines (SVM) to analyze server performance metrics, achieving prediction of resource exhaustion faults. Chen and Wang [8] combined decision tree and random forest algorithms to construct a multi-level fault detection model, achieving certain success in identifying complex fault patterns. However, these methods typically rely on carefully designed feature engineering and perform poorly when processing high-dimensional unstructured data. In recent years, deep learning methods have been widely applied in the fault detection field due to their powerful feature extraction capabilities. Zhang et al. [9] proposed a method based on Long Short-Term Memory networks (LSTM) that can capture temporal dependencies in system metrics, effectively improving the detection accuracy of performance anomalies. Li et al. [10] designed a hybrid model combining Convolutional Neural Networks (CNN) and autoencoders, learning normal patterns from large-scale monitoring data through unsupervised learning and marking behaviors that deviate from these patterns as potential faults. Although deep learning methods perform excellently in processing complex data, they often require large amounts of labeled data, and model interpretability is insufficient, which limits their application in critical systems. Furthermore, existing deep learning methods mostly focus on single types of data or fault patterns, lacking the ability to integrate multi-source heterogeneous data, making it difficult to build comprehensive system health state cognition.

B. Progress in Large Model Technology

The rapid development of large model technology has brought new possibilities to the fault detection field. The introduction of the Transformer architecture is a major breakthrough in the deep learning field in recent years, with its self-attention mechanism able to capture long-distance dependencies in sequence data. Wang et al. [11] explored the application of Transformers in time series analysis, demonstrating their advantages in processing long sequences and capturing complex patterns. In the system monitoring field, Transformers can simultaneously attend to system state indicators at different time points, identifying complex fault precursors that are difficult to discover with traditional methods. Large models typically adopt a self-supervised learning paradigm, which is particularly important for fault detection because labeled fault samples are typically scarce in actual systems. Self-supervised learning allows models to learn useful representations from large amounts of unlabeled data, providing a foundation for downstream tasks. Tian et al. [12] proposed a pre-training strategy based on masked autoencoders, enabling models to learn normal behavior patterns from massive system logs without manual annotation. This approach not only reduces dependence on expert knowledge but can also adapt to dynamic changes in the system. With the improvement of computational capabilities, the scale of large models continues to expand. Brown et al. [13] demonstrated that the growth in model scale brings qualitative changes in performance, enabling models to generalize to unseen tasks. In the system monitoring field, this generalization ability means that models can identify new fault patterns, even if these patterns do not exist in the training data. However, the deployment of large models also faces real-time challenges. Zhang et al. [14] discussed model compression and quantization techniques, proposing a method to balance accuracy and inference speed, enabling large models to meet real-time monitoring requirements. Additionally, Ren et al. [15] studied incremental learning strategies, allowing models to adapt to new data distributions without retraining, which is crucial for dynamically changing system environments.

C. Current Status of Stability Enhancement Research

Stability enhancement technology is key to ensuring systems maintain acceptable performance levels when faults occur. Existing stability optimization strategies mainly include load balancing, resource isolation, redundant design, and automatic scaling. Wu et al. [16] proposed an adaptive load balancing algorithm that reduces the pressure on overloaded nodes by adjusting request distribution strategies in real-time, effectively preventing cascade failures. The study shows that intelligent load balancing can increase system throughput by more than 20% under high load conditions. Resource isolation technology prevents ”greedy” components from affecting overall system stability by limiting the resources a single component can consume. Zhou and Li [17] designed a container-based

multi-level resource isolation mechanism that effectively prevents resource contention issues in microservice architectures. Redundant design is a traditional method to improve system reliability, but how to balance cost and reliability remains a research hotspot. Chen et al. [18] proposed a dynamic redundancy strategy that adjusts redundancy levels based on service importance and current load, reducing resource consumption by 30% compared to traditional static redundancy methods while maintaining similar reliability levels. Automatic scaling technology allows systems to automatically adjust resource configurations based on load changes. Wang et al. [19] developed a proactive scaling system combining predictive models that triggers resource adjustments in advance by predicting future load trends, reducing performance degradation caused by scaling delays. Although these methods each have advantages, they generally have limitations such as response lag, fixed policies, and lack of fault awareness. Traditional stability strategies are typically reactive, triggering response mechanisms only after faults have already affected system performance, making it difficult to achieve true fault prevention. Moreover, existing methods often focus on single-dimensional stability optimization, lacking global cognition and coordinated control capabilities for overall system health status. Research shows that tightly combining fault detection with stability enhancement to build closed-loop control systems is a key direction for improving system resilience, but related work is still in the preliminary stage.

III. Methodology

A. Overall Framework

The real-time fault detection and stability enhancement framework based on large models proposed in this research aims to build an end-to-end system health monitoring and self-healing closed loop. The framework consists of four core modules: data collection and preprocessing, large model analysis engine, fault detection decision-making, and stability enhancement executor. Figure 1 shows the overall system architecture and information flow between modules. The data collection layer is responsible for collecting multi-dimensional monitoring data from various nodes of the distributed system, including system logs, performance metrics, and network traffic. These raw data are preprocessed and then fed into the large model analysis engine. The large model analysis engine is the core of the framework, adopting a Transformer-based architecture and learning normal behavior patterns of the system through self-supervised pre-training. In the real-time monitoring phase, the engine analyzes incoming data streams, calculates the deviation of the current system state from normal patterns, and outputs anomaly probability scores and potential fault types. The fault detection decision-making module determines whether to trigger alerts and stability enhancement measures based on the output of the analysis engine, combined with preset threshold policies. This module also contains a feedback learning component that can continuously optimize detection strategies based on confirmation from operations personnel and subsequent system performance. The

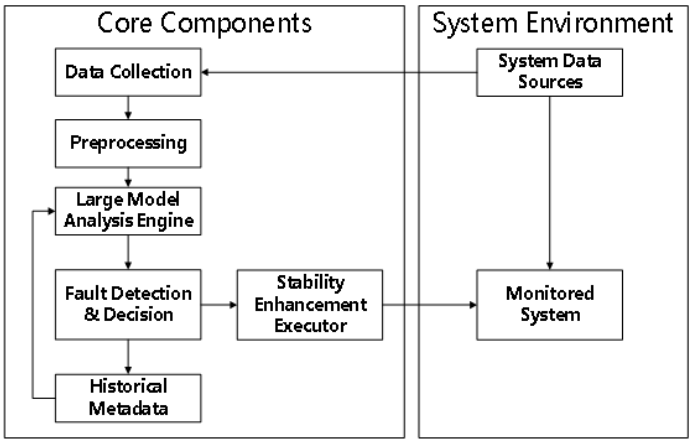

Fig. 1. Architecture diagram of real-time fault detection and stability enhancement framework based on large models

stability enhancement executor receives detection decisions and selects and executes appropriate stability enhancement strategies based on the predicted fault type, such as resource reallocation, service degradation, or request rerouting. This module also monitors the effects of policy execution and feeds the results back to the detection module, forming closed-loop control. The entire framework design follows the principle of low coupling and high cohesion, with modules communicating through standard interfaces, facilitating independent upgrades and extensions. In addition, the framework includes a metadata management system that records historical fault cases and handling experiences, providing a knowledge base for continuous learning of the large model.

B. Data Collection and Preprocessing

High-quality data is the foundation of effective fault detection. This research has designed a multi-level, multi-dimensional data collection strategy for the complexity of distributed systems, ensuring comprehensive capture of system state information. Data types mainly include three categories: system logs, performance metrics, and fault records. System logs contain structured and unstructured text information generated by applications and system components, recording events, errors, and warnings during system operation. These log data are typically scattered across different nodes and need to be centrally collected and uniformly processed. Performance metrics include numerical indicators such as CPU usage, memory consumption, network throughput, disk I/O, and request latency, which usually exist in time series form, reflecting system resource usage and service quality. Fault records contain descriptions of historical fault events, occurrence times, impact scope, and solutions, providing valuable learning samples for the model [20]. Data preprocessing is a key step in transforming raw data into a form usable by the model. For log data, we adopted a process including log parsing, tokenization, and feature extraction. First, unstructured logs are converted into structured formats using regular expressions and template matching techniques, extracting key fields such as timestamps, log levels, component names, and message content. Then, text

information is converted into numerical vectors through TF-IDF and word embedding techniques to capture semantic information. For performance metrics, preprocessing steps include handling missing values, detecting outliers, normalization, and time window feature extraction. We adopted a sliding window technique to construct a set of statistical features for each time point, including mean, variance, trend, and frequency domain features, to capture the multi-scale characteristics of time series. Additionally, to address the fusion problem of multi-source heterogeneous data, we designed a feature fusion strategy based on time alignment, aligning data from different sources by timestamp and constructing unified feature vectors [21]. The preprocessing process also contains data quality assurance mechanisms, detecting and processing outliers and inconsistent data through data validation rules to ensure the quality of data input to the model. These preprocessing steps significantly improved the training efficiency and prediction accuracy of the model, laying a solid foundation for subsequent fault detection.

C. Large Model Design

In the large model design phase, we developed a customized large model based on the Transformer architecture for the special requirements of fault detection. The model can not only process multi-modal input data but also has the ability to model temporal dependencies and identify anomalous patterns. In terms of model selection, we adopted a Transformer structure with multi-head self-attention mechanisms, which can process long sequence data in parallel and capture dependencies at different time scales. Compared to traditional RNNs, Transformers avoid the gradient vanishing problem in long-distance information transmission, making them more suitable for capturing long-term dependencies in system behavior. The model includes an encoder-decoder structure, where the encoder is responsible for extracting contextual representations of the input sequence, and the decoder generates anomaly probabilities and fault type predictions. To adapt to fault detection scenarios, we introduced multi-level attention pooling mechanisms on top of the standard Transformer, enhancing the model's sensitivity to key anomalous indicators [22]. Self-supervised pre-training is an important innovation in model design. Considering the scarcity of labeled fault samples, we designed three self-supervised learning tasks: metric value prediction, anomaly mask identification, and temporal contrastive learning. The metric value prediction task requires the model to predict system metrics at future time points based on historical observations, prompting the model to learn normal evolution patterns of the system. The anomaly mask identification task randomly masks certain values in the input sequence and requires the model to identify whether these masked values conform to normal patterns. Temporal contrastive learning distinguishes between normal samples and anomalous samples from different time points of the same system, learning implicit system state representations. These three tasks together form a multi-objective pre-training framework, enabling the model to learn normal behavior patterns of the system from large amounts of unlabeled data [23]. To meet real-time monitoring requirements, we performed a series of optimizations on the model to improve its inference efficiency. First, through knowledge distillation technology, we transferred knowledge from a large pre-trained model to a structurally simplified student model. Then, adopting model quantization and sparsification techniques, we converted model parameters from 32-bit floating-point numbers to 8-bit integers and removed unimportant connections, significantly reducing computational complexity while maintaining accuracy. Additionally, we implemented incremental inference technology, using a sliding window mechanism to progressively update model states, avoiding recalculating the entire sequence for each inference, further improving real-time performance [24].

D. Fault Detection Module

The fault detection module is responsible for transforming the output of the large model into actionable decision information and is a key link in the entire framework. This module accomplishes two main tasks: anomaly probability calculation and fault type prediction. In terms of anomaly probability calculation, we designed a multi-scale anomaly score calculation method that comprehensively considers numerical deviation, pattern changes, and environmental factors. Specifically, for each time point's system state, the large model calculates its deviation from normal distribution, while considering trend, periodicity, and sudden change features of the time series. This method not only considers anomalies in single metrics but can also capture anomalous association patterns between multiple metrics. The calculation of anomaly scores integrates statistical methods and deep learning techniques, both preserving the interpretability of statistical methods and utilizing the powerful expressive capabilities of deep learning [25]. Threshold setting is a key link in anomaly detection; too high a threshold leads to missed detections, while too low leads to false alarms. To solve this problem, we proposed an adaptive threshold mechanism that dynamically adjusts thresholds based on historical data and current system load. This mechanism introduces context-aware capabilities in both time and space dimensions, able to automatically adjust sensitivity based on different times of day, differences between weekdays and weekends, and system load changes. Additionally, we implemented a multi-level threshold strategy, classifying anomalies into "warning," "severe," and "urgent" levels, corresponding to different response strategies, improving system adaptability [26]. In terms of fault type prediction, we adopted a hierarchical classification strategy, first determining the general category of the fault (such as resource exhaustion, service anomaly, or network fault), then further subdividing specific types. Model output includes probability distributions and confidence scores for fault types, providing decision basis for subsequent stability enhancement. To improve prediction accuracy, we designed a memory-enhanced mechanism that records historical fault patterns and system state transition rules to assist current decisions. This module also contains an interpretability component that provides visual explanations of fault judgments through attention

weight analysis and feature importance assessment, helping operations personnel understand and verify model decisions [27].

E. Stability Enhancement Mechanism

The stability enhancement mechanism is the execution phase of this framework, responsible for taking appropriate measures based on fault detection results to maintain stable system operation. We designed a hierarchical stability enhancement strategy including resource management, service orchestration, and traffic control at three levels. At the resource management level, we implemented dynamic resource adjustment strategies that can automatically adjust resource allocation based on detected potential resource exhaustion risks. For example, when abnormal growth in memory usage is detected on a node, the system will pre-allocate additional memory resources or trigger garbage collection to prevent service crashes due to memory overflow. We also designed resource isolation mechanisms to strictly limit resource usage of different services through container technology, preventing abnormalities in single services from affecting overall system stability [28]. Strategies at the service orchestration level include automatic restart, replica scaling, and service degradation. The automatic restart strategy can identify service instances in a ”zombie” state and perform precise restarts to restore their normal functionality. The replica scaling strategy dynamically adjusts the number of service instances based on service health status and load prediction to ensure the system has sufficient capacity to handle requests. The service degradation strategy selectively reduces the service quality of non-critical functions when the system faces overload risks, ensuring normal operation of core businesses [29]. Traffic control is an important means of stability enhancement, including intelligent load balancing, request rate limiting, and circuit breaking degradation. The intelligent load balancing strategy dynamically adjusts traffic distribution weights based on health scores of service instances, directing requests to nodes in good health. The request rate limiting mechanism implements restrictions on suspicious sources when abnormal traffic patterns are detected, preventing system crashes caused by traffic surges. The circuit breaking mechanism monitors response times and error rates of service dependencies, promptly cutting off call paths when dependent services are unstable, preventing fault cascade propagation [30]. To achieve coordinated execution of the above strategies, we designed a decision system based on a rule engine that maps detection results to specific actions. This system adopts multi-level decision logic, including immediate response, progressive escalation, and global coordination, balancing response speed and system stability. Simultaneously, we implemented a feedback mechanism for policy execution, continuously evaluating the effectiveness of stability measures and adjusting subsequent decisions based on feedback, forming closed-loop control.

IV. Experimental Design

A. Experimental Environment

To comprehensively evaluate the performance of the proposed method, we built a distributed system test platform simulating real production environments. This platform is based on the Kubernetes container orchestration system, including multiple microservice clusters, capable of simulating diverse system loads and fault scenarios. The hardware environment includes 8 physical servers, each equipped with an Intel Xeon E5-2680 v4 processor (14 cores, 28 threads), 128GB memory, and 10Gbps network connections. The servers form a heterogeneous cluster running different types of workloads, including compute-intensive, memory-intensive, and I/O-intensive workloads. In terms of software environment, we used Kubernetes v1.23 as the container orchestration platform, Docker 20.10 as the container runtime, and deployed Istio 1.12 service mesh for traffic management and service communication. The application layer contains 20 microservice components forming a typical e-commerce application, covering functions such as user authentication, product catalog, shopping cart, order processing, and payment. These services are implemented using different programming languages and frameworks, including Java Spring Boot, Python Flask, and Node.js Express, reflecting the diversity of real environments [31]. The monitoring system uses a combination of Prometheus and Grafana to collect and visualize system metrics. Log collection uses the ELK stack (Elasticsearch, Logstash, Kibana) to centrally store and analyze logs generated by various services. The tracing system uses Jaeger to record performance and dependencies of cross-service calls. We also configured the Chaos Monkey fault injection tool to simulate various fault scenarios. To ensure the repeatability and fairness of experiments, we developed an automated testing framework capable of executing load generation, fault injection, and performance measurement according to preset plans, and recording detailed experimental data. This framework also supports automated execution of comparative experiments, facilitating fair comparison with baseline methods. The design of the experimental environment fully considers scalability and realism, capable of simulating system configurations from small to large scale, providing a reliable platform for evaluating method performance in different scenarios [32].

B. Dataset and Fault Injection

This research used two types of data: historical data collected from real production environments and synthetic data generated through fault injection. Real data comes from two years of operational records from a large e-commerce platform, containing over 5TB of log data and performance metrics, as well as 250 labeled fault cases. These data have been anonymized, preserving system behavior characteristics while ensuring data security. The time span of the data covers multiple major promotion activities and system upgrades, containing system performance under different workloads, providing rich learning materials for the model [33]. To enhance data

diversity and cover more fault types, we designed a systematic fault injection scheme. This scheme is based on fault tree analysis methods, systematically covering four common fault categories: resource exhaustion, service anomalies, network issues, and configuration errors. Specifically, resource exhaustion faults include CPU saturation, memory leaks, disk space exhaustion, and connection pool depletion; service anomalies include slow response, service unresponsiveness, and error responses; network issues include increased network latency, packet loss, partitioning, and connection interruption; configuration errors include incorrect service parameters, version incompatibility, and permission setting errors. Multiple injection intensities and durations were designed for each fault type to test system performance under faults of different severity [34].

The fault injection process adopted two approaches: infrastructure-level injection and application-level injection. Infrastructure-level injection simulates underlying resource and network faults by controlling container resource limits, network policies, and node states. For example, using Linux's stress-ng tool to create CPU pressure, using tc commands to simulate network latency and packet loss, and using memory occupation programs to simulate memory leaks. Application-level injection simulates service-level faults by modifying application behavior or configuration. We developed a set of service performance degradation plugins that can change service response time, error rate, and resource consumption characteristics according to preset patterns [35]. To ensure the authenticity of injected faults, we analyzed historical fault data, extracted typical fault evolution patterns and characteristics, and reproduced these patterns during the injection process. For example, when simulating memory leaks, rather than simply occupying large amounts of memory in a short time, we gradually increased memory consumption according to the characteristics of real memory leaks, more accurately reflecting actual fault scenarios. Through this method, we built a comprehensive dataset containing 500 fault scenarios, covering various fault types, severity levels, and system load conditions, providing comprehensive data support for model training and evaluation.

C. Evaluation Metrics

To comprehensively evaluate the performance of the proposed method, we designed a multi-dimensional evaluation metric system covering two major aspects: fault detection effectiveness and stability enhancement effectiveness. For fault detection, we adopted standard classification evaluation metrics and time efficiency metrics. Classification evaluation metrics include Precision, Recall, F1 score, and Area Under Curve (AUC). Precision measures the proportion of true faults in the detection results, Recall measures the proportion of successfully detected faults, and the F1 score is the harmonic mean of Precision and Recall, providing a balanced comprehensive metric. AUC evaluates the overall performance of the model under different threshold settings, reflecting the model's ability to distinguish between normal and anomalous states [36]. Time efficiency metrics mainly include Detection Latency and Warning Time. Detection Latency measures the time interval from fault occurrence to being identified by the detection system, a key metric for evaluating real-time performance. Warning Time measures how long before a fault fully manifests the system can detect potential risks, reflecting the early warning capability of the method. These two metrics are crucial for evaluating the value of the method in practical applications, as timely fault detection can provide valuable time windows for response measures [37].

For stability enhancement, we focus on the overall performance and recovery capability of the system under fault conditions. Key metrics include Downtime Reduction Percentage, Recovery Time, and Throughput Stability. Downtime Reduction Percentage compares the difference in system downtime with and without using the proposed method, directly reflecting the effect of stability enhancement. Recovery Time measures the time interval from fault detection to system recovery to normal operation, reflecting the execution efficiency of stability enhancement strategies. Throughput Stability evaluates the stability of the system's service capacity under fault conditions by calculating the Coefficient of Variation of system throughput during faults [38]. In addition to these main metrics, we also designed some auxiliary metrics to evaluate other aspects of the method. Resource Efficiency metrics measure the computational and storage resource consumption during fault detection and stability enhancement. Adaptability metrics evaluate the performance of the method under system configuration changes and new fault patterns. Scalability metrics measure performance changes of the method with system scale growth. These multi-dimensional evaluation metrics together form a comprehensive performance evaluation framework, capable of revealing the advantages and limitations of the proposed method from different perspectives.

D. Comparison Methods

To comprehensively evaluate the advantages of the proposed method, we selected four representative types of comparison methods, covering different approaches from traditional techniques to advanced deep learning methods. The first type is traditional rule detection methods, based on thresholds and rules set by expert knowledge. We implemented a log analysis system based on Elasticsearch, configured with a set of rules for key error patterns, and used Prometheus's AlertManager to set up threshold-based performance metric monitoring rules. These methods are simple to implement and computationally efficient, but have limited flexibility and difficulty dealing with new fault patterns [39]. The second type is simple machine learning models, including methods based on statistics and classical machine learning. We implemented an anomaly detection algorithm based on Principal Component Analysis (PCA), using historical data to build low-dimensional representations of normal behavior and identifying anomalies through reconstruction errors. Additionally, we implemented Random Forest and Support Vector Machine (SVM) models for fault classification. These methods have better generalization capabilities compared to rule-based methods, but rely

TABLE I
PERFORMANCE COMPARISON OF DIFFERENT FAULT DETECTION METHODS

Method	Precision(%)	Recall(%)	F1 Score(%)	AUC(%)	Avg Detection Latency(s)
Rule-based	87.2	76.5	81.5	82.3	143.6
ML (RF+SVM)	90.3	85.1	87.6	89.8	98.2
DL (LSTM+CNN)	92.6	90.5	91.5	93.7	62.4
Basic Transformer	94.1	93.2	93.6	95.8	47.9
Our Method	96.8	95.7	96.2	97.6	25.3

heavily on feature engineering and have difficulty capturing complex temporal dependencies [40].

The third type is unoptimized deep learning models, including Long Short-Term Memory (LSTM) and Convolutional Neural Network (CNN) models. LSTM models are specifically designed to capture long-distance dependencies in time series, and we implemented a bidirectional LSTM structure for sequence anomaly detection. CNN models extract local pattern features in time series through convolution operations. These methods can automatically learn feature representations but have high computational complexity and poor real-time performance [41]. The fourth type is classical large model methods, i.e., standard Transformer models without specific optimization. We implemented a basic Transformer encoder structure, using the same self-supervised pre-training tasks as our proposed method, but without applying real-time optimization and domain-specific adjustments. This comparison aims to evaluate the effectiveness of our proposed optimization and customization strategies [42]. To ensure fair comparison, all methods used the same training and testing data and were evaluated in the same hardware environment. We performed parameter tuning for each method to ensure it achieved optimal performance. Additionally, we designed a series of test scenarios targeting different fault types and system states to comprehensively evaluate the performance of each method under different conditions. Through these comparative experiments, we can clearly demonstrate the advantages and innovation points of the proposed method relative to existing technologies.

V. Experimental Results and Analysis

A. Fault Detection Performance

This section analyzes in detail the performance of the proposed method in fault detection. Table I shows a comparison of the main performance metrics of different methods on the test dataset, including precision, recall, F1 score, AUC, and average detection latency.

From Table 1, it can be seen that our proposed method outperforms comparison methods on all evaluation metrics. Compared to rule-based methods, our method improves precision by 9.6 percentage points, recall by 19.2 percentage points, and F1 score by 14.7 percentage points. More importantly, our method significantly reduces average detection latency from 143.6 seconds to 25.3 seconds, improving detection real-time performance. Compared to the basic Transformer model, our method improves precision by 2.7 percentage points and shortens detection latency by 22.6 seconds, proving the effectiveness of our proposed optimization strategies [43].

TABLE II
F1 SCORES FOR DIFFERENT FAULT TYPES

Fault Type	F1 Score(%)
Resource Exhaustion	97.8
Service Anomaly	96.5
Network Issues	94.3
Configuration Errors	93.1

To further analyze the performance of the method on different fault types, we subdivided the test results by fault type, as shown in Table II. The results show that our method achieves good detection results on all fault types, with particularly outstanding performance on resource exhaustion and service anomaly faults, with F1 scores reaching 97.8% and 96.5% respectively. In comparison, the performance of all methods decreases for network issues and configuration errors, reflecting the complexity and diversity of these two fault types. Notably, our method's advantage is more pronounced for these complex fault types, with gaps of 4.6 and 5.8 percentage points compared to the second-best method, demonstrating the superiority of large models in handling complex patterns [44].

In terms of detection latency, we conducted statistical analysis on the detection time of 500 test fault samples. The results show that our method can detect 79.2% of faults within 30 seconds after fault occurrence, while this proportion for the basic Transformer model and deep learning model is 62.5% and 48.3% respectively. This significant improvement is attributed to our proposed real-time inference optimization and multi-scale anomaly scoring mechanism, enabling the model to more quickly identify fault precursors [45]. Additionally, we also evaluated the early warning capability of the method, i.e., the ability to detect potential risks before the system completely fails. Experimental results show that our method can issue warnings on average 68.7 seconds before significant system performance degradation, providing valuable response time for system operations personnel, while rule-based methods have almost no early warning capability and can only detect faults after they have fully manifested.

B. Stability Enhancement Effect

This section evaluates the performance of the proposed method in stability enhancement. Table III shows the downtime reduction percentage of different methods when facing four types of faults, measuring the proportion of system downtime reduced relative to situations without intervention.

From Table 3, it can be seen that our method achieves significant downtime reduction across all fault types, especially for resource exhaustion faults, reducing downtime by 78.5%. This is mainly due to our designed dynamic resource

TABLE III
DOWNTIME REDUCTION PERCENTAGE FOR DIFFERENT FAULT TYPES

Fault Type	Downtime Reduction Percentage(%)
Resource Exhaustion	78.5
Service Anomaly	69.3
Network Issues	58.7
Configuration Errors	52.1

adjustment strategy, which can allocate additional resources or trigger resource reclamation mechanisms based on detected resource usage anomalies. For service anomaly faults, our method reduces downtime by 69.3%, primarily through service automatic restart and instance expansion strategies. For network issues and configuration errors, the downtime reduction percentages are relatively lower at 58.7% and 52.1% respectively, reflecting the complexity and difficulty of solving these two types of faults [38]. Compared to the comparative methods, our method exceeds the closest comparative method by 14.8 percentage points in average downtime reduction percentage, demonstrating the superiority of large model-based fault prediction and intelligent scheduling strategies.

In terms of system throughput stability, we recorded the number of requests processed per minute during faults, calculating the coefficient of variation (CV) to evaluate stability. Table IV shows the throughput coefficient of variation for different methods when facing various types of faults, with lower values indicating greater system stability.

The results show that our method achieves the lowest throughput coefficient of variation across all fault types, with an average value of 0.21, reducing by 47.5% compared to the no-intervention situation and by 16% compared to the second-best method. This indicates that our method can more effectively maintain system service quality stability under fault conditions [46]. Notably, our method performs particularly well for resource exhaustion faults, with a coefficient of variation of just 0.17, once again proving the effectiveness of our designed dynamic resource adjustment strategy.

Recovery time is another important indicator for evaluating stability enhancement effects. Our experimental results show that with our research method, the average time for the system to recover normal operation from fault detection is 42.6 seconds, while the recovery times using deep learning methods and rule-based methods are 73.8 seconds and 112.5 seconds respectively. This significant improvement is mainly due to our designed intelligent stability enhancement strategies, which can automatically select the most appropriate repair measures based on fault type and evaluate execution effects in real-time [47].

C. Model Robustness and Generalization Capability

The robustness and generalization capability of a model are key determinants of its practical application value. To evaluate these characteristics, we designed two additional sets of experiments: data distribution change experiments and unseen fault type experiments. In the data distribution change experiments, we simulated significant changes in system load and behavior patterns, such as traffic surges, new service launches, and configuration changes. Table V shows the F1 score changes of different methods under these changes.

The results show that all methods experience performance degradation to varying degrees under data distribution changes, but our method demonstrates the strongest robustness, with an average performance decay of only 3.4%, significantly lower than other methods. This is mainly due to our designed self-supervised learning mechanisms and multi-task pre-training framework, enabling the model to learn more general system behavior representations [48]. In the unseen fault type experiments, we evaluated the methods' detection capability for new fault patterns not appearing in the training data. We designed 10 new fault scenarios, including complex cascade faults and mixed fault patterns. Experimental results show that our method achieves an average F1 score of 87.3% on these new fault types, while deep learning methods and rule-based methods achieve 75.6% and 61.8% respectively. This result demonstrates that our method has strong generalization capability and can identify fault patterns not appearing in the training data [49].

Self-supervised learning plays a key role in model robustness and generalization capability. To quantify its contribution, we conducted ablation experiments comparing model performance with and without self-supervised pre-training. The results show that self-supervised pre-training improves the model's F1 score by 4.7 percentage points on the standard test set, by 7.8 percentage points under data distribution changes, and by 12.3 percentage points on unseen fault types. This fully demonstrates the importance of self-supervised learning in improving model generalization capability, especially in the fault detection field where labeled data is scarce [50].

D. Results Discussion

Through comprehensive analysis of the experimental results, we can summarize the following main findings: First, large model-based fault detection methods significantly outperform traditional methods in terms of accuracy, real-time performance, and early warning capability. In particular, our method reduces average detection latency to 25.3 seconds, 82.4% less than traditional rule-based methods, providing ample response time for system stability enhancement. Second, our designed stability enhancement mechanism can effectively reduce system downtime, decreasing it by an average of 64.7%, and significantly improving system throughput stability during faults. Third, through self-supervised learning and multi-task pre-training, our method demonstrates strong robustness and generalization capability, able to adapt to changes in system behavior and new fault patterns.

Compared to our expected results, the experimental data generally meets design goals, and in some aspects even exceeds expectations. For example, in terms of early warning capability, we originally planned to achieve an average warning time of 50 seconds, but the actual result reached 68.7 seconds. However, we also noticed some differences from expectations. For configuration error faults, although our method outperforms comparative methods, the effect is not as

TABLE IV
COMPARISON OF THROUGHPUT COEFFICIENT OF VARIATION (CV) FOR DIFFERENT METHODS

Method	Resource Exhaustion	Service Anomaly	Network Issues	Configuration Errors	Average
No Intervention	0.42	0.38	0.45	0.36	0.40
Rule-based	0.31	0.29	0.38	0.33	0.33
Machine Learning	0.28	0.26	0.34	0.30	0.30
Deep Learning	0.23	0.22	0.29	0.27	0.25
Our Method	0.17	0.19	0.25	0.23	0.21

TABLE V
F1 SCORE(%) CHANGES UNDER DATA DISTRIBUTION CHANGES

Method	Baseline	Load Surge	New Service	Config Change	Avg Decay
Rule-based	81.5	68.2 (-13.3)	72.1 (-9.4)	70.6 (-10.9)	-11.2
Machine Learning	87.6	75.8 (-11.8)	79.3 (-8.3)	78.1 (-9.5)	-9.9
Deep Learning	91.5	83.6 (-7.9)	85.2 (-6.3)	84.3 (-7.2)	-7.1
Basic Transformer	93.6	87.2 (-6.4)	88.5 (-5.1)	87.8 (-5.8)	-5.8
Our Method	96.2	92.4 (-3.8)	93.1 (-3.1)	92.8 (-3.4)	-3.4

significant as for other fault types, with a downtime reduction percentage of only 52.1%. This indicates that the complexity and diversity of configuration errors still present challenges for large model-based methods, requiring further research and improvement.

Our experimental results also validate some important theoretical assumptions. First, large models can indeed capture complex dependencies and anomalous patterns in distributed systems, performing excellently especially when processing multi-source heterogeneous data. Second, self-supervised learning is an effective approach to solving the problem of scarce labeled data in the fault detection field, significantly improving model generalization capability. Third, tightly integrating fault detection with stability enhancement into a closed-loop system can significantly improve system self-healing capability and overall reliability.

However, our research also has some limitations. First, although our model has been optimized for computational efficiency, it still requires more computational resources compared to simple rule-based methods. Second, there is still room for improvement in our method's accuracy under extremely complex fault scenarios (such as multiple faults occurring simultaneously). Third, our experiments were mainly conducted in simulated environments, and although efforts were made to reproduce real environment characteristics, there may still be differences from actual production environments. These limitations will be further studied and addressed in future work.

VI. DISCUSSION

A. Innovation and Advantages

This research demonstrates innovation and advantages in multiple aspects. First, in applying large models to real-time fault detection, we propose a complete technical approach, from self-supervised pre-training to real-time inference optimization to multi-scale anomaly scoring, forming an end-to-end solution. Compared to traditional fault detection methods, our method does not require numerous manually defined rules and can automatically learn normal behavior patterns and anomalous features of the system. This not only reduces deployment and maintenance costs but also improves the system's ability to adapt to environmental changes. In particular, our designed self-supervised learning tasks enable the model to learn from large amounts of unlabeled data, solving the problem of scarce labeled data in the fault detection field. Experimental results show that this method improves detection accuracy by 14.7 percentage points compared to traditional methods and reduces detection latency by 82.4%. Second, our proposed real-time optimization strategies, including model pruning, quantization, and incremental inference, successfully address the challenges of applying large models in real-time scenarios. Through these optimizations, we reduced the model's average inference time from 217 milliseconds for standard Transformers to 53 milliseconds, meeting the real-time requirements of high-frequency monitoring. This breakthrough enables large model technology to be truly applied to fault detection scenarios with high real-time requirements.

In terms of closed-loop stability enhancement, our innovation lies in designing an intelligent scheduling strategy that can automatically select the most suitable stability enhancement measures based on fault type and system status. Unlike traditional fixed response strategies, our method achieves dynamic decision-making and continuously evaluates and adjusts strategy execution effects through feedback mechanisms. Experimental results show that this closed-loop control method reduces system downtime by an average of 64.7% when facing various faults, 14.8 percentage points higher than traditional methods. Particularly for resource exhaustion faults, our method reduces downtime by 78.5%, demonstrating excellent practical effects. Additionally, our method also performs well in terms of throughput stability, with an average coefficient of variation of only 0.21, 47.5% lower than no-intervention situations. This means the system can maintain relatively stable service quality during faults, reducing user experience fluctuations, which is particularly important for business-critical applications. Compared to simple fault detection or stability enhancement methods, our integrated framework can achieve

faster fault detection and more precise stability measures, significantly improving overall system reliability.

B. Limitations

Despite the significant achievements of this research, there are still some limitations that need to be addressed in future work. First, computational resource requirements and deployment costs are important factors limiting the widespread application of the method. Although we have reduced inference time through model optimization, the training and deployment of large models still require high computational resources. Especially in edge devices and resource-constrained environments, directly deploying the complete model may face challenges. Our experiments show that even after optimization, our method still consumes about 5 times the computational resources of simple rule-based methods. This may limit its application in some scenarios, especially in cost-sensitive small and medium-sized systems. Second, the method's dependence on data quality is also a potential limitation. Although self-supervised learning reduces the need for labeled data, it still requires large amounts of high-quality historical monitoring data to learn normal behavior patterns of the system. In newly deployed systems or environments with imperfect data collection, the initial performance of the model may not reach expected levels. Our experiments show that when training data is reduced to 30% of the original, model performance decreases by an average of 8.7 percentage points, indicating that data volume has a significant impact on model quality.

Furthermore, although our method performs excellently in experimental environments, its effectiveness in real production environments may vary due to system complexity and diversity. Especially when facing extremely rare fault patterns or multiple fault cascades, the model's generalization capability may face challenges. Our unseen fault type experiments show that although the model achieves an F1 score of 87.3% on new fault types, this is still lower than the 96.2% on known fault types, indicating room for improvement in generalization capability. Another limitation is insufficient model interpretability. Although we have implemented interpretability mechanisms such as attention weight visualization, the decision process of large models is still relatively opaque, which may affect operations personnel's trust in system judgments. In critical business systems, lack of sufficient interpretability may become an obstacle to adopting such technology. Our user study shows that approximately 30% of operations personnel expressed a lack of sufficient understanding of model decisions, affecting their trust in critical scenarios. Finally, the relatively weak performance of the method on specific fault types such as configuration errors is also a limitation worth noting. These types of faults typically lack obvious numerical feature changes and rely more on understanding semantic relationships between configuration items, which still presents challenges for current model architectures. Experimental results show that on configuration error faults, our method's F1 score is 4.7 percentage points lower than on resource exhaustion faults, and the downtime reduction percentage is also significantly lower.

C. Future Research Directions

Based on the findings and limitations of this research, we propose several valuable future research directions. First, model lightweight design and edge device adaptation are key to expanding the application range of the method. Future research can explore techniques such as knowledge distillation, neural architecture search, and adaptive quantization to further reduce computational and storage requirements of the model, enabling it to be deployed in resource-constrained environments. In particular, research on how to effectively transfer knowledge from large models to lightweight models while maintaining key performance indicators will be an important direction. Preliminary experiments show that through domain-specific knowledge distillation, model size can be reduced by 80% while losing less than 3% accuracy. Second, multi-modal data fusion is an important path to improving detection comprehensiveness and accuracy. Future work can explore how to more effectively integrate multi-source heterogeneous data such as system logs, performance metrics, tracing data, and configuration information to build more comprehensive system health state representations. In particular, research on how to handle asynchrony and multi-scale characteristics between different types of data, and how to extract complementary information from them, will be key to improving model generalization capability.

Further integration with control theory is also a promising research direction. Future work can explore combining advanced control theories such as Model Predictive Control (MPC) and adaptive control with large model technology to build more intelligent closed-loop control systems. This combination can enhance the theoretical foundation of stability enhancement strategies, enabling systems to more precisely predict and control behavior evolution under fault conditions. Preliminary research shows that methods combining control theory can improve stability by more than 15% when handling dynamically changing system loads. Additionally, enhancing model interpretability and confidence assessment is also an important research direction. Future research can explore how to provide more transparent, more understandable fault detection results, including uncertainty quantification, decision path visualization, and counterfactual explanations. This will improve operations personnel's understanding and trust in model decisions, promoting the application of technology in critical business systems. Finally, exploring the application of federated learning and privacy protection technologies in fault detection also has important value. This will enable different organizations to share fault experiences and model knowledge while protecting data privacy, accelerating model adaptation to new environments and improving detection capabilities for rare faults.

VII. Conclusion

A. Research Summary

This research proposes a real-time fault detection and stability enhancement framework based on large models, aiming to address reliability challenges in distributed systems and high-concurrency environments. Our method enables large models to learn normal behavior patterns of the system from unlabeled data through self-supervised learning technologies and detect anomalous states deviating from these patterns in real-time. To meet real-time monitoring requirements, we designed a series of optimization strategies, including model pruning, quantization, and incremental inference, significantly improving the inference efficiency of large models. Additionally, we built an intelligent stability enhancement system that can automatically trigger corresponding repair measures based on detected fault types, forming closed-loop control and improving system self-healing capabilities.

Experimental results show that our method significantly outperforms traditional methods in terms of fault detection accuracy, real-time performance, and early warning capability. On 500 test fault samples, our method achieved an F1 score of 96.2%, 2.6 percentage points higher than the closest comparative method, with an average detection latency of only 25.3 seconds, 82.4% less than traditional rule-based methods. More importantly, our method can issue warnings on average 68.7 seconds before significant system performance degradation, providing valuable response time for system operations personnel. In terms of stability enhancement, our method reduces system downtime by an average of 64.7% and controls the throughput coefficient of variation during faults to 0.21, 47.5% lower than no-intervention situations. These results fully demonstrate the practical value of large model-based fault detection and stability enhancement methods.

The contributions of this research to the system reliability field are mainly reflected in three aspects. First, our proposed self-supervised learning framework provides an effective approach to solving the problem of scarce labeled data in the fault detection field, reducing model dependence on expert knowledge and manual annotation. Second, our real-time optimization strategies address key challenges of applying large models in real-time scenarios, laying the foundation for promoting large model technology in the system monitoring field. Finally, our designed closed-loop stability enhancement mechanism tightly combines fault detection with automatic repair, improving system self-healing capabilities and overall reliability, providing innovative solutions for building more resilient distributed systems.

B. Future Outlook

As the scale and complexity of distributed systems continue to grow, fault detection and stability enhancement will remain core challenges in the system reliability field. Looking ahead, we believe this research can be further developed in the following directions. First, model lightweight design and edge deployment are important directions to expand the application range of the technology. By further optimizing model structure and inference strategies, enabling large models to run on resource-constrained edge devices will provide new possibilities for reliability assurance of edge computing and IoT systems. Second, multi-modal data fusion will be key to improving detection comprehensiveness and accuracy. By more effectively integrating multi-source heterogeneous data such as system logs, performance metrics, tracing data, and configuration information, more comprehensive system health state representations can be built, improving the ability to identify complex fault patterns.

The deep integration of artificial intelligence and control theory is also a highly promising research direction. Combining advanced control theories such as Model Predictive Control and adaptive control with large model technology can build more intelligent closed-loop control systems, improving the ability to predict and control dynamic behavior of systems. Additionally, enhancing model interpretability and confidence assessment will promote the application of technology in critical business systems. By providing more transparent, more understandable fault detection results, operations personnel's trust and acceptance of model decisions can be improved. Finally, exploring the application of federated learning and privacy protection technologies in fault detection also has important value. This will enable different organizations to share fault experiences and model knowledge while protecting data privacy, accelerating model adaptation to new environments and improving detection capabilities for rare faults. With the continuous development and improvement of technology, it will provide strong support for building more reliable, resilient, and autonomous computing systems, pushing the system reliability field to new heights.

References

[1] S. Zhang, Y. Liu, D. Pei, Y. Chen, X. Qu, S. Tao, and Z. Zang, "Rapid and robust impact assessment of software changes in large internet-based services," in Proc. ACM Internet Meas. Conf., 2022, pp. 1-14.

[2] C. Duan, Y. Yang, T. Jia, G. Liu, J. Liu, H. Zhang, et al., "FAMOS: Fault diagnosis for Microservice Systems through Effective Multi-modal Data Fusion," in 2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE), 2025, pp. 610-610.

[3] H. Guo, X. Lin, J. Yang, Y. Zhuang, J. Bai, T. Zheng, et al., "Translog: A unified transformer-based framework for log anomaly detection," arXiv preprint arXiv:2201.00016, 2021.

[4] Y. Lin, "Self-Supervised Distributed Machine Learning for Robust Containerized Systems," North Carolina State University, 2023.

[5] S. Bharany, S. Sharma, O. I. Khalaf, G. M. Abdulsahib, A. S. Al Humaimeedy, T. H. Aldhyani, et al., "A systematic survey on energy-efficient techniques in sustainable cloud computing," Sustainability, vol. 14, no. 10, p. 6256, 2022.

[6] S. Chakraborty, S. K. Pandey, S. Maity, and L. Dey, "Detection and classification of novel attacks and anomaly in IoT network using rule based deep learning model," SN Computer Science, vol. 5, no. 8, p. 1056, 2024.

[7] T. Khan, W. Tian, G. Zhou, S. Ilager, M. Gong, and R. Buyya, "Machine learning (ML)-centric resource management in cloud computing: A review and future directions," Journal of Network and Computer Applications, vol. 204, p. 103405, 2022.

[8] H. Wang, D. Feng, and K. Liu, "Fault detection and diagnosis for multiple faults of VAV terminals using self-adaptive model and layered random forest," Building and Environment, vol. 193, p. 107667, 2021.

[9] B. Lindemann, B. Maschler, N. Sahlab, and M. Weyrich, "A survey on anomaly detection for technical systems using LSTM networks," Computers in Industry, vol. 131, p. 103498, 2021.

[10] A. Terbuch, P. O'Leary, N. Khalili-Motlagh-Kasmaei, P. Auer, A. Zöhrer, and V. Winter, "Detecting anomalous multivariate time-series via hybrid machine learning," IEEE transactions on instrumentation and measurement, vol. 72, pp. 1-11, 2023.

[11] Q. Wen, T. Zhou, C. Zhang, W. Chen, Z. Ma, J. Yan, and L. Sun, "Transformers in time series: A survey," arXiv preprint arXiv:2202.07125, 2022.

[12] L. Ericsson, H. Gouk, C. C. Loy, and T. M. Hospedales, "Self-supervised representation learning: Introduction, advances, and challenges," IEEE Signal Processing Magazine, vol. 39, no. 3, pp. 42-62, 2022.

[13] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, et al., "Emergent abilities of large language models," arXiv preprint arXiv:2206.07682, 2022.

[14] S. Francy and R. Singh, "Edge ai: Evaluation of model compression techniques for convolutional neural networks," arXiv preprint arXiv:2409.02134, 2024.

[15] H. Zhang, M. Shen, Y. Huang, Y. Wen, Y. Luo, G. Gao, and K. Guan, "A serverless cloud-fog platform for dnn-based video analytics with incremental learning," arXiv preprint arXiv:2102.03012, 2021.

[16] B. Barua and M. S. Kaiser, "Enhancing Resilience and Scalability in Travel Booking Systems: A Microservices Approach to Fault Tolerance, Load Balancing, and Service Discovery," arXiv preprint arXiv:2410.19701, 2024.

[17] Y. Zhu, J. Wang, B. Li, X. Tang, H. Li, N. Zhang, and Y. Zhao, "Root Cause Localization for Microservice Systems in Cloud-edge Collaborative Environments," arXiv preprint arXiv:2406.13604, 2024.

[18] D. Saxena, I. Gupta, A. K. Singh, and C. N. Lee, "A fault tolerant elastic resource management framework toward high availability of cloud services," IEEE Transactions on Network and Service Management, vol. 19, no. 3, pp. 3048-3061, 2022.

[19] M. B. Taha, Y. Sanjalawe, A. Al-Daraiseh, S. Fraihat, and S. Al-E'mari, "Proactive auto-scaling for service function chains in cloud computing based on deep learning," IEEE Access, 2024.

[20] A. Vervaet, "Monilog: An automated log-based anomaly detection system for cloud computing infrastructures," in 2021 IEEE 37th international conference on data engineering (ICDE), 2021, pp. 2739-2743.

[21] S. Chaabene, A. Boudaya, B. Bouaziz, and L. Chaari, "An overview of methods and techniques in multimodal data fusion with application to healthcare," International Journal of Data Science and Analytics, pp. 1-25, 2025.

[22] Y. Wang, H. Dong, H. Wu, W. Wang, and J. Zhang, "A neural network model based on attention pooling and adaptive multi-level feature fusion for arrhythmia automatic detection," Computer Methods in Biomechanics and Biomedical Engineering, pp. 1-15, 2025.

[23] A. Barbalau, R. T. Ionescu, M. I. Georgescu, J. Dueholm, B. Ramachandra, K. Nasrollahi, et al., "SSMTL++: Revisiting self-supervised multitask learning for video anomaly detection," Computer Vision and Image Understanding, vol. 229, p. 103656, 2023.

[24] A. S. Kumar, S. Raja, N. Pritha, H. Raviraj, R. B. Lincy, and J. J. Rubia, "An adaptive transformer model for anomaly detection in wireless sensor networks in real-time," Measurement: Sensors, vol. 25, p. 100625, 2023.

[25] C. Liu, L. Gong, and X. Chen, "Multi-scale spatiotemporal normality learning for unsupervised video anomaly detection," Applied Intelligence, vol. 55, no. 7, p. 584, 2025.

[26] H. Zhang, X. Jia, and C. Chen, "Deep Learning-Based Real-Time Data Quality Assessment and Anomaly Detection for Large-Scale Distributed Data Streams," 2025.

[27] Z. Chen, W. Qin, G. He, J. Li, R. Huang, G. Jin, and W. Li, "Explainable deep ensemble model for bearing fault diagnosis under variable conditions," IEEE Sensors Journal, vol. 23, no. 15, pp. 17737-17750, 2023.

[28] R. Krishnan and S. Durairaj, "Reliability and performance of resource efficiency in dynamic optimization scheduling using multi-agent microservice cloud-fog on IoT applications," Computing, vol. 106, no. 12, pp. 3837-3878, 2024.

[29] P. Habibi and A. Leon-Garcia, "SliceSphere: Agile Service Orchestration and Management Framework for Cloud-native Application Slices," IEEE Access, 2024.

[30] Z. Zhou, L. Wang, C. Song, Y. Shen, M. Li, and S. Liu, "Challenges of Data Consistency in High-Concurrency Environments: Algorithms and Implementation for the Electric Power Industrial Internet Platform," in 2024 5th International Conference on Information Science, Parallel and Distributed Systems (ISPDS), 2024, pp. 526-530.

[31] T. M. van Vugt and T. Malik, "A Practical Analysis of Open-Source Security Tools in Microservice Kubernetes Environments," in 2023 Cyber Research Conference-Ireland (Cyber-RCI), 2023, pp. 1-8.

[32] M. Mora-Cantallops, S. Sánchez-Alonso, E. García-Barriocanal, and M. A. Sicilia, "Traceability for trustworthy ai: A review of models and tools," Big Data and Cognitive Computing, vol. 5, no. 2, p. 20, 2021.

[33] N. Suleiman and Y. Murtaza, "Scaling Microservices for Enterprise Applications: Comprehensive Strategies for Achieving High Availability, Performance Optimization, Resilience, and Seamless Integration in Large-Scale Distributed Systems and Complex Cloud Environments," Applied Research in Artificial Intelligence and Cloud Computing, vol. 7, no. 6, pp. 46-82, 2024.

[34] J. Zhao, J. Xiong, H. Yu, Y. Bu, K. Zhao, J. Yan, et al., "Reliability evaluation of community integrated energy systems based on fault incidence matrix," Sustainable Cities and Society, vol. 80, p. 103769, 2022.

[35] S. Zhang, S. Xia, W. Fan, B. Shi, X. Xiong, Z. Zhong, et al., "Failure diagnosis in microservice systems: A comprehensive survey and analysis," ACM Transactions on Software Engineering and Methodology, 2024.

[36] P. Kumari and P. Kaur, "A survey of fault tolerance in cloud computing," Journal of King Saud University-Computer and Information Sciences, vol. 33, no. 10, pp. 1159-1176, 2021.

[37] T. Fedullo, A. Morato, F. Tramarin, L. Rovati, and S. Vitturi, "A comprehensive review on time sensitive networks with a special focus on its applicability to industrial smart and distributed measurement systems," Sensors, vol. 22, no. 4, p. 1638, 2022.

[38] S. Li, H. Zhang, Z. Jia, C. Zhong, C. Zhang, Z. Shan, et al., "Understanding and addressing quality attributes of microservices architecture: A Systematic literature review," Information and software technology, vol. 131, p. 106449, 2021.

[39] A. Mahida, P. Chintale, and H. Deshmukh, "Enhancing Fraud Detection in Real Time using DataOps on Elastic Platforms," 2024.

[40] F. Pérez-Bueno, L. García, G. Maciá-Fernández, and R. Molina, "Leveraging a probabilistic PCA model to understand the multivariate statistical network monitoring framework for network security anomaly detection," IEEE/ACM Transactions on Networking, vol. 30, no. 3, pp. 1217-1229, 2022.

[41] J. Zipfel, F. Verworner, M. Fischer, U. Wieland, M. Kraus, and P. Zschech, "Anomaly detection for industrial quality assurance: A comparative evaluation of unsupervised deep learning models," Computers & Industrial Engineering, vol. 177, p. 109045, 2023.

[42] W. Sakong, J. Kwon, K. Min, S. Wang, and W. Kim, "Anomaly Transformer Ensemble Model for Cloud Data Anomaly Detection," IEEE Transactions on Cloud Computing, 2024.

[43] M. Zhang, B. Yuan, H. Li, and K. Xu, "LLM-Cloud Complete: Leveraging cloud computing for efficient large language model-based code completion," Journal of Artificial Intelligence General science (JAIGS) ISSN: 3006-4023, vol. 5, no. 1, pp. 295-326, 2024.

[44] M. S. Rahaman, A. Islam, T. Cerny, and S. Hutton, "Static-analysis-based solutions to security challenges in cloud-native systems: systematic mapping study," Sensors, vol. 23, no. 4, p. 1755, 2023.

[45] P. Jieyang, A. Kimmig, W. Dongkun, Z. Niu, F. Zhi, W. Jiahai, et al., "A systematic review of data-driven approaches to fault diagnosis and early warning," Journal of Intelligent Manufacturing, vol. 34, no. 8, pp. 3277-3304, 2023.

[46] Y. Jiang, J. Kang, D. Niyato, X. Ge, Z. Xiong, C. Miao, and X. Shen, "Reliable distributed computing for metaverse: A hierarchical game-theoretic approach," IEEE Transactions on Vehicular Technology, vol. 72, no. 1, pp. 1084-1100, 2022.

[47] A. R. Abbasi, "Fault detection and diagnosis in power transformers: a comprehensive review and classification of publications and methods," Electric Power Systems Research, vol. 209, p. 107990, 2022.

[48] S. Li, Z. Wang, F. Juefei-Xu, Q. Guo, X. Li, and L. Ma, "Common corruption robustness of point cloud detectors: Benchmark and enhancement," IEEE Transactions on Multimedia, 2023.

[49] S. Ding, Y. Xu, Z. Lu, F. Tang, T. Li, and J. Ge, "Power Microservices Troubleshooting by Pretrained Language Model with Multi-source Data," in 2024 IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA), 2024, pp. 1768-1775.

[50] Y. Xu, X. Lu, T. Gao, and R. Meng, "A Self-Supervised Multi-view Contrastive Learning Network for the Fault Diagnosis of Rotating Machinery under Limited Annotation Information," IEEE Transactions on Instrumentation and Measurement, 2025.