International Journal of Networking and Computing – www.ijnc.org, ISSN 2185-2847
Volume 16, Number 2, pages 199–223, June 2026

Research on Anomaly Detection Methods for HPC Systems

Seiya Yaguchi
Graduate School of Engineering
Tokyo Denki University
Tokyo, Japan
24kmc25@ms.dendai.ac.jp

Ryusuke Egawa
Tokyo Denki University
Cyberscience Center, Tohoku University
Tokyo, Sendai, Japan
egawa@mail.dendai.ac.jp

Received: February 20, 2026
Revised: May 5, 2026
Accepted: June 3, 2026
Communicated by Shuichi Ichikawa

Abstract

Detecting anomalies in high-performance computing (HPC) systems is essential to keeping them reliable. However, this task is difficult because modern systems are very complex and there is little labeled failure data. Because of this, we need unsupervised methods that can learn only from normal operating data. Many existing methods mainly look at static system states. They often ignore the dynamic behaviors that show how faults start to develop. As a result, they may miss the first signs of problems and only detect failures after they have grown. This paper introduces a Difference Analysis (DA) method to solve this problem. This unsupervised method measures how reconstructed data from an LSTM-based autoencoder (RUAD) changes over time. Unlike methods that use only reconstruction error, DA directly models the instability of system behavior. We evaluated our method on monitoring data from the Marconi100 supercomputer. DA achieved an ROC AUC of 0.881, while the conventional reconstruction error method scored 0.770. These results show that studying dynamic state changes is a more effective way than relying only on static reconstruction error to improve the reliability of large-scale HPC systems. DA may also provide useful signals for anomaly anticipation, which we leave for future study.

Keywords: High-Performance Computing (HPC), Anomaly Detection, Unsupervised Learning, LSTM Autoencoder, Time-Series Analysis

1 Introduction

1.1 Background

In exascale computing, high-performance computing (HPC) systems are becoming much larger and more complex. Modern supercomputers have hundreds of thousands of nodes and tens of millions of cores. Their power use can reach several megawatts. These systems support tasks that require

high reliability, such as weather forecasting, drug discovery, and simulations of social infrastructure. As systems get larger, component failures tend to become more frequent, which reduces the mean time between failures. When a failure occurs, long-running jobs are interrupted, and a lot of time is spent on recomputation.

In recent HPC operations, power efficiency has also become an important issue[19]. When failures force recomputation, they waste power resources. Therefore, anomaly detection is important not only for efficient maintenance, but also for saving power and using power resources effectively.

For stable system operation, we need technologies that detect anomalies accurately and reduce downtime and power loss. However, anomaly detection in HPC systems is much harder than in general IT systems. A main reason is HPC-specific performance variation[21, 20]. In HPC environments, multiple jobs compete for resources such as network and storage bandwidth. As a result, even during normal operation, sensor values such as performance, power, and temperature can change greatly.

The target of this study is serious failures that need system administrator action or node shutdown. These failures may look sudden, but system behavior can become unstable around such failure events. Because normal performance variation also exists, it is very hard to separate anomaly-related instability from normal load changes using only the value at one time point. This difficulty can lead to false positives.

HPC systems are also updated regularly, and different systems often use very different architectures. For example, RIKEN's "Fugaku" has more than 150,000 CPU-only nodes[16], while Oak Ridge National Laboratory's "Frontier" has one CPU and four GPUs per node[15]. Some systems also use a mixed setup where CPU nodes and GPU nodes coexist, such as Europe's "LUMI"[11]. Because hardware components and system layout differ across systems, a model that depends strongly on a specific hardware configuration or on the behavior of specific metrics is hard to reuse for other systems or future systems. Therefore, there is a strong need for a general anomaly detection model that uses only standard monitoring data collected in many HPC systems and does not rely on a specific architecture.

Anomalies in logs and metrics do not always appear as sudden spikes. In many cases, anomaly-related changes appear as changes in behavior, such as changes in correlations between variables or less stable behavior. With multivariate data where hundreds to thousands of sensors are linked in complex ways, it is hard to monitor such behavioral changes by hand. We need automated, data-driven methods such as machine learning.

Machine-learning-based anomaly detection is widely studied, but in HPC there is often a lack of labels for training data. In real operations, anomalies are rare and their types are often unknown. Therefore, unsupervised approaches are promising because they do not require anomaly labels and can use the large amount of daily monitoring data directly for training[12, 1]. Among them, autoencoder-based methods are widely used because they learn correlations in normal data and can detect deviations from these patterns. However, many existing methods use time-series models but still rely on the size of a single value called reconstruction error when making the anomaly decision. As a result, they often fail to capture dynamic behavioral changes that appear over time.

1.2 Objective

The objective of this study is to develop an accurate anomaly detection method for HPC systems. To do this, we propose a new indicator that measures changes in system behavior over time: the time difference of reconstructed data, hereafter DA. With this approach, we aim to reduce downtime and wasted power caused by failures during job execution, and to improve the availability and reliability of HPC systems.

As described above, many conventional studies have used static anomalies at a certain time point as the basis for detection, such as the reconstruction error of an autoencoder. However, in HPC systems, metrics can fluctuate even in normal times due to load changes. For this reason, static indicators have a high risk of mistaking normal variation for anomalies. In addition, failures can be associated with a process where the system state changes from stable behavior to unstable behavior.

Therefore, in this study, we introduce DA based on differences between time points in the reconstructed time-series data produced by an LSTM autoencoder, RUAD[12]. DA measures temporal fluctuations of the reconstructed series, not the size of the value itself. We evaluate anomaly detection using DA and analyze its behavior by comparing it with the conventional method based on reconstruction error.

The main contributions of this study are as follows.

- **A new indicator DA that captures dynamic behavior:** The main contribution of this study is that DA, which measures temporal change in reconstructed data, is an effective feature for system anomaly detection. While reconstruction error measures a static deviation, DA can capture unstable behavior over time. As a result, DA can capture temporal changes that are hard to represent using a single-time reconstruction error.

- **An anomaly detection method using DA:** Using DA as an anomaly score, we built an unsupervised anomaly detection framework that does not require labeled anomaly data for training. DA can be computed from sensor data that are usually available in systems. Because it does not require specific anomaly patterns or hardware knowledge, it can be used across systems with different architectures.

- **Evaluation with real operational data:** We evaluated the method using a large-scale real operational dataset, M100 ExaData, from Marconi100, a Tier-0 supercomputer operated by CINECA in Italy[2]. The results show that the proposed method achieves higher detection accuracy and more stable detection than the conventional method.

1.3 Organization of This Paper

The organization of this paper is as follows.

In Section 2, we review related work on anomaly detection in HPC systems. We describe log-based methods, metric-based methods, and anomaly anticipation, and clarify the position of this study.

In Section 3, we describe the background techniques and the proposed method. First, we explain the basic ideas of autoencoders, LSTM, and RUAD as the base model. Next, we define DA introduced in this study and its computation process.

In Section 4, we present evaluations to verify the effectiveness of the proposed method. After describing the evaluation environment and the basic reconstruction capability of RUAD, we examine whether DA captures unstable behavior associated with anomalies. We also compare accuracy with the conventional method and provide further analysis from multiple viewpoints, such as per-node detection stability, system time scales, and comparison with results using raw data.

In Section 5, we summarize this study and describe conclusions and future work.

2 Anomaly Detection in HPC Systems

As HPC systems grow larger and more complex, automatic anomaly detection has become an important research topic for keeping systems reliable and available. In this section, we review existing work, from classic statistical baselines to recent deep learning models, and then anomaly anticipation.

Existing methods can be grouped mainly by (1) what data they use and (2) how complex their algorithms are. Two data sources are common: text-based system logs produced by the OS and applications, and performance metrics that are continuous numeric data such as CPU utilization and temperature. Section 2.1 covers traditional methods before deep learning. Sections 2.2 and 2.3 explain deep-learning methods for logs and metrics. Finally, Section 2.4 describes recent work on anomaly anticipation and the position of this study.

2.1 Conventional Statistical and Machine Learning Approaches

Conventional anomaly detection often uses statistical and machine learning methods such as principal component analysis (PCA) and support vector machines.

Xu et al. proposed a method that counts occurrences of system log messages, maps the data into a low-dimensional space using PCA, and detects points with large reconstruction error as anomalies[22]. Lou et al. proposed a method that automatically finds invariant relations among program variables from log data and treats violations of these relations as anomalies[10]. For numeric metrics, common baselines include One-Class SVM by Schölkopf et al.[17] and Isolation Forest by Liu et al.[9]. Isolation Forest uses the idea that anomalous points are easier to isolate than normal points, and it separates anomalies efficiently using random decision trees.

These methods are attractive because they have low computational cost and their results are relatively easy to interpret. However, linear models such as PCA cannot fully capture the complex nonlinear relationships often seen in HPC systems. Also, many conventional machine learning models such as Isolation Forest treat each data point as independent, so they do not model time-series dependencies such as order and context.

2.2 Deep Learning Based on Log Data

To capture complex order patterns in system logs, deep learning—especially recurrent neural networks—has been increasingly used in recent years.

DeepLog converts unstructured log messages into log keys (fixed templates) and parameters, and learns time-series patterns of log keys using LSTM[4]. It models the normal transition patterns of log keys, and flags a case as anomalous when logs appear in an order different from the prediction. This method is effective for detecting changes in log execution flow, but it can lose detailed parameter values and the meaning of the original text when converting logs into log keys.

To address this limitation, recent work has applied natural language processing to system logs. For example, FALL does not convert logs into log keys; instead, it inputs raw log text into a language model such as BERT[7]. FALL trains using only normal data and detects unknown failures by capturing small vocabulary changes and unusual context. However, logs are often recorded after an event happens. To capture early signs before a failure, continuously measured performance metrics are often more suitable.

2.3 Deep Learning Based on Performance Metrics

Performance metrics such as CPU utilization, memory usage, temperature, and power provide continuous and quantitative information about system state. For this reason, they are central to real-time anomaly detection.

2.3.1 Performance Variation and Difficulty of Diagnosis

A key difficulty in metric analysis for HPC systems is performance variation: values can change greatly even during normal operation. Tuncer et al. pointed out that resource contention and OS noise during HPC application runs introduce large variability in runtime and sensor values, and they proposed a framework to diagnose this using machine learning[20, 21]. Their work tries to separate normal load variation from anomalies using statistical feature extraction and classifiers. This supports the idea that simple threshold monitoring is not effective in HPC environments, and it motivates models that can learn dynamic fluctuation patterns.

2.3.2 Unsupervised Learning with Autoencoders

In real HPC operations, anomaly data are hard to obtain. Therefore, unsupervised learning methods that train only on normal data have become common. RUAD, which is also the baseline in this study, showed the effectiveness of unsupervised anomaly detection using an LSTM autoencoder on the Marconi100 system[12]. Sukhija et al. also performed anomaly detection focused on power

consumption and discussed its impact on reducing operational cost[19]. These methods use reconstruction error (the difference between input and reconstructed output) as the anomaly score. In other words, they detect points that deviate from normal correlations learned by the model and cannot be reconstructed well.

Some work also combines probabilistic RNNs and variational autoencoders, and learns the data distribution to improve robustness, such as OmniAnomaly[18]. However, these approaches still rely on static indicators at a single time point, such as error size or probability, and they do not directly measure the amount of behavioral change over time.

2.4 Development Toward Anomaly Anticipation

While traditional anomaly detection focuses on finding failures after they occur, recent research has moved toward anomaly anticipation, which aims to detect signs minutes to hours before a failure.

2.4.1 Prediction-based Approach

A common approach for anticipation is to predict future metric values and monitor (1) whether the prediction shows a worsening trend or (2) whether measured values start to deviate from the prediction. Borghesi et al. proposed a deep learning model that uses event logs from monitoring tools such as Nagios as ground-truth labels and predicts the probability of future failures from the current system state[3]. Nguyen et al. also predicted future values with LSTM and detected anomalies using prediction error in supply chain data[14], and the same idea can be applied to metric prediction in HPC.

More recently, generative models have also been used for anticipation. Lee et al. used a conditional diffusion model to generate and predict future metrics and to capture signs earlier than real time[8].

2.4.2 Utilization of Spatial Correlation

An HPC system is a cluster of many interconnected nodes. An anomaly may spread from one node to nearby nodes or even to an entire rack. Molan et al. proposed an anticipation method that uses a graph neural network to model spatial relationships such as physical placement and connection structure among nodes[13]. This can capture subtle early signs that span multiple nodes, which may be missed when analyzing each node alone.

2.5 Positioning of This Study

This study belongs to the research line of unsupervised anomaly detection using performance metrics. As discussed by Borghesi et al., because anomalies are very rare in HPC, an autoencoder tends to learn normal operation that dominates the data, and it does not learn rare anomalous patterns well. This property allows unlabeled data to work as an effective normal model, and it is a useful approach[1]. In this study, we also adopt RUAD, which models time-series behavior, as the baseline[12].

The key difference from existing work is the indicator used for anomaly detection. Table 1 compares related work with this study. Conventional methods including RUAD use reconstruction error, which is the difference between input and reconstructed data at a time point, as the anomaly score. This is a static indicator because it measures how far the system is from normal at that moment. Anticipation models such as Maat[8] use future prediction to estimate later system states, but this is not the main task of this study.

In contrast, DA in this study is the time difference of reconstructed data. DA measures dynamic behavior, that is, how the reconstructed system state changes over time. The goal is to improve anomaly detection by capturing temporal instability in reconstructed behavior. Thus, reliable future failure prediction is outside the primary scope of this study. Instead, this study focuses on current anomaly detection and shows that temporal changes in reconstructed data provide useful information that is not represented by single-time reconstruction error.

Table 1: Comparison of related work and this study

Study	**Data**	**Model**	**Indicator for anomaly detection**	**Characteristics / issues**
Invariants Mining [10]	Log	Invariant mining	Violation of invariants	Fixed rules and vulnerability to noise
Isolation Forest [9]	Metrics	Decision-tree ensemble	Isolation degree	Lack of time-series modeling
DeepLog [4]	Log	LSTM	Deviation from order patterns	Focus on execution order
Tuncer et al. [21]	Metrics	Random forest, etc.	Classification score	Diagnosis of known anomalies
RUAD [12]	Metrics	LSTM autoencoder	Reconstruction error	**Static state estimation**
Maat [8]	Metrics	Diffusion model	Deviation from predicted values	Anticipation of future state
This study	**Metrics**	**LSTM autoencoder**	**DA**	**Dynamic behavioral transition**

3 Anomaly Detection in HPC Systems and the Proposed Method

In this section, we describe our anomaly detection framework based on the time difference of reconstructed data. We designed this framework to support stable operation of HPC systems.

As discussed in Section 2, many existing methods rely on a static reconstruction error. This makes it hard to capture system state transitions and small behavioral changes associated with anomalies. To address this, we introduce a dynamic indicator based on how reconstructed data change over time. Our goal is to better separate normal performance variation from anomaly-related unstable behavior.

Section 3.1 gives an overview of the framework. Section 3.2 explains data preprocessing and feature generation. Section 3.3 describes RUAD, the baseline reconstruction model. Finally, Section 3.4 defines DA (time-difference analysis of reconstructed data) and explains how we compute the anomaly score.

3.1 Overview

The proposed framework aims to capture dynamic behavioral changes from operational HPC data and detect anomalous behavior. It has four main stages: data preprocessing, reconstruction by a model, computation of a dynamic indicator, and anomaly decision.

First, we aggregate multivariate time-series data with a fixed time window and convert them into feature vectors using simple statistics. Next, we reconstruct the input using RUAD, a trained LSTM autoencoder. This step reduces noise and extracts main behavioral components. Then we compute the amount of temporal change in the reconstructed sequence and define it as the dynamic indicator DA. Finally, we compute an anomaly score from DA and detect anomalies using a threshold.

3.2 Data Preprocessing

Monitoring data in HPC systems often include irregular sampling and noise, so preprocessing is needed before model input. In this study, we generate features for each compute node as follows.

First, we split raw data into fixed time frames. We use 15-minute intervals. Within each time frame, we compute four statistics: mean, standard deviation, maximum, and minimum. This captures not only the value level but also the strength of fluctuation and peak values in the interval. If

there are X sensor metrics, the feature at each time point has $X \times 4$ dimensions. For example, in the Marconi100 dataset, $X = 88$, so we obtain a 352-dimensional feature vector.

Because metrics have different physical units (temperature, power, fan speed, etc.), their scales differ across dimensions. To stabilize neural network training, we apply Min-Max normalization to scale each dimension to the range $[0, 1]$ before model input.

3.3 Base Model

3.3.1 Autoencoders in Anomaly Detection

An autoencoder is a neural network that learns data features by compressing the input and then reconstructing it[6]. As shown in Fig. 1, it consists of an encoder that maps input data to a low-dimensional latent space and a decoder that reconstructs the original data.

A key idea of autoencoders is the information bottleneck: the latent layer has fewer dimensions than the input. This forces the network to learn important patterns instead of memorizing the input. As a result, the model can reduce small noise and keep the essential system state.

For anomaly detection, the autoencoder is trained using only normal data to minimize reconstruction error (the difference between input and output). After training, normal patterns can be reconstructed well. However, if an unseen abnormal pattern is given, reconstruction becomes poor and the reconstruction error increases. Many conventional methods directly use this error magnitude as the anomaly score.

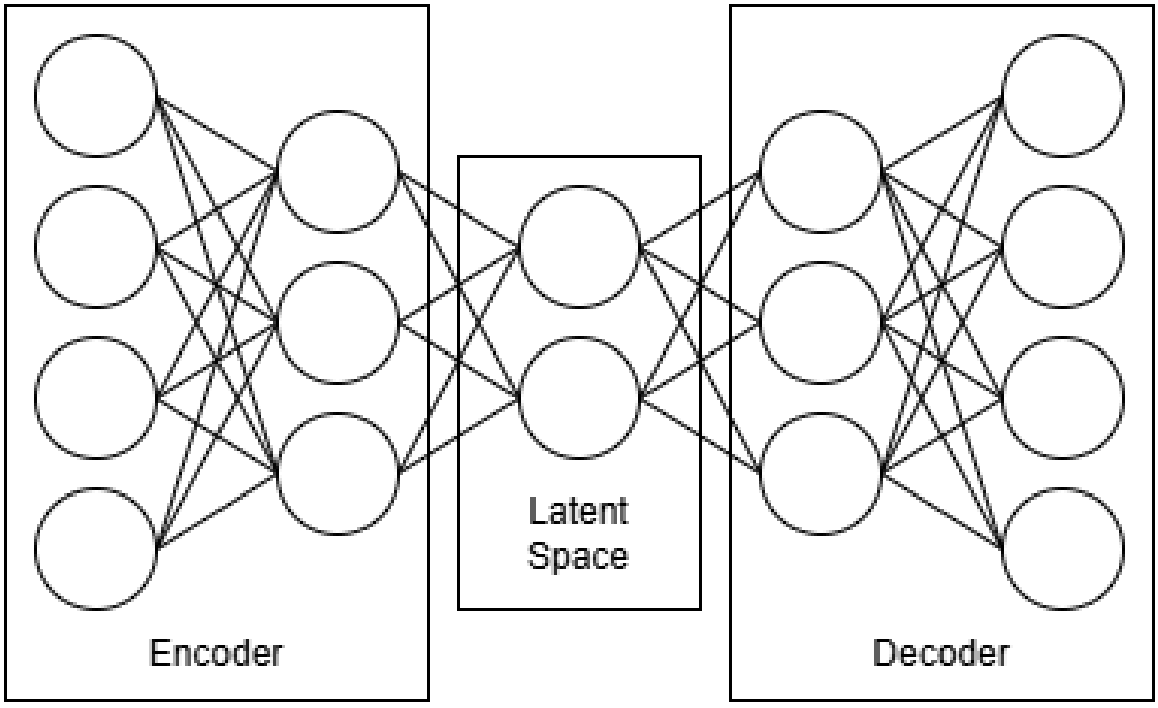

Figure 1: Structure of the autoencoder

3.3.2 Time-series Learning with LSTM

Log and sensor data in HPC systems are meaningful not only at a single time point but also in how they change over time. To handle time-series data, recurrent neural networks (RNNs) are commonly used. However, standard RNNs can forget older information over long sequences and can suffer from vanishing gradients, which makes learning difficult.

LSTM was introduced to reduce these problems[5]. LSTM has a memory cell that can keep information for a long time. It also uses gates to control how much new information to store and how much old information to forget. This makes it possible to learn long-range dependencies, for example, when current temperature is affected by a workload increase several hours earlier.

Figure 2 shows the structure of an LSTM block. The equations are as follows. The key point is that the forget gate f_t removes unnecessary memory and the input gate i_t adds only necessary information, so the memory cell C_t keeps important information over time.

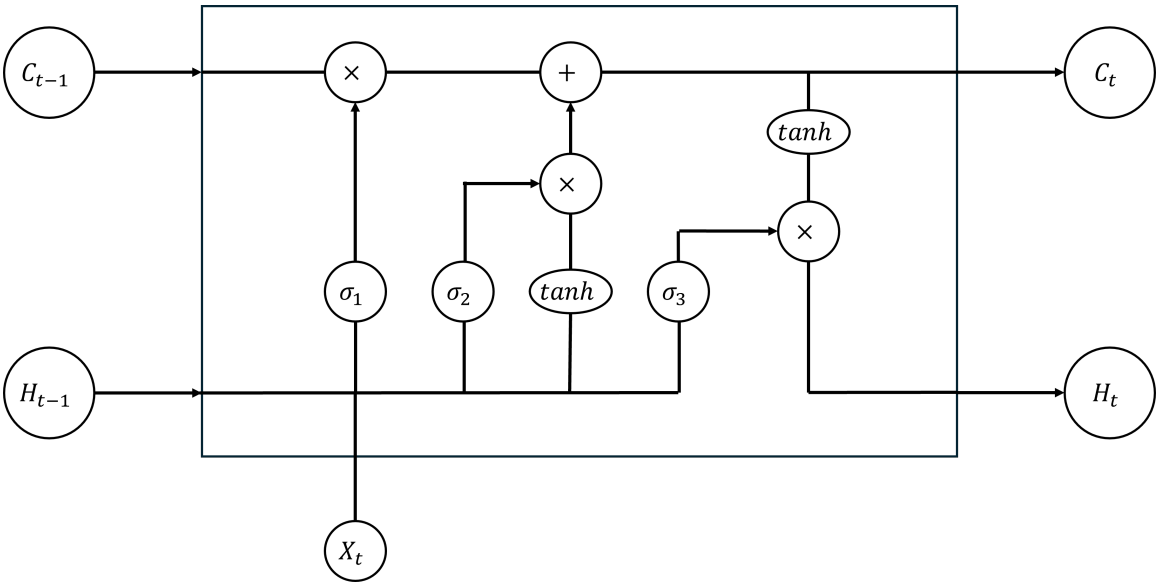

Figure 2: Structure of an LSTM block

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{1}$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{2}$$

$$\tilde{C}_t = \tanh(W_c[h_{t-1}, x_t] + b_c) \tag{3}$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{4}$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{5}$$

$$h_t = o_t \odot \tanh(C_t) \tag{6}$$

Here, σ is the sigmoid function, and $\odot$ is element-wise multiplication.

3.3.3 Adoption of the Base Model RUAD

In this study, we use RUAD to generate reconstructed data[12]. RUAD is an RNN-based unsupervised anomaly detection model designed for HPC systems.

RUAD is designed for large-scale systems. As shown in Fig. 3, it uses different structures for the encoder and decoder. The encoder uses LSTM to learn time-series features. The decoder uses fully connected layers, which are cheaper than LSTM.

If the decoder also used LSTM, the cost would be much higher, and it would be difficult to run the model on thousands of nodes. By simplifying the decoder, RUAD achieves practical speed while keeping good reconstruction capability.

In the original RUAD method, the reconstruction error between input x_t and reconstructed data $\hat{x}_t$ is used as the anomaly score. In this study, we use RUAD mainly as a generator that removes noise and produces a reconstructed sequence $\hat{x}_t, \hat{x}_{t+1}, \ldots$ that reflects system behavior. We then apply DA difference analysis to this reconstructed sequence.

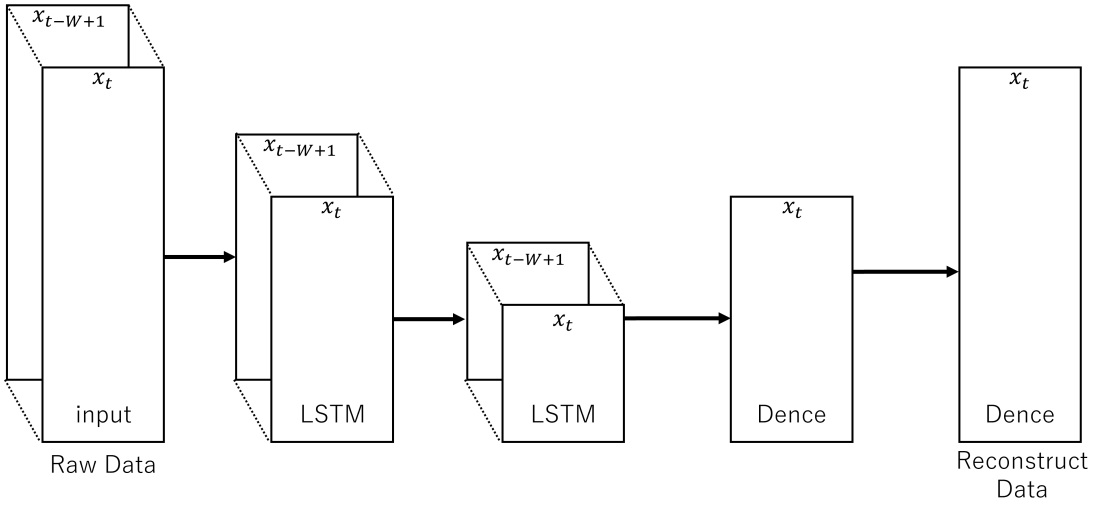

Figure 3: Structure of the RUAD model

3.4 Proposed Method: Time-Difference Analysis of Reconstructed Data (DA Difference Analysis)

3.4.1 Overview of the Method and Viewpoint

In order to improve sensitivity and stability in anomaly detection for HPC systems, we propose DA, a new indicator that focuses on temporal changes in reconstructed data produced by RUAD.

Conventional autoencoder-based detection defines reconstruction error, the difference between input x_t and reconstruction $\hat{x}_t$ at time t, as the anomaly score. This is a static indicator: it measures how different the system is from normal at a single time point.

However, in dynamic and complex systems such as HPC systems, anomalies are not always represented only by sudden value jumps. They can also be associated with unstable behavior over time.

Therefore, we focus on behavioral change rather than a static state. We quantify time differences in the reconstructed sequence. During normal operation, reconstructed data tend to change in a relatively stable way. In contrast, during anomaly-related unstable periods, reconstructed data may change more strongly over time. We define this amount of change as DA and use it for anomaly detection.

3.4.2 Process for Computing the DA Anomaly Score

In the proposed method, we compute the anomaly score as follows.

1. **Generation of reconstructed data:** We input time-series data into a trained RUAD model and generate a reconstructed vector $\hat{x}_t$ for each time t. This reconstruction reduces noise and keeps the main behavioral features of the system.

2. **Computation of time difference (DA):** To capture behavioral change, we compute the difference between reconstructed data at fixed time intervals. In this study, we focus on one-hour changes. For time t, we compute differences at one-hour steps in the past, using pairs $(\hat{x}_{t-k}, \hat{x}_{t-(k+1)})$ for $k = 0, 1, \ldots, N-1$. Let the mean squared error (MSE) of each pair be d_k:

$$d_k(t) = ||\hat{x}_{t-k} - \hat{x}_{t-(k+1)}||^2 \tag{7}$$

   Each d_k is a local DA value that reflects how strongly the reconstructed behavior changes at that time interval.

3. **Definition of the DA anomaly score:** We define the final DA anomaly score at time t as the average of DA values over the past N hours:

$$S_{Da}(t) = \frac{1}{N} \sum_{k=0}^{N-1} d_k(t) \tag{8}$$

A larger score means that the reconstructed behavior has been more unstable in the most recent N hours. Therefore, it is used as an anomaly score.

4 Evaluation

4.1 Evaluation Dataset

4.1.1 Examon Monitoring Framework and Collected Data

For evaluation, we use the public dataset of Marconi100, a Tier-0 HPC system operated by CINECA in Italy[2]. This dataset was collected by Examon, a monitoring framework developed by CINECA. Examon is designed for large HPC centers. It collects many kinds of operational data using a plugin-based architecture, including data from compute nodes and from facility equipment.

Table 2 lists the nine main plugins in Examon and the data collected by each. Examon monitors a wide range of information, including OS-level performance, hardware sensor values, job history, and facility data such as power and cooling, as well as outside weather data.

Table 2: List of main data collection plugins of Examon

Plugin name	**Collected data contents and role**
Ganglia	Collects OS-level performance metrics (CPU utilization, memory usage, network I/O, disk I/O, etc.).
IPMI	Acquires hardware physical sensor values (temperature, power, voltage, fan speed) that are independent of the OS via the baseboard management controller (BMC).
Job table	Databases historical information about completed jobs (job ID, user, runtime, used resources, exit status, etc.).
Logics	Collects text data such as system logs (Syslog) and application logs, and extracts specific error patterns and events.
Nagios	Works with the monitoring tool Nagios and acquires health status (OK, WARNING, CRITICAL) and alert information for each node and service.
SLURM	Acquires, in real time, the current queue state, node allocation status, partition information, etc. from the job scheduler SLURM.
Schneider	Collects facility-level metrics related to power and cooling from Schneider Electric facility equipment (such as PDUs and cooling devices).
Vertiv	Collects state data related to power supply and thermal management in the data center from Vertiv facility equipment (such as UPSs and air-conditioning units).
Weather	Acquires weather conditions outside the data center (outside air temperature, humidity, etc.) and uses them for analyses such as air-conditioning efficiency.

4.1.2 Data and Parameters Used in This Study

Examon collects many types of data (Table 2). In this study, to focus on hardware-level anomaly detection, we use data from **IPMI** and **Nagios**.

As input data, we use sensor values collected by the **IPMI** plugin, following the RUAD prior work. This is because OS-level metrics (e.g., Ganglia) are strongly affected by running applications, while IPMI data reflect more direct physical behavior of hardware. The input data include 88 sensor metrics, summarized in Table 3. They include system power, detailed temperature values, voltage/current values, and fan speeds.

Table 3: Details of the IPMI metrics used in this study (input parameters)

Category	**Examples of parameter names (variable names)**	**# items**
Power	`sys_power`, `proc_power`, `Gpu_power`, `dimm_power`, etc.	22
Temperature	`dimm[0-31]_temp`, `Gpu_gpu_temp`, `Gpu_mem_temp`, `proc_temp`, `ambient_temp`, etc.	42
Volt/Curr	`p_vdd_voltage`, `p_vcs_voltage`, `sys_12v_current`, etc.	16
Fan	`fan[0-7]_speed` (rotational speed of each fan module)	8
Total	**88 items**	

For these 88 metrics, we apply the preprocessing described in Section 3.2. We compute four statistics (mean, standard deviation, maximum, minimum) for each 15-minute interval. This produces a 352-dimensional feature vector.

4.1.3 Definition of Ground-Truth Labels

For ground-truth labels, we use event logs collected by the **Nagios** plugin. In Marconi100 operations, Nagios continuously monitors hardware and system conditions and reports alerts. In the Examon dataset, each Nagios event has a severity level.

Nagios statuses have four levels: “OK”, “WARNING”, “CRITICAL”, and “UNKNOWN” (Table 4). In this study, we treat anomaly detection as a binary classification problem. We define “OK” as normal and any of “WARNING”, “CRITICAL”, or “UNKNOWN” as anomaly. Based on this rule, we assign label 0 (normal) or 1 (anomaly) to each time step and evaluate detection accuracy.

Table 4: Definition of ground-truth labels based on Nagios status information

Status	**Nagios State**	**State description**	**Label in this study**
0	**OK**	Normal operation	**Normal**
1	**WARNING**	Caution (such as threshold exceedance)	**Anomaly**
2	**CRITICAL**	Critical failure	**Anomaly**
3	**UNKNOWN**	Unknown state / communication loss	**Anomaly**

4.1.4 Experimental Settings and Model Hyperparameters

The RUAD model follows the original structure[12]. It uses an asymmetric five-layer structure: two LSTM layers in the encoder and two fully connected layers in the decoder. Unless noted otherwise, the input window size is $W = 5$. We use stochastic gradient descent for training and mean squared error as the loss function. We train for 3 epochs. These settings follow the RUAD prior work[12], and we use them because our method relies on the same reconstruction process.

4.2 Basic Evaluation of the Time-series Reconstruction Capability of RUAD

The proposed method (DA difference analysis) depends on whether RUAD can learn and reconstruct normal time-series behavior well. If RUAD simply outputs the input (an identity mapping), or if it ignores time dependencies, then taking differences of its outputs would not be meaningful. Therefore, before evaluating DA, we check how well RUAD captures long-term dependencies and periodic patterns in HPC data.

For this check, we modify the reconstruction task. Instead of reconstructing the time step immediately after the input window, we set RUAD to predict x_{t+N}, which is N steps in the future. We evaluate several values of N: 1, 5, 9, 13, 32, 64, and 96. These correspond to prediction horizons from 15 minutes ahead to 24 hours ahead. We evaluate predictive accuracy using mean squared error (MSE) and compare it between normal and anomalous periods. ROC-AUC is then computed for anomaly discrimination.

Figure 4 shows AUC scores for each prediction horizon for multiple nodes within one rack. In many nodes, the AUC score at $N = 32$ (about 8 hours ahead) is higher than at $N = 1$ (15 minutes ahead). Accuracy remains relatively high up to about 16 hours ahead, but it drops at 24 hours ahead.

In general time-series prediction, uncertainty increases as the horizon becomes longer, so accuracy often decreases. However, our results do not follow this simple trend. This result suggests that the reconstructed data may contain temporal structures beyond short-term 15-minute fluctuations. The origin of this multi-hour behavior cannot be determined from this reconstruction evaluation alone. Therefore, in the next section, we further examine system time scales using job execution statistics and spectral analysis of reconstructed data.

These results suggest that RUAD does not mainly model only short-term fluctuations. Instead, the reconstructed outputs appear to contain temporal structures over several hours. This observation is important for our DA design. DA is computed from differences between reconstructed data and then averaged over a multi-hour window. If the reconstructed series remains stable during normal operation, DA should stay small. When the reconstructed behavior becomes unstable around anomalous periods, DA is expected to increase. Therefore, these results motivate the use of DA to analyze reconstructed behavior over multiple hours rather than relying only on a single-time reconstruction error.

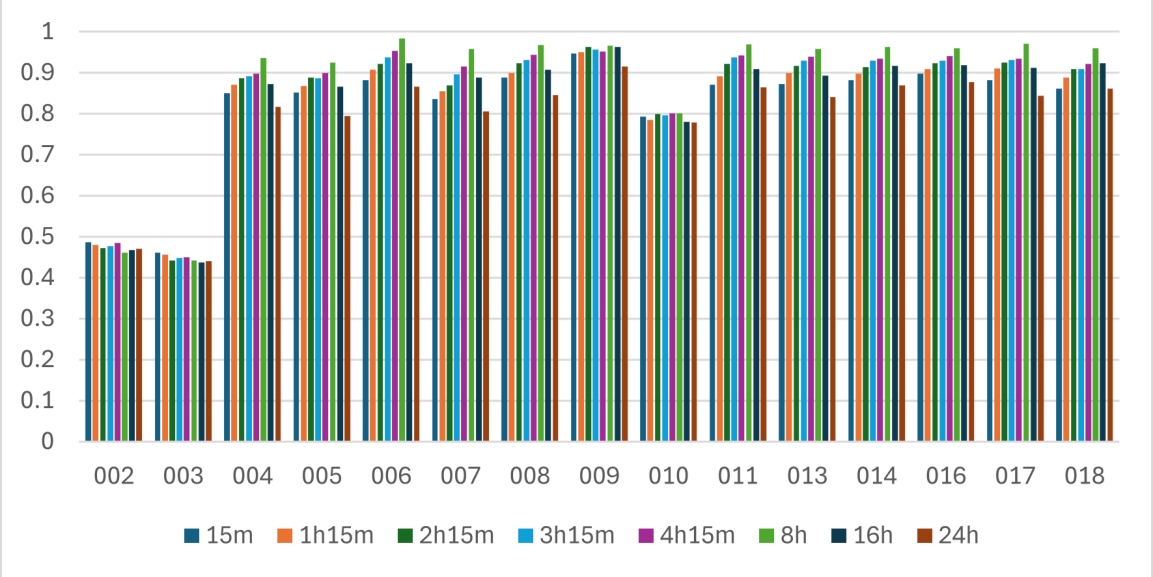

Figure 4: Transition of AUC scores for each prediction time in each node (one rack)

4.3 Analysis of System Time Scales

In Section 4.2, RUAD showed higher AUC at several-hour prediction horizons than at a very short horizon. This result suggests that the data may contain temporal structures over several hours. However, the origin of this time scale cannot be determined from the reconstruction evaluation alone. Therefore, we further analyze the system time scales from two viewpoints: job execution time and the power spectral density of reconstructed data.

First, we analyze the distribution of job execution time using the Job table data. Figure 5 shows the distribution of job duration. Most jobs are short. About 79.6% of jobs finish within 20 minutes, and more than 80% finish within 25 minutes. In contrast, jobs between 20 and 24 hours account for only about 4.3%. This result shows that the job execution time distribution is dominated by short jobs.

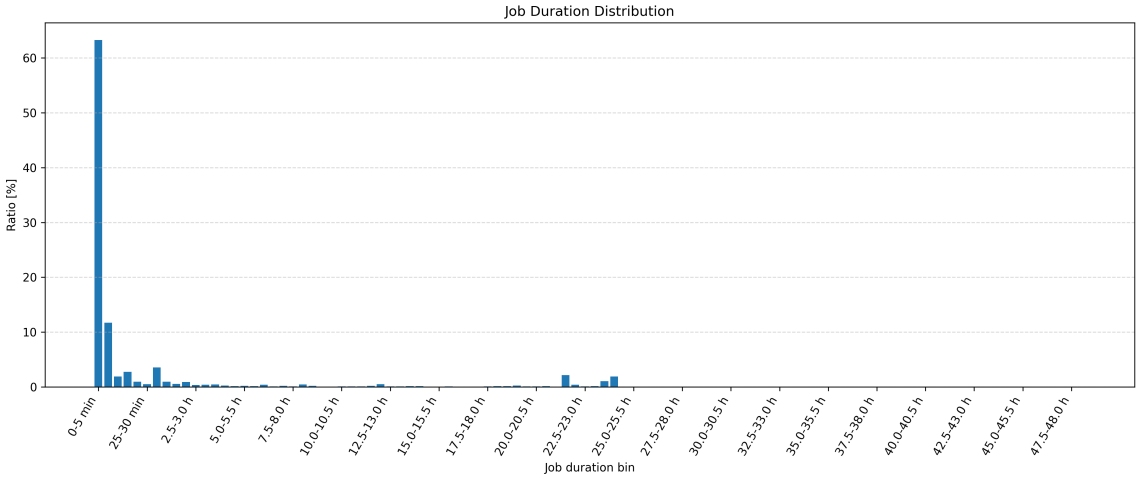

Figure 5: Distribution of job execution time

Next, we analyze the temporal components of the reconstructed data. We apply Welch power spectral density analysis to the reconstructed sequence. To make the result easier to interpret, we convert frequency into its corresponding period and plot the PSD against period. The horizontal axis in Figure 6 represents the period in hours, and the vertical axis represents the mean PSD value.

Figure 6 shows that the reconstructed data have strong components in long-period ranges. Local peaks are observed around 24 hours and 168 hours. This suggests that the reconstructed data include temporal components at daily and weekly scales. However, this result does not mean that the system behavior is governed by a single strict periodic cycle.

The job execution time distribution and the PSD result show different time scales. Job execution time is mainly concentrated in several minutes, while the PSD of reconstructed data shows stronger components at much longer time scales. Therefore, a direct correspondence between job execution time and the long-period components was not confirmed.

These results also do not directly explain why RUAD showed higher AUC around the 8-hour prediction horizon in Section 4.2. The improvement in reconstruction-based discrimination cannot be explained only by job duration or by the dominant temporal components observed in the PSD. This suggests that RUAD may capture more complex temporal structures that are not represented by a single job duration or a single periodic component. Thus, in this study, we treat the multi-hour behavior observed in Section 4.2 as an empirical property of the reconstructed data, rather than attributing it to a specific operational cycle.

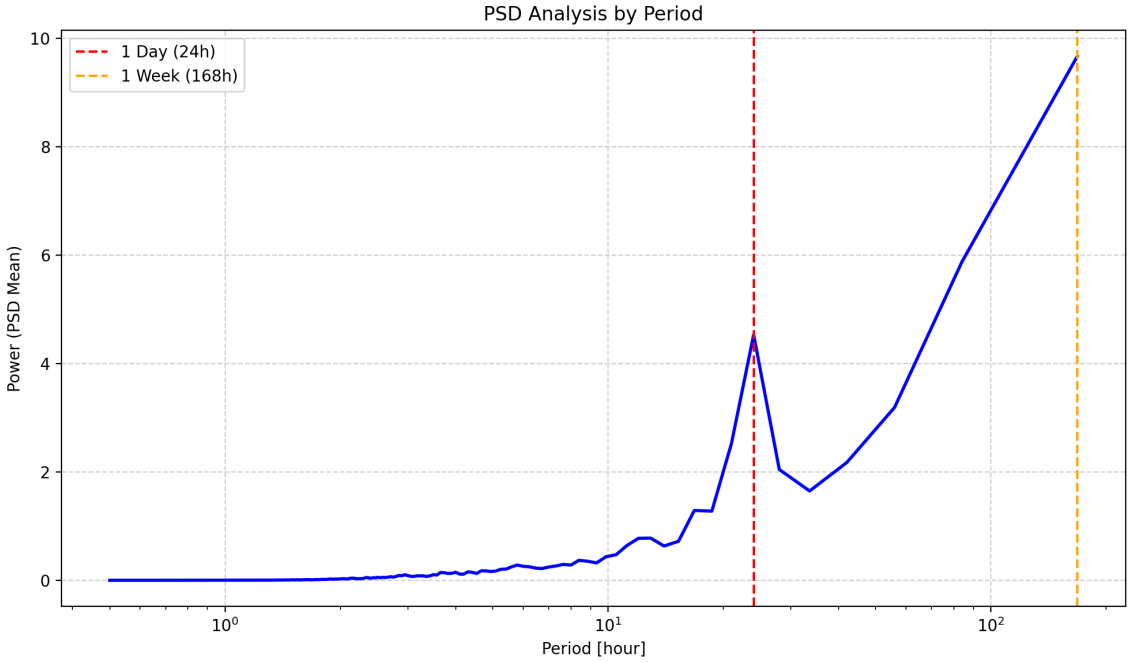

Figure 6: Welch PSD of reconstructed data plotted by period

4.4 Validity Evaluation of DA Difference Analysis

In this section, we examine whether DA can capture unstable behavior around anomalous periods. Using the ground-truth labels, we split the data into normal and anomalous periods and compare average time-difference trends.

We evaluate DA from the following two viewpoints.

1. **Long-term divergence from a reference time**

   Figure 7 shows a representative example for node 198. We compute the difference between reconstructed data at a reference time t and reconstructed data at past times $t-1, t-2, \ldots, t-24$ within the past 24 hours. In the figure, the blue line shows the normal reference time. It stays almost flat near zero, which indicates that the reconstructed behavior is stable during normal operation.

   In contrast, the orange line shows the anomalous reference time. It clearly diverges from past reconstructed data. This indicates that the reconstructed behavior becomes less stable around anomalous periods. Anomalous behavior can take different forms. In our data, we observed several patterns, such as sudden jumps, gradual drift, and oscillating growth. Across these patterns, anomalous periods tend to deviate more strongly from past behavior than normal periods.

2. **Short-term change between consecutive times**

   Next, following the DA definition in the proposed method, we compute differences between reconstructed data at consecutive one-hour steps. Figure 8 shows the result for node 198.

   The blue line shows normal periods and stays at very small values. The red line shows anomalous periods and has larger values across most of the recent 24 hours. Although the fluctuation range depends on the anomaly type, anomalous periods consistently show larger time differences than normal periods.

   This result indicates that anomalous periods are not characterized only by a single momentary change. Instead, the reconstructed behavior can remain unstable over multiple hours. This

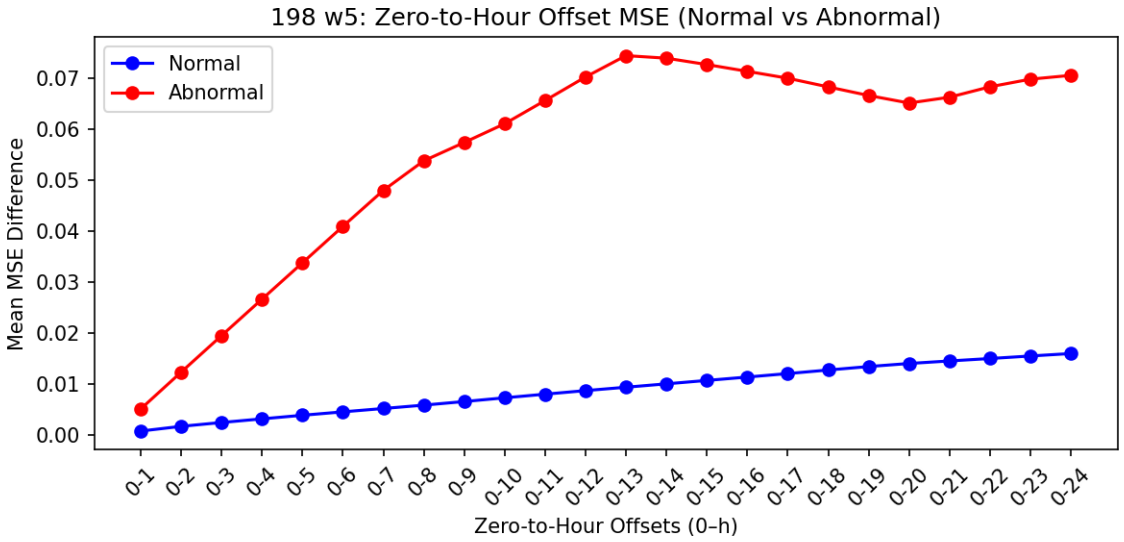

Figure 7: Transition of differences of reconstructed data at the reference time and past times (node 198)

supports the use of DA as an indicator that accumulates temporal changes in reconstructed data.

These results show that DA can separate stable normal behavior from unstable anomalous behavior in the reconstructed data. The long-term divergence shown in Fig. 7 and the consecutive one-hour differences shown in Fig. 8 are difficult to capture using only static reconstruction error. Therefore, DA provides a useful dynamic feature for anomaly detection in HPC systems. The detection performance of this feature is evaluated quantitatively in the next section.

4.5 Evaluation of Detection Accuracy and Stability of the Proposed Method

4.5.1 Accuracy Comparison with the Conventional Method

Here we compare DA difference analysis (proposed) with anomaly detection using RUAD reconstruction error (conventional). We use ROC AUC, which does not depend on a specific threshold. Figure 9 shows results when we change the RUAD input window size W to 5, 10, 15, and 20. For each W, we compute the average AUC over all 800 nodes.

In Fig. 9, **DA** is the proposed method and **Recon** is reconstruction error. For all window sizes, DA achieves clearly higher AUC than Recon. For example, at $W = 5$, Recon has AUC 0.764, while DA has AUC 0.881 (an improvement of about 0.12). This advantage remains even when W changes. At $W = 20$, DA keeps a high AUC of 0.883 compared with 0.770 for Recon.

An AUC around 0.7 often leads to many missed detections and false alarms, while an AUC near 0.9 can support practical monitoring. These results suggest that shifting from a static error to a dynamic change is effective. Reconstruction error checks only whether the system state at one moment is outside the normal range. In contrast, DA can capture temporal instability in reconstructed behavior, which is not represented by a single-time reconstruction error. As a result, DA achieves high accuracy without strong sensitivity to the window size.

4.5.2 Evaluation of Detection Stability for Each Node

An HPC system is a large cluster with hundreds to thousands of nodes. Nodes differ due to manufacturing variation and differences in their thermal environment inside racks. A practical anomaly

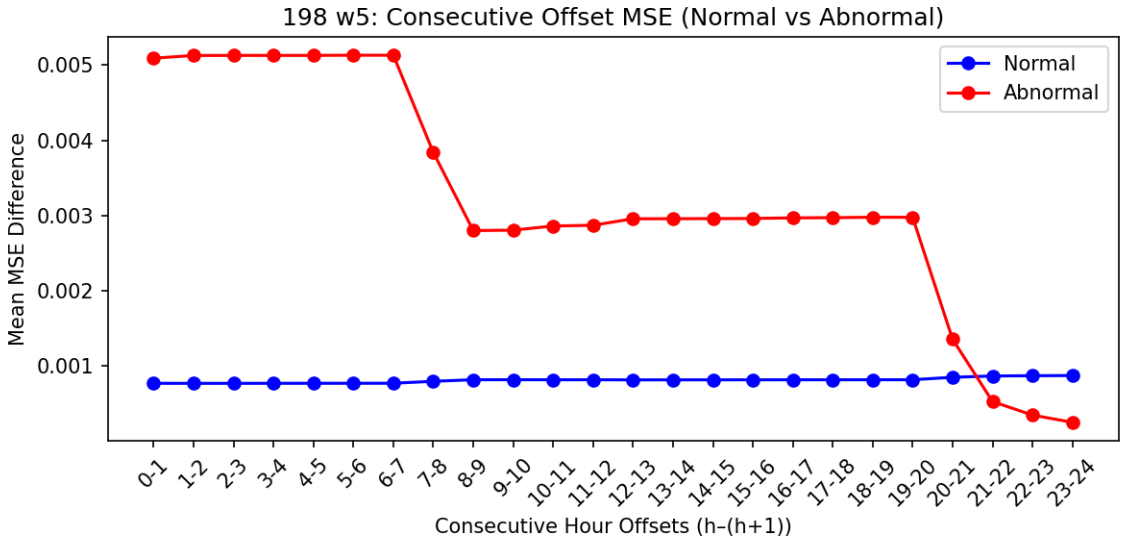

Figure 8: Transition of differences between reconstructed data every consecutive one hour (node 198)

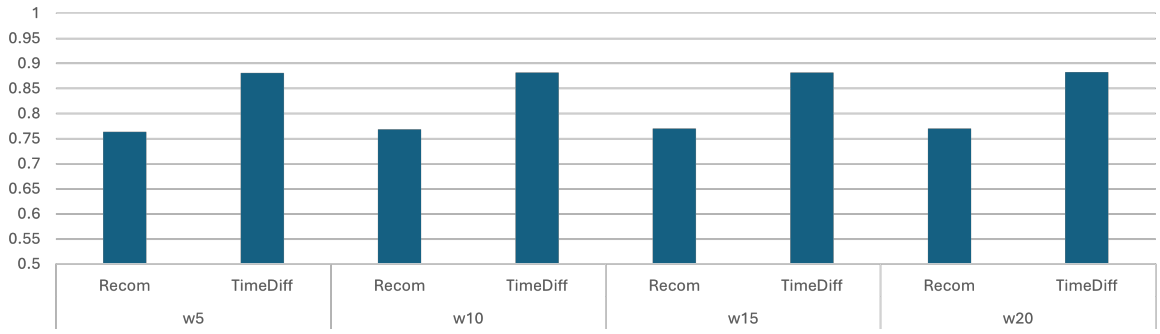

Figure 9: Comparison of AUC scores between the conventional method and the proposed method for each window size

detection method should perform well not only on some nodes but across the whole system. Therefore, we analyze the distribution of per-node AUC scores to evaluate detection stability.

Figure 10 shows the AUC distribution across 800 nodes at $W = 5$ for the conventional method and the proposed method. In the conventional method (left), many nodes are in the 0.8 range, but there is also a noticeable tail of low-accuracy nodes (0.7 and 0.6 ranges). Only a small number of nodes exceed AUC 0.9. This suggests that reconstruction-error-based detection is affected by node differences and environment. Static reconstruction error can include node-specific offsets (e.g., nodes that are always hotter), which can reduce stability.

In contrast, the proposed method (right) shows a much more stable distribution. The distribution shifts strongly to the right. More than 600 nodes fall into AUC 0.9 or higher, and the low-accuracy tail is almost removed.

This means DA difference analysis improves not only average accuracy but also robustness to system heterogeneity. Because DA uses differences from each node's own past behavior, it reduces the impact of static offsets and node-specific baselines. As a result, it provides consistently reliable monitoring for most nodes.

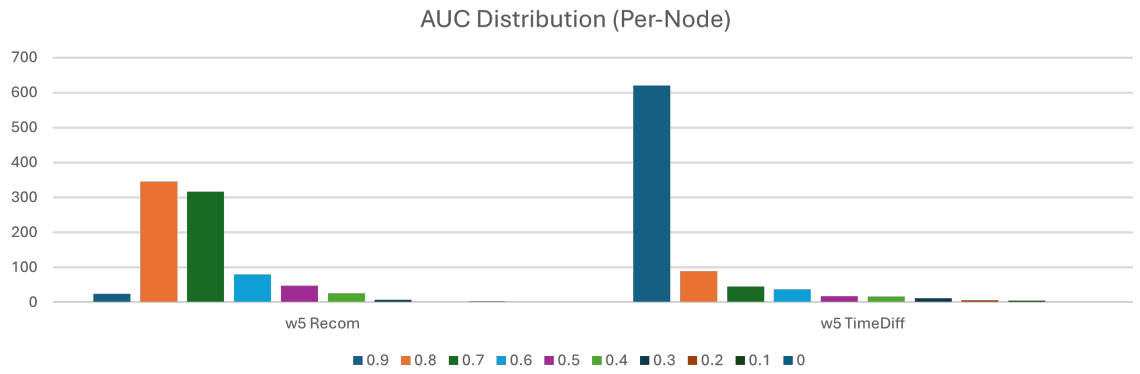

Figure 10: Comparison of distributions of AUC scores for each node (w5)

4.6 Detailed Analysis of the Proposed Method

In this section, we further analyze the proposed method. We focus on (1) whether RUAD reconstruction is necessary and (2) how parameters in DA computation affect accuracy.

4.6.1 Comparison with Time-Difference Analysis on Raw Data

In the proposed method, DA is computed from reconstructed data produced by RUAD. However, one may ask whether we can detect anomalies by simply taking time differences of raw data, without using a neural network model. If raw-data differences achieve enough accuracy, then RUAD would be less necessary. To test this, we define **RawDiff** as follows. We compute the same time-difference score directly on the preprocessed raw data and average it over the past 24 hours, using the same procedure as DA. We then compare its detection accuracy.

Figure 11 compares the average AUC scores of **DA** (proposed), **Recon** (conventional reconstruction error), and **RawDiff**. RawDiff achieves only about 0.61 AUC. This is far below Recon (about 0.76) and also far below DA (about 0.88). Therefore, simple raw-data differences do not work well for anomaly detection.

This drop in accuracy is likely due to the difference between raw data and reconstructed data. Raw sensor data include sensor noise, measurement errors, and short-term fluctuations during normal operation. If we take differences directly on raw data, we also detect these small normal fluctuations as instability. This increases false positives.

In contrast, RUAD (as an autoencoder) learns major normal patterns and tends to ignore small, non-essential noise that it does not learn. In this sense, RUAD works as a strong noise filter. Therefore, by taking differences on reconstructed data, we can focus more on essential behavioral changes and less on raw noise.

From these results, we can conclude that time differences alone are not enough. Time differences become useful only after feature extraction and reconstruction by RUAD. In other words, high accuracy is achieved by combining RUAD (static feature extraction / denoising) and DA (dynamic change analysis).

4.6.2 Influence of Averaging Time on Difference Computation

In the proposed method, we define the anomaly score as the average of DA values over the past 24 hours. To evaluate how the averaging window length H affects accuracy, we vary H from 2 hours to 96 hours and measure the AUC score. Figure 12 shows the results.

With very short windows such as $H = 2$ and $H = 4$, the AUC stays near 0.5. This is likely because the averaging period is too short to reduce short spikes and temporary noise. As a result, normal variations can be mistaken as anomalies.

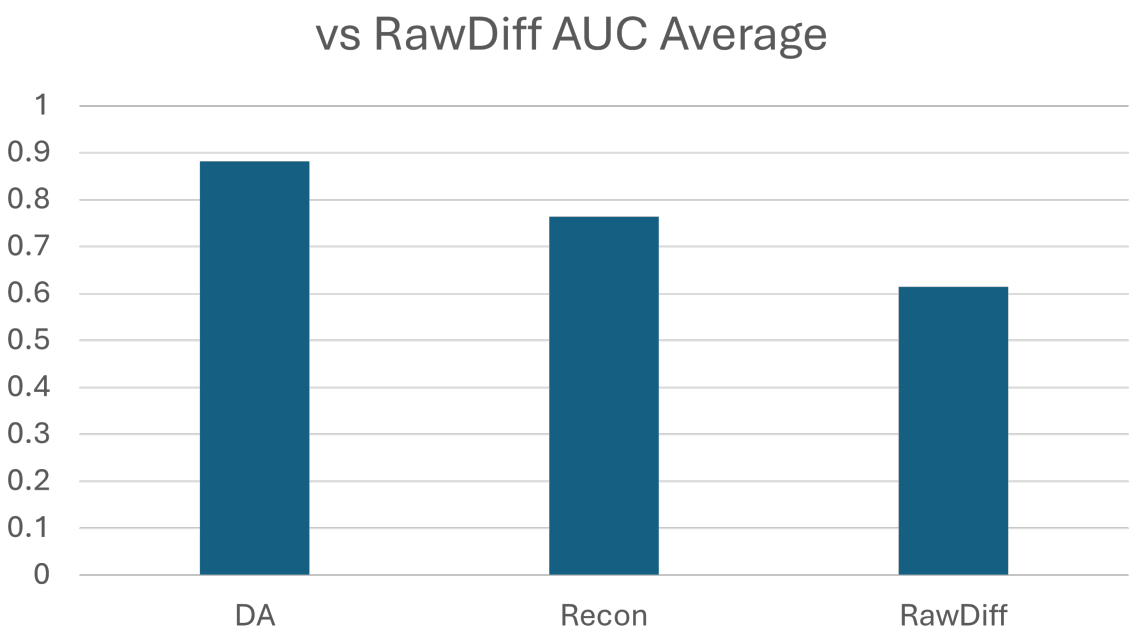

Figure 11: Accuracy comparison among the proposed method (DA), the conventional method (Recon), and raw-data difference (RawDiff)

As H increases, accuracy improves quickly. At $H = 12$, the AUC reaches about 0.88 and becomes stable. High accuracy is also maintained at $H = 16$ and $H = 24$. This range of 12–24 hours gives the best performance in this evaluation. These results suggest that averaging DA over a certain length of time is important. A short window is not enough to reduce temporary fluctuations. In contrast, a window of 12–24 hours can reduce short-term fluctuations while keeping anomaly-related changes. This does not mean that the result is directly caused by a specific periodic cycle. Rather, the result shows that DA works well when temporal changes are accumulated over a moderate time window.

However, when H becomes very long (e.g., $H = 72$ and $H = 96$), the AUC gradually decreases. If we average over too long a period, many normal samples are included and the signal of anomaly-related change can be diluted. This reduces sensitivity and responsiveness. Overall, the results suggest that H around 12–24 hours is a good choice in terms of balancing noise reduction and sensitivity. The 24-hour setting used in this study is within this good range.

4.6.3 Increase in False Positives and Future Issues

The proposed method achieves much higher AUC than the conventional method. However, a closer look shows a new issue: false positives can increase because DA is sensitive.

Because the DA anomaly score uses DA values over the past 24 hours, we compare distributions of DA samples that contribute to the score. We define an **anomaly period** as a time t with anomaly label (A), and a **normal period** as a time without anomaly label. For each time t, let the DA samples used in the score be $\{\mathrm{DA}(t), \mathrm{DA}(t-1\mathrm{h}), \ldots, \mathrm{DA}(t-23\mathrm{h})\}$. For anomaly periods, we add these DA samples to the anomaly-side distribution. For normal periods, we collect DA samples in the same way, but we exclude samples that are included in the anomaly-side set. From the normal-side distribution, we extract only the top 1% (99th percentile or higher) and compare it with the anomaly-side distribution.

Figure 13 shows the distributions for a representative node (node 198). The horizontal axis is $\log_{10}(\mathrm{DA} + \varepsilon)$, where ε is a small constant to avoid issues when $\mathrm{DA} = 0$. The vertical axis is probability density, normalized so that the histogram area is 1.

The anomaly-side distribution tends to have larger values, but the top 1% of normal data can

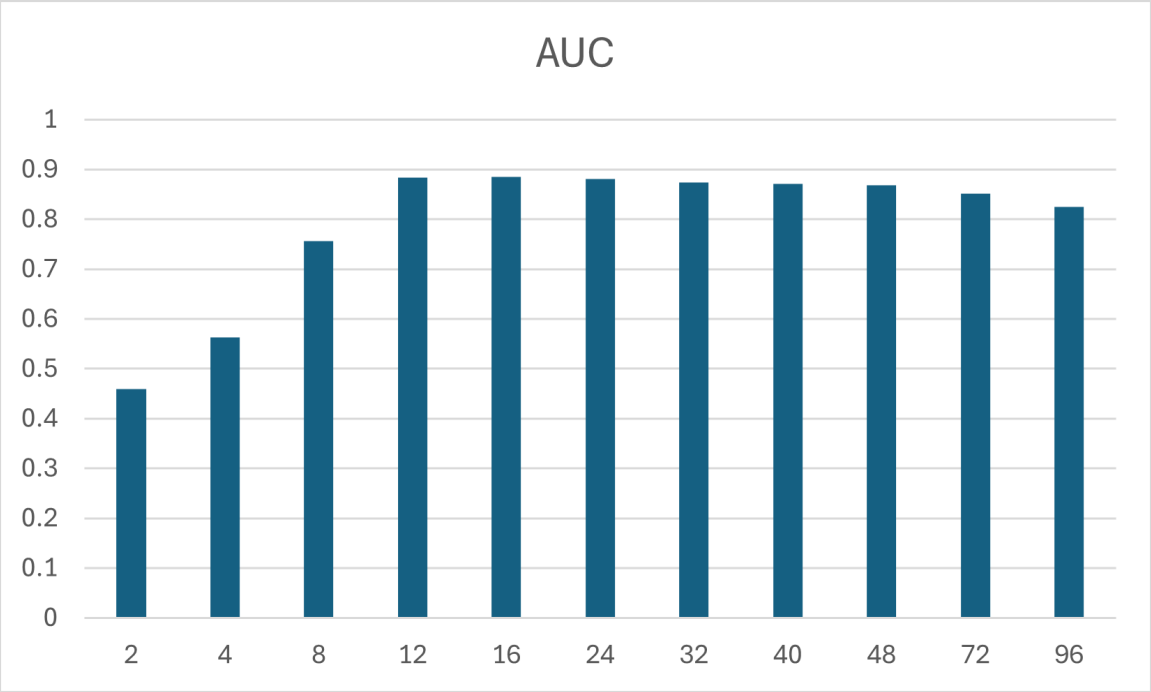

Figure 12: Transition of AUC scores by changing the averaging time window H

still overlap and even exceed some anomaly values. In other words, DA captures non-stationary behavior, but it can also react to strong normal state transitions. For example, when power and temperature move to a new stable level in a short time due to normal operational changes, the difference between $\hat{x}_t$ and $\hat{x}_{t-1h}$ can become large and DA can spike. Even with averaging over H, such spikes cannot be fully removed, which increases false positives.

To quantify the impact, we compare the proposed method and the conventional method using the following two metrics.

- **Recall**: the fraction of anomalies that are correctly detected among all anomalous data ($\mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$).
- **False discovery rate**: the fraction of detected anomalies that are actually normal ($\mathrm{FP}/(\mathrm{TP}+\mathrm{FP})$).

In our comparison, the recall of the proposed method is 0.74, which is 0.16 higher than the conventional method (0.58). However, the false discovery rate of the proposed method is 0.78, which is 0.25 higher than the conventional method (0.53).

This shows a clear trade-off. DA is very sensitive to dynamic change, but it can also react too strongly to normal high-load transitions. Therefore, to make the method more practical, we need a mechanism that keeps high recall while reducing false positives. In the next subsection, we introduce one example: a score that suppresses DA using reconstruction error, and we compare it using recall and false discovery rate in addition to AUC.

4.7 Application Examples of the Proposed Indicator DA

In this section, we present two application examples of DA. First, we design a suppression score $S(t)$ to reduce false positives. Second, we evaluate whether DA-derived indicators can predict future anomaly labels. These examples are not the main claim of this study, but they show how DA can be extended.

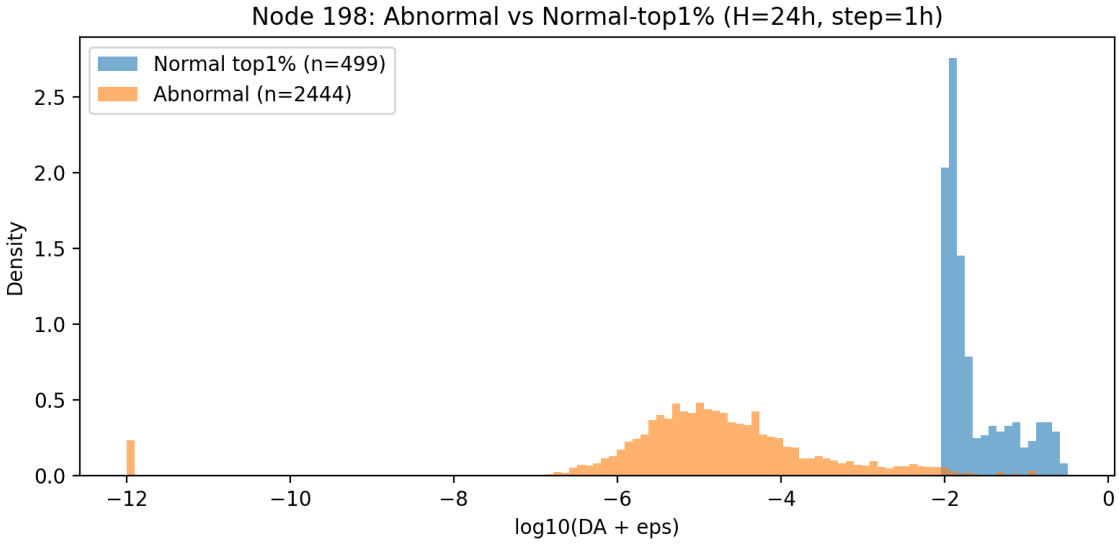

Figure 13: Comparison of DA value distributions between anomalous times and the top 1% of normal times (node 198)

4.7.1 Examination of DA Score Suppression

As discussed above, DA can react strongly to some normal state transitions. Therefore, we define a suppression score $S(t)$ that reduces DA when reconstruction error is small.

Let the reconstruction error at time t be $r(t)$. We define a gating coefficient $g(t)$ as follows:

$$g(t) = \sigma\left(\frac{r(t) - \tau}{T}\right), \tag{9}$$

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{10}$$

Here, τ is a reference value for reconstruction error, and T is a temperature parameter that controls how smooth the transition is. When $r(t) \ll \tau$, $g(t) \approx 0$. When $r(t) \gg \tau$, $g(t) \approx 1$.

Let the DA-based score be $DA(t)$. We define the final score $S(t)$ as:

$$S(t) = DA(t) \cdot g(t). \tag{11}$$

This means that when reconstruction error is small, DA spikes are suppressed. When reconstruction error is large, $S(t) \approx DA(t)$.

To avoid data leakage, we determine τ and T only from the first 80% of the time series. Let the reconstruction error sequence be $\{r(t)\}_{t=1}^{n}$ and let $\texttt{split} = \lfloor 0.8n \rfloor$. The training side is $\{r(t)\}_{t=1}^{\texttt{split}}$. We set τ to the 95th percentile of reconstruction errors on the training side. We set $T = \alpha\,\mathrm{std}$ with $\alpha = 0.1$, where std is the standard deviation of reconstruction errors on the training side. If std is very small, we use $T = 10^{-6}$ for numerical stability.

Because $S(t)$ is continuous, we evaluate it using the ROC curve and ROC-AUC. Table 5 shows AUC, recall, and false discovery rate for each method. The AUC of $S(t)$ is 0.89. Recall is 0.82 and false discovery rate is 0.52. Compared with DA alone, $S(t)$ keeps recall high while reducing false positives. It also achieves higher recall than the conventional method, with a similar false discovery rate.

4.7.2 Preliminary Evaluation of DA for Future Anomaly Prediction

From an operational viewpoint, predicting future anomalies is useful. Therefore, we evaluate whether DA-derived indicators can predict anomaly labels within the next 24 hours. This evaluation is

Table 5: Comparison of detection performance of each method

Method	AUC score	Recall	False discovery rate
Conventional method	0.77	0.58	0.53
Proposed method (DA)	0.88	0.74	0.78
Suppression score (S)	0.89	0.82	0.52

preliminary and is not the main claim of this study.

We define a 24-hour look-ahead task. The positive class is when an anomaly occurs within 24 hours from the current time. The negative class is when no anomaly occurs within 24 hours. As prediction features, we define seven derived indicators computed from the DA time series. For each indicator, we use the past 24 hours of DA data.

- **A1 (EWMA):** Exponentially weighted moving average of DA with half-life 6 hours. It reduces sudden noise and captures trend components.
- **A2 (Streak):** Consecutive length above a threshold. The threshold is the 95th percentile of DA in the past 24 hours. It measures how long DA remains at a high level.
- **A3 (PosDeltaCum):** Cumulative sum of only positive increases of DA over the past 24 hours. It ignores decreases and measures accumulated growth of instability.
- **A4 (LocalZ):** Local Z-score using mean and standard deviation over the past 24 hours. The result is smoothed by EWMA with half-life 6 hours. It measures how much recent DA deviates from its local distribution.
- **A5 (Variance):** Variance of DA over the past 24 hours. It reflects expansion of fluctuation rather than value level.
- **A6 (Short/Long):** Ratio of a 6-hour moving average to a 24-hour moving average. It captures short-term worsening relative to the long-term baseline.
- **A7 (ExceedRatio):** Fraction of time steps in the past 24 hours where DA exceeds the 95th percentile. Unlike A2, it does not require continuity.

Table 6 shows average AUC scores for each indicator. The best performance is A1 with an AUC of 0.590, followed by A3 with 0.572 and A5 with 0.567. In contrast, A2, A4, A6, and A7 remain around 0.50–0.52, which is close to random prediction.

These results provide two observations. First, A1 and A3 perform better than the other indicators. This suggests that DA-derived indicators may contain weak information related to future anomaly labels. For example, a rising baseline or accumulated instability may be related to future anomaly occurrence. The fact that A3 is higher than A5 suggests that accumulated increases are more informative than variance alone.

Second, even the best AUC is below 0.6. This performance is not sufficient to claim practical anomaly anticipation. A likely reason is that simple fixed-window statistics cannot handle diverse failure patterns in HPC systems. This result is also clearly lower than the anomaly detection performance at the current time.

Overall, DA is effective for capturing behavioral changes at anomaly times. However, the simple indicators used in this preliminary evaluation are not sufficient for reliable future anomaly prediction. More advanced models may be needed to use DA-derived features for anomaly anticipation. This remains an important direction for future work.

Table 6: Future anomaly prediction accuracy using DA-derived indicators

Indicator ID	Indicator overview and parameter setting	AUC Score
A1	Exponentially weighted moving average with half-life 6h	**0.590**
A2	Consecutive period of threshold exceedance with threshold 95%ile and window 24h	0.501
A3	Cumulative sum of increases with window 24h	**0.572**
A4	EWMA of local Z-score with half-life 6h and window 24h	0.503
A5	Variance with window 24h	**0.567**
A6	Short-term 6h moving average divided by long-term 24h moving average	0.503
A7	Threshold exceedance ratio with threshold 95%ile	0.521

5 Conclusion

5.1 Summary

In this study, we demonstrated the effectiveness of DA, a new anomaly detection indicator that focuses on temporal changes in reconstructed data generated by RUAD. The aim of this study is to support stable operation of HPC systems, which are becoming larger and more complex.

Conventional methods often use reconstruction error at a single time point as the basis for anomaly detection. This is a static indicator. In contrast, DA measures temporal fluctuations in reconstructed data. By treating anomalies as unstable behavior over time rather than only as static deviations at one time point, DA provides a dynamic viewpoint for anomaly detection.

Evaluation using the Marconi100 dataset showed the following five main findings.

First, DA improved detection accuracy. Anomaly detection using DA improved the AUC score by more than 0.11 points at maximum compared with the conventional method based on reconstruction error. This result shows that a dynamic viewpoint based on time differences can capture behavioral changes that are difficult to represent using a single-time reconstruction error.

Second, DA improved detection stability. The analysis of per-node AUC distributions showed that DA achieved high accuracy for most nodes in the system. This indicates that DA is robust to node heterogeneity in large-scale systems. Because DA uses differences from each node's own past behavior, it can reduce the effect of static offsets caused by node-specific characteristics and installation environments.

Third, the combination of RUAD and DA was important. When time differences were computed directly from raw data, the accuracy was much lower than when DA was computed from reconstructed data. This shows that time differences alone are not sufficient for noisy real operational data. RUAD works as a feature extractor and noise reducer, and DA evaluates temporal fluctuations of the reconstructed features.

Fourth, the averaging window length was important. The sensitivity analysis of the averaging window showed that DA achieved high accuracy when H was set to 12–24 hours. This result indicates that accumulating DA over a sufficiently long window is effective for reducing temporary fluctuations and capturing sustained unstable behavior. When the window was too short, the score was strongly affected by short-term fluctuations. When the window was too long, anomaly-related changes could be diluted.

Fifth, DA has useful extensibility, but also clear limitations. DA captures behavioral changes with high sensitivity, but it can also react to normal state transitions. As an application example, we introduced the suppression score $S(t)$, which suppresses DA based on reconstruction error. This reduced the false discovery rate from 0.78 to 0.52 while achieving recall of 0.82. We also evaluated DA-derived indicators for future anomaly prediction. Some indicators performed slightly better than random prediction, but the best AUC was 0.590. This result shows that simple statistical indicators derived from DA are not sufficient for practical future anomaly prediction.

Overall, these results show that dynamic behavioral changes in reconstructed data provide useful information for anomaly detection in HPC systems. DA improves detection accuracy and stability compared with static reconstruction error. At the same time, the results also show that DA should

be used carefully because it can react to normal state transitions.

5.2 Future Issues and Outlook

Although this study achieved high anomaly detection accuracy, several issues remain before practical use.

The greatest issue is the trade-off between recall and false discovery rate. In the evaluation, the recall of the proposed method was 0.74, which was 0.16 points higher than that of the conventional method. However, the false discovery rate of the proposed method was 0.78, which was 0.25 points higher than that of the conventional method. Detailed analysis showed that DA can react strongly even to normal operational changes. For example, when power and temperature move to a new stable level in a short time, DA can spike. Because DA is sensitive to dynamic changes, it can also react to such normal changes.

Therefore, future work should introduce mechanisms that reduce false positives while keeping high recall. One direction is to adjust thresholds according to system state. Another direction is to introduce more advanced filtering methods that can separate normal state transitions from anomaly-related instability. In this study, the suppression score $S(t)$ showed one possible approach. It improved recall while reducing the false discovery rate to a level close to the conventional method. However, the reference value and temperature parameter used in this score are design parameters. A more reproducible method for setting these parameters is needed.

Future anomaly prediction is also an important direction. In the application example in Section 4, we evaluated simple statistical indicators derived from DA for a 24-hour future anomaly prediction task. The best AUC was below 0.6, which shows that simple fixed-window statistics are not sufficient for reliable future anomaly prediction. At the same time, indicators such as the moving average and cumulative increase of DA performed slightly better than random prediction. This suggests that DA-derived features may contain weak information related to future anomaly labels.

To use this information more effectively, more advanced time-series models may be needed. For example, Transformers or diffusion models may be able to learn nonlinear and long-term patterns from DA-derived features. This remains future work.

Furthermore, this study targeted a single HPC system. To further examine the architecture-independence of DA, we plan to evaluate the method using datasets from multiple HPC systems with different configurations, such as Fugaku and LUMI. We expect that the dynamic analysis framework presented in this study will contribute to more stable and reliable operation of future exascale systems.

References

[1] Andrea Borghesi, Andrea Bartolini, Michele Lombardi, Michela Milano, and Luca Benini. A semisupervised autoencoder-based approach for anomaly detection in high performance computing systems. *Engineering Applications of Artificial Intelligence*, 85:634–644, 2019.

[2] Andrea Borghesi, Carmine Di Santi, Martin Molan, Mohsen Seyedkazemi Ardebili, Alessio Mauri, Massimiliano Guarrasi, Daniela Galetti, Mirko Cestari, Francesco Barchi, Luca Benini, Francesco Beneventi, and Andrea Bartolini. M100 ExaData: a data collection campaign on the CINECA's Marconi100 Tier-0 supercomputer. *Scientific Data*, 10:288, 2023.

[3] Andrea Borghesi, Martin Molan, Michela Milano, and Andrea Bartolini. Anomaly detection and anticipation in high performance computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 33(4):739–750, 2022.

[4] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. DeepLog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017.

[5] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12(10):2451–2471, 2000.

[6] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[7] Jaeyoon Jeong, Insung Baek, Byungwoo Bang, Junyeon Lee, Uiseok Song, and Seoung Bum Kim. FALL: Prior failure detection in large scale system based on language model. *IEEE Transactions on Dependable and Secure Computing*, 2024.

[8] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, and Michael R. Lyu. Maat: Performance metric anomaly anticipation for cloud services with conditional diffusion. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2023.

[9] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining (ICDM)*, pages 413–422, 2008.

[10] Jian-Guang Lou, Qiang Fu, Shengqi Yang, Yi-Feng Xu, and Jiang Li. Mining invariants from console logs for system problem detection. In *Proceedings of the 2010 USENIX Annual Technical Conference (USENIX ATC '10)*, pages 24–24, 2010.

[11] LUMI Supercomputer Helpdesk. LUMI Hardware Architecture. `https://docs.lumi-supercomputer.eu/hardware/`, 2024. Accessed: 2024-12-28.

[12] Martin Molan, Andrea Borghesi, Daniele Cesarini, Luca Benini, and Andrea Bartolini. RUAD: Unsupervised anomaly detection in HPC systems. *Future Generation Computer Systems*, 141:542–554, 2023.

[13] Martin Molan, Junaid Ahmed Khan, Andrea Borghesi, and Andrea Bartolini. Graph neural networks for anomaly anticipation in HPC systems. In *Companion of the ACM/SPEC International Conference on Performance Engineering (ICPE '23 Companion)*, 2023.

[14] H. D. Nguyen, K. P. Tran, S. Thomassey, and M. Hamad. Forecasting and anomaly detection approaches using LSTM and LSTM autoencoder techniques with the applications in supply chain management. *International Journal of Information Management*, 57:102282, 2021.

[15] Oak Ridge Leadership Computing Facility. Frontier System Overview. `https://docs.olcf.ornl.gov/systems/frontier_user_guide.html`, 2024. Accessed: 2024-12-28.

[16] RIKEN Center for Computational Science. Fugaku System Configuration. `https://www.r-ccs.riken.jp/en/fugaku/about/system/`, 2024. Accessed: 2024-12-28.

[17] Bernhard Schölkopf, Robert C. Williamson, Alex J. Smola, John Shawe-Taylor, and John C. Platt. Support vector method for novelty detection. In *Advances in Neural Information Processing Systems 12 (NIPS 1999)*, pages 582–588. MIT Press, 2000.

[18] Ya Su, Youjian Zhao, Chenhao Niu, Rong Liu, Wei Sun, and Dan Pei. Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2828–2837, 2019.

[19] Nitin Sukhija, Elizabeth Bautista, Drake Butz, and Cary Whitney. Towards anomaly detection for monitoring power consumption in HPC facilities. In *ACM International Conference on Management of Emergent Digital EcoSystems (MEDES)*, 2022.

[20] Ozan Tuncer, Emre Ates, Yijia Zhang, Ata Turk, Jim Brandt, Vitus J. Leung, Manuel Egele, and Ayse K. Coskun. Diagnosing performance variations in HPC applications using machine learning. In Julian M. Kunkel, Rio Yokota, Pavan Balaji, and David Keyes, editors, *High Performance Computing*, volume 10266 of *Lecture Notes in Computer Science*, pages 355–373. Springer International Publishing, 2017.

[21] Ozan Tuncer, Emre Ates, Yijia Zhang, Ata Turk, Jim Brandt, Vitus J. Leung, Manuel Egele, and Ayse K. Coskun. Online diagnosis of performance variation in HPC systems using machine learning. *IEEE Transactions on Parallel and Distributed Systems*, 30(4):883–896, April 2019.

[22] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I. Jordan. Detecting large-scale system problems by mining console logs. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP '09)*, pages 117–132, 2009.