GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN IN PUBLICA COMMODA SEIT 1737

HPS

Seminar Report

Machine Learning for Predictive Maintenance on a HPC Nodes

Karim Elezabawy

MatrNr: 24175359

Supervisor: Christian Boehme

Georg-August-Universität Göttingen
Institute of Computer Science

March 25, 2026

Abstract

High-performance computing (HPC) systems are critical infrastructure for scientific and industrial workloads, yet they remain vulnerable to hardware failures whose costs and frequency make reactive maintenance economically prohibitive. The scale and complexity of modern HPC clusters, with thousands of interdependent components operating under continuous high utilization, render manual monitoring impractical and motivate the use of data-driven approaches to anticipate failures before they occur. This report surveys machine learning methods for predictive maintenance on HPC nodes, organised by data type and supervision paradigm. The methods covered span traditional supervised classifiers, autoencoder-based anomaly detection, deep learning approaches that exploit temporal structure in logs and sensor streams, Large Language Model (LLM) methods for log analysis, and graph-based techniques that encode time-series topology as embeddings. For each family of methods, the core principles, representative systems, and key limitations are discussed. The methods are compared across heterogeneous HPC datasets, with an explicit acknowledgement that differences in evaluation conditions limit direct comparison. The report additionally includes a practical reproduction experiment applying a graph-based approach to an independent subset of a publicly available production dataset, providing empirical evidence on the generalisability of the method beyond its original evaluation. Open challenges identified include label scarcity, data contamination in production telemetry, system-specific generalisation, and the absence of a unified benchmarking framework for HPC anomaly detection.

Declaration on the use of ChatGPT and comparable tools in the context of examinations

In this work I have used ChatGPT or another AI as follows:

- ☐ Not at all
- ☑ During brainstorming
- ☑ When creating the outline
- ☐ To write individual passages, altogether to the extent of 0% of the entire text
- ☐ For the development of software source texts
- ☐ For optimizing or restructuring software source texts
- ☑ For proofreading or optimizing
- ☐ Further, namely: -

I hereby declare that I have stated all uses completely.
Missing or incorrect information will be considered as an attempt to cheat.

Contents

List of Tables

List of Figures

List of Abbreviations

HPC High-Performance Computing

AI Artificial Intelligence

ML Machine Learning

ODA Operational Data Analytics

DCDB Data Center Data Base

CPU Central processing unit

FINJ Fault Injection Tool

LDMS Lightweight Distributed Metric Service

HPAS HPC Performance Anomaly Suite

AE Auto Encoder

VAE Variational Auto Encoder

MoE Mixture-of-Experts

LLM Large Language Model

NVG Natural Visibility Graph

1 Introduction

HPC systems underpin scientific and commercial workloads ranging from climate modelling to large-scale Artificial Intelligence (AI) training. They refer to the use of powerful compute clusters to solve computationally intensive problems, and their availability directly determines the throughput of the research and industry they serve. Yet these systems are vulnerable: surveys report that 50% of HPC sites experience failures monthly, with costs between $100,000 and $1,000,000 per day [Hyp20], and a nine-year study at Los Alamos National Laboratory documented up to 700 failures per year on a single cluster [SG06]. Anticipating failures before they occur in systems where anomalous conditions are rare, labels are scarce, and normal behaviour shifts with every workload transition is the core challenge addressed in this report.

Existing maintenance approaches are inadequate: reactive strategies incur the full cost of downtime, while preventive fixed-schedule maintenance wastes resources without preventing spontaneous failures. Predictive maintenance uses operational data and machine learning to enable just-in-time intervention. Early HPC approaches relied on supervised classifiers such as Random Forests trained on synthetically injected faults [Net+19; Aks+21], but these struggle to generalise to real production anomalies. This report surveys a broader landscape of Machine Learning (ML) techniques, organised by data type and supervision paradigm, including autoencoders, deep learning, large language models, and graph-based methods. Figure 1 illustrates the taxonomy of methods covered.

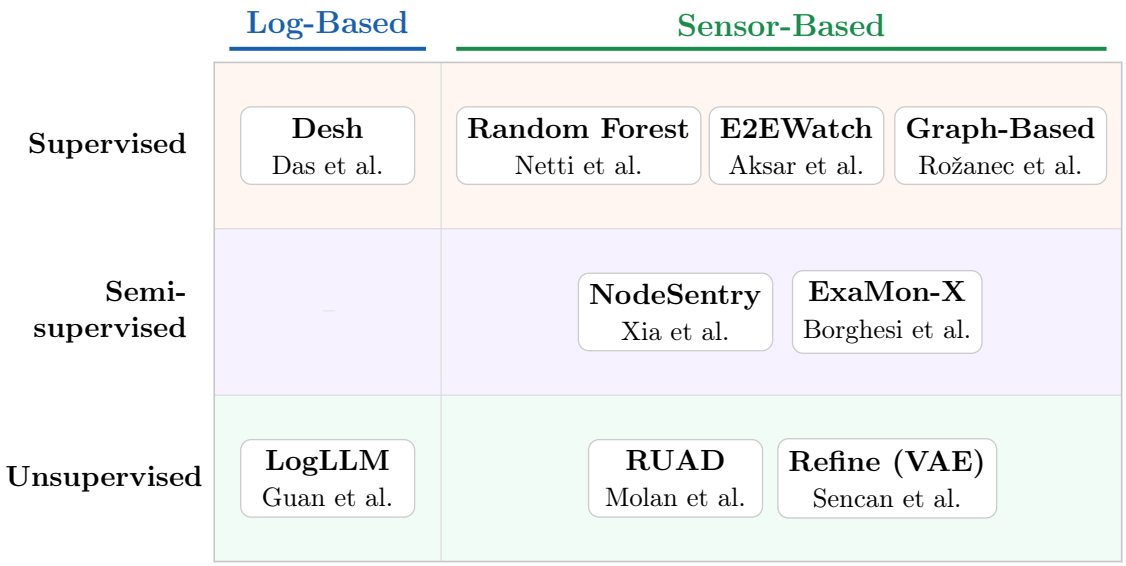

Figure 1: Taxonomy of ML methods for HPC predictive maintenance, organised by data type (columns) and supervision paradigm (rows).

Methods are evaluated against published results across a range of real and synthetic HPC datasets. The report also includes a practical reproduction experiment applying the graph-based approach of Rožanec et al. [Roz+25] to two independent node sets, providing empirical evidence on generalisability beyond the original evaluation.

Contributions

- A structured survey of nine ML methods organised by data type and supervision paradigm.
- A comparative analysis with an explicit fairness disclaimer on non-uniform evaluation conditions.
- A practical reproduction experiment on two Marconi100 node sets, providing empirical evidence on cross-node generalisability.
- Identification of open challenges: label scarcity, data contamination, system-specific generalisation, and the absence of a unified HPC benchmarking framework.

The remainder is structured as follows. Section 2 covers background concepts. Sections 3–7 survey each method family. Section 8 presents the reproduction experiment. Section 9 discusses and compares all methods. Section 10 concludes.

2 Background

2.1 Types of Maintenance

HPC maintenance falls into three strategies with distinct cost–risk trade-offs. **Reactive maintenance** addresses faults after they occur, incurring full downtime costs; a single node failure in a tightly coupled cluster can abort jobs across the entire system. **Preventive maintenance** services components on a fixed schedule regardless of condition, wasting resources and failing to prevent spontaneous failures. **Predictive maintenance** uses continuously collected operational data and ML models to anticipate failures and enable targeted, just-in-time intervention which is the kind of strategy explored throughout this report. Figure 2 illustrates the three approaches.

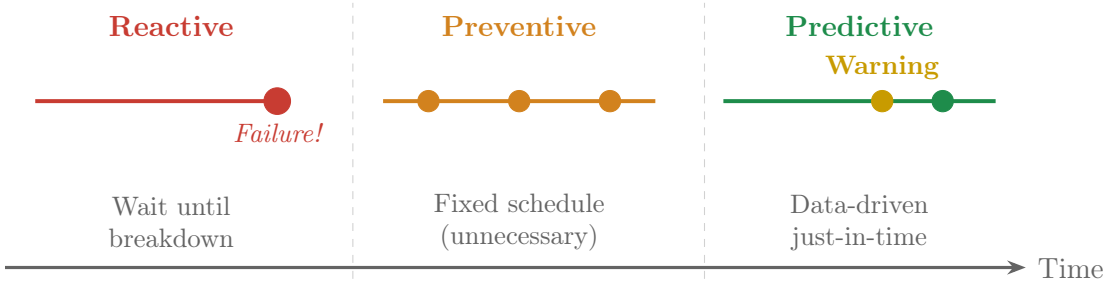

Figure 2: The three maintenance strategies: reactive, preventive, and predictive.

2.2 Types of Monitoring Data

Predictive maintenance systems rely on two complementary data sources. **Log-based data** consists of discrete, timestamped text records from the OS, firmware, and applications. Logs are rich in semantic content and well-suited for detecting fault event sequences, but challenging to parse at scale due to template drift caused by software updates. **Sensor-based (telemetry) data** consists of continuous numeric time-series from

hardware sensors such as temperatures, power draw, fan speeds, and network counters. Telemetry is regular and dense, making it suited to statistical and deep learning methods, but requires careful feature engineering to surface subtle pre-failure degradation. Table 1 summarises the key differences.

Table 1: Comparison of the two primary monitoring data types.

Property	**Log data**	**Sensor data**
Format	Unstructured text	Numeric time series
Collection rate	Event-driven	Fixed interval
Semantic content	High	Low
Key challenge	Parsing, template drift	Feature engineering, noise

2.3 Operational Data Analytics Framework

Netti et al. [Net+21] propose the Operational Data Analytics (ODA) framework for deploying ML-based monitoring in production HPC systems. It comprises three tightly coupled layers: (1) a **monitoring layer** that collects sensor and log data continuously (implemented via the Data Center Data Base (DCDB) system); (2) a **machine learning layer** that applies anomaly detection and fault classification models; and (3) a **combination layer** that translates model outputs into actionable alerts for operators and job schedulers. A central principle is that only the tight integration of all three layers delivers operational value so, a high-accuracy detector without an alerting pipeline provides no practical benefit. This end-to-end perspective motivates attention throughout this survey to deployment characteristics such as scalability and scheduler integration, not only to model accuracy.

3 Traditional Machine Learning

The earliest ML approaches to HPC predictive maintenance extract statistical features from fixed-length sensor time-series windows and train supervised classifiers on synthetically labelled fault data. Two representative works are the Random Forest framework of Netti et al. [Net+19] and the E2EWatch ensemble of Aksar et al. [Aks+21].

3.1 Random Forest: Online Fault Classification

Netti et al. [Net+19] deploy a Random Forest classifier on the Antarex cluster (ETH Zurich). The key design choice is *per-resource-type* models: separate classifiers for Central processing unit (CPU), memory, and network faults, reducing feature-space interference between subsystems. Faults are injected synthetically via Fault Injection Tool (FINJ). Raw telemetry is segmented into 60-second windows (10-second step), and statistical features (min, max, mean, variance, skewness, kurtosis, percentiles) are computed per metric per window. The framework is illustrated in Figure 3 and achieves an F1-score of 0.98.

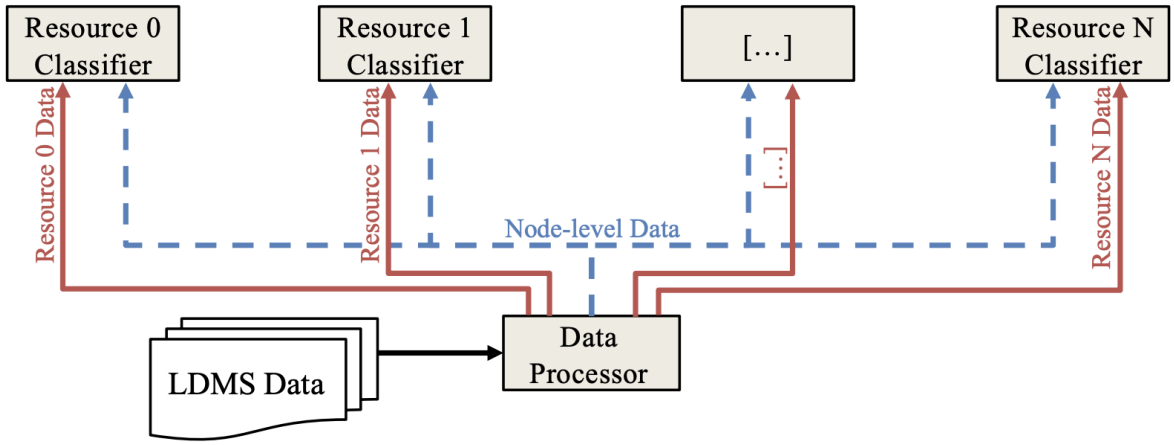

Figure 3: Per-resource Random Forest architecture of Netti et al. [Net+19]. Separate classifiers are trained per resource type and combined for final fault classification.

3.2 E2EWatch: End-to-End Anomaly Diagnosis

Aksar et al. [Aks+21] deploy E2EWatch on Eclipse (Sandia National Laboratories, 1,488 nodes), targeting *multi-class* anomaly diagnosis rather than binary detection. Telemetry from Lightweight Distributed Metric Service (LDMS) is windowed (60 s, 15 s step), statistical features extracted, and feature selection applied via the Kolmogorov-Smirnov test. An LGBM/XGBoost ensemble is then trained, deployed to a monitoring server, and visualised in a Grafana dashboard. Faults are injected via HPC Performance Anomaly Suite (HPAS). The framework achieves F1 0.91.

3.3 Comparison and Limitations

Table 2: Comparison of traditional ML approaches.

Feature	**Random Forest**	**E2EWatch**
Dataset	Antarex (ETH Zurich)	Eclipse (Sandia, 1,488 nodes)
Fault injection	Synthetic (FINJ / HPAS)	
Window size	60 s (10 s step)	60 s (15 s step)
Classifier	Random Forest	LGBM / XGBoost
Output	Fault type per resource	Multi-class diagnosis
F1-score	0.98	0.91

Both methods share three structural limitations. First, they are fully supervised, requiring labelled examples of each fault class — data that is rarely available in production. Second, training on synthetic faults (FINJ, HPAS) may not capture the diversity of real hardware degradation, firmware bugs, or workload-induced failures. Third, both are evaluated on a single cluster each, making direct transfer to other systems non-trivial. These limitations motivate the unsupervised and semi-supervised approaches in the following sections.

4 Autoencoders

Autoencoder-based approaches address the label scarcity problem of traditional ML by learning exclusively from normal system behaviour. An Auto Encoder (AE) compresses input $\mathbf{x}$ into a latent vector $\mathbf{z}$ and reconstructs it as $\hat{\mathbf{x}}$, minimising $\|\mathbf{x} - \hat{\mathbf{x}}\|^2$. A model trained only on normal data produces high reconstruction error on anomalies, which are flagged by thresholding. A Variational Auto Encoder (VAE) extends this by encoding $\mathbf{x}$ as a Gaussian distribution $(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$ rather than a fixed point, adding a KL-divergence regularisation term to keep the learned latent distribution close to a prior (typically a standard normal), enabling smooth sampling and generalisation. This produces a smoother latent space and makes VAEs more robust to contaminated training data, a critical property in production HPC environments where truly clean telemetry is rare. Table 3 summarises the key differences.

Table 3: Key differences between AE and VAE.

Property	**AE**	**VAE**
Latent representation	Fixed vector	Distribution ($\boldsymbol{\mu}$, $\boldsymbol{\sigma}^2$)
Training objective	Reconstruction loss	Reconstruction loss + KL divergence
Robustness to noise	Lower	Higher
Typical HPC use	Semi-supervised (ExaMon-X)	Unsupervised (Refine)

4.1 Refine: Robust Unsupervised Anomaly Detection

Sencan et al. [Sen+25] tackle a practical deployment problem: production HPC telemetry is never truly clean, and training a VAE on contaminated data degrades its model of normality. Refine uses an iterative filtering strategy to purge anomalous samples without ground-truth labels. A VAE is trained on the full dataset; after each cycle, samples whose reconstruction error exceeds a knee point in the error distribution are removed, and the VAE is retrained on the filtered set. This repeats until convergence. Figure 4 shows the published training diagram. At inference, anomalies are flagged by thresholding reconstruction error against the cleaned distribution. Evaluated on Eclipse (a 1,488-node production cluster at Sandia National Laboratories) and Volta (a GPU-focused research cluster) with up to 10% synthetic contamination, Refine achieves F1 0.88, and 100% accuracy on real production anomalies.

4.2 ExaMon-X: Hybrid Autoencoder Framework

Borghesi et al. [BBB23] propose ExaMon-X, deployed on Marconi (a Tier-0 supercomputer at CINECA, Bologna) and D.A.V.I.D.E. (a GPU-accelerated cluster at CINECA used for deep learning workloads). Its anomaly detection component first trains a dense AE on normal telemetry, then uses the encoder's latent representations as features for a small supervised classifier trained on a handful of labelled anomaly examples. This hybrid design reduces annotation effort while providing explicit fault typing. ExaMon-X is integrated end-to-end with the ExaMon monitoring stack and achieves F1 0.85 on real production anomalies.

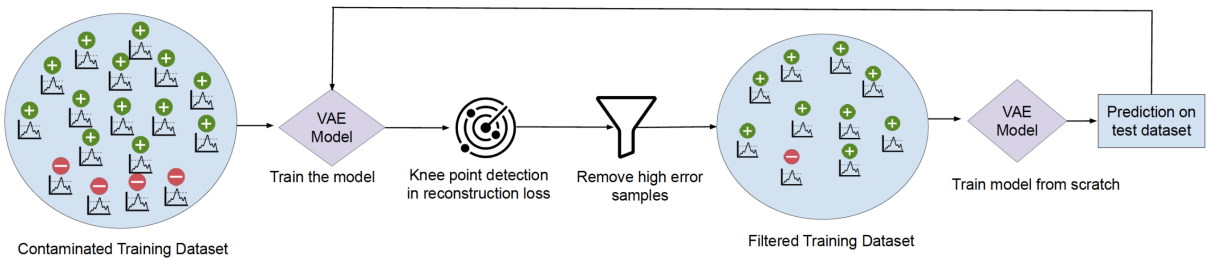

Figure 4: Iterative VAE training process of Refine [Sen+25]. High-error samples are progressively removed and the model retrained, recovering a clean normality model from contaminated data.

4.3 Comparison and Limitations

Table 4: Comparison of autoencoder-based methods.

Feature	**Refine (VAE)**	**ExaMon-X (AE)**
Dataset	Eclipse, Volta	Marconi, D.A.V.I.D.E.
Training paradigm	Fully unsupervised	Semi-supervised
Key innovation	Iterative contamination filtering	Hybrid AE + classifier
F1-score	0.88 (10% contamination)	0.85

Table 4 summarizes the comparison between the two methods. Both of them eliminate the need for large labelled fault datasets: Refine through iterative filtering, ExaMon-X through minimal annotation. Key limitations include: Refine's knee-point heuristic becomes unreliable above 10% contamination; ExaMon-X reintroduces a labelling burden for the classifier stage; and both methods require retraining when deployed on systems with different hardware profiles or workload characteristics.

5 Deep Learning

Traditional ML treats each telemetry window independently, ignoring how failures evolve over time. Deep learning captures temporal dependencies through recurrent architectures or Transformers. This section covers three representative systems: Desh [DMR18], RUAD [Mol+23], and NodeSentry [Xia+25].

5.1 Desh: Log-Based Failure Prediction

Das et al. [DMR18] deploy Desh on Cray XC30, XE6, and XC40 supercomputers (large-scale petascale systems used at US national laboratories). Desh predicts which node will fail and how many minutes remain, with a typical lead time of three minutes. A two-layer stacked LSTM with skip-gram embeddings learns failure chains, recurring log event sequences that precede node outages. The output layer regresses lead time ΔT and the failing node identity simultaneously. Evaluated on real Cray failure data, Desh achieves 85% recall.

5.2 RUAD: Recurrent Unsupervised Anomaly Detection

Molan et al. [Mol+23] propose RUAD, an LSTM-based autoencoder trained without labels, evaluated on the full ten-month history of Marconi100 (a Tier-0 supercomputer at CINECA with 980 nodes). The LSTM encoder maintains hidden state across time steps, capturing temporal dependencies that a dense encoder discards: sensor sequence → LSTM encoder → latent representation → dense decoder → reconstruction. Unsupervised training is viable because anomalies represent only 0.035% of all timestamps. RUAD achieves AUC 0.767, outperforming the prior semi-supervised AE baseline (0.747) and the clustering baseline (0.548).

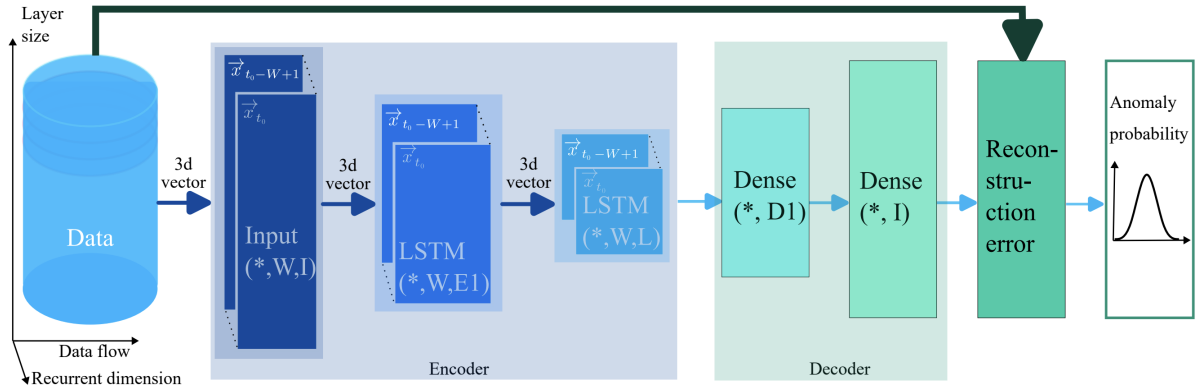

Figure 5: RUAD LSTM-AE architecture [Mol+23]. The LSTM encoder captures temporal structure; anomaly scores derive from reconstruction error.

5.3 NodeSentry: Job-Aware Transformer with MoE

Xia et al. [Xia+25] address a key blind spot: sensor profiles change with every job transition, causing single-model detectors to misclassify job changes as anomalies. NodeSentry uses a two-stage approach. Telemetry is first segmented by Slurm job boundaries and clustered via Hierarchical Agglomerative Clustering into operating-mode groups. Within each cluster, a Transformer autoencoder with a Mixture-of-Experts (MoE) feed-forward layer is trained where a gating network routes each input to the top-k relevant experts, specialising for sub-patterns without per-pattern models. NodeSentry achieves F1 0.876–0.891 on two production HPC datasets, with ablations confirming job-segmentation as the critical driver.

5.4 Comparison and Limitations

All three methods confirm that temporal context is essential. Desh shows log failure chains carry enough signal for lead-time prediction; RUAD shows unsupervised training is viable when anomalies are rare; NodeSentry shows job-awareness is critical for production deployment. Key limitations: Desh's lead-time accuracy degrades at longer horizons; RUAD requires per-node retraining on hardware changes; NodeSentry has a detection blind spot in the first minutes of each new job.

Table 5: Comparison of deep learning approaches for HPC predictive maintenance.

Feature	**Desh**	**RUAD**	**NodeSentry**
Data type	Log events	Sensor time-series	Sensor time-series
Dataset	Cray XC30/XE6/XC40	Marconi100 (980 nodes)	Production HPC
Model	Stacked LSTM	LSTM autoencoder	Transformer + MoE
Supervision	Supervised	Unsupervised	Semi-supervised
Anomalies		Real	
Performance	Recall 85%	AUC 0.767	F1 0.876–0.891

6 LLMs for Log Anomaly Detection

Traditional log anomaly detection extracts fixed templates from raw messages and applies sequence models to template IDs; a pipeline that might breaks when software updates introduce new log formats. LLMs bypass the parser partially, mapping unseen log messages to semantic vectors through pre-trained language understanding.

This section covers LogLLM [Gua+24]. Although it was not designed specifically for HPC infrastructure, its strong performance on general software system log datasets demonstrates capabilities directly relevant to HPC monitoring: robustness to evolving log formats caused by frequent firmware and OS updates, semantic understanding of free-form error messages without requiring a fixed log parser, and efficient handling of large log volumes through per-message embedding. These properties address longstanding challenges in HPC log analysis and make LogLLM a compelling candidate for adaptation to production HPC environments.

6.1 Architecture and Training

LogLLM combines a transformer encoder and decoder in a single pipeline:

1. **BERT** (bert-base-uncased, 110M parameters) processes each log message individually, producing a fixed-length semantic embedding via bidirectional attention — capturing full message meaning regardless of whether the exact template was seen during training.

2. A learned **projector** maps BERT's output into Llama's embedding space, resolving the representational mismatch between the two pre-training objectives.

3. **Llama-3-8B** receives the sequence of projected per-message embeddings and classifies the entire log window as normal or anomalous.

Using per-message BERT embeddings rather than tokenising the raw log string avoids out-of-memory failures for long sequences. Training proceeds in three stages: BERT fine-tuning, projector alignment with frozen LLMs, then joint end-to-end fine-tuning.

6.2 Performance and HPC Relevance

LogLLM outperforms all baselines as reported in Table 6. For HPC, the key promise is robustness to unstable logs caused by firmware and OS updates, and the potential to process hours of log history using modern LLMs' large context windows.

Table 6: The average F1 Score of LogLLM compared to other methods over four datasets [Gua+24].

Method	Architecture	Avg F1-score
LogLLM	BERT + projector + Llama-3-8B	**0.959**
NeuralLog	BERT + transformer classifier	0.893
RAPID	BERT + distance-based comparison	0.602
LogBERT	BERT reconstruction	0.456
FastLogAD	BERT reconstruction	0.341

7 Graph-Based Anomaly Detection

Rožanec et al. [Roz+25] propose representing sensor time-series as graphs, encoding their topological structure via graph embeddings, and feeding these into a gradient-boosted classifier to predict compute node downtimes. Unlike flat statistical features, graph representations can embed domain knowledge like sensor identity and type directly into the feature vector, providing structural context about what is being measured.

7.1 Method

The pipeline is illustrated in Figure 6 and proceeds in five stages. Raw telemetry from 16 Marconi100 nodes (13 sensor metrics, sampled every 15 minutes, March 2020 – September 2022) is first **pre-processed**: missing values are forward-filled, a change detection algorithm segments each signal into piecewise-constant regions, and values are quantised into five bins, yielding a discrete sequence of system states.

Each sensor's state window (up to ten preceding states) is then converted into a Natural Visibility Graph (NVG), where each data point is a node and two nodes are connected if their line-of-sight is unobstructed by intermediate points; a heuristic that captures the shape and monotonicity of the series. The NVGs are enriched with sensor ID and sensor type metadata and encoded into fixed-length vectors using Graph2Vec [Nar+17], which generates random walks over each graph and trains a skip-gram model to produce the embeddings. Embeddings from all sensors in a state are concatenated into one feature vector.

Finally, a CatBoostclassifier is trained in two configurations: a *local* model per node and a *global* model pooling all 16 nodes, evaluated at three prediction horizons (1–3 states ahead, ≈165–495 minutes).

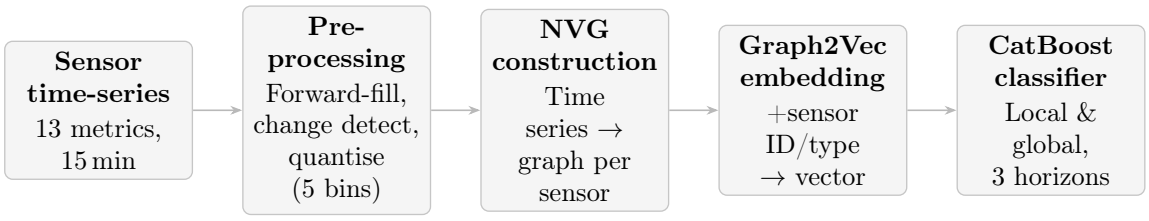

Figure 6: End-to-end pipeline of the graph-based approach [Roz+25].

7.2 Results and Limitations

Table 7: Mean AUC ROC across 16 Marconi100 nodes at three prediction horizons [Roz+25].

Horizon	Local model	Global model
1 state (≈165 min)	0.848	**0.923**
2 states (≈330 min)	0.855	**0.893**
3 states (≈495 min)	0.800	**0.887**

Global models consistently outperform local models, indicating that pooling multi-node data captures failure patterns that individual node histories miss. The best overall AUC ROC of 0.8085 is achieved with enriched graph embeddings. Key limitations are the small evaluation scale (16 nodes, single rack), the sensitivity of the preprocessing pipeline to its hyperparameters, and the coarse minimum prediction horizon of approximately 165 minutes.

8 Discussion

8.1 Fairness Disclaimer

The methods surveyed cannot be compared on strictly equal terms. Each was evaluated on a different HPC system, at a different scale, against different anomaly types, and using different metrics. Supervised methods trained on synthetic faults have an inherent advantage on clean benchmark data that does not reflect production noise. Unsupervised methods evaluated on real anomalies face a harder detection problem but demonstrate more realistic deployment conditions. F1-score, AUC ROC, and recall reward different aspects of detector behaviour and are not directly comparable.

8.2 Comprehensive Comparison

Table 8 summarises all nine methods across dimensions relevant to HPC deployment decisions.

8.3 Key Observations

- **Synthetic anomalies inflate performance.** Random Forest (F1 0.98) and E2EWatch (F1 0.91) top the table but are trained on controlled synthetic faults (FINJ, HPAS), making their numbers less comparable to methods evaluated on real production anomalies.
- **Unsupervised methods are most deployment-ready.** Refine and RUAD require no labelled data and can be deployed on a new HPC system without annotation effort — a critical advantage given label scarcity in production environments.
- **Global models generally outperform local models.** Pooling data across nodes improves detection, but assumes sufficient telemetry homogeneity — an assumption that may break across heterogeneous hardware generations.

Table 8: Comparison of ML methods for HPC predictive maintenance. Results are not directly comparable (see Section 8.1).

Method	Metric	Anomalies	Supervision	Data type	Scalability
Random Forest	F1 0.98	Synthetic	Supervised	Statistical metrics	Global
E2EWatch	F1 0.91	Synthetic	Supervised	Statistical metrics	Global
LogLLM	F1 0.959	Real	Supervised	Textual logs	Global
NodeSentry	F1 0.876–0.891	Real	Semi-supervised	Time series	Per-node
Refine (VAE)	F1 0.88	Real	Unsupervised	Time series	Per-node
ExaMon-X	F1 0.85	Real	Semi-supervised	Time series	Per-node
RUAD	AUC 0.767	Real	Unsupervised	Time series	Per-node
Desh	Recall 85%	Real	Supervised	Textual logs	Per-node
Graph-based	AUC 0.809	Real	Supervised	Embeddings	Global/per-node

- **Log and sensor methods are complementary.** Log-based methods (Desh, LogLLM) suit discrete fault event detection and lead-time prediction; sensor-based methods (RUAD, Refine, NodeSentry) suit continuous health monitoring of gradual degradation. A production system would benefit from combining both.
- **No single method dominates.** Each occupies a different trade-off between label requirements, computational cost, and generalisability. The right choice depends on data availability, system homogeneity, and operational latency requirements.

9 Practical Implementation

To evaluate how well existing approaches generalise beyond their original experimental setups, we decided to reproduce one of the surveyed methods ourselves. We chose the graph-based approach of Rožanec et al. [Roz+25] due to its distinctive preprocessing and training pipeline. Since the source code is not publicly available, we implemented the pipeline ourselves based on the description provided in the original paper. We applied our implementation to two independent node sets drawn from the publicly available Marconi100 dataset [Bor+23]. The original study evaluated only 16 nodes from a single rack; our experiment tests whether the same approach yields reliable predictions on different node groups from the same supercomputer.

9.1 Experimental Setup

We implemented the five-stage pipeline described in Section 7.1 in Python, following the published methodology: forward-fill imputation, change-detection segmentation, five-bin quantisation, Natural Visibility Graph construction, Graph2Vec embedding enriched with sensor ID and type, and CatBoost classification. The 13 sensor features used are listed in Table 9. Each model is evaluated at three prediction horizons (1–3 states ahead, corresponding to approximately 165, 330, and 495 minutes) using mean AUC ROC.

Two CatBoost model configurations are trained and compared:

- **Local model:** one model trained per compute node, using only that node's own historical data. This represents a node-specific approach that requires no cross-node data sharing.

Table 9: Sensor features selected for the practical implementation, following the feature set of Rožanec et al. [Roz+25].

Feature	Description
`ambient_avg`	Ambient temperature
`dimm0_temp_avg`	DIMM0 temperature
`fan_disk_power_avg`	Fan disk power
`gpu0_core_temp_avg`	GPU0 core temperature
`gpu0_mem_temp_avg`	GPU0 memory temperature
`p0_io_power_avg`	P0 I/O power
`p0_mem_power_avg`	P0 memory power
`p0_power_avg`	P0 power
`ps0_input_power_avg`	PS0 input power
`ps0_output_curre_avg`	PS0 output current
`ps0_output_volta_avg`	PS0 output voltage
`fan0_0_avg`	Fan speed
`p0_vdd_temp_avg`	P0 VDD temperature

- **Global model:** one model trained on the pooled data of all nodes in the set, then evaluated on each node individually.

The two node sets are: **Node Set 1** (17 nodes, IDs 20–29, 31–33, 35–38) and **Node Set 2** (17 nodes, IDs 40–59).

9.2 Results and Discussion

Figure 7 shows the mean AUC ROC for both model types across all three horizons on both node sets.

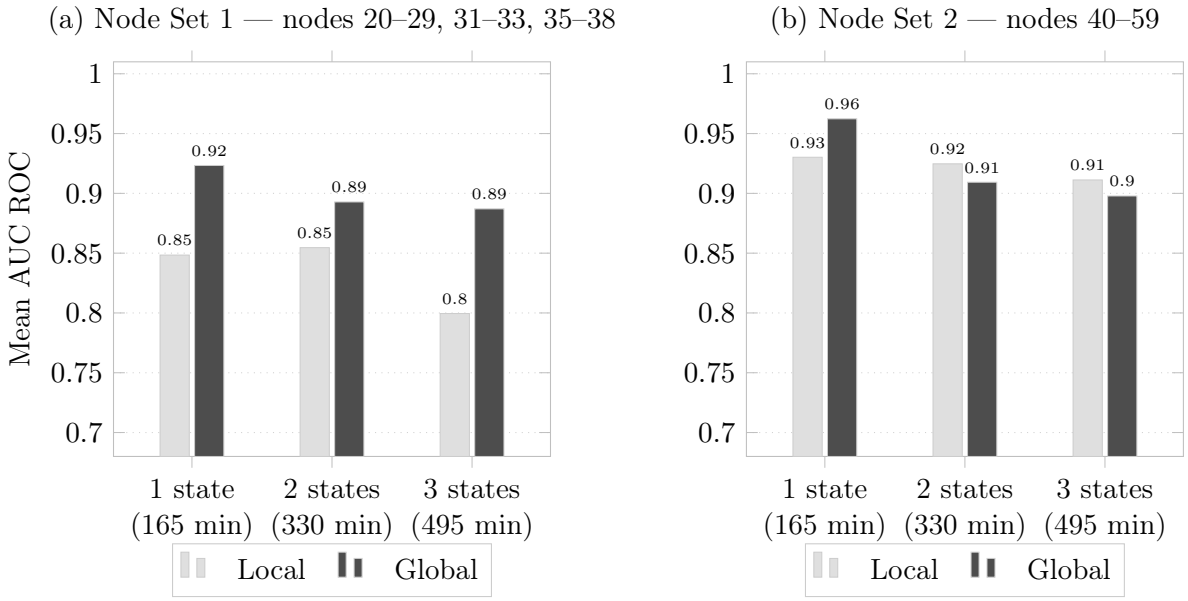

Figure 7: Mean AUC ROC of local and global CatBoost models across three prediction horizons on two independent Marconi100 node sets.

Both node sets achieve AUC scores well above 0.80 across all horizons, confirming

that the pipeline generalises beyond the single rack used in the original study. Node Set 2 consistently outperforms Node Set 1 (0.90–0.96 vs. 0.80–0.92), suggesting that failure behaviour in nodes 40–59 is more regular and thus easier to model. This inter-rack variability underscores that performance should be validated independently for each deployment target.

The global versus local ranking differs between the two sets. In Node Set 1, the global model dominates across all horizons, replicating the original paper's finding that cross-node pooling improves detection. In Node Set 2, local and global models are competitive, with local models slightly ahead at the two longer horizons. This reversal indicates that the benefit of data pooling depends on the homogeneity of failure patterns within a node set which is a practically important consideration when deciding whether to deploy a shared or per-node model in a production cluster.

10 Conclusion

This report surveyed machine learning approaches for predictive maintenance on HPC nodes across five method families: traditional supervised classifiers, autoencoder-based anomaly detection, deep learning with temporal models, LLM-based log analysis, and graph-based time-series representations. The survey reveals a clear progression in how the field has responded to the fundamental constraints of production HPC environments. Early supervised methods such as Random Forest and E2EWatch achieve high reported accuracy but rely on synthetic fault injection and single-system evaluation, limiting their practical applicability. Subsequent unsupervised and semi-supervised approaches (Refine, RUAD, and NodeSentry) address label scarcity more realistically, though each introduces its own deployment trade-off between annotation burden, retraining cost, and sensitivity to workload variation.

Three challenges recur consistently across all families. *Label scarcity* constrains the applicability of supervised methods, since production HPC systems rarely record structured fault annotations. *Data contamination* degrades unsupervised models trained on real telemetry that contains unlabelled anomalies. And *system-specific generalisation* remains unsolved: models trained on one cluster routinely require retraining or re-evaluation before being trusted on another, as the practical experiment in Section 9 confirms through inter-rack performance differences on Marconi100 itself.

The practical experiment adds a concrete finding to the literature: the graph-based pipeline of Rožanec et al. [Roz+25] generalises beyond its original single-rack evaluation, achieving AUC above 0.80 across both independent node sets, but with meaningful inter-rack variability that underscores the need for deployment-specific validation. Among the methods surveyed, unsupervised approaches are the most immediately deployable in new environments, while graph-based and LLM-based methods show the strongest potential for cross-system generalisation as their respective fields mature.

11 Future Work

Three directions are most pressing for advancing HPC predictive maintenance in practice.

Unified HPC benchmarking framework. No standardised benchmark exists for comparing anomaly detection methods on HPC telemetry. Each method surveyed was evaluated on a different cluster, with different fault types, evaluation metrics, and data collection pipelines, making direct comparison impossible. A community benchmark providing shared datasets, standardised fault injection protocols, and common metrics would allow methods to be compared on equal terms and accelerate progress in the field in the same way that ImageNet shaped the development of computer vision.

Data labelling tools and standards. Labelled anomaly data remains the limiting resource for supervised and semi-supervised methods. Active learning pipelines that surface candidate anomalies for operator confirmation, or standardised schemas for recording fault events alongside operational telemetry, would substantially reduce the annotation burden and make supervised methods viable in more production settings.

Integration with the GWDG HPC infrastructure. A natural and locally relevant next step is to evaluate the most promising methods on the production systems operated by the GWDG at Georg-August-Universität Göttingen. The GWDG infrastructure is heterogeneous in hardware generation and workload profile, making it a demanding testbed for methods that claim cross-system generalisability. Applying the approaches reviewed in this report to GWDG telemetry would provide direct empirical evidence on how well results from systems such as Marconi100 transfer to a different operational environment

References

[Aks+21] Burak Aksar et al. "E2EWatch: An End-to-End Anomaly Diagnosis Framework for Production HPC Systems". In: *Proceedings of the International Conference on Parallel Processing (ICPP)*. ACM, 2021. DOI: 10.1007/978-3-030-85665-6_5.

[BBB23] Andrea Borghesi, Alessio Burrello, and Andrea Bartolini. "ExaMon-X: a Predictive Maintenance Framework for Automatic Monitoring in Industrial IoT Systems". In: *IEEE Internet of Things Journal* 10.4 (2023), pp. 2995–3005. DOI: 10.1109/JIOT.2021.3125885.

[Bor+23] Andrea Borghesi et al. *M100 dataset: time-aggregated data for anomaly detection.* Version 1.0.0. Zenodo, Jan. 2023. DOI: 10.5281/zenodo.7541722.

[DMR18] Saurabh Das, Frank Mueller, and Barry Rountree. "Desh: Deep Learning for System Health Prediction of Lead Times to Failure in HPC". In: *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*. ACM, 2018, pp. 40–51. DOI: 10.1145/3208040.3208051.

[Gua+24] Wei Guan et al. "LogLLM: Log-Based Anomaly Detection Using Large Language Models". In: *arXiv preprint* (2024). DOI: 10.48550/arXiv.2411.08561.

[Hyp20] Hyperion Research. *HPC Market Update and Trends.* Tech. rep. Hyperion Research, 2020.

[Mol+23] Martin Molan et al. "RUAD: Recurrent Unsupervised Anomaly Detection in High Performance Computing Systems". In: *Future Generation Computer Systems* 145 (2023), pp. 229–240. DOI: 10.48550/arXiv.2208.13169.

[Nar+17] Annamalai Narayanan et al. "graph2vec: Learning Distributed Representations of Graphs". In: *CoRR* abs/1707.05005 (2017). arXiv: 1707.05005. URL: http://arxiv.org/abs/1707.05005.

[Net+19] Alessio Netti et al. "Online Fault Classification in HPC Systems Through Machine Learning". In: *Euro-Par 2019: Parallel Processing.* Springer, 2019, pp. 1–16. DOI: 10.1007/978-3-030-29400-7_1.

[Net+21] Alessio Netti et al. "From Facility to Application Sensor Data: Modular, Continuous and Holistic Monitoring with DCDB". In: *Concurrency and Computation: Practice and Experience* 33.6 (2021). DOI: 10.1145/3295500.335619.

[Roz+25] Joze M. Rozanec et al. "Learning anomalies from graph: predicting compute node failures on HPC clusters". In: *Proceedings of the 6th Northern Lights Deep Learning Conference (NLDL)*. Ed. by Tetiana Lutchyn, Adín Ramírez Rivera, and Benjamin Ricaud. Vol. 265. Proceedings of Machine Learning Research. PMLR, July 2025, pp. 213–219. URL: https://proceedings.mlr.press/v265/rozanec25a.html.

[Sen+25] Ahmet Sencan et al. "Refine: A Robust Approach to Unsupervised Anomaly Detection for Production HPC Systems". In: *ISC High Performance 2025.* IEEE, 2025. DOI: 10.23919/ISC.2025.11018307.

[SG06] Bianca Schroeder and Garth A. Gibson. "A Large-Scale Study of Failures in High-Performance Computing Systems". In: *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*. Philadelphia, PA, USA, 2006, pp. 249–258. DOI: 10.1109/DSN.2006.5.

[Xia+25] Lingfei Xia et al. "Effective Node-Level Anomaly Detection in HPC Systems via Coarse-Grained Clustering and Fine-Grained Model Sharing". In: *Proceedings of the ACM International Conference on Supercomputing (ICS)*. ACM, 2025. DOI: 10.1145/3712285.3759794.