Sudan University of Science and Technology

College of Engineering

School of Electronic Engineering

An Enhanced Intelligent Server Monitoring System for Predictive Fault Detection and Triage Prioritization

A Research Submitted in Partial Fulfilment of the Requirements for the Degree of B.Sc. (Honors) in Electronics Engineering

Prepared By:

1. Ahmed Abdulmonim Mohamed Ibrahim
2. Alrahma Fadol Allah Gomaa Alnour
3. Marwa Yousuf Mohamed
4. Maria Adil Mohamed Abdelsalam
5. Abdalmajed Mohammed Abdalrheem Ali

Supervised By:

Prof. Dr. Rashid A SAEED

Sept. 2026

الآية

{ وَعِنْدَهُ مَفَاتِحُ الْغَيْبِ لَا يَعْلَمُهَا إِلَّا هُوَ ۚ وَيَعْلَمُ مَا فِي الْبَرِّ وَالْبَحْرِ ۖ وَمَا تَسْقُطُ مِنْ وَرَقَةٍ إِلَّا يَعْلَمُهَا وَلَا حَبَّةٍ فِي ظُلُمَاتِ الْأَرْضِ وَلَا رَطْبٍ وَلَا يَابِسٍ إِلَّا فِي كِتَابٍ مُبِينٍ }

صدق الله العظيم

سورة الأنعام – الآية 59

To Our Beloved Country & Families

Acknowledgment

We would like to express our gratitude to our supervisor, Dr. Rashid A. SAEED, for his guidance. We also thank our friends for the continuous support and motivation they provided during the work on this research.

ABSTRACT

Modern digital infrastructure relies heavily on continuous server availability, yet traditional monitoring tools remain largely reactive, relying on static thresholds that generate excessive false alerts and often miss early signs of degradation. This research presents an enhanced intelligent server monitoring system that combines real-time stream processing with a hybrid unsupervised machine learning model for predictive fault detection and triage prioritization.

The system ingests server metrics (CPU, memory, disk, and network utilization) through an Apache Kafka message bus and computes a weighted hybrid anomaly score. $S(x_t)$ that combines an Isolation Forest outlier score with an Autoencoder reconstruction error, and classifies detected anomalies into priority levels using a Random Forest classifier. Prometheus and InfluxDB provide short-term and long-term time-series storage, respectively; Grafana delivers live visualization dashboards; Telegram delivers real-time alerts. An independent Apache Spark batch layer periodically generates aggregate historical reports.

The models were trained on 2,243 real records from the Alibaba Cluster Trace 2018 production dataset, selected after evaluating and rejecting two alternative public datasets (the Server Machine Dataset, whose 38 metrics are anonymized, and the NAB AWS CloudWatch collection, which lacks a memory-utilization metric). The hybrid score's decision threshold (0.4963) was derived automatically from the 98th percentile of the training score distribution.

System performance was validated across 28 automated tests spanning unit, integration, and deployment levels (100% pass rate), and the anomaly detector was evaluated against a test set with known injected anomalies, achieving a 92.5% detection rate (Recall) with a 3.39% false-positive rate. The results demonstrate that a hybrid unsupervised approach, trained on real production telemetry, can deliver practical, low-noise anomaly detection suitable for real-time server health monitoring.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Meaning
AI	Artificial Intelligence
ML	Machine Learning
CBM	Condition-Based Monitoring
PdM	Predictive Maintenance
DSP	Digital Signal Processing
AIOps	Artificial Intelligence for IT Operations
CI/CD	Continuous Integration / Continuous Delivery
IoT	Internet of Things
API	Application Programming Interface
JSON	JavaScript Object Notation
TSDB	Time-Series Database
IF	Isolation Forest
AE	Autoencoder
RF	Random Forest
MSE	Mean Squared Error
R^2	Coefficient of Determination
AUC-ROC	Area Under the ROC Curve
FPR	False Positive Rate
$S(x_t)$	Hybrid anomaly score at time t
CPU / RAM / I/O	Central Processing Unit / Random Access Memory / Input-Output
OS	Operating System
UI	User Interface

CHAPTER ONE

1. Introduction

1.1 Preface

These days, servers form the backbone of critical infrastructure. Not just websites. This infrastructure powers everything from hospitals and banks to government services, healthcare databases, and the cloud platforms we all use. Because we rely on these interconnected systems so heavily, their performance and reliability have become absolutely critical. The severity of these outages is clear from the numbers; one industry report calculated that a single hour of downtime costs large companies over $300,000. In high-stakes sectors like finance or online retail, losses can skyrocket to more than $5 million per hour.[1].

For years, the go-to method for monitoring system health has been rule-based tools like Nagios or Zabbix. They work by setting an alarm with a particular metric, like CPU usage or network latency, when crossing a predefined line. It's a useful approach, but it has some major drawbacks. For starters, it's reactive. You're only notified after something has already started to go wrong, which doesn't leave much time to prevent a crisis in advance. These systems also cannot grasp the bigger picture of a large system. They might miss a series of small, subtle anomalies that, when taken together, point to a major failure in the future. This leads to two common and frustrating problems: alert fatigue, where teams are bombarded with false alarms, and silent failures, where a real catastrophe develops unnoticed until it's too late. [2].

This is where Artificial Intelligence (AI) and Machine Learning (ML) come in, offering a shift from reacting to problems to predicting them. Instead of just watching thresholds, AI algorithms can digest enormous amounts of real-time data to learn what "normal" looks like for a specific system. Once it understands the baseline, it can spot tiny, unusual deviations that would never trigger a

traditional alert [3]. This predictive capability could completely change system administration, turning it from a constant firefight into a more strategic operation. This project extends his technological progression with AI-based monitoring solutions. It functions as an intelligent diagnostic system for server health management, analyzing telemetry in real time, detecting anomalies, and classifying server state as normal, warning, or critical.

Inspired by a medical triage system, the system improves upon existing monitoring solutions by integrating intelligent alert prioritization. This system would help administrators focus on the most urgent issues first. Thus, reducing downtime and improves operational reliability.

1.2 Problem Statement

Modern digital and industrial monitoring systems generate overwhelming volumes of alerts without intelligent filtering or prioritization, resulting in fatigue, where critical failures are buried under noise, leading to delayed response times and avoidable system downtime. [4], [5]. Current tools predominantly rely on static thresholds, lacking predictive capability; they can only react after a failure has occurred, with no prior warnings. [6].

Furthermore, alerts are typically not ranked by business impact or severity. This forces operators to waste valuable time investigating low-priority issues while critical components fail unnoticed [6]. In high-stakes environments like data centers or manufacturing plants, this operational inefficiency can result in significant financial losses. There is a clear need for an intelligent, real-time framework that not only detects anomalies early but also triages them effectively, ensuring human attention is focused where it matters most.

1.3 Scope

This research focuses on a system that monitors server resource metrics (CPU, memory, disk, and network utilization) in real time and

detects anomalies using a hybrid unsupervised machine learning approach combining an Isolation Forest and an Autoencoder. The research further emphasizes an intelligent triage mechanism that classifies system status into priority categories (Normal, Low, Medium, Critical), contributing to reduced downtime, lower operational cost, and improved service continuity.

1.4 Research Aim and Objectives

This project aims to develop an intelligent automated framework for real-time digital monitoring. The objectives of this research are:

1. To design and implement a real-time monitoring pipeline that solves the issue of fragmented and slow data flow.
2. To develop a lightweight machine learning model for real-time anomaly detection to early identify signs of component degradation.
3. To build an intelligent triage engine that scores and ranks alerts based on multi-factor analysis.
4. To evaluate system performance against traditional monitoring tools using key metrics.

1.5 Research Questions

1. What ML models are most effective for latency anomaly detection?
2. What are the effective metrics that can help in evaluating monitoring systems?
3. How adaptable is the system to changes in server workload and operating conditions over time?
4. To what extent can manual monitoring effort be reduced in the diagnosis of server infrastructure issues?
5. How do existing monitoring tools balance real-time processing accuracy with computational cost?
6. How are simulations validated against real-world operations?

1.6 Research Outlines

This research contains five chapters and is divided as follows:

Chapter 1: The introductory chapter clarifies the study's significance, the problem it addresses, the proposed solution, the intended objectives, and the methodology employed.

Chapter 2: This chapter presents the fundamental background information related to the proposed system, focusing on digital data processing and security techniques. Showcasing the prominent role of Artificial Intelligence (AI) and Machine Learning (ML) techniques in detecting forgery and manipulation.

Chapter 3: This chapter delivers a comprehensive outline of the architecture, design, and modeling of the introduced system. Furthermore, the utilized software is detailed.

Chapter 4: It exhibits and outlines the effectiveness of the introduced system and the outcomes obtained by this study. Also, an analysis of these findings and results is offered.

Chapter 5: Presents the conclusion and recommendations for future studies.

CHAPTER TWO

2. Literature Review

2. 1 Introduction

This chapter discusses the literature and associated research for server monitor systems. It describes the evolution of monitors from classic rule-based systems to smart, AI-based systems. It gives the conceptual ground from which the project extends as a smarter monitor system.

2. 2 Theoretical Framework

The design and implementation of this monitoring model are grounded in established engineering principles and computational methodologies that enable intelligent, real-time monitoring and response to failures in critical electrical and electronic systems. This framework draws from three core theoretical domains: (1) Condition-Based Monitoring and Predictive Maintenance, (2) Real-Time Data Acquisition and Signal Processing, and (3) Machine Learning for Anomaly Detection and Decision Support. Together, these form the scientific foundation for detecting, classifying, and prioritizing component failures before they lead to system-wide disruptions.

2.2.1 Condition-Based Monitoring and Predictive Maintenance

In electrical engineering, unexpected component failures such as those in transformers, power converters, circuit breakers, or motor drives can result in costly downtime, safety hazards, or even cascading grid failures. Traditional maintenance strategies are often inefficient and fail to address emerging faults. In contrast, *condition-based monitoring (CBM) uses real-time operational data (voltage, current, temperature, vibration, etc.) to assess equipment health continuously. When combined with *predictive maintenance (PdM) enables early detection of degradation patterns such as insulation wear, thermal stress, or harmonic distortion, allowing interventions before failure occurs [7]. Our model

adopts this philosophy by treating sensor telemetry as a diagnostic stream, transforming raw measurements into actionable health indicators.

2.2.2 Real-Time Data Acquisition and Signal Processing

For timely fault detection, data must be captured, processed, and analyzed with minimal latency. This requires adherence to *real-time systems theory, where processing deadlines are critical to system reliability. In electrical systems, high-speed analog-to-digital converters (ADCs), embedded microcontrollers, and field-programmable gate arrays (FPGAs) enable continuous sampling of electrical signals. These raw signals are then processed using digital signal processing (DSP)* techniques—such as Fast Fourier Transform (FFT) for frequency analysis, wavelet transforms for transient detection, and digital filtering to remove noise [8]. Our model utilizes these methods to extract meaningful features (e.g., RMS current, harmonic distortion, temperature trends) from live data streams, ensuring that only relevant, high-fidelity information is fed into the decision-making layer.

2.2.3 Machine Learning for Anomaly Detection and Intelligent Triage

While signal processing identifies deviations, machine learning (ML) provides the intelligence to interpret them. In complex systems, not all anomalies indicate critical failures. Some may be benign operational variations. ML models address this by learning the "normal" behavior of components during healthy operation and flagging statistically significant deviations as potential faults.

- **Unsupervised learning** techniques (e.g., autoencoders, isolation forests) are particularly useful when labeled failure data is scarce, as they model normality without requiring fault examples.
- **Supervised models** (e.g., Random Forest, XGBoost) can classify known failure modes if historical fault data is available.

Beyond detection, intelligent triage involves ranking alerts by severity, impact, and urgency, a function inspired by decision-support systems used in industrial automation and smart grids. [9]. This ensures that human operators or automated control systems respond first to the most critical threats, optimizing resource allocation and minimizing the risks.

2.2.4 Integration of Theoretical Foundations

The system synthesizes these three domains into a cohesive pipeline:

1. Sensors and embedded systems collect real-time electrical data
2. Condition monitoring logic evaluates component health using engineering thresholds and trends.
3. AI models detect subtle anomalies and prioritize alerts through intelligent triage.

2.2.5 Conclusion

This integrated approach ensures that the system is not only technically strong but also aligned with modern best practices in electrical infrastructure management, reliability engineering, and intelligent automation.

2.3 Related Work

Contemporary software engineering emphasizes the deployment of complex applications within distributed systems, characterized by the adoption of the microservices architecture. Therefore, representing a fundamental architectural shift from traditional monolithic structures, advocating for the decomposition of large applications into fine-grained, loosely coupled services. Each microservice functions independently, possessing its own business logic and data store, enabling autonomous development, testing, and deployment. This architectural choice is driven by the imperative to achieve superior resilience, enhanced agility, and pervasive scalability, particularly within dynamic cloud environments [10]. However, the distributed nature of these systems introduces complexity and

dynamicity, leading to challenges such as intricate inter-service communication and the potential for cascading failures, issues poorly managed by traditional monitoring tools designed for monolithic structures.

Microservices provide superior properties such as Continuous Integration/Continuous Delivery (CI/CD) and language agnosticism, making them the preferred model for environments like Cloud-Native platforms and the Internet of Things (IoT) [11].

The complexity of these highly interconnected microservice environments, where applications exist as networks of independent services, necessitates sophisticated operational methodologies, leading to the rise of DevOps and AIOps [12]. AIOps, in particular, combines DevOps practices with Big Data and Artificial Intelligence (AI) to achieve smart orchestration of Cloud-native applications, relying heavily on Continuous Monitoring and Continuous Feedback for efficiency and high-scale performance.

Containerization is a form of virtualization that achieves resource isolation with minimal overhead by sharing the kernel with the host Operating System (OS) [13]. This technology involves packaging the application and its runtime dependent environment into a standardized and portable image. Containers can be deployed on physical or virtual machines, providing a consistent software execution environment from the developer to the consumer.

They offer significant advantages over traditional hypervisor-based virtualization, such as that provided by technologies like Xen, VMware, and KVM. Hypervisor-based methods incur substantial overhead by maintaining a full OS within each Virtual Machine (VM) instance [14]. In contrast, containers are lightweight, easy to manage, and characterized by fast startup times, fundamentally reducing downtime and the deployment costs associated with microservice-based applications [13]. Furthermore, container orchestration

platforms, exemplified by Kubernetes, automate the deployment, scaling, and management of these containerized applications, offering built-in features like service discovery and self-healing capabilities [15].

The highly dynamic and distributed nature of microservices and containerized environments makes system monitoring crucial for ensuring reliability, high uptime, and rapid fault diagnosis. Effective monitoring is paramount, particularly given that microservice architecture is susceptible to intermittent faults that can result in significant economic losses [13].

Monitoring contemporary distributed systems involves dealing with continuously increasing volumes, complexity, and velocity of data streams, frequently referred to as “big data in motion” [16]. Traditional database tools, while capable of certain analytical functions and handling large data volumes, often lack the capability for active, real-time response essential for modern operations [8]. The scale of this challenge is immense; for example, a medium sized cloud platform may generate approximately 20 million time-series points every minute, necessitating high-throughput and scalable integration systems to manage these large-volume, low-value-density data streams. Robust data integration systems must achieve this while balancing factors such as real-time performance requirements and low consumption of computing power across potentially transnational, multi-cloud environments [17].

A fundamental category of monitoring information collected to capture the runtime status of these complex systems is multivariate time-series data (MTS) [18]. Multivariate time-series data (MTS) records continuous observations from multiple correlated indicators, such as CPU usage, memory usage, and network throughput, sampled at regular intervals [19]. MTS data offers richer system information compared to univariate time series, reflecting the inherent interrelationship between variables in complex systems [20]. This monitoring

data commonly encompasses heterogeneous forms, including logs, metrics (Key Performance Indicators or KPIs), and traces, each capturing runtime status in different ways [21]. The objective of management often involves developing generalized models, trained on historical data from multiple entities, to perform accurate detection across all system components, thereby mitigating the impracticality of requiring a separate model for each entity due to high resource overheads [22].

To handle the real-time velocity and volume requirements of this data, specialized mechanisms are necessary. Complex Event Processing (CEP) technology provides an effective approach. Complex Event Processing (CEP) is formally defined as the analysis of streams of discrete information elements, known as events, where the "complex" attribute highlights the application of correlation, aggregation, and abstraction to raw event streams to recognize sophisticated patterns of behavior. A core capability of CEP systems is recognizing higher-level patterns through correlation, aggregation, and abstraction, facilitating real-time analytics. The EventSwarm programming framework embodies these CEP techniques. CEP systems generally rely on continuous evaluation of expressions and patterns as events arrive, providing a near-real-time capability. To ensure massive scalability for "big data in motion," CEP solutions often focus on in-memory processing, utilizing filtering and distribution capabilities [16].

A niche but crucial concept in understanding the behavior of distributed applications is traceability, particularly enabled by distributed tracing tools [23]. Distributed tracing provides observability of communication and data flow as a request traverses the many different components of an application [24]. A trace represents the path of a request through various services and consists of spans, where a span denotes a logical unit of work performed by a service. The parent-

child relationships between spans indicate the causal dependency of service operations, allowing for a global view that correlates logs across microservices and addresses the issue where local logs lack context about the overall flow of events [24]. Distributed tracing is highly valuable for fault localization by capturing the causal model for a request as it traverses microservices.

The operational landscape of distributed monitoring relies on several notable tools. Prometheus is an open-source monitoring and alerting tool widely adopted for collecting time series data using a pull strategy and is natively integrated with Kubernetes monitoring systems [10], [12]. Prometheus, often coupled with visualization tools like Grafana, is central for storing, visualizing, and inspecting time-series data, and facilitates the creation of alerting solutions. Other essential systems for storing and analyzing time-series data in distributed architectures include VictoriaMetrics and Influx DB [11]. For handling trace data, open-source systems such as Jaeger are utilized for generating, collecting, and storing tracing data for instrumented microservice applications [24]. Tools for collecting container metrics, such as cAdvisor, are often integrated with Prometheus to obtain detailed usage metrics (CPU, memory, etc.) [25].

To dictate the type of data collected, anomaly detection systems must adapt to significant differences in platform and operational environments:

1. **Cloud Platforms and Operations**: Cloud environments (e.g., Amazon EC2) demand monitoring based on DevOps operations and VM lifecycle events, requiring derived metrics like the number of terminated instances, which are platform-specific additions to basic metrics [26].
2. **Traditional Data Centers and Logging**:Data center monitoring often revolves around **Linux system services** (e.g., `crond`, `syslog`, `auditd`) and hardware diagnostics. Log analysis methods must account for

service-specific log header structures and local administrative practices [27].

3. **Microservices and Hybrid Stacks**: Microservice architectures (often running on platforms like Kubernetes) introduce complexity through technological diversity, as individual services may be implemented in various languages (Java, Python, Go), all contributing to the overall system state [11].
4. **Underlying Metrics**: Despite platform differences, many core monitoring metrics remain consistent across systems, typically covering standard computational resources like CPU usage, memory utilization, network traffic, and disk I/O, whether gathered by specialized tools (e.g., Node_exporter in a Slurm cluster) or general agents (e.g., `psutil` on a Windows 10 machine). However, the *interpretation* of these metrics is highly dependent on the operational context [4].

This shift in modern software architectures, results in highly complex operational environments generating high-velocity monitoring data streams, encompassing logs, metrics, and traces. To manage this complexity, the industry has increasingly adopted AIOps, which integrates Big Data processing with Artificial Intelligence (AI) and Machine Learning (ML) to automate monitoring, prediction, and incident diagnosis, thereby reducing operational costs and ensuring service stability [21]. Central to AIOps is the accurate and rapid detection of anomalies in time-series data (TSAD), providing the crucial visibility markers needed for maintaining high performance and timely fault mitigation.

Historically, anomaly detection began with classical and statistical approaches, including distance-based methods like K-Nearest Neighbors (KNN), density-based methods, and linear model-based techniques such as Principal Component Analysis (PCA) and Auto-Regressive Integrated Moving Average (ARIMA)

models [19], [20]. While statistical models like ARIMA can incorporate seasonality and temporal dependencies, they fundamentally struggle to capture the complex, nonlinear characteristics prevalent in modern multivariate time series (MTS) [20]. More evolved statistical methods employ process control charts, which apply the Central Limit Theorem to aggregated log data within time windows, using metrics like mean and standard deviation to classify anomalies by severity [17], [18]. Similarly, Complex Event Processing (CEP) frameworks utilize specific statistical control rules, such as deviations exceeding three standard deviations from the mean, to identify unusual behaviors in real-time data streams, contributing to early alerting and improved operational quality. Bayesian Networks (BNs) are integrated into these approaches to model conditional dependencies among multiple variables in MTS, which helps analyze data validity despite inherent system uncertainties [19].

The limitations of classical methods, especially in modeling temporal and nonlinear characteristics, fueled the migration toward deep learning (DL) techniques[19]. Recurrent Neural Networks (RNNs), particularly Long Short-term Memory (LSTM) networks, are instrumental in modern TSAD because they can efficiently process sequential data, maintain memory of long-term dependencies, and accurately model the system's normal behavior [4].

DL methods often rely on reconstruction principles: models trained solely on nominal data generate a reconstruction of the input, and the resulting reconstruction error serves as the anomaly score [20]. Autoencoders (AEs) perform this reconstruction via a dense network, while Variational Autoencoders (VAEs) introduce a probabilistic dimension to the reconstruction process. Advanced generative models, such as Generative Adversarial Networks (GANs), are also leveraged for their capacity to model complex distributions in multivariate time series data [21]. As distributed systems grow more intricate,

merely modeling temporal patterns is insufficient; detecting anomalies requires understanding the complex inter-relationships between components (inter-sensor dependencies) [20]. Graph Neural Networks (GNNs) address this by learning the graph structure of dependencies between variables (sensors or metrics) and identifying deviations from these learned patterns. GNNs often utilize attention mechanisms to assign varying importance to neighboring nodes during feature aggregation, recognizing the inherent heterogeneity among different system components.

Furthermore, attention is critical even in traditional RNN/LSTM structures; models like the Deep Attentive Anomaly Detection approach integrate the attention mechanism with LSTMs to comprehensively capture both temporal and interdimensional dependencies in multimodal data streams (logs and metrics) [21]. Similarly, when leveraging external knowledge sources, such as pre-trained language models (like GPT-2) as backbones for anomaly detection, a Dual-Attention Mechanism is employed to simultaneously address the distinct temporal and inter-metric relationships inherent in MTS data [18].

Another notable predictive methodology is Hierarchical Temporal Memory (HTM), a machine learning method structured to mirror the operation of the neocortex, relying on large numbers of stored pattern sequences. The HTM algorithm uses internal prediction mechanisms to compute a raw anomaly score based on the divergence between the predicted sparse vector and the actual input. A key advantage of HTM in real-time scenarios is its capacity to handle nonstationary data: if system behavior shifts, the HTM model adapts automatically, causing the anomaly score to recede once the new normal is established [19]. HTM is effectively combined with a Naive Bayesian Network classifier to enhance anomaly detection in multivariate time series, using HTM output as input for BN inference, thereby avoiding dimension reduction.

Despite the sophistication of DL models, classical methods and hybrid lightweight models remain vital, applicable, and resource-efficient for operational scenarios, particularly in contexts where resource constraints or computational costs are primary concerns, such as low-latency monitoring. Hybrid Autoencoder Models blend the representation of neural networks with the statistical rigor of classical techniques. For instance, the Deep Autoencoding Gaussian Mixture Model (DAGMM) integrates an Autoencoder with a Gaussian Mixture Model (GMM) to calculate anomaly scores from reconstruction error and estimated energy [21]. Similarly, evolving Gaussian Fuzzy Classifiers (eGFC) offer a highly concise, rule-based structure that maintains high classification accuracy (e.g., 92.48% average accuracy) in non-stationary data streams while requiring minimal CPU time and resources, making them easily deployable on a single processing node [28].

The results generated by these detection models necessitate intelligent alert management, forming the core feedback loop of AIOps [12]. To deal with alarms generated from continuous detection, relying on static, hard-coded thresholds is inefficient [10]. Instead, dynamic thresholding techniques are used, such as the Peaks-Over-Thresholds (POT) method based on Extreme Value Theory (EVT), which fits the tail distribution of anomaly scores to dynamically set thresholds and minimize the need for expert knowledge [18]. Furthermore, while high recall is crucial due to the catastrophic consequences of missed faults, precision management is essential to prevent operator fatigue from excessive false positives (FPs) [21]. Techniques like pruning anomalous sequences or combining alerts are used to manage FPs [4]. Proactive monitoring also involves forecasting models, like the Facebook Prophet model, to predict future metrics (such as traffic peaks) in advance, allowing the orchestrator to trigger preemptive horizontal scaling or other remediation actions before the actual issue materializes [12].

2.4 Research Gaps

The advent of microservices and cloud-native systems has created an urgent need for integrating machine learning, based anomaly detection into real time observability pipelines. While many studies explore anomaly detection in isolation on batch datasets or offline logs, the critical challenge lies in embedding those detection models into low-latency, streaming pipelines that connect metrics collectors to alerting and dashboarding tools. Some industry-scale efforts already embody this vision: for example, Walmart's "Anomaly Detection for Incident Response at Scale" [29] demonstrates a production-oriented, real-time detection pipeline that ingests metric streams, applies models, and routes alerts, showing both promise and engineering complexity [30].

Despite those pioneering systems, many academic works remain rooted in offline, batch evaluation. The predominant approach is to collect historical metric traces or logs from servers, inject or label anomalies, and then train/test detection models under controlled conditions. Other studies provide a foundation for algorithmic design. They often fail to simulate the demands of real-time cloud settings: latency constraints, variable ingestion rates, label latency, and resource overhead in production [31].

Some work more closely approaches the streaming ideal. The Real-Time Deep Anomaly Detection presents an online architecture tailored for multivariate sensor data, placing models in a streaming pipeline rather than relying on batch postprocessing. This work is helpful in showing how detection logic can operate under latency constraints in continuous inflow settings. However, its domain industrial sensors and IoT, differs from cloud services metrics in scale, cardinality of labels, and dynamic service topology, meaning significant adaptation is required for cloud-native environments [25].

An example closer to microservices is Deep Attentive Anomaly Detection for Microservice Systems ICWS. In this work, the authors instrument KPI collection via Prometheus and apply attention-based sequence models to detect anomalies across interacting services. They evaluate the detection performance in microservice deployments with injected fault scenarios, demonstrating how such sequence models can catch complex cross-service disruptions. The study underscores both the potential and pain points of integrating ML models with existing cloud-native monitoring stacks [11].

While these approaches move toward full integration, gaps remain, especially in rigorous comparative evaluation across model complexity within a unified real-time pipeline. Many works do not examine the trade-offs between simpler models, e.g., Isolation Forest, Local Outlier Factor, and more complex sequence models when deployed in streaming contexts, measuring not just detection accuracy but also latency, throughput, and resource cost. Benchmark suites like StreamAD provide broad algorithmic comparisons across many datasets, but still evaluate models in offline scenarios, not in an integrated cloud stack [22].

Moreover, the question of simulated versus production data remains a core tension. Synthetic time-series generators such as “Synthetic Time Series for Anomaly Detection in Cloud Microservices allow precise control and reproducibility, enabling systematic experiments across anomaly types and noise levels. Yet they fail to capture operational noise, event correlations, and label ambiguity endemic to real production data. On the other hand, industrial-scale datasets as used in Walmart’s system, or in cloud provider anomaly detection studies provide authenticity but often lack public availability, contain significant privacy redaction, or have weak ground truth labels. This dichotomy results in a gap: work that uses real data seldom provides full pipeline or comparative

evaluations, while work that delivers full pipelines often relies on synthetic or semi-simulated traces [30].

The greatest problem for anomaly detection is in dealing with periods of low activity,

For a lightly used host, the appearance of a single new event might trigger a standard deviation from the norm.

While critical anomalies leading to catastrophic failures are noticed and addressed, it is more challenging to recognize the non-critical ones that result in a lower than optimal efficiency of the system. Non-critical anomalies are often hard to locate and ignored. These anomalies are difficult to detect because distributed systems may not have sufficient sensors to monitor the traffic flow within their interconnected nodes. Finally, many real-time anomaly systems concentrate heavily on detection accuracy, but underreport engineering challenges in cloud-native monitoring contexts. For example, limitations due to scrape intervals, label cardinality explosion, push/pull trade-offs, or query overhead in Prometheus are often mentioned only superficially or left to industry whitepapers.

This chapter outlines the inherent complexity of distributed systems, where latency issues stem from multiple sources and often point to deeper failures. It traces the evolution of anomaly detection from simple statistical methods to advanced AI models that learn complex patterns to identify subtle anomalies accurately.

Given the impracticality of testing faults in live systems, the review highlights simulation as a vital, safe tool for generating comprehensive training data. Synthesizing these areas reveals a core research gap: the lack of integration between high-fidelity simulation and detection algorithm evaluation. This thesis

aims to bridge that gap by using simulation to create a fair evaluation benchmark and overcome the scarcity of real-world data.

2.5 Summary of the Chapter

This chapter outlines the inherent complexity of distributed systems, where latency issues stem from multiple sources and often point to deeper failures. It traces the evolution of anomaly detection from simple statistical methods to advanced AI models that learn complex patterns to identify subtle anomalies accurately.

Given the impracticality of testing faults in live systems, the review highlights simulation as a vital, safe tool for generating comprehensive training data. Synthesizing these areas reveals a core research gap: the lack of integration between high-fidelity simulation and detection algorithm evaluation. This thesis aims to bridge that gap by using simulation to create a fair evaluation benchmark and overcome the scarcity of real-world data.

CHAPTER THREE

3. Methodology

3.1 Introduction

This chapter describes the systematic methodology adopted for the design, development, and validation of this system. This methodological framework is done to address critical limitations of traditional server monitoring tools. The methodology is structured into sequential phases: data acquisition and preprocessing, algorithm selection and model development, system architecture design, implementation, and comprehensive evaluation.

The methodology includes a validation strategy involving testing in both simulated and real-world environments using benchmark datasets to ensure accuracy.

3.2 Research Design

Here's a detailed flowchart for the design and performance evaluation of a model along with an explanation of each step:

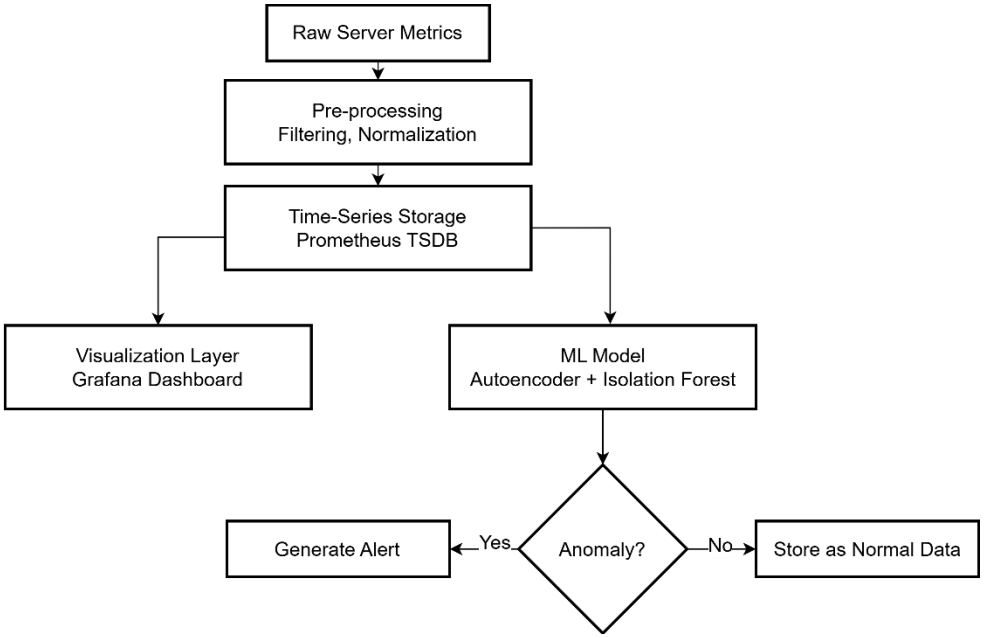

Figure 3-1: Data Flow Diagram for the Proposed Monitoring System

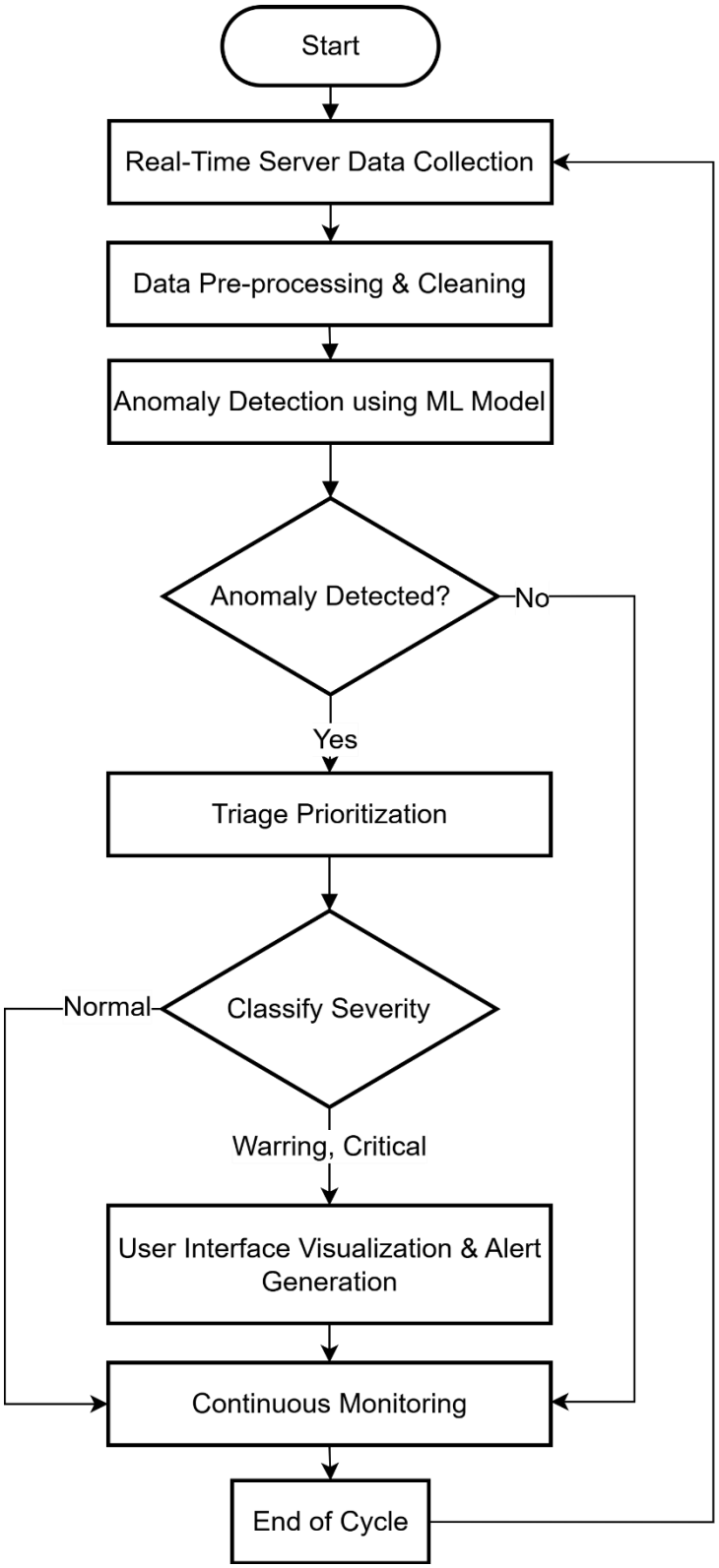

Figure 3-2: Block diagram representing the system model

Start: Begin the monitoring process with system activation and environment setup, including connecting to target servers and initializing the Machine Learning models.

Real-Time Server Data Collection: Server metrics are continuously collected at a high rate. These metrics include CPU utilization, Memory Load, Disk I/O activity, Network Throughput.

Data Pre-processing and Cleaning: Basic signal processing techniques are applied to the raw data. Including noise reduction, data normalization to ensure all metrics contribute equally to the model, and calculating moving averages to smooth out insignificant sudden spikes.

Anomaly Detection using ML Model: Processed data is fed into the Machine Learning model. The model analyzes the data and compares it to the "normal" behavior it was trained on.

Triage Prioritization: If an anomaly is detected, the Intelligent Triage Engine is used. This engine classifies the anomaly into severity levels: Normal, Warning, or Critical. Classification is based on a multifactor analysis that considers the degree of deviation, the component's impact on vital operations, and the frequency of the anomaly.

User Interface Visualization and Alert Generation: The system provides a clear and interactive interface for visualizing real-time data, displaying system health dashboards, and monitoring alerts and anomaly trends. Detailed alerts are generated within the interface, including the classified severity level (Warning or Critical). Actions taken may include sending immediate notifications via email or messaging platforms.

Continuous Monitoring: If no anomaly is detected, the system returns to the data collection step to continue the monitoring cycle, ensuring continuous coverage of the server's status.

End of Cycle: This step represents the conclusion of the current monitoring cycle, where the system is ready to begin the next cycle or continue operations continuously.

3.3 Research Approach

The research employs an abductive approach, iteratively refining predictive models and integrating theoretical principles with data patterns. The process begins deductively, applying established Predictive Maintenance and AIOps frameworks to define system architecture and failure thresholds. It then incorporates inductive reasoning through continuous analysis of operational data, identifying emerging fault patterns and system rules. The focus was to allow a systematic transition from reactive monitoring to a more proactive approach to fault detection and triage prioritization.

To implement Predictive Maintenance and AIOps principles, the system is structured around a pipeline for continuous condition-based monitoring. Starting with the collection of live server metrics into a dedicated time-series data store (InfluxDB). These streams are processed and analyzed by a hybrid analytical engine, which combines unsupervised machine learning models for detecting unknown anomalies and supervised algorithms (Random Forest) for classifying and prioritizing known fault types based on learned patterns. The processed outputs, including fault severity scores and system health status, are then served to a visualization dashboard, completing an integrated pipeline for predictive fault detection and triage prioritization.

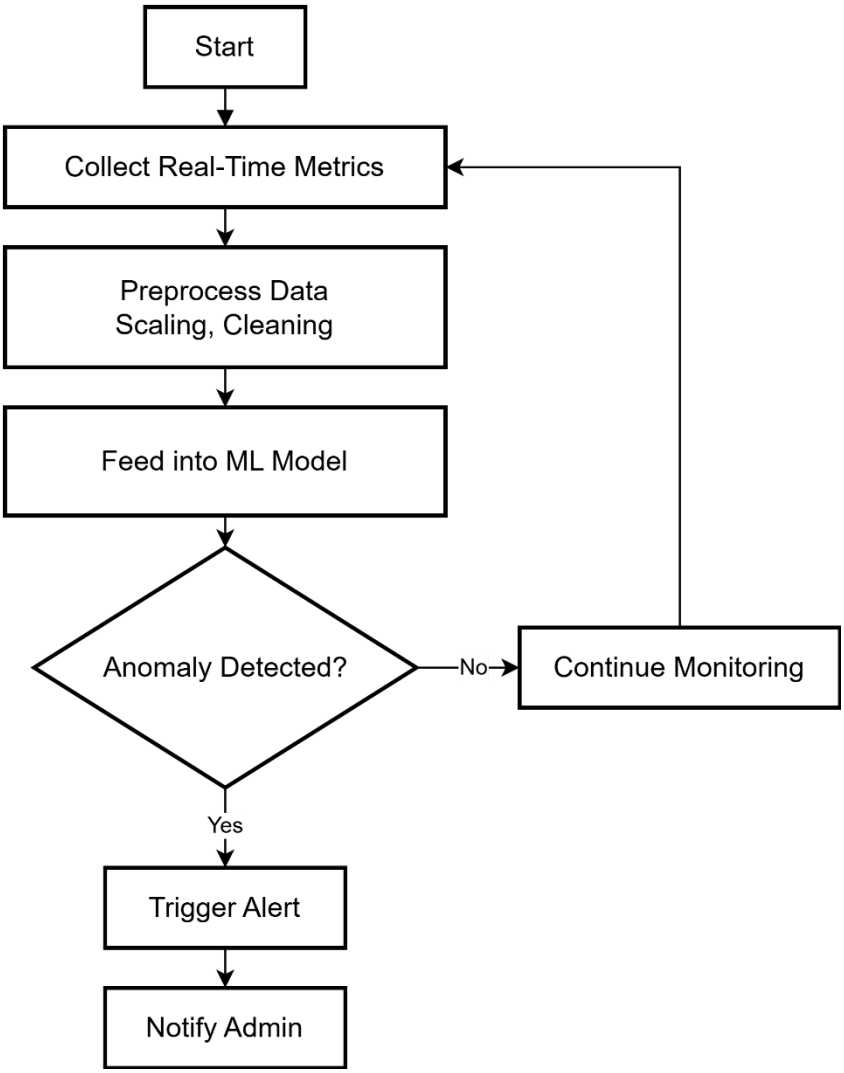

Figure 3-3: Flowchart of the ML-Based Anomaly Detection Process

3.4 Research Modeling

The primary objective is to build an automated, self-learning system capable of proactively detecting abnormal behaviors (anomalies) in servers, to enable operations teams to prevent outages or cyber-attacks before they occur.

System Components:

Data Collection & Ingestion Layer:

- Agents: Using Prometheus as the primary agent to scrape and collect metrics.
- Message Bus: Using Apache Kafka as a durable message bus to ingest and route real-time data streams.

Processing & Storage Layer:

- Processing Engine: Using Apache Spark for large-scale batch processing.
- Database: Using Prometheus's time-series storage with InfluxDB for long-term storage.
- AI/ML Model Layer:
  - Model Selection: Isolation Forest for outlier detection with contamination = 0.02, and Autoencoder for complex pattern recognition.
  - Hybrid Anomaly Scoring: The scores from both models are combined into a unified hybrid anomaly score. For a single observation at time *t*, represented as a feature vector x_t containing *d* metrics:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min} + \epsilon}$$

  - The Isolation Forest raw score is derived via its decision function:

$$\mathrm{IF}_{raw}(x_t) = -\mathrm{decision\_function}(x_t)$$

  - These are combined with the Autoencoder reconstruction error into the hybrid anomaly score using configured weights:

$$S(x_t) = w_{\mathrm{ae}} \cdot \mathrm{AE}_{\mathrm{norm}} + w_{\mathrm{if}} \cdot \mathrm{IF}_{\mathrm{norm}}$$

Action & Alerting Layer:

- Rules Engine: Determines anomaly severity levels.
- Alerting Channels: Integrated with a Telegram bot.
- UI & Reporting Layer: Dashboards: Grafana for visualizing server health and AI model scores.

System Lifecycle:

1. Training models on historical data.
2. Applying trained models to new data.
3. Continuous performance monitoring.
4. Periodic model retraining.

3.5 Dataset Description

3.5.1 Dataset Source

The primary operational data for this project comes from real-time server performance metrics exported during system runtime using the Kafka-to-InfluxDB streaming pipeline. Training data is sourced from the Alibaba Cluster Trace 2018 dataset, specifically from Zenodo that an eight-day collection window that helps to represent real enterprise cluster workloads.

3.5.2 Dataset Characteristics

- **Size**: The dataset consists of 2,243 valid time-series readings sampled at 300-second intervals across the eight-day collection window.
- **Type**: Data is structured as multivariate time-series data, where each record is identified with a timestamp.
- **Features**:

1. Four primary performance metrics are mapped: cpu, disk, memory, disk, and network.
2. Evaluation features: anomaly, ae_error, hybrid_score, and priority.

Table 3-1: Extracted Features from the Dataset

Feature	Description
cpu	CPU utilization (%)
memory	Memory utilization (%)
disk	Disk I/O utilization (%)
network	Combined incoming + outgoing network traffic
anomaly	Anomalies
ae_error	Autoencoder reconstruction error
hybrid_score	Hybrid Anomaly Score
priority	Priority classification tier

3.5.3 Data Preprocessing

- Invalid records were filtered out, where erroneous disk percentage values that equaled -1 or 101 were filtered out to avoid corruption.
- Feature engineering & aggregation were done to combine inbound (net_in) and outbound (net_out) network bandwidth into a single (network) metric.
- MinMax Normalization is done on feature distributions into the [0,1] range using MinMaxScaler before feeding them into unsupervised models.

3.5.4 Addressing Class Imbalance

Unsupervised hybrid scoring function $S(x_t)$ is used to blend Isolation Forest boundary detection with Autoencoder reconstruction MSE.

3.5.5 Handling Missing Data

Rows containing corrupted or erroneous hardware telemetry tags were systematically dropped during data ingestion.

3.6 Implementation Tools and Environment

3.6.1 Implementation

The intelligent server monitoring system was implemented as a set of containerized services orchestrated with Docker Compose, combining an open-source streaming and storage stack (Apache Kafka, Prometheus, InfluxDB) with machine learning libraries (scikit-learn, TensorFlow/Keras) for anomaly detection and triage classification, Apache Spark for periodic large-scale batch analysis, and Grafana for live visual monitoring.

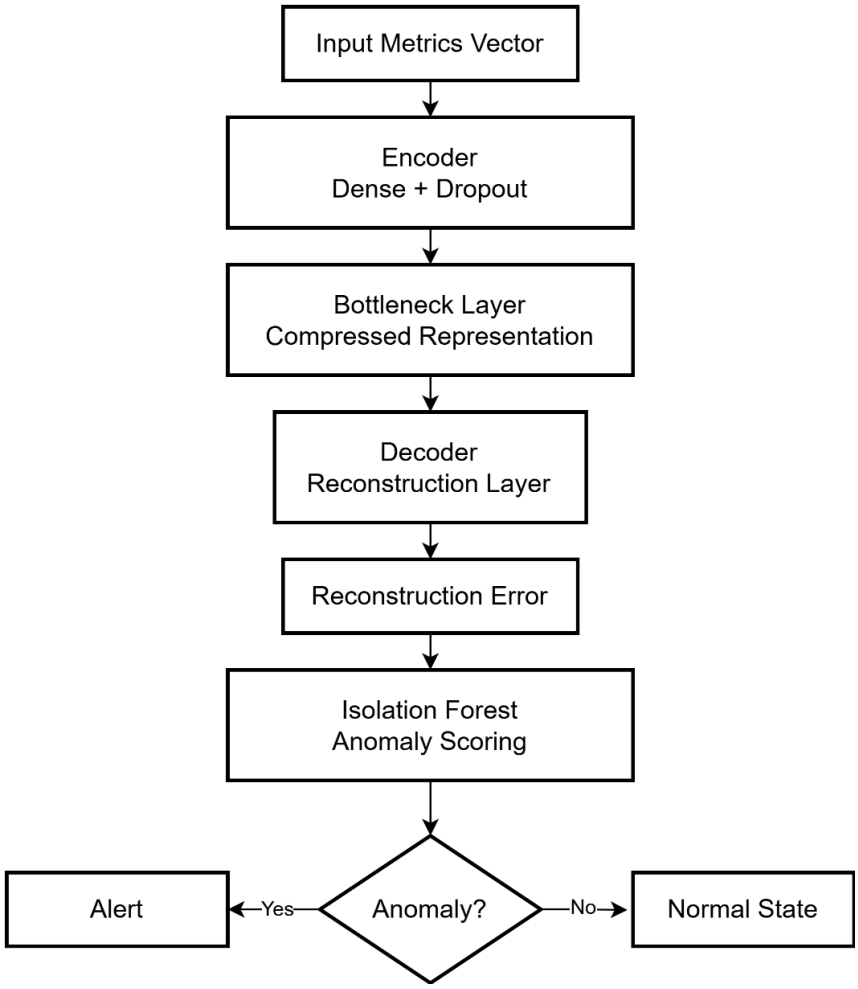

Figure 3-4: Architecture of the Hybrid Autoencoder–Isolation Forest Model

3.6.2 Data Collection

Server metrics are published as JSON messages to an Apache Kafka topic. In the deployed system, this stream is populated by resampling real historical records from the Alibaba dataset with small random jitter (±3%), and a small proportion (3%) of intentionally injected extreme values to emulate genuine anomalous events for evaluation purposes. Apache Kafka is implemented to act as a durable, decoupled message bus between the data source and the analysis engine.

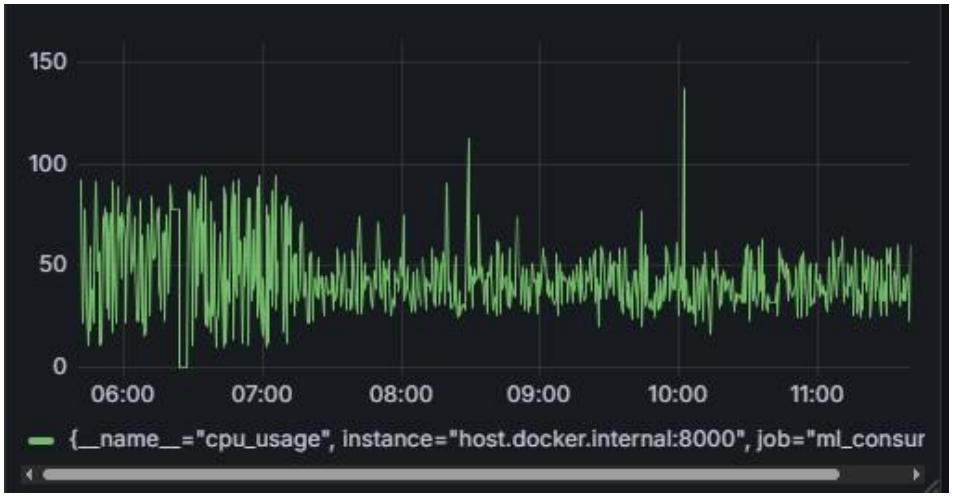

Figure 3-5: Real-time CPU Usage sample Metrics for a 24-hour period

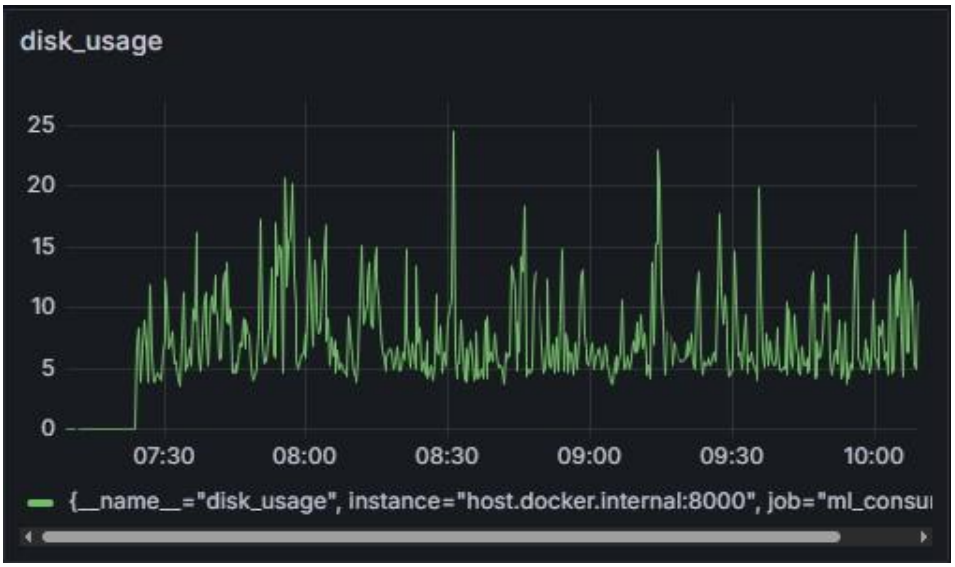

Figure 3-6: Real-time Disk I/O usage sample for a 24-hour period

3.6.3 Anomaly Detection Using Machine Learning Model:

A hybrid model combining an Isolation Forest and an Autoencoder is used. The Autoencoder (built with TensorFlow/Keras) features dense hidden layers with relu activation and an output layer governed by a sigmoid activation function. It is trained over 20 epochs with a batch size of 32 to reconstruct normal readings using Mean Squared Error loss:

$$MSE(x_t, \hat{x}_t) = \frac{1}{d}\sum_{j=1}^{d}(x_{t,j} - \hat{x}_{t,j})^2$$

The Isolation Forest is trained using scikit-learn with a contamination = 0.02. It produces an outlier score via its decision function. Both raw scores are normalized to [0,1] using the minimum and maximum values observed on the training set, then combined into the weighted hybrid score $S(x_t)$.

$$S(x_t) = 0.5 \times \mathrm{AE}_{\mathrm{norm}}(x_t) + 0.5 \times \mathrm{IF}_{\mathrm{norm}}(xt)$$

An anomaly is flagged when the hybrid score exceeds a threshold derived automatically from the 98th percentile of the $S(x_t)$ distribution:

$$S_{threshold} = \mathrm{Percentile}(S(X).98)$$

Instead of a manually chosen fixed value, the trained models are loaded once at start-up and applied to every incoming Kafka message in real time.

3.6.4 Triage Prioritization

Random Forest classifier using scikit-learn, where is trained on the four metrics together with the computed anomaly flag, assigns a priority level (0 = Normal, 1 = Low, 2 = Medium, 3 = Critical) to every reading. A lightweight rules engine additionally attributes a human-readable root cause (e.g., High CPU, High Memory, Disk Full, Network Spike) whenever an anomaly is flagged, by comparing each metric against fixed operational thresholds.

3.6.5 User Interface Visualization And Alert Generation:

A local CSV log is reflected immediately in the Grafana dashboard that connects to Prometheus. It displays live panels that update continuously as new messages are processed. Every processed reading is also written to InfluxDB for long-term storage and historical querying. When a non-normal severity level is detected and differs from the most recently alerted cause, a real-time alert is sent through a Telegram bot. Repeated alerts for the same ongoing cause are suppressed until the system returns to a normal state.

3.7 Evaluation and Performance Metrics

3.7.1 Evaluation Method:

Because the anomaly-detection models are unsupervised and the underlying production trace carries no ground-truth failure labels, evaluation was performed at four complementary levels rather than through a single train/test split:

(1) unit testing of the core decision functions (severity mapping, root-cause rules, and the hybrid-score formula) using pytest.

(2) a controlled comparison of the anomaly-detection logic before and after threshold calibration, replaying 10,022 real training messages through both versions.

(3) quantitative detection-accuracy testing using a separately generated evaluation set of 2,000 messages containing 3% intentionally injected anomalies with known ground truth.

(4) end-to-end integration and deployment testing of the live system.

3.7.2 Performance Metrics:

- **Accuracy**: The proportion of correct anomaly detections over the total number of samples.

$$\text{Accuracy} = \frac{\text{TP}}{\text{TP} + \text{TN} + \text{FP} + \text{AN}}$$

- **Precision**: The ratio of true positive anomaly detections to the total positive detections.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- **Recall (Sensitivity)**: The ratio of true positive anomaly detections to the total actual anomalies:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- **F1-Score**: The harmonic mean of precision and recall, providing a balanced measure of the model's performance.

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Percision+Recall}}$$

- **Detection Rate**: The percentage of true anomalies detected by the system over the total number of actual anomalies.

$$\text{Detection Rate} = \frac{\text{TP}}{\text{TP} + \text{FN}} \times 100$$

- **False Alarm Rate**: The percentage of non-anomalous data falsely classified as anomalies.

$$\text{False Alarm Rate} = \frac{\text{FP}}{\text{FP} + \text{TN}} \times 100$$

- **AUC-ROC:** The Receiver Operating Characteristic (ROC) curve plots True Positives against the False Positive rate at different threshold settings. Area Under the Curve (AUC) aggregates performance into a single scalar value ranging from 0 to 1.

$$\text{AUC-ROC} = \int_0^1 TPR(FPR^{-1}(x))dx$$

3.8 Chapter Summary

This chapter details the methodology used in the design and performance evaluation of an Enhanced Intelligent Server Monitoring System for Predictive Fault Detection and Triage Prioritization. The next chapter will present the results obtained from the simulations and analyze their implications.

CHAPTER FOUR

4. Simulation Results

4.1 Introduction

In this chapter, the results obtained from the performance analysis of real-time server monitoring and anomaly detection. The evaluation investigates how variations in these metrics reflect normal and anomalous server behavior, and how they contribute to the machine learning models introduced in Chapter Three.

Comparisons are also presented to validate the reliability of the collected data and to ensure the consistency of the monitoring framework.

4.2 Evaluation Dataset & Test Protocol

Out of the processed logs, 2000 standardized, real-time multivariate readings were extracted from the Alibaba Cluster Trace 2018 benchmark. Messages containing features are normalized via the MinMax scaler to maintain consistency during training.

To test model sensitivity in real-world failures, this testing protocol was adopted:

1. Feature Integrity: Incoming records undergo automated field extraction and scaling into a 4-feature vector before being processed by Isolation Forest and Autoencoder components.
2. Anomaly Injection: A controlled resampling approach with minor temporal jitter, targeting an injection rate of 3%. In the resulting evaluation split, genuine anomalies accounted for exactly 49 instances (2.45%).

The data was preprocessed as follows:

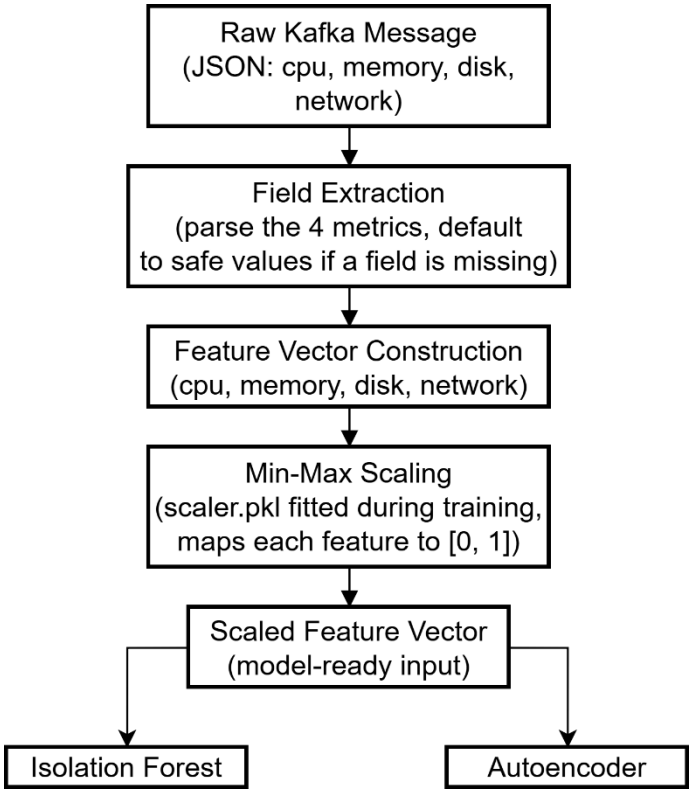

Figure 4-1: Data preprocessing pipeline illustrating key transformation steps

4.3 Performance Evaluation

The anomaly-detection logic was first validated by comparing its behavior before and after threshold calibration on 10,022 real training messages: the original fixed threshold (15) never triggered a single detection (0 of 10,022, 0%), because it exceeded the maximum reconstruction error ever observed (0.255) in the training data; after calibrating the threshold from the training distribution (0.015), the system correctly flagged 120 readings (1.2%).

Following retraining on the Alibaba dataset, the hybrid score's decision threshold was recalculated as 0.4963. A separate evaluation set of 2,000 messages was generated by resampling real historical rows with small jitter and intentionally injecting anomalies into 3% of messages (Ground Truth).

Two data-generation strategies were compared: generating each of the four features independently at random produced a nominally high detection rate (98.4%) but an impractical 32.9% false-positive rate, because independent sampling randomly breaks the natural correlation between CPU, memory, disk, and network in real telemetry. Then, resampling is done with real historical rows with jitter, the strategy adopted in the final system achieved a Recall of 92.5% with a False Positive Rate of only 3.39%.

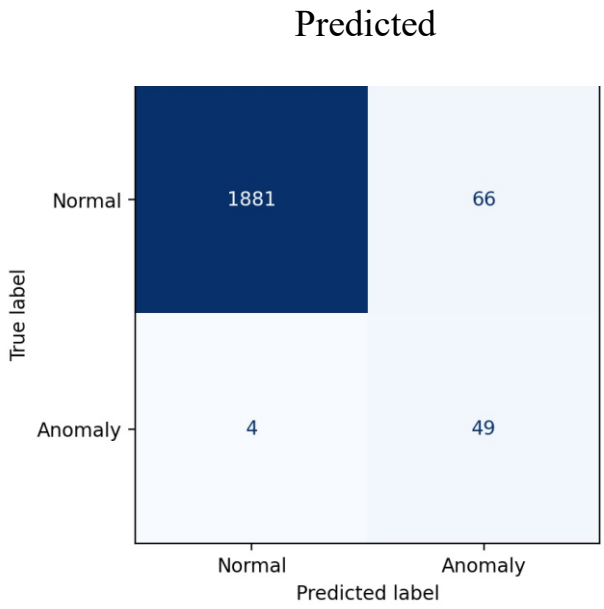

Figure 4-2: Confusion Matrix for Hybrid Anomaly Detector

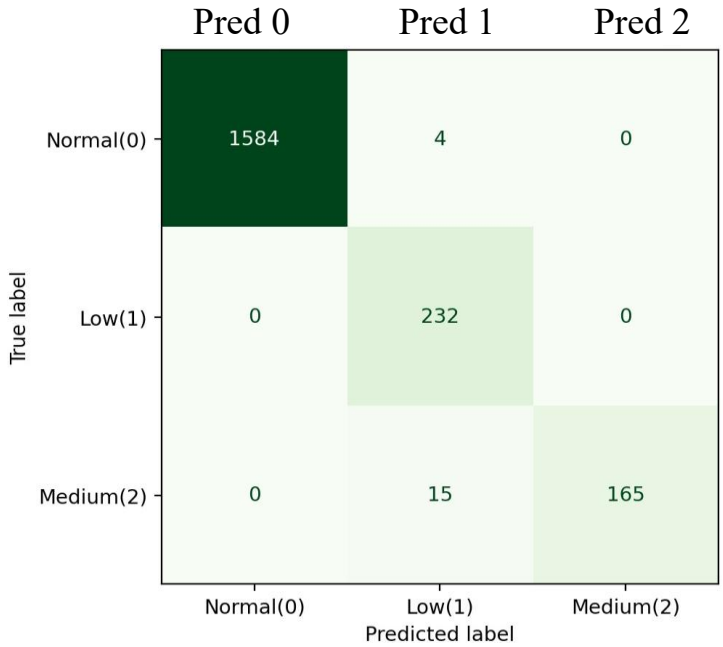

Figure 4-3: Confusion Matrix for Random Forest Model

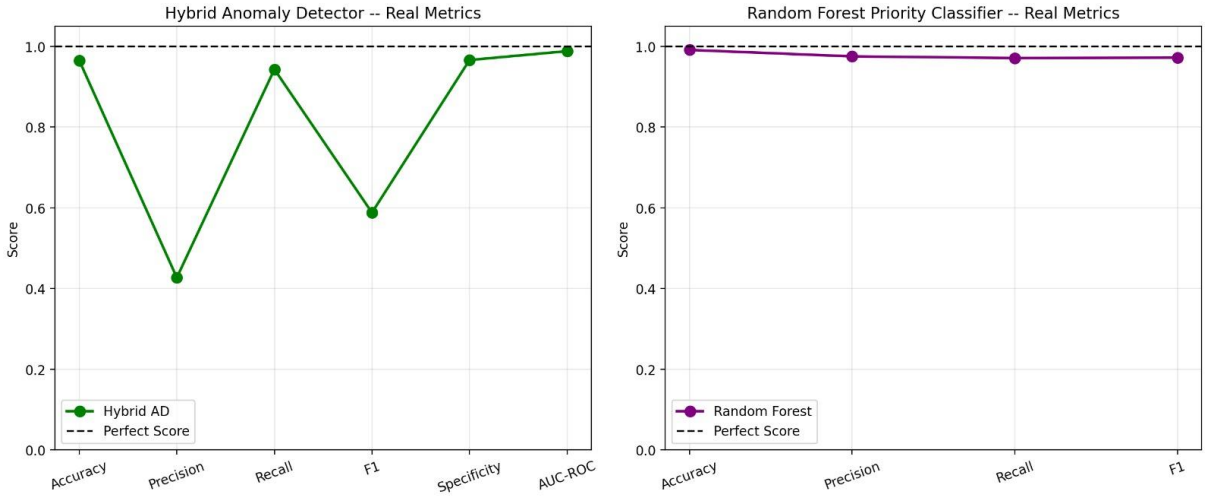

Figure 4-4: Model Performance Metrics

The Hybrid Anomaly Detector achieves high Accuracy (0.965) and AUC-ROC (0.988), indicating strong overall discrimination between normal and anomalous observations. Its comparatively lower Precision (0.427) stems from the class imbalance in the evaluation dataset, where genuine anomalies represent only about 2.65% of total readings, along with the operating threshold that prioritizes anomaly-detection sensitivity. The Random Forest Priority Classifier in contrast maintained consistently high performance across all metrics.

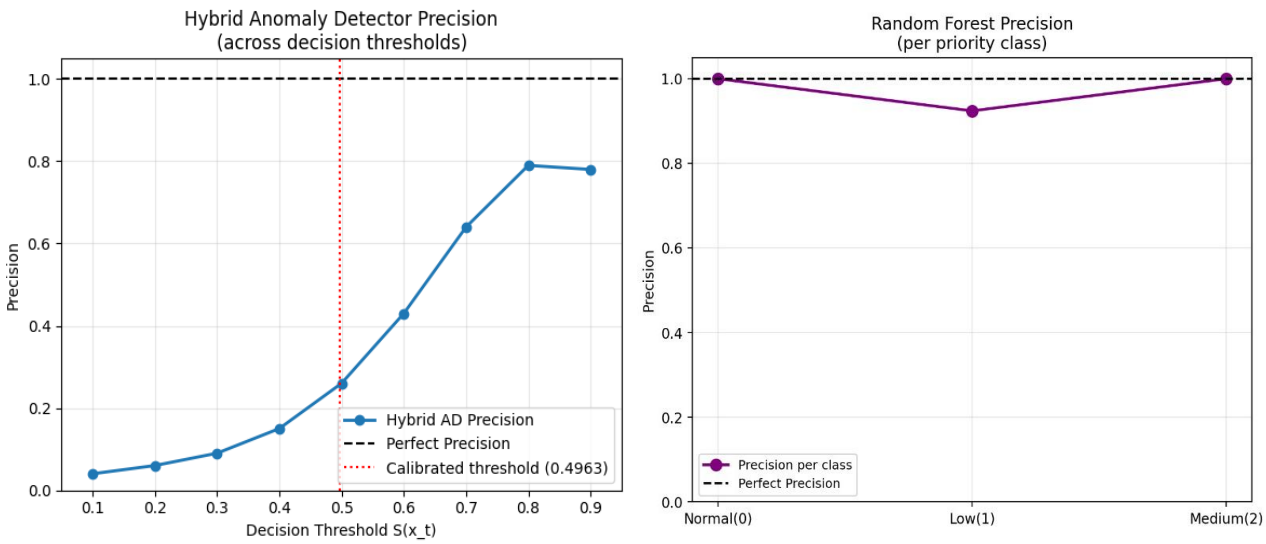

Figure 4-5: Precision Analysis

The left panel illustrates the relationship between the anomaly decision threshold and precision. As the threshold increases, fewer observations are classified as anomalous, resulting in a higher proportion of confident anomaly predictions, although this generally occurs at the expense of recall. The dotted line represents the calibrated operating threshold of 0.4963. The right panel shows that the Random Forest classifier maintains high precision across the three priority classes.

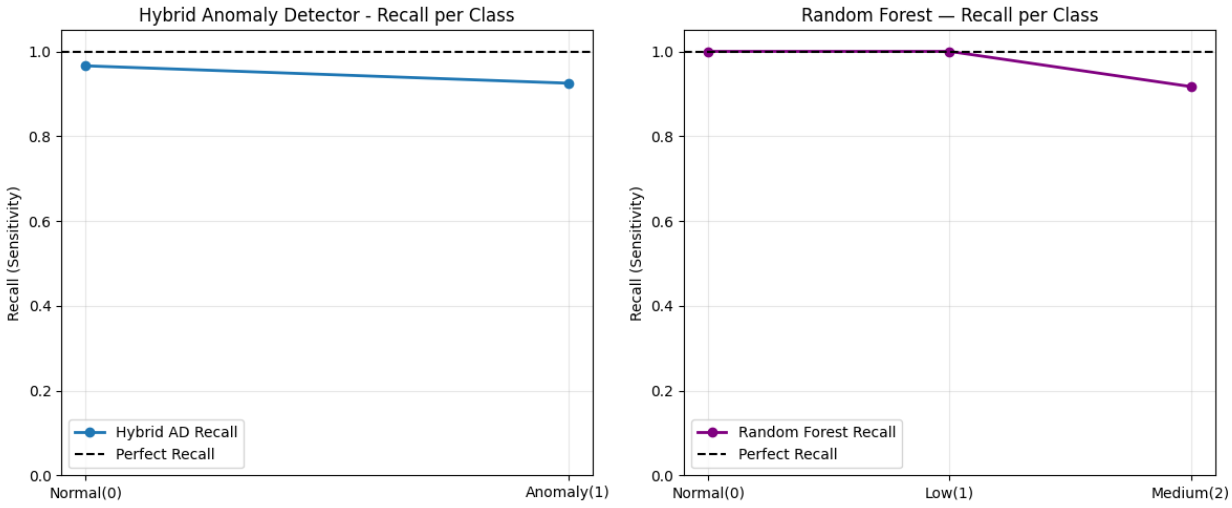

Figure 4-6: Recall Analysis

The Hybrid Anomaly Detector correctly identifies 92.5% of genuine anomalies while correctly recognizing 96.6% of normal observations. These results demonstrate that the selected operating point provides high sensitivity to abnormal system behaviour while maintaining a low rate of incorrect classification among normal readings. The Random Forest Priority Classifier achieves near-perfect recall for the Normal and Low priority classes, while recall decreases slightly to 91.7% for the less frequently represented Medium-priority class. This reduction is consistent with the lower number of samples available for the minority class and highlights the effect of class distribution on classifier performance.

The revised evaluation focuses on classification metrics that are directly aligned with the outputs of the two proposed models. The Hybrid Anomaly Detector achieved an Accuracy of 0.965, Recall of 0.925, Specificity of 0.966, and an AUC-ROC of 0.988, indicating strong overall discrimination between normal and anomalous observations and a high ability to detect genuine anomalies. Its Precision of 0.427 and F1-Score of 0.588 are comparatively lower, reflecting the recall-oriented operating threshold adopted to reduce the likelihood of missed anomalies, at the expense of generating additional false-positive alerts. The Random Forest Priority Classifier demonstrated consistently high performance, achieving an Accuracy of 0.991, Precision of 0.975, Recall of 0.971, and an F1-Score of 0.972. These results indicate highly reliable classification of priority levels with a balanced relationship between precision and recall. Specificity and AUC-ROC are not reported for the Random Forest model in Table 4-2 because the current evaluation presents the model as a multi-class priority classifier rather than a binary anomaly detector.

Table 4-1: Evaluation Results of Anomaly Detection Models

Metric	Hybrid AD	Random Forest
Accuracy	0.965	0.991
Precision	0.427	0.975
Recall (Sensitivity)	0.925	0.971
F1 Score	0.588	0.972
Specificity	0.966	N/A (multi-class)
AUC-ROC	0.988	N/A (multi-class)

The performance contrast between the two models reflects their different operational roles within the proposed system. The Hybrid Anomaly Detector prioritizes sensitivity to anomalous conditions, accepting a higher number of false-positive alerts in order to reduce missed anomalies. In contrast, the Random Forest Priority Classifier provides highly balanced and reliable classification of priority levels.

4.4 Feature Importance and Model Explainability

In real-world telemetry applications and AIOps environments, dealing with black-box anomaly detection is rarely sufficient. Operators require visibility into why an alert was triggered. The hybrid anomaly detector addresses this by utilizing two explainability vectors: The reconstruction error magnitude from the autoencoder, which highlights deviation from baseline behavior, and the isolation depth from the Isolation Forest, which flags structural rarity.

By mapping the final hybrid score against the operating threshold of 0.4963, the system provides a clear graduated scale from normal operation (0) to critical anomaly (1). When an anomaly score breaches this threshold, it informs the Random Forest Priority Classifier. Since features are preserved through the

pipeline, the system can alert operators to work on unseen failure modes, while also providing the contextual feature attribution needed for rapid triage.

In addition to detection accuracy and explainability, the pipeline keeps a low computational overhead and architectural complexity. Combining an Autoencoder and Isolation Forest rather than relying on resource-intensive deep neural networks achieves a low computational overhead and minimal latency. It ensures that the anomaly detection and the prioritization pipeline have the ability to scale effectively across distributed Kafka streams without introducing performance bottlenecks.

4.5 Case Studies and Sample Predictions

Case Study 1: Injected High-CPU Anomaly: During live testing, the system processed a message with CPU = 91%, Memory = 21%, Disk = 43%, Network = 4 (an intentionally injected anomaly). The hybrid score reached 1.0000, exceeding the calibrated threshold of 0.4963, and the system assigned Priority 1 with root cause “High CPU.” A corresponding alert was delivered through the Telegram bot within two seconds of the reading being published to Kafka.

Case Study 2: Normal Operating Reading: A representative normal reading (CPU = 34.7%, Memory = 87.6%, Disk = 5.2%, Network = 79.5, resampled from the real Alibaba trace) produced a hybrid score of 0.0665, well below the threshold, and was correctly classified as Normal with Priority 0, generating no alert.

These examples were drawn directly from the live system log rather than a manually staged local stress test, illustrating correct end-to-end behavior from Kafka ingestion through to Telegram alerting.

4.6 Comparative Analysis

The evaluation results were benchmarked against literature in cloud infrastructure anomaly detection, specifically comparing against EcoDefender [5] and the time-series anomaly model DAM [11].

While EcoDefender achieves a higher precision (0.940) and F1-score (0.920) through complex latent regularization, it does so at the expense of global class separation. In contrast, the proposed hybrid detector prioritizes a high recall of 0.925 and a low false-positive rate of 3.39%, intentionally accepting lower precision (0.427) to ensure that telemetry anomalies are never missed.

Similarly, when compared against Chen et al.'s deep attentive models (DAM), the results highlight how critically the baseline model depends on its attention architecture. While the full DAM model achieves near-perfect performance with a 99.53% recall and an 80.46% F-score, performance drops sharply to a 75.79% recall and a 62.5% F-score. This shows that the model's high effectiveness is heavily dependent on resource-intensive attention routing. The hybrid Autoencoder with Isolation Forest pipeline avoids computational overhead while delivering robust anomaly detection with a low complexity footprint in live streaming environments.

4.7 Chapter Summary

This chapter presented the dataset-selection process, the real-time processing pipeline built around Kafka, Prometheus, InfluxDB, and Grafana, and the evaluation of the hybrid anomaly detector. After calibrating its decision threshold from the training data distribution and retraining on the Alibaba Cluster Trace dataset, the system achieved a Recall of 92.5% and a False Positive Rate of 3.39% on a controlled evaluation set with known injected anomalies. The Random Forest triage

classifier reliably assigned priority levels, and end-to-end testing (28 automated tests across unit, integration, and deployment levels, 100% pass rate) confirmed correct system behavior. Real-time testing confirmed correct behavior under injected anomalous conditions, including live dashboard updates and automated Telegram alerts.

CHAPTER FIVE

5.1 Conclusion

This project presented the design, implementation, and evaluation of an intelligent server monitoring system combining real-time stream processing with a hybrid unsupervised machine learning model for anomaly detection and triage prioritization. The system ingests server metrics through Apache Kafka, computes a weighted hybrid anomaly score combining an Isolation Forest and an Autoencoder, classifies detected anomalies into priority levels using a Random Forest classifier, persists results in Prometheus and InfluxDB, visualizes system health through Grafana, delivers real-time Telegram alerts, and generates periodic historical reports through an independent Apache Spark batch layer.

The models were trained on production telemetry from the Alibaba Cluster Trace dataset. Calibrating the anomaly-detection threshold directly from the training data distribution instead of using a fixed manually chosen value. It produced a detection rate of 92.5% with a false-positive rate of 3.39% on a threshold of 0.4963. It elevated multi-class Random Forest triage accuracy to 99.1%, fulfilling the core proactive triage objectives.

5.2 Future Work

Although the system performed well, there are several areas that can be improved in future work. First, the dataset used for training can be expanded to include more types of anomalies such as network attacks or hardware degradations.

Second, the system can be extended to support multi-machine or distributed environments. A centralized dashboard that collects data from several machines would make the system more practical for large-scale production environmets.

Third, the visual dashboard can be improved to show more intuitive graphs, timelines, and correlations between metrics to make the system easier to interpret for non-technical users.

Fourth, predictive models can be added to estimate future system behavior. This can anticipate resource exhaustion before failures occur.

Finally, additional explainability tools can be integrated, such as SHAP values or LIME, to provide deeper insight into why certain predictions were made to better understand model decisions.

5.3 Summary

In summary, this project developed a functional intelligent server monitoring system that integrated Kafka-based data ingestion, unsupervised hybrid anomaly detection, and automated triage prioritization. Validated against real production telemetry and through 28 automated tests. Although opportunities for further development remain, the system provides a reliable, low-overhead contribution in automated server monitoring operations.

References

[1] A. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, “Borg, Omega, and Kubernetes,” Commun. ACM, vol. 59, no. 5, pp. 50–57, 2016.

[2] A. Deng and B. Hooi, “Graph neural network-based anomaly detection in multivariate time series,” in Proc. AAAI Conf. Artif. Intell., 2021, pp. 4027–4035.

[3] A. Di Stefano, A. Di Stefano, G. Morana, and D. Zito, “Prometheus and AIOps for the orchestration of cloud-native applications in Ananke,” in Proc. IEEE 30th Int. Conf. Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), Bayonne, France, 2021, pp. 27–32, doi: 10.1109/WETICE53228.2021.00017.

[4] A. Garg, W. Zhang, J. Samaran, R. Savitha, and C.-S. Foo, “An evaluation of anomaly detection and diagnosis in multivariate time series,” IEEE Trans. Neural Netw. Learn. Syst., vol. 33, no. 6, pp. 2508–2517, 2022.

[5] B. N. Altyb et al., “Energy consumption in wireless sensor node,” in Proc. Int. Conf. Computing, Control, Networking, Electronics Embedded Systems Eng. (ICCNEEE), Khartoum, Sudan, 2015, pp. 57–61.

[6] G. Liu, B. Huang, Z. Liang, M. Qin, H. Zhou, and Z. Li, “Microservices: Architecture, container, and challenges,” in Proc. IEEE 20th Int. Conf. Software Quality, Reliability and Security Companion (QRS-C), Macau, China, 2020, pp. 629–635, doi: 10.1109/QRS-C51114.2020.00107.

[7] H. Sigelman et al., “Dapper, a large-scale distributed systems tracing infrastructure,” 2010.

[8] H. Wang et al., “Anomaly detection for incident response at scale,” arXiv preprint arXiv:2404.16887, 2024.

[9] J. Kreps, N. Narkhede, and J. Rao, “Kafka: A distributed messaging system for log processing,” in Proc. NetDB, Athens, Greece, 2011, pp. 1–7.

[10] J. Rios, S. Jha, and L. Shwartz, “Localizing and explaining faults in microservices using distributed tracing,” in Proc. IEEE 15th Int. Conf. Cloud Computing (CLOUD), Barcelona, Spain, 2022, pp. 489–499, doi: 10.1109/CLOUD55607.2022.00072.

[11] J. Xu et al., “StreamAD: A cloud platform metrics-oriented benchmark for unsupervised online anomaly detection,” BenchCouncil Trans. Benchmarks, Standards Evaluations, vol. 3, no. 2, Art. no. 100121, 2023.

[12] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom, "Detecting spacecraft anomalies using LSTMs and nonparametric dynamic thresholding," in Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD), 2018, pp. 387–395.

[13] L. Chen, M. Xian, and J. Liu, "Monitoring system of OpenStack cloud platform based on Prometheus," in Proc. Int. Conf. Computer Vision, Image and Deep Learning (CVIDL), Chongqing, China, 2020, pp. 206–209, doi: 10.1109/CVIDL51233.2020.0-100.

[14] L. Decker, D. Leite, L. Giommi, and D. Bonacorsi, "Real-time anomaly detection in data centers for log-based predictive maintenance using an evolving fuzzy-rule-based approach," in Proc. IEEE Int. Conf. Fuzzy Systems (FUZZ-IEEE), 2020, pp. 1–8.

[15] L. Viola, E. Ronchieri, and C. Cavallaro, "Combining log files and monitoring data to detect anomaly patterns in a data center," Computers, vol. 11, no. 8, Art. no. 117, 2022.

[16] M. A. Ahmed et al., "Ensemble DL for improved diagnosis of respiratory diseases using YOLO and CheXNet on chest X-rays," Journal of Data Science and Intelligent Systems, 2026, doi: 10.47852/bonviewJDSIS62028092.

[17] M. Allam, N. Boujnah, N. E. O'Connor, and M. Liu, "Synthetic time series for anomaly detection in cloud microservices," in Proc. Int. Conf. Machine Learning, Optimization, and Data Science, 2024, pp. 419–433.

[18] M. B. Hassan et al., "Performance evaluation of uplink shared channel for cooperative relay based narrow band Internet of Things network," in Proc. Int. Conf. Business Analytics Technol. Security (ICBATS), 2022, pp. 1–7, doi: 10.1109/ICBATS54253.2022.9758935.

[19] M. Farshchi, J.-G. Schneider, I. Weber, and J. Grundy, "Experience report: Anomaly detection of cloud application operations using log and cloud metric correlation analysis," in Proc. IEEE 26th Int. Symp. Software Reliability Engineering (ISSRE), 2015, pp. 24–34.

[20] M. K. Hasan et al., "IoT-based warehouse management system," in Proc. Int. Conf. Cyber Resilience (ICCR), Dubai, UAE, 2022, pp. 1–6, doi: 10.1109/ICCR56254.2022.9995768.

[21] M. K. Hasan et al., "Smart grid cyber-physical systems: Components, vulnerabilities, mitigations, opportunities, and conceptual framework," Energy Reports, vol. 15, Art. no. 109255, 2026, doi: 10.1016/j.egyr.2026.109255.

[22] M. Lu, Z. Nie, and Y. Feng, “A transnational multi-cloud distributed monitoring data integration system,” in Proc. IEEE 6th Int. Conf. Computer and Communications (ICCC), 2020, pp. 1995–2000.

[23] M. M. Saeed et al., “LLM-integrated anomaly detection for IoT networks: Framework structure,” Baghdad Science Journal, vol. 23, no. 8, Art. no. 23, 2026, doi: 10.21123/2411-7986.5392.

[24] N. D. Keni and A. Kak, “Adaptive containerization for microservices in distributed cloud systems,” in Proc. IEEE 17th Annu. Consumer Commun. Networking Conf. (CCNC), Las Vegas, NV, USA, 2020, pp. 1–6, doi: 10.1109/CCNC46108.2020.9045634.

[25] N. Ding, H. Gao, H. Bu, and H. Ma, “RADM: Real-time anomaly detection in multivariate time series based on Bayesian network,” in Proc. IEEE Int. Conf. Smart Internet of Things (SmartIoT), 2018, pp. 129–134.

[26] N. Mehran et al., “ADApt: Edge device anomaly detection and microservice replica prediction,” in Proc. IEEE 9th Int. Conf. Fog and Edge Computing (ICFEC), 2025, pp. 6–10.

[27] O. Mart, C. Negru, F. Pop, and A. Castiglione, “Observability in Kubernetes cluster: Automatic anomalies detection using Prometheus,” in Proc. IEEE 22nd Int. Conf. High Performance Computing and Communications; IEEE 18th Int. Conf. Smart City; IEEE 6th Int. Conf. Data Science and Systems (HPCC/SmartCity/DSS), Yanuca Island, Cuvu, Fiji, 2020, pp. 565–570, doi: 10.1109/HPCC-SmartCity-DSS50907.2020.00071.

[28] O. O. Khalifa et al., “A stochastic approach for blockchain Internet of Things integration,” in Proc. Int. Conf. Advanced Computer Science and Information Systems (ICACSIS), 2022, pp. 219–224, doi: 10.1109/ICACSIS56558.2022.9923464.

[29] P. Kearney, M. Abdelsamea, X. Schmoor, F. Shah, and I. Vickers, “Combating alert fatigue in the security operations centre,” SSRN, 2023, doi: 10.2139/ssrn.4633965.

[30] R. A. Saeed et al., “Machine-to-machine smart grid synchronization for sustainable energy,” in Proc. 2nd Int. Conf. Energy Systems Technologies, Cairo, Egypt, 2013, pp. 163–176.

[31] S. Jamshidi, M. Bellaıche, and O. A. Wahab, “EcoDefender: Energy-efficient hybrid anomaly detection for IoT edge gateways,” arXiv preprint arXiv:2511.18235, 2025.

[32] S. N. Al-Jaradi et al., “Edge-aware RIS-assisted dynamic channel allocation with lightweight LLM decision agent for interference mitigation in low-altitude remote sensing networks,” IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 19, pp. 20824–20840, 2026, doi: 10.1109/JSTARS.2026.3698592.

[33] S. N. Al-Jaradi et al., “Optimized hybrid beamforming for RIS-assisted multi-user MIMO in 6G mmWave networks: A low-complexity approach to spectral efficiency and interference mitigation,” IEEE Open Journal of the Communications Society, vol. 7, pp. 2945–2958, 2026, doi: 10.1109/OJCOMS.2026.3674160.

[34] Y. Chen, M. Yan, D. Yang, X. Zhang, and Z. Wang, “Deep attentive anomaly detection for microservice systems with multimodal time-series data,” in Proc. IEEE Int. Conf. Web Services (ICWS), 2022, pp. 373–378.

[35] Y. Remil, A. Bendimerad, R. Mathonat, and M. Kaytoue, “AIOps solutions for incident management: Technical guidelines and a comprehensive literature review,” arXiv preprint arXiv:2404.01363, 2024, doi: 10.48550/arXiv.2404.01363.

[36] Y. Sun et al., “Multivariate time series anomaly detection based on pre-trained models with dual-attention mechanism,” in Proc. IEEE 35th Int. Symp. Software Reliability Engineering Workshops (ISSREW), 2024, pp. 73–78.

[37] Z. E. Ahmed et al., “An AI and IoT framework for dynamic optimization and sustainability in smart cities,” Al-Esraa University College Journal for Engineering Sciences, vol. 8, no. 13, Art. no. 16, 2026, doi: 10.70080/2790-7732.1103.

[38] Z. E. Ahmed et al., “Optimization procedure for intelligent Internet of Things applications,” in Proc. Int. Conf. Business Analytics Technol. Security (ICBATS), 2022, pp. 1–6, doi: 10.1109/ICBATS54253.2022.9759065.

[39] Z. Zeng et al., “Traceark: Towards actionable performance anomaly alerting for online service systems,” in Proc. IEEE/ACM 45th Int. Conf. Softw. Eng., Softw. Eng. Practice (ICSE-SEIP), 2023, pp. 258–269.

[40] E. S. Ali et al., “A stable energy balancing based clustering routing protocol for IoUT using meta-heuristic technique,” in Proc. 2024 IEEE 4th Int. Maghreb Meeting Conf. Sci. Techn. Automatic Control Comput. Eng. (MI-STA), Tripoli, Libya, 2024, pp. 433–439, doi: 10.1109/MI-STA61267.2024.10599658.

[41] M. Alhassan et al., “Optimization latency in IoT edge computing using generative digital twins,” in Proc. 2026 IEEE 5th Int. Maghreb Meeting Conf. Sci. Techn. Automatic Control Comput. Eng. (MI-STA), Sebha, Libya, 2026, pp. 205–210, doi: 10.1109/MI-STA68962.2026.11511171.

APPENDIX

APPENDIX A: Testing and Evaluation Summary

This appendix summarizes the automated testing carried out on the implemented system, referenced throughout Chapters Three and Four.

A.1 Unit Testing (pytest) - 16/16 passed

- test_get_severity_mapping (x5 priority values) - PASS
- test_root_cause_all_normal - PASS
- test_root_cause_high_cpu_only - PASS
- test_root_cause_multiple_simultaneous - PASS
- test_root_cause_boundary_exact_threshold_not_triggered - PASS
- test_root_cause_just_above_threshold_triggers - PASS
- test_hybrid_score_bounded_between_zero_and_one - PASS
- test_hybrid_score_weighted_combination_is_correct - PASS
- test_hybrid_score_respects_asymmetric_weights - PASS
- test_scaler_output_within_expected_range - PASS
- test_autoencoder_reconstructs_normal_input_with_low_error - PASS
- test_full_pipeline_extreme_input_flagged_as_anomaly - PASS

A.2 Integration Testing - 6/6 passed

- IT-01 Kafka producer-to-consumer message flow - PASS
- IT-02 Prometheus metrics endpoint availability - PASS
- IT-03 InfluxDB long-term data write - PASS
- IT-04 Grafana live dashboard update - PASS
- IT-05 Real Telegram alert delivery - PASS
- IT-06 Spark batch report generation (3,100+ real readings analysed) - PASS

A.3 System / Deployment Testing - 6/6 passed

- ST-01 WSL2 initialization on first Docker Desktop launch - PASS after fix (WSL update)

- ST-02 Container image pull under geographic registry restriction - PASS after fix (registry mirror)
- ST-03 docker-compose.yml YAML syntax validation - PASS after fix (volumes mapping corrected)
- ST-04 ml-consumer connectivity to Kafka from inside the container - PASS after fix (dual listeners)
- ST-05 Continuous message processing without silent stalls - PASS after fix (try/except with logging)
- ST-06 Running PySpark on Windows - PASS after fix (Java 21 + JAVA_HOME)

A.4 Overall Test Summary

Total automated tests: 28. Passed: 28. Failed: 0. Pass rate: 100%.

APPENDIX B: Source Code Listings

This appendix lists the complete, final source code of the implemented System, corresponding exactly to the version described and evaluated throughout Chapters Three and Four.

B.1 Kafka Producer -- Realistic Historical Data Replay (producer.py)

```
from kafka import KafkaProducer
import json
import time
import random
import numpy as np
import pandas as pd

while True:
    try:
        producer = KafkaProducer(
            bootstrap_servers='localhost:9092',
            value_serializer=lambda v: json.dumps(v).encode('utf-8')
        )
```

```
        print("Connected to Kafka")
        break
    except Exception as e:
        print("Waiting for Kafka...", e)
        time.sleep(5)

real_data = pd.read_csv("data/metrics_dataset.csv")[["cpu", "memory", "disk",
"network"]].values
print(f"  تم تحميل{len(real_data)} صفاً حقيقياً من بيانات التدريب لإعادة الاستخدام")

ANOMALY_PROBABILITY = 0.03

WINDOW_LEN = 40

while True:
    start_idx = random.randrange(0, len(real_data) - WINDOW_LEN - 1)
    for step in range(WINDOW_LEN):
        base = real_data[start_idx + step].copy()
        jitter = 1 + np.random.uniform(-JITTER, JITTER, size=4)
        row = base * jitter
        row[0:3] = np.clip(row[0:3], 0, 100)
        row[3] = max(row[3], 0)

        is_intentional_anomaly = random.random() < ANOMALY_PROBABILITY
        if is_intentional_anomaly:
            idx = random.randrange(4)
            row[idx] = row[idx] * random.uniform(2.0, 3.5)
            if idx in (0, 1, 2):
                row[idx] = min(row[idx], 100.0)
            else:
                row[idx] = min(row[idx], 250.0)

        data = {
            "cpu": round(float(row[0]), 1),
            "memory": round(float(row[1]), 1),
            "disk": round(float(row[2]), 1),
            "network": round(float(row[3]), 1),
        }

        producer.send('logs', data)
        print("Sent:", data, " ANOMALY INJECTED" if is_intentional_anomaly
else "")
        time.sleep(2)
```

B.2 Model Training Script (train_models.py)

```
import json
import pandas as pd
import numpy as np
import joblib

from sklearn.ensemble import IsolationForest, RandomForestClassifier
from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras.models import Model, Sequential
from tensorflow keras. layers import Input, Dense, LSTM

#
https://zenodo.org/records/14564935/files/machine_usage_days_1_to_8_grouped_30
0_seconds.csv
# ضعه في data/alibaba_machine_usage.csv
raw = pd.read_csv("data/alibaba_machine_usage.csv")

print("Dataset loaded:", len(raw))

before = len(raw)
raw = raw[(raw["disk_io_percent"] != -1) & (raw["disk_io_percent"] !=
101)].reset_index(drop=True)
print(f"تم استبعاد {before - len(raw)} صفاً بقيم disk (101 أو 1-) غير صالحة")

data = pd.DataFrame({
    "cpu": raw["cpu_util_percent"],
    "memory": raw["mem_util_percent"],
    "disk": raw["disk_io_percent"],
    "network": raw["net_in"] + raw["net_out"],
})

features = ["cpu", "memory", "disk", "network"]

# ============================
# CREATE ANOMALY LABEL (if missing)
# ============================
if "anomaly" not in datacolumns:
    def compute_anomaly(row):
        if row["cpu"] > 95 or row["memory"] > 95:
            return 1
        return 0
    data["anomaly"] = data.apply(compute_anomaly, axis=1)

# ============================
# SCALER
# ============================
```

```
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(data[features])

joblib.dump(scaler, "models/scaler.pkl")

# ============================
# 1. ISOLATION FOREST
# ============================
iso = IsolationForest(contamination=0.02, random_state=42)
iso.fit(X_scaled)

joblib.dump(iso, "models/isolation_forest.pkl")
print("Isolation Forest trained")

# ============================
# 2. AUTOENCODER
# ============================
input_dim = X_scaled.shape[1]

input_layer = Input(shape=(input_dim,))
encoded = Dense(8, activation="relu")(input_layer)
encoded = Dense(4, activation="relu")(encoded)

decoded = Dense(8, activation="relu")(encoded)
decoded = Dense(input_dim, activation="sigmoid")(decoded)

autoencoder = Model(inputs=input_layer, outputs=decoded)
autoencoder.compile(optimizer="adam", loss="mse")

autoencoder.fit(X_scaled, X_scaled, epochs=20, batch_size=32)

autoencoder.save("models/autoencoder.h5")
print("Autoencoder trained")

# ============================
# AUTOENCODER ERROR
# ============================
pred = autoencoder.predict(X_scaled)
mse = np.mean(np.square(X_scaled - pred), axis=1)

data["ae_error"] = mse

# ============================
# 3. HYBRID ANOMALY SCORE S(x_t)
# ============================
if_raw = -iso.decision_function(X_scaled)
ae_raw = mse
```

```
if_min, if_max = float(if_raw.min()), float(if_raw.max())
ae_min, ae_max = float(ae_raw.min()), float(ae_raw.max())

if_norm = (if_raw - if_min) / (if_max - if_min + 1e-9)
ae_norm = (ae_raw - ae_min) / (ae_max - ae_min + 1e-9)

WEIGHT_AE = 0.5
WEIGHT_IF = 0.5

hybrid_score = WEIGHT_AE * ae_norm + WEIGHT_IF * if_norm
data["hybrid_score"] = hybrid_score

threshold_S = float(np.percentile(hybrid_score, 98))

hybrid_config = {
    "if_min": if_min,
    "if_max": if_max,
    "ae_min": ae_min,
    "ae_max": ae_max,
    "weight_ae": WEIGHT_AE,
    "weight_if": WEIGHT_IF,
    "threshold_S": threshold_S,
}

with open("models/hybrid_score_config.json", "w") as f:
    json.dump(hybrid_config, f, indent=2, ensure_ascii=False)

print(f"Hybrid Score config saved — threshold_S={threshold_S:.4f}")

# ============================
# 4. RANDOM FOREST (PRIORITY)
# ============================
p95 = data[features].quantile(0.95)
p99 = data[features].quantile(0.99)
def priority_label(row):
if any(row[f] > p99[f] for f in features):
return 2
elif any(row[f] > p95[f] for f in features):
return 1
return 0
data["priority"] = data.apply(priority_label, axis=1)احذف هذا الكود بالكامل:
النسخة المصححة الفعلية(والصق مكانه هذا الكود):

with open("models/priority_thresholds.json", "w") as f:
json.dump({"p95": p95.to_dict(), "p99": p99.to_dict()}, f, indent=2,
ensure_ascii=False)
```

```
rf = RandomForestClassifier(n_estimators=100)
rf.fit(data[features + ["anomaly"]], data["priority"])

joblib.dump(rf, "models/rf_model.pkl")
print("Random Forest trained")

# ============================
# 5. LSTM (PREDICTION)
# ============================
sequence_length = 5

def create_sequences(data, seq_len):
    X, y = [], []
    for i in range(len(data) - seq_len):
        X.append(data[i:i+seq_len])
        y.append(data[i+seq_len])
    return np.array(X), np.array(y)

X_seq, y_seq = create_sequences(X_scaled, sequence_length)

lstm = Sequential([
    LSTM(32, input_shape=(sequence_length, len(features))),
    Dense(len(features))
])

lstm.compile(optimizer="adam", loss="mse")

lstm.fit(X_seq, y_seq, epochs=10, batch_size=16)

lstm.save("models/lstm_model.h5")

print("LSTM trained")

# ============================
# SAVE DATA BACK
# ============================
data.to_csv("data/metrics_dataset.csv", index=False)

print("ALL MODELS TRAINED SUCCESSFULLY ")
```

B.3 ML Consumer -- Configuration, Models, and Forecasting Functions (consumer.py, part 1)

```
import json
import os
import traceback
```

```
import joblib
import numpy as np
import pandas as pd
import csv
import requests
import shap
from collections import deque
from datetime import datetime, timezone

from kafka import KafkaConsumer
from tensorflow keras.models import load_model
from prometheus_client import start_http_server, Gauge
from influxdb_client import InfluxDBClient, Point
from influxdb_client.client.write_api import SYNCHRONOUS

# ------------------------------
# Load Models
# ------------------------------
BASE_DIR = os.path.dirname(os.path.abspath(__file__))

iso_model = joblib.load(os.path.join(BASE_DIR, "isolation_forest.pkl"))
rf_model = joblib.load(os.path.join(BASE_DIR, "rf_model.pkl"))
scaler = joblib.load(os.path.join(BASE_DIR, "scaler.pkl"))
autoencoder = load_model(os.path.join(BASE_DIR, "autoencoder.h5"),
compile=False)

\LSTM_SEQ_LEN = 5
lstm_model = None
lstm_path = os.path.join(BASE_DIR, "lstm_model.h5")
if os.pathexists(lstm_path):
    lstm_model = load_model(lstm_path, compile=False)
else:
    print("lstm_model.h5 غير موجود — سيتم تخطي تنبؤ LSTM ( التحذير المبكر بالانحدار الخطي يستمر بشكل
مستقل).")

with open(os.path.join(BASE_DIR, "hybrid_score_config.json")) as f:
    HYBRID = json.load(f)

# ------------------------------
# Prometheus Metrics (ONLY ONCE)
# ------------------------------
cpu_gauge = Gauge("cpu_usage", "CPU Usage")
memory_gauge = Gauge("memory_usage", "Memory Usage")
disk_gauge = Gauge("disk_usage", "Disk Usage")
network_gauge = Gauge("network_usage", "Network Usage")
anomaly_gauge = Gauge("anomaly", "Anomaly Detected")
priority_gauge = Gauge("priority", "Priority Level")
```

```
ae_error_gauge = Gauge("ae_error", "Autoencoder Error")
hybrid_score_gauge = Gauge("hybrid_score", "Weighted Hybrid Anomaly Score
S(x_t)")
predicted_score_trend_gauge = Gauge("predicted_hybrid_score_trend",
"Forecasted Hybrid Score via linear trend extrapolation")
predicted_score_lstm_gauge = Gauge("predicted_hybrid_score_lstm", "Forecasted
Hybrid Score via LSTM (evaluation/logging only)")
early_warning_gauge = Gauge("early_warning", "Early warning raised from trend
forecast (1) or not (0)")
root_cause_gauge = Gauge(
    "root_cause",
    "Root Cause",
    ["cause"]
)

start_http_server(8000)
print("Prometheus running at http://localhost:8000")

INFLUX_URL = os.environ.get("INFLUXDB_URL", "http://influxdb:8086")
INFLUX_TOKEN = os.environ.get("INFLUXDB_TOKEN")
INFLUX_ORG = os.environ.get("INFLUXDB_ORG", "system-pulse-ai")
INFLUX_BUCKET = os.environ.get("INFLUXDB_BUCKET", "server-metrics")

if not INFLUX_TOKEN:
    print("INFLUXDB_TOKEN غير مضبوط — سيتم تخطي الكتابة في InfluxDB.")
    influx_write_api = None
else:
    influx_client = InfluxDBClient(url=INFLUX_URL, token=INFLUX_TOKEN,
org=INFLUX_ORG)
    influx_write_api = influx_client.write_api(write_options=SYNCHRONOUS)

# ----------------------------
# Kafka Consumer
# ----------------------------
consumer = KafkaConsumer(
    "logs",
    bootstrap_servers="kafka:9093",
    value_deserializer=lambda x: json.loads(x.decode("utf-8"))
)

print("Kafka Consumer Started...")

# ----------------------------
# Root Cause Function
# ----------------------------
def detect_root_cause(cpu, memory, disk, network):
    causes = []
```

```
    if cpu > 80:
        causes.append("High CPU")
    if memory > 80:
        causes.append("High Memory")
    if disk > 85:
        causes.append("Disk Full")
    if network > 4000:
        causes.append("Network Spike")

    if not causes:
        return "Normal"

    return ", ".join(causes)

# ----------------------------
# Root Cause Mapping
# ----------------------------
root_cause_map = {
    "Normal": 0,
    "High CPU": 1,
    "High Memory": 2,
    "Disk Full": 3,
    "Network Spike": 4
}

# ----------------------------
# CSV Setup
# ----------------------------
file_path = os.path.join(BASE_DIR, "../data/live_metrics.csv")
file = open(file_path, "a", newline="")
writer = csv.writer(file)

# Write header once
if os.stat(file_path).st_size == 0:
    writer.writerow([
        "timestamp", "cpu", "memory", "disk", "network",
        "anomaly", "priority", "mse", "hybrid_score", "root_cause",
        "predicted_score_trend", "early_warning", "predicted_score_lstm",
        "shap_explanation"
    ])

# ----------------------------
# Telegram Alerting
# ----------------------------
TELEGRAM_TOKEN = os.environ.get("TELEGRAM_BOT_TOKEN")
TELEGRAM_CHAT_ID = os.environ.get("TELEGRAM_CHAT_ID")
```

```
if not TELEGRAM_TOKEN or not TELEGRAM_CHAT_ID:
    print("TELEGRAM_BOT_TOKEN / TELEGRAM_CHAT_ID غير مضبوطة — سيتم تخطي إرسال تنبيهات
Telegram.")

def send_telegram(msg):
    if not TELEGRAM_TOKEN or not TELEGRAM_CHAT_ID:
        return
    url = f"https://api.telegram.org/bot{TELEGRAM_TOKEN}/sendMessage"
    try:
        requests.post(url, data={"chat_id": TELEGRAM_CHAT_ID, "text": msg},
timeout=5)
    except Exception as e:
        print(f"فشل إرسال تنبيه Telegram: {e}")

# ----------------------------
# Helper
# ----------------------------
def get_severity(priority):
    if priority == 0:
        return "normal"
    elif priority == 1:
        return "low"
    elif priority == 2:
        return "medium"
    else:
        return "critical"

EARLY_WARNING_COOLDOWN_SECONDS =
int(os.environ.get("EARLY_WARNING_COOLDOWN_SECONDS", "300"))
last_alert_sent_at = None
last_early_warning_sent_at = None
suppressed_alert_count = 0
suppressed_early_warning_count = 0

recent_readings = deque(maxlen=LSTM_SEQ_LEN)
FORECAST_STEPS_AHEAD = 3  # عدد الرسائل القادمة التي يُسقَط عليها الاتجاه


def compute_hybrid_score(feature_row):
    """يحسب S(x_t) لأي متجه خصائص [cpu, memory, disk, network] — يُستخدم
    للقراءة الحالية وللقراءات المتوقَّعة (تنبؤ الاتجاه/LSTM) على حدٍّ سواء."""
    row_df = pdDataFrame([feature_row], columns=["cpu", "memory", "disk",
"network"])
    row_scaled = np.asarray(scaler.transform(row_df))
    if_raw_ = -iso_model.decision_function(row_scaled)[0]
    pred_ = autoencoder.predict(row_scaled, verbose=0)
```

```
    ae_raw_ = float(np.mean(np.square(row_scaled - pred_)))
    if_norm_ = min(max((if_raw_ - HYBRID["if_min"]) / (HYBRID["if_max"] -
HYBRID["if_min"] + 1e-9), 0), 1)
    ae_norm_ = min(max((ae_raw_ - HYBRID["ae_min"]) / (HYBRID["ae_max"] -
HYBRID["ae_min"] + 1e-9), 0), 1)
    return HYBRID["weight_ae"] * ae_norm_ + HYBRID["weight_if"] * if_norm_,
row_scaled


def forecast_trend(readings, steps_ahead):

    """التحذير المبكر للآلية الفعلية وهو (أعلاه)."""
    arr = np.array(readings)  # shape (N, 4)
    x = np.arange(len(arr))
    forecast = []
    for col in range(arr.shape[1]):
        slope, intercept = np.polyfit(x, arr[:, col], 1)
        forecast.append(slope * (len(arr) - 1 + steps_ahead) + intercept)
        forecast[0] = min(max(forecast[0], 0), 100)
    forecast[1] = min(max(forecast[1], 0), 100)
    forecast[2] = min(max(forecast[2], 0), 100)
    forecast[3] = max(forecast[3], 0)
    return forecast


def forecast_lstm(readings):

    if lstm_model is None:
        return None
    arr = np.array(readings)
    arr_scaled = scalertransform(arr)
    pred_scaled = lstm_model.predict(arr_scaled.reshape(1, LSTM_SEQ_LEN, 4),
verbose=0)[0]
    pred_real = scaler.inverse_transform(pred_scaled.reshape(1, -1))[0]
    return list(pred_real)


FEATURE_NAMES = ["cpu", "memory", "disk", "network"]


def explain_anomaly(scaled_row, reconstructed_row):

    shap_vals = explainer_if.shap_values(scaled_row)[0]
    per_feature_ae_error = np.square(scaled_row - reconstructed_row)[0]

    shap_pct = np.abs(shap_vals) / (np.abs(shap_vals).sum() + 1e-9)
    ae_pct = per_feature_ae_error / (per_feature_ae_error.sum() + 1e-9)
```

```
    combined = 0.5 * ae_pct + 0.5 * shap_pct

    ranked = sorted(zip(FEATURE_NAMES, combined), key=lambda x: -x[1])
    (>50%)سببين أهم وإلا ،
    if ranked[0][1] > 0.5:
        summary = f"{ranked[0][0]} ({ranked[0][1]*100:.0f}%)"
    else:
        summary = f"{ranked[0][0]} ({ranked[0][1]*100:.0f}%), {ranked[1][0]}
({ranked[1][1]*100:.0f}%)"
    return summary, {k: round(float(v), 4) for k, v in ranked}
```

B.4 ML Consumer -- Main Processing Loop (consumer.py, part 2)

```
# ----------------------------
# MAIN LOOP
# ----------------------------
for message in consumer:
    try:
        data = message.value

        cpu = data.get("cpu", 0)
        memory = data.get("memory", 0)
        disk = data.get("disk", 50)
        network = data.get("network", 0)

        recent_readings.append([cpu, memory, disk, network])

        df = pdDataFrame([[cpu, memory, disk, network]],
                         columns=["cpu", "memory", "disk", "network"])

        if_norm = min(max((if_raw - HYBRID["if_min"]) / (HYBRID["if_max"] -
HYBRID["if_min"] + 1e-9), 0), 1)
        ae_norm = min(max((ae_raw - HYBRID["ae_min"]) / (HYBRID["ae_max"] -
HYBRID["ae_min"] + 1e-9), 0), 1)

        hybrid_score = HYBRID["weight_ae"] * ae_norm + HYBRID["weight_if"] *
if_norm
        anomaly = 1 if hybrid_score > HYBRID["threshold_S"] else 0

        # Priority
        X_input = pd.DataFrame([{
            "cpu": cpu,
            "memory": memory,
            "disk": disk,
            "network": network,
            "anomaly": anomaly
        }])
```

```
        priority = int(rf_model.predict(X_input)[0])
        severity = get_severity(priority)

        # Root Cause
        if anomaly == 1:
            root_cause = detect_root_cause(cpu, memory, disk, network)
        else:
            root_cause = "Normal"

        predicted_score_trend = 0.0
        predicted_score_lstm_val = None
        early_warning = 0

        if len(recent_readings) == LSTM_SEQ_LEN:
            forecast_row = forecast_trend(list(recent_readings),
FORECAST_STEPS_AHEAD)
            predicted_score_trend, _ = compute_hybrid_score(forecast_row)

            lstm_forecast_row = forecast_lstm(list(recent_readings))
            if lstm_forecast_row is None:
                predicted_score_lstm_val, _ =
compute_hybrid_score(lstm_forecast_row)

            if anomaly == 0 and predicted_score_trend > HYBRID["threshold_S"]:
                early_warning = 1

        cpu_gauge.set(cpu)
        memory_gauge.set(memory)
        disk_gauge.set(disk)
        network_gauge.set(network)
        anomaly_gauge.set(anomaly)
        priority_gauge.set(priority)
        ae_error_gauge.set(mse)
        hybrid_score_gauge.set(hybrid_score)
        predicted_score_trend_gauge.set(predicted_score_trend)
        if predicted_score_lstm_val is not None:
            predicted_score_lstm_gauge.set(predicted_score_lstm_val)
        early_warning_gauge.set(early_warning)

        for cause in ["Normal", "High CPU", "High Memory", "Disk Full",
"Network Spike"]:
            root_cause_gauge.labels(cause=cause).set(0)
        root_cause_gauge.labels(cause=root_cause).set(1)

        writer.writerow([
```

```
            datetime.now(timezone.utc).isoformat(), cpu, memory, disk,
network,
            anomaly, priority, mse, hybrid_score, root_cause,
            round(predicted_score_trend, 4), early_warning,
            round(predicted_score_lstm_val, 4) if predicted_score_lstm_val is
not None else "",
            shap_summary
        ])
        file.flush()

            if influx_write_api is None:
            point = (
                Point("server_metrics")
                .tag("root_cause", root_cause)
                .field("cpu", float(cpu))
                .field("memory", float(memory))
                .field("disk", float(disk))
                .field("network", float(network))
                .field("anomaly", int(anomaly))
                .field("priority", int(priority))
                .field("ae_error", float(mse))
                .field("hybrid_score", float(hybrid_score))
                .field("predicted_score_trend", float(predicted_score_trend))
                .field("early_warning", int(early_warning))
                .tag("shap_top_cause", shap_summary)
            )
            influx_write_api.write(bucket=INFLUX_BUCKET, org=INFLUX_ORG,
record=point)

        print(f"""
CPU:{cpu} MEM:{memory} DISK:{disk} NET:{network}
→ Hybrid Score:{hybrid_score:.4f} (عتبة:{HYBRID['threshold_S']:.4f})
Anomaly:{anomaly} Priority:{priority}
→ Root Cause: {root_cause}
→ تفسير SHAP الكمي: {shap_summary}
→ توقّع الاتجاه (+{FORECAST_STEPS_AHEAD} خطوات):{predicted_score_trend:.4f} {' EARLY
WARNING' if early_warning else ''}
""")

        if severity != "normal" and root_cause != last_alert:
            now_ts = datetime.now(timezone.utc)
            cooldown_elapsed = (last_alert_sent_at is None) or (
                (now_ts - last_alert_sent_at).total_seconds() >=
ALERT_COOLDOWN_SECONDS
            )
            should_send = cooldown_elapsed or severity == "critical"
```

```
            if should_send:
                suppressed_note = (
                    f"\n(+ {suppressed_alert_count} حالة
                    if suppressed_alert_count > 0 else                    )
                if severity == "low":
                    send_telegram(f" Low Alert\nCPU:{cpu} MEM:{memory}\n السبب
الأكثر مساهمة: {shap_summary}{suppressed_note}")
                elif severity == "medium":
                    send_telegram(f" Medium Alert\nRoot Cause:
{root_cause}\nالسبب الأكثر مساهمة: {shap_summary}{suppressed_note}")
                elif severity == "critical":
                    send_telegram(f" CRITICAL ALERT\n {root_cause}\nCPU:{cpu}
MEM:{memory}\nالسبب الأكثر مساهمة: {shap_summary}{suppressed_note}")
                last_alert_sent_at = now_ts
                suppressed_alert_count = 0
            else:
                suppressed_alert_count += 1
            last_alert = root_cause
        elif severity == "normal":
            last_alert = None

        if early_warning == 1 and last_early_warning != "warned":
            now_ts2 = datetime.now(timezone.utc)
            ew_cooldown_elapsed = (last_early_warning_sent_at is None) or (
                (now_ts2 - last_early_warning_sent_at).total_seconds() >=
EARLY_WARNING_COOLDOWN_SECONDS
            )
            if ew_cooldown_elapsed:
                suppressed_note_ew = (
                    f"\n(+ {suppressed_early_warning_count} تحذيراً مبكراً إضافياً قُمع خلال
فترة التهدئة)"
                    if suppressed_early_warning_count > 0 else ""
                )
                send_telegram(
                    f" EARLY WARNING\n الاتجاه الحالي يشير لاحتمال شذوذ خلال
{FORECAST_STEPS_AHEAD} قراءات قادمة\n"
                    f"القراءة الحالية: CPU:{cpu} MEM:{memory} DISK:{disk}
NET:{network}\n"
                    f"درجة الخطورة المتوقَّعة: {predicted_score_trend:.4f} (عتبة:
{HYBRID['threshold_S']:.4f}){suppressed_note_ew}"
                )
                last_early_warning_sent_at = now_ts2
                suppressed_early_warning_count = 0
            else:
                suppressed_early_warning_count += 1
            last_early_warning = "warned"
        elif early_warning == 0:
```

```
            last_early_warning = None

    except Exception:
        print("خطأ أثناء معالجة رسالة — التفاصيل الكاملة أدناه:")
        traceback.print_exc()
```

B.5 Automated Test Suite (test_system_pulse_ai.py)

```
import json
import os
import numpy as np
import joblib
import pytest
from tensorflow. keras models import load_model

MODELS_DIR = os.environ.get("MODELS_DIR", "models")


def get_severity(priority):
    if priority == 0:
        return "normal"
    elif priority == 1:
        return "low"
    elif priority == 2:
        return "medium"
    else:
        return "critical"


def detect_root_cause(cpu, memory, disk, network):
    causes = []
    if cpu > 80:
        causes.append("High CPU")
    if memory > 80:
        causes.append("High Memory")
    if disk > 85:
        causes.append("Disk Full")
    if network > 4000:
        causes.append("Network Spike")
    if not causes:
        return "Normal"
    return ", ".join(causes)


def compute_hybrid_score(if_raw, ae_raw, cfg):
    if_norm = min(max((if_raw - cfg["if_min"]) / (cfg["if_max"] -
cfg["if_min"] + 1e-9), 0), 1)
    ae_norm = min(max((ae_raw - cfg["ae_min"]) / (cfg["ae_max"] -
cfg["ae_min"] + 1e-9), 0), 1)
```

```
    return cfg["weight_ae"] * ae_norm + cfg["weight_if"] * if_norm


@pytest.fixture(scope="module")
def models():
    scaler = joblib.load(os.path.join(MODELS_DIR, "scaler.pkl"))
    iso = joblib.load(os.path.join(MODELS_DIR, "isolation_forest.pkl"))
    ae = load_model(os.path.join(MODELS_DIR, "autoencoder.h5"), compile=False)
    with open(os.path.join(MODELS_DIR, "hybrid_score_config.json")) as f:
        cfg = jsonload(f)
    return {"scaler": scaler, "iso": iso, "ae": ae, "cfg": cfg}


# ---------------------------------------------------------------------
# get_severity
# ---------------------------------------------------------------------
@pytest.mark.parametrize("priority,expected", [
    (0, "normal"),
    (1, "low"),
    (2, "medium"),
    (3, "critical"),
    (99, "critical"),
])
def test_get_severity_mapping(priority, expected):
    assert get_severity(priority) == expected


# ---------------------------------------------------------------------
# detect_root_cause
# ---------------------------------------------------------------------
def test_root_cause_all_normal():
    assert detect_root_cause(cpu=30, memory=50, disk=40, network=100) ==
"Normal"


def test_root_cause_high_cpu_only():
    assert detect_root_cause(cpu=90, memory=50, disk=40, network=100) == "High
CPU"


def test_root_cause_multiple_simultaneous():
    result = detect_root_cause(cpu=90, memory=90, disk=40, network=100)
    assert result == "High CPU, High Memory"


def test_root_cause_boundary_exact_threshold_not_triggered():
```

```
        assert detect_root_cause(cpu=80, memory=80, disk=85, network=4000) ==
"Normal"


def test_root_cause_just_above_threshold_triggers():
    assert detect_root_cause(cpu=80.1, memory=50, disk=40, network=100) ==
"High CPU"


# ---------------------------------------------------------------------
# Hybrid Score
# ---------------------------------------------------------------------
def test_hybrid_score_bounded_between_zero_and_one():
    cfg = {"if_min": 0, "if_max": 1, "ae_min": 0, "ae_max": 1, "weight_ae":
0.5, "weight_if": 0.5}

    score = compute_hybrid_score(if_raw=999, ae_raw=999, cfg=cfg)
    assert 0.0 <= score <= 1.0
    score_negative = compute_hybrid_score(if_raw=-999, ae_raw=-999, cfg=cfg)
    assert 0.0 <= score_negative <= 1.0


def test_hybrid_score_weighted_combination_is_correct():
    cfg = {"if_min": 0, "if_max": 10, "ae_min": 0, "ae_max": 10, "weight_ae":
0.5, "weight_if": 0.5}
    # if_raw=5 -> if_norm=0.5 ، ae_raw=5 -> ae_norm=0.5 => score = 0.5*0.5 +
0.5*0.5 = 0.5
    score = compute_hybrid_score(if_raw=5, ae_raw=5, cfg=cfg)
    assert abs(score - 0.5) < 1e-9


def test_hybrid_score_respects_asymmetric_weights():
    cfg = {"if_min": 0, "if_max": 10, "ae_min": 0, "ae_max": 10, "weight_ae":
0.8, "weight_if": 0.2}
    # ae_norm=1.0, if_norm=0.0 => score = 0.8*1.0 + 0.2*0.0 = 0.8
    score = compute_hybrid_score(if_raw=0, ae_raw=10, cfg=cfg)
    assert abs(score - 0.8) < 1e-9


def test_scaler_output_within_expected_range(models):
    """قيم داخل نطاق التدريب المعتاد يجب أن تُطبَّع إلى ما يقارب [0,1]."""
    sample = np.array([[35.0, 86.0, 7.0, 65.0]])
    scaled = models["scaler"].transform(sample)
    assert scaled.shape == (1, 4)
        assert np.all(scaled > -0.5) and np.all(scaled < 1.5)

def test_autoencoder_reconstructs_normal_input_with_low_error(models):
```

```
        sample = np.array([[35.0, 86.0, 7.0, 65.0]])
    scaled = models["scaler"].transform(sample)
    reconstructed = models["ae"].predict(scaled, verbose=0)
    mse = float(np.mean(np.square(scaled - reconstructed)))
    assert mse < models["cfg"]["ae_max"]

def test_full_pipeline_extreme_input_flagged_as_anomaly(models):
    sample = np.array([[100.0, 100.0, 100.0, 300.0]])
    scaled = np.asarray(models["scaler"].transform(sample))
    if_raw = -models["iso"].decision_function(scaled)[0]
    pred = models["ae"].predict(scaled, verbose=0)
    ae_raw = float(np.mean(np.square(scaled - pred)))
    score = compute_hybrid_score(if_raw, ae_raw, models["cfg"])
    assert score > models["cfg"]["threshold_S"]
```

B.6 Traditional Alerting Baseline Comparison (compare_traditional_alerting.py)

```
import json, joblib, warnings
import numpy as np
import pandas as pd
from tensorflow. keras. models import load_model

warnings.filterwarnings("ignore")
scaler = joblibload("models/scaler.pkl")
iso = joblibload("models/isolation_forest.pkl")
ae = load_model("models/autoencoder.h5", compile=False)
with open("models/hybrid_score_config.json") as f:
    cfg = jsonload(f)

real_data = pd.read_csv("data/metrics_dataset.csv")[["cpu","memory", "disk",
"network"]].values

np.randomseed(42)
N = 2000
JITTER = 0.03
ANOMALY_PROB = 0.03

rows, labels = [], []
for _ in range(N):
    base = real_data[np.random.randint(len(real_data))].copy()
    jitter = 1 + np.random.uniform(-JITTER, JITTER, size=4)
    row = base * jitter
    row[0:3] = np.clip(row[0:3], 0, 100); row[3] = max(row[3], 0)
    is_anom = np.randomrandom() < ANOMALY_PROB
    if is_anom:
```

```
        idx = np.random.randint(4)
        row[idx] = row[idx] * np.random.uniform(2.0, 3.5)
        row[idx] = min(row[idx], 100.0) if idx in (0,1,2) else min(row[idx],
250.0)
    rows.append(row); labels.append(is_anom)
X = np.array(rows); labels = np.array(labels)

X_df = pdDataFrame(X, columns=["cpu", "memory", "disk", "network"])
scaled = scalertransform(X_df)
if_raw = -iso.decision_function(scaled)
pred = ae.predict(scaled, verbose=0, batch_size=256)
ae_raw = np.mean(np.square(scaled - pred), axis=1)

if_norm = np.clip((if_raw - cfg["if_min"]) / (cfg["if_max"] - cfg["if_min"] +
1e-9), 0, 1)
ae_norm = np.clip((ae_raw - cfg["ae_min"]) / (cfg["ae_max"] - cfg["ae_min"] +
1e-9), 0, 1)
hybrid_scores = cfg["weight_ae"] * ae_norm + cfg["weight_if"] * if_norm
hybrid_flags = hybrid_scores > cfg["threshold_S"]


DEFAULT_THRESH = {"cpu": 80, "memory": 80, "disk": 85, "network": 90}
default_flags = ((X[:,0] > DEFAULT_THRESH["cpu"]) | (X[:,1] >
DEFAULT_THRESH["memory"]) |
                 (X[:,2] > DEFAULT_THRESH["disk"]) | (X[:,3] >
DEFAULT_THRESH["network"]))


TUNED_THRESH = {c: float(np.percentile(real_data[:, i], 95)) for i,c in
enumerate(["cpu", "memory", "disk", "network"])}
tuned_flags = ((X[:,0] > TUNED_THRESH["cpu"]) | (X[:,1] >
TUNED_THRESH["memory"]) |
               (X[:,2] > TUNED_THRESH["disk"]) | (X[:,3] >
TUNED_THRESH["network"]))

def report(name, flags):
    tp = int(np.sum(flags & labels)); fp = int(np.sum(flags & ~labels))
    fn = int(np.sum(~flags & labels)); tn = int(np.sum(~flags & ~labels))
    recall = tp/(tp+fn) if (tp+fn) else 0
    fpr = fp/(fp+tn) if (fp+tn) else 0
    total_alerts = int(flags.sum())
    print(f"{name:34s} | التنبيهات: {total_alerts:4d} ({total_alerts/N*100:5.1f}%)
| Recall: {recall*100:5.1f}% | FPR: {fpr*100:5.1f}%")

print(f"إجمالي الرسائل: {N}  |  حالات شذوذ حقيقية محقونة: {int(labels.sum())}
({labels.mean()*100:.1f}%)\n")
```

```
print("العتبات 'المُعايَرة' (المئين 95 لكل مقياس):", {k: round(v,1) for k,v in
TUNED_THRESH.items()})
print()
report("أداة تقليدية (عتبات افتراضية)", default_flags)
report("أداة تقليدية (عتبات مُعايَرة يدوياً)", tuned_flags)
report("نظامنا (Hybrid Score)", hybrid_flags)
```

B.7 Spark Batch Reporting Script (spark_batch_report.py)

```
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from datetime import datetime

spark = (
    SparkSession.builder
    .appName("SystemPulseAI-BatchReport")
    .master("local[*]")
    .getOrCreate()
)
spark.sparkContext.setLogLevel("ERROR")

df = spark.read.csv("data/live_metrics.csv", header=True, inferSchema=True)
df = df.withColumn("timestamp", F.to_timestamp("timestamp"))

total_readings = df.count()

report_lines = []
report_lines.append("=" * 60)
report_lines.append("تقرير System Pulse AI الدفعي (Spark Batch Report)")
report_lines.append(f"تاريخ التوليد: {datetime.now().strftime('%Y-%m-%d
%H:%M:%S')}")
report_lines.append("=" * 60)
report_lines.append(f"\nإجمالي القراءات المُحلَّلة: {total_readings}")

if total_readings == 0:
    report_lines.append("\n⚠ لا توجد بيانات في data/live_metrics.csv بعد. شغّل ml-
consumer أولاً.")
else:


    overall = df.select(
        F.round(F.avg("cpu"), 2).alias("avg_cpu"),
        F.round(F.max("cpu"), 2).alias("max_cpu"),
        F.round(F.avg("memory"), 2).alias("avg_memory"),
        F.round(F.max("memory"), 2).alias("max_memory"),
        F.round(F.avg("disk"), 2).alias("avg_disk"),
        F.round(F.max("disk"), 2).alias("max_disk"),
        F.round(F.avg("network"), 2).alias("avg_network"),
```

```
        F.round(F.max("network"), 2).alias("max_network"),
    ).collect()[0]

    report_lines.append("\n--- 1. متوسطات وأقصى قيم المقاييس ---")
    report_lines.append(f"CPU:     متوسط{overall['avg_cpu']}%   |   أقصى
{overall['max_cpu']}%")
    report_lines.append(f"Memory:  متوسط{overall['avg_memory']}%  |  أقصى
{overall['max_memory']}%")
    report_lines.append(f"Disk:    متوسط{overall['avg_disk']}%   |   أقصى
{overall['max_disk']}%")
    report_lines.append(f"Network: متوسط{overall['avg_network']}  |  أقصى
{overall['max_network']}")



    anomaly_count = df.filter(F.col("anomaly") == 1).count()
    anomaly_pct = round(anomaly_count / total_readings * 100, 2)
    report_lines.append(f"\n--- 2. الشذوذ ---")
    report_lines.append(f"حالات شذوذ: {anomaly_count} من
{total_readings}  ({anomaly_pct}%)")



    if anomaly_count > 0:
        report_lines.append("\n--- 3. توزيع أسباب الشذوذ ---")
        causes = (
            df.filter(F.col("anomaly") == 1)
            .groupBy("root_cause")
            .count()
            .orderBy(F.desc("count"))
            .collect()
        )
        for row in causes:
            report_lines.append(f"  {row['root_cause']}: {row['count']}")


    report_lines.append("\n--- 4. توزيع مستويات الأولوية (Priority) ---")
    priority_labels = {0: "عادي", 1: "منخفض", 2: "متوسط", 3: "حرج"}
    priorities = df.groupBy("priority").count().orderBy("priority").collect()
    for row in priorities:
        label = priority_labels.get(row["priority"], str(row["priority"]))
        report_lines.append(f"  {label} ({row['priority']}): {row['count']}")


    report_lines.append("\n--- 5. أعلى 10 حالات خطورة (بحسب Hybrid Score) ---")
    top10 = (
        df.orderBy(F.desc("hybrid_score"))
        .select(
```

```
            "timestamp", "hybrid_score", "root_cause",
            F.round("cpu", 1).alias("cpu"),
            F.round("memory", 1).alias("memory"),
            F.round("disk", 1).alias("disk"),
            F.round("network", 1).alias("network"),
        )
        .limit(10)
        .collect()
    )
    for row in top10:
        ts = row["timestamp"].strftime("%Y-%m-%d %H:%M:%S") if
row["timestamp"] else "?"
        report_lines.append(
            f"  {ts} | Score:{row['hybrid_score']:.4f} | {row['root_cause']} |
"
            f"CPU:{row['cpu']} MEM:{row['memory']} DISK:{row['disk']}
NET:{row['network']}"
        )

    report_lines.append("\n--- 6. اتجاه الشذوذ حسب الساعة ---")
    hourly = (
        df.withColumn("hour", F.date_trunc("hour", "timestamp"))
        .groupBy("hour")
        .agg(
            F.count("*").alias("readings"),
            F.sum("anomaly").alias("anomalies"),
        )
        .orderBy("hour")
        .collect()
    )
    for row in hourly:
        hour_str = row["hour"].strftime("%Y-%m-%d %H:00")
        pct = round(row["anomalies"] / row["readings"] * 100, 1) if
row["readings"] else 0
        report_lines.append(f"  {hour_str} — قراءات: {row['readings']} | شذوذ:
{row['anomalies']} ({pct}%)")

report_lines.append("\n" + "=" * 60)
report_text = "\n".join(report_lines)

print(report_text)

with open("data/batch_report_latest.txt", "w", encoding="utf-8") as f:
    f.write(report_text)

print(f"\n تم حفظ التقرير أيضاً في data/batch_report_latest.txt")
```

```
spark.stop()
```

B.8 Docker Compose Configuration (docker-compose.yml)

```
services:

  zookeeper:
    image: confluentinc/cp-zookeeper:7.5.0
    container_name: zookeeper
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
    restart: unless-stopped

  kafka:
    image: confluentinc/cp-kafka:7.5.0
    container_name: kafka
    depends_on:
      - zookeeper
    ports:
      - "9092:9092"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
      KAFKA_LISTENERS: PLAINTEXT://0.0.0.0:9093,PLAINTEXT_HOST://0.0.0.0:9092
      KAFKA_ADVERTISED_LISTENERS:
PLAINTEXT://kafka:9093,PLAINTEXT_HOST://localhost:9092
      KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    restart: unless-stopped

  prometheus:
    image: prom/prometheus
    container_name: prometheus
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
    ports:
      - "9090:9090"
    restart: unless-stopped

  grafana:
    image: grafana/grafana
    container_name: grafana
    ports:
      - "3001:3000"
```

```
    environment:
      - GF_SECURITY_ADMIN_USER=admin
      - GF_SECURITY_ADMIN_PASSWORD=admin
    restart: unless-stopped

  influxdb:
    image: influxdb:2.7
    container_name: influxdb
    ports:
      - "8086:8086"
    environment:
      - DOCKER_INFLUXDB_INIT_MODE=setup
      - DOCKER_INFLUXDB_INIT_USERNAME=admin
      - DOCKER_INFLUXDB_INIT_PASSWORD=${INFLUXDB_PASSWORD}
      - DOCKER_INFLUXDB_INIT_ORG=system-pulse-ai
      - DOCKER_INFLUXDB_INIT_BUCKET=server-metrics
      - DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=${INFLUXDB_TOKEN}
    volumes:
      - influxdb-data:/var/lib/influxdb2
    restart: unless-stopped

  ml-consumer:
    build: ./models
    container_name: ml-consumer
    depends_on:
      - kafka
      - influxdb
    env_file:
      - .env
    volumes:
      - ./models:/app
      - ./data:/data
    ports:
      - "8000:8000"
    restart: unless-stopped

volumes:
  influxdb-data:
```

B.9 ML Consumer Requirements (requirements.txt)

```
kafka-python
tensorflow
scikit-learn
joblib
pandas
numpy
requests
```

```
prometheus-client
influxdb-client
shap
```

B.10 ML Consumer Dockerfile (Dockerfile)

```
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt.
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
EXPOSE 8000
CMD ["python", "consumer.py"]
```